UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

The Mask: Masking the effects of Edge Nodes being unavailable

Ilia Zhakun

INF-3990: Master's Thesis in Computer Science
November 2021

To Kate, Lev and Lilia.

“Simplicity is prerequisite for reliability.”
–Edsger Dijkstra

Abstract

The arctic tundra is observed to collect data to be used for climate research. Data can be collected by cyber-physical computers with sensors. However, the arctic tundra has a limited availability of energy. Consequently, the nodes rely on batteries and sleep most of the time to increase the battery-limited operational lifetime. In addition, only a few nodes can expect to be in reach of a back-haul wireless data network. Consequently, the nodes have on-node wireless local area networks to reach nearby neighbor nodes.

To increase the availability for remote clients to the data collected by the nodes, a set of shadow nodes are used. These are always on, and always have access to a back-haul network. Data from an edge node on the arctic tundra propagates to the shadow nodes either directly over a back-haul network, or via a neighbor node with a back-haul network. The purpose is to make the data produced by an edge node available to a client even when the edge node sleeps or no network access is available.

A statistical analysis is done to characterize the prototype's behavior under a set of edge-node behaviors. To validate the statistical analysis a prototype system is developed and used in a set of performance-measuring experiments. Experiments are done with 10 to 1,000,000 nodes, different probabilities of nodes being awake, and different probabilities of the back-haul network being available. Edge and shadow nodes are emulated as Go functions and executed on a high-performance computer with thousands of cores. Different wireless networks are emulated albeit in a simplified way. A run-time simulation system is developed to control the prototype and conduct the experiments.

The results for the prototype show that if the single synchronization chance is low or the desired time to get the latest data should be minimized, an additional data delivery path should be considered on the edge node's side. Synchronization via the right neighbor principle adds an extra communication channel which increases the data availability level by 50%-100%, but the resource demand grows by 30% per unit. The time required to get the latest data from edge nodes decreases by a factor of 1.75.

The results for the simulation show that the cumulative network throughput of approximately ≈ 2100 MB/s and the Generated Data Amount ≈ 25000 MB/s can be achieved at the cost of ≈ 80 KB RAM per emulated node.

The results show that the statistical analysis and the results from the prototype as used by the simulation system match, but the statistical expectation considers a limited range of factors. Statistically derived values can be used as the input for the simulation, where they would be adjusted to get a more comprehensive result.

The conclusions are that the Mask provides instant access to data storage for edge nodes. The Mask is fronted to clients which become able to retrieve the data asynchronously, even when edge nodes are offline.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Professor Otto Anshus, for his guidance, expert advice, prompt and precise feedback throughout the development process of this master's project. Thank you to the DAO project, Research Council of Norway, project number 270672, for providing an interesting research context for my master project.

I would also like to say thanks to the Department of Computer Science of the University of Tromsø and especially to Jan Fuglesteg for the personalized approach to the study process planning.

My sincere thanks also go to my employer, the IT Department of the UiT, and especially to the project manager Steinar Trædal-Henden for his comprehension and for the high level of flexibility in the work process, which made completing this thesis possible. I am also grateful to NRIS - Norwegian Research Infrastructure Services for providing access to the high-performance computing resources needed for the simulation.

Last but not least, I would also like to thank my family. Thanks to my wife Ekaterina for her great support even while conducting her PhD research. Thanks to my 3 years old son Lev for his patience while his dad was away to do the master's project work.

Contents

List of Figures

List of Tables

List of Equations

List of Listings

List of Definitions

List of Abbreviations

BLE Bluetooth 5.2

Bps Bytes per second

COAT Climate-ecological Observatory for Arctic Tundra

DAO The Distributed Arctic Observatory

GDA Generated Data Amount

HPC High Performance Computing

ID Identificator

IO Input/Output

Kbps Kilobytes per second

MPI Message Passing Interface

MTU Maximum Transfer Unit

NEIB Neighbour Protocol

NRD Node Resource Demand

OS Operating System

OU Observation Unit

PMF Probability Mass Function

PTT Packet Travel Time

RAM Random-Access Memory

SR Shadow Router

SU Shadow Unit

UiT University of Tromsø

UR Unit Router

1

Introduction

Observation of wildlife is an exciting and challenging process. It provides information about nature using non-invasive means which are environmentally friendly for biocenosis. Such research requires complex systems for monitoring, data collection, processing, and transfer. One example is the network of observation units in the Arctic tundra. Such a system comprises a set of cyber-physical nodes with sensors. Only a few nodes have a back-haul network. However, they all have one or several on-node local area networks allowing a single ad hoc node-to-node connection at a time.

The nodes are battery-limited and mostly sleep to conserve energy.

This project proposes to have a remote always-on node per mostly-sleeping edge node.

The purpose is to make the data produced by an edge node available to a client even when the edge node sleeps.

A prototype of the proposed approach is developed. The prototype comprises emulated virtual edge nodes, emulated virtual always-on nodes, and emulated networks. The prototype is executed on high-performance computer with thousands of cores.

The performance characteristics of the prototype are documented through a set of performance measuring experiments.

1.1 COAT Project

Climate-ecological Observatory for Arctic Tundra (COAT)[1] project designed and implemented a network of observation units for Norwegian Arctic tundra [13]. Those units are spread across the Varanger Peninsula and Svalbard. The COAT project provides the data to other researchers and projects.

The Definition 1. Observation Unit

Observation Unit (OU) is an edge node in the observation network that monitors the environment, stores and transfers the received data over the network.

Edge nodes - Observation Units are placed in remote areas with no usual infrastructure such as roads or power supply. That is why OUs are battery-powered. Battery replacement and hardware repairs require physical manipulation. Such expeditions are resource-demanding and can't be conducted on a daily basis.

OUs can be fully unavailable because of technical failures or external reasons such as interaction with animals or severe weather.

The connection to the back-haul network is provided for a limited time. OUs are in sleep mode most of the time in order to save battery. When OUs wake up, they check the back-haul network availability in order to synchronize and transfer the data. If there is no connection, nodes make another synchronization attempt during the next wake-up.

Several Observation Units can operate in the existing network reach of each other. This is a neighborhood of nodes.

The Definition 2. Neighborhood

Neighborhood is the abstraction that includes all the Observation Units operating in the existing network reach of each other.

The Definition 3. Observation Network

Observation Network is the abstraction that includes all the Observation Units, combining all the neighborhoods of OUs together.

1. The COAT project: `https://www.coat.no` (accessed: 2021-11-10)

The Definition 4. Shadow Unit

There is a set of shadow nodes located in, say, the project's data center. Those nodes make the data available to clients. A shadow node contains a recent copy of data from the Observation Unit.

A Shadow Unit (SU) is the permanently available node that contains the replica of the OU and which is fronted to the user.

The Definition 5. Shadow Network

All the Shadow Units form a shadow network. Shadow Network is the abstraction that includes all the Shadow Units and combines all the local networks of SUs together.

The shadow network is exposed to the user, it has enough energy to be always-on and network connection with sufficient bandwidth. When an OU becomes available, it gets the latest software updates from, and transfers the observational data to, the shadow network. An SU, that represents one particular OU, receives media, environmental data and metadata. Even if the SU has not received updates for a longer period, the old data would be still available for users.

1.2 Distributed systems for data collection

Network of Observation Units is a practical scientific tool for empirical data collection. As any distributed computer system, it has goals and challenges.

1.2.1 Goals

The main goal of the COAT network is wildlife monitoring. It provides data for complex environmental research purposes. Data collected in the Arctic tundra helps to identify trends in the wild nature and make forecasts.

The Distributed Arctic Observatory (DAO)[2] project maintains the entire system [39]. Additionally, such a system serves as a test platform for improvements of network communication and orchestration patterns.

2. The DAO project: `https://site.uit.no/dao` (accessed: 2021-11-10)

1.2.2 Challenges

There are several technical challenges actual for the DAO nodes.

First, physical limitations of battery capacity require design based on the energy-saving principle. The unavailability of a back-haul network at many nodes means that a temporary network channel with sufficient bandwidth must be delivered to OUs. Distributed nature of the network assumes that extra attention must be paid to data synchronization, communication, recovery from failures, and node orchestration.

1.3 Problem identification

After combining goals and challenges, one problem can be identified. The edge nodes are usually not available when clients need data from them. A shadow network of always-on nodes caching the data from the edge nodes can be used to mask edge node unavailability. However, observational data in shadow network can be outdated, not synchronized due to edge nodes' unavailability.

1.3.1 Motivation

The motivation here is to suggest a way how to increase the data availability level. Recent observational data should be accessible by users.

1.3.2 Research question

What are the benefit and cost of masking edge node's unavailability?

1.3.3 Research tasks

It is required to fulfill the following list of research tasks in order to come closer to the research question solution:

1. create a simplified statistical model to get the expectation and the starting point of the input for the prototype;

2. develop a prototype suited for emulation of several thousand OUs;

3. develop a shadow network emulation connected to the OU network;

4. emulate several network types with their technical characteristics;

5. make the prototype suitable for execution in hardware environments with large memory;

6. run set of experiments to test scaling ability, expected network behavior, resource demand;

7. make assumptions for the real-world network behavior based on the results of the simulation.

1.3.4 Additional tasks

Two additional tasks can help to make a solution more comprehensive:

1. suggest and implement an algorithm for synchronization and data transfer using neighbor nodes;

2. assess resource demand and data availability effect of employing synchronization via a neighbor node;

3. add support for hundreds of thousands OUs to check scaling effects.

1.3.5 Scope and limitations

The scope of the thesis includes modeling of general data delivery paths, the schematic architecture of observational network represented by a computer simulation with limited functionality. When it comes to limitations, the prototyping doesn't assume real-world testing or interaction with the real observational network. Another limitation is that the model doesn't include all the characteristics of the real-world network. Only some aspects that are subjects for study in this thesis are taken into consideration in the model.

For instance, several types of networks with physical limitations are emulated, but software updates propagation, clock synchronization patterns, energy consumption, data storage and processing aspects can be absent or oversimplified. Both the statistical approximation and the prototype are simplified representations of the OU network. Results derived from a set of experiments do not pretend for absolute accuracy, as the simulation is not the exact copy of a real-world observation network.

1.4 Thesis outline

This master's thesis has the following structure:

Chapter 1 - Introduction is the current chapter that defines the motivation, research question, tasks, scope and limitations;

Chapter 2 - Theoretical Background illustrates main concepts and principles employed in the master's thesis;

Chapter 4 - Related Work shortly presents results of work of similar projects with similar goals;

Chapter 3 - Methodology describes the chosen research method and helps to logically connect all parts of the thesis;

Chapter 5 - Statistical Expectation provides the preliminary input for the simulation and demonstrates how probability is estimated;

Chapter 6 - Prototype outlines architecture, design and implementation of the prototype - the simulation of the network of ou;

Chapter 7 - Evaluation contains benchmarking, stress tests of the prototype and description of experiments;

Chapter 8 - Results & Discussion lists the research results and highlights their practical meaning;

Chapter 9 - Contributions - summarizes the main findings;

Chapter 10 - Future Work suggests possible vectors of the future development;

Chapter 11 - Conclusion gives a short summary of the master's thesis.

2

Theoretical Background

The goal of this chapter is to clarify some theoretical terms and concepts which are not covered by other parts of the thesis. Such clarification eases the overall model perception and serves as an introduction to the prototype development process.

2.1 Observational distributed systems

The idea to use a distributed system of observational nodes to monitor the wildlife and environment is not new. Some relevant examples are discussed in chapter 4 - Related Work. One of the primary questions in such systems is how to organize the network and balance the workload among network members. Sensors are the main data sources for observational distributed systems and for the COAT project [13] in particular. Another important question is how and where to process the data before the transfer to the end-user.

2.1.1 Edge computing

If the data is sent right away, the observational node would remain idle, would have unused computational resources, but save some energy. If the data is processed in place, there will be a growing demand for computational resources. Edge computing means that the data is processed closer to its source. From the

perspective of data security, the sensitive information is stored locally avoiding centralized data aggregators [48]. From the point of workload balancing, edge computing reduces the network workload by processing parts of data in place [35, p.17]. A schematic representation of the data flow and the workload distribution is illustrated in figure 2.1 (adaptation of illustration of "cloud-fog-edge" principle [8] to the network of Observation Units).

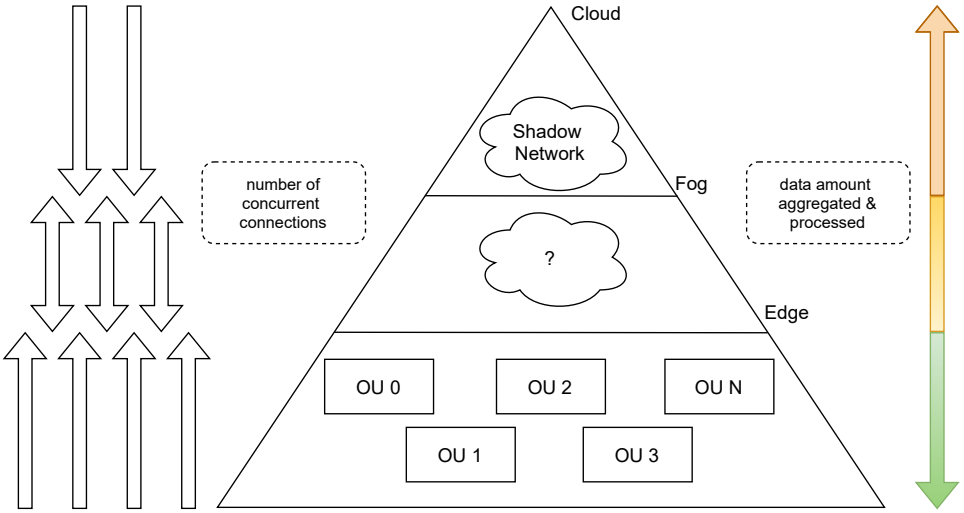


Figure 2.1: Cloud-Edge network principle (adaptation of illustration [8])

The simulation of the observational network can be described using the cloud-edge approach [35, p.17]. Figure 2.1 shows how observational network elements correlate with cloud-edge principles. Here, in figure 2.1, Observation Units represent edge nodes, the shadow network is a cloud with sufficient computational power. Thus, the simulated Observation Unit network can be seen as a network of the edge nodes which partially do the data processing because they collect, store and send the sensor data. The energy-saving perspective assumes extra attention to the balance between in-place data processing and the network availability rate. Even permanently available networks with sufficient bandwidth like 5G may require in-place data processing, edge computation can reduce the communication overhead [55]. Most data processing resources should be spent where they are inexpensive - in the data center/cloud, as illustrated in figure 2.1. From the perspective of the data flow forwarding and processing, some fog abstraction as shown in figure 2.1 might be required in future work (see chapter 10). The simulation can benefit from an adaptive mechanism of edge load distribution [30]. More details on resource-demanding cases are discussed in section 6.3.8 (NEIB mode), results can be found in the experiment 4 description (see section 7.8).

2.1.2 Parallel & concurrent computing

As illustrated in figure 2.1, there may be fewer concurrent connections on the way from edge to cloud, but the data amount remains the same. Receivers sometimes have to process data from several senders. It is natural for routers in networking and for simulations emulating several independent data sources. For several simultaneous data streams, parallel computing may help to distribute the workload among several threads, processes, cores, or nodes. In accordance with [42], in parallel computing, several processes are solving the same issue simultaneously, while in concurrent programming several processes are running simultaneously.

In the prototype construction (chapter 6), the principles of parallel computing help, for example, to simultaneously extract router data from the network using several goroutines. Nevertheless, all units are running concurrently: independently, but at the same time.

2.2 Network scalability

When it comes to communication between nodes, there is a need for a network - an abstraction for the data exchange. Simultaneous connection requires synchronization. Simultaneous access to shared variables assumes mutual exclusion. The network can be static or dynamic, but the number of nodes may vary from run to run. It means that routines in the prototype should be flexible to handle any number of network elements within a given range.

2.2.1 Communication & synchronization

Despite the shared memory availability and support for the global configuration, the simulation should emulate independent edge nodes to have similar behavior to the network of Observation Units. Communication happens both between members of the same local network and between members of different remote networks. That is why data transfer algorithms are needed.

Parameters are passed from the level of the shadow network in a form of multicast. Multicast takes place when the data from the control center is propagated to all the edge nodes via routers and other network layers [51]. The entire network communication is based on the message-queuing model, where data is transferred in chunks and can be delivered even after the original sender was shut down [51]. Channels in Go support buffering [20] and can accept several or several hundreds of messages.

The intra-node communication can be based on the persistent message-oriented model [51]. Such systems do not require all members to be active at the moment of connection [51]. Message passing via Go channels using messages with key primitives can be used to implement this kind of model. This approach opens for the data synchronization via neighbor nodes (see 6.3.8).

2.2.2 Mutex

When several nodes are trying to send the data via the same non-private channel, simultaneous access may affect the message order between processes. Despite the fact that the mutual exclusion in shared memory is mainly an issue for systems with simultaneous multiprocessor access to the data [4], race conditions can occur between several processes as well. Some orchestration approaches are lock-free and non-blocking, but if data channels would be accessed simultaneously by several processes, the lock-based approach should be considered. Maekawa's algorithm [33] is permission-based and demonstrates good results when applied to communication within linked lists structures [32]. That is why the principle of asking for permission before entering the critical region is applied in the prototype. A simplified permission-based lock implementation is described in chapter 6.

2.3 Network topology

There is an issue, unique for distributed systems - incomplete knowledge about the distant elements. This issue may be partially resolved by data synchronization [43]. But what is the proper organization of the network and how many ranks and roles there should be? Peleg [43] introduces locality-sensitive distributed algorithms. Even in case of extensive network growth, build in a locality-sensitive way, the algorithm will operate in the given relatively small part of the data path without knowledge on the entire system. So the principle of locality is employed in the simulation prototyping, more details on design baselines are in section 6.2.

2.3.1 Ranked networks

Some systems scale without errors, for some scaling may result in failures after extensive growth. Synchronization algorithms may not cope with a changed number of nodes in some cases. Observation Units are homogeneous edges - identical nodes of the same level. All of them have to follow identical paths to push the data to its destination as explained in the next subsection. So

the simulation is the one-rank network. Routers have different roles - they belong to the transport layer and are not considered in the ranked model. The flat model still supports various options of node-to-node and node-to-router communication.

2.3.2 Linked lists

One of the most primitive forms of the flat, one-ranked network organization - is the linked list. An example of such a list is illustrated in figure 2.2.

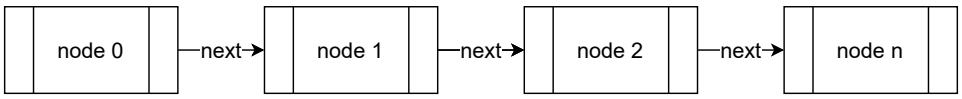


Figure 2.2: Linked list principle

For the simulation without neighbor-empowered synchronization (section 6.3.8), the organization remains flat without any links. Nodes are only connected to the router as discussed in chapter 6. But in some cases, an extra connection is needed. In ordinary linked lists, every list entry has a reference to the next entry [16, p.12]. The circular linked list's tail has a reference to the head. The double-linked list assumes that every element has a reference to a successor and to a predecessor [16, p.18]. The circular double-linked list is illustrated in figure 2.3.

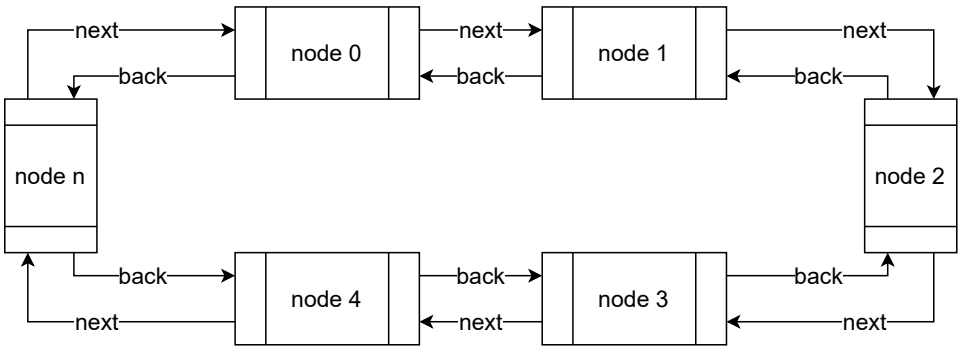


Figure 2.3: Double-linked list principle

The form of intra-node communication illustrated in figure 2.3 has the advantage of simplicity. Iteration complexity in big-endian notation is O(1) for any node-node step and O(1) to go from any node to the nearest router. With this approach, complex network traversing can be avoided.

2.3.3 Shadow networks

While remote nodes might be unavailable, the "shadow" layer represents edge nodes and is exposed to a user. In fact, shadow nodes are designed to contain copies of data from remote edge nodes. In this thesis, such an extra layer is called a shadow network. Practically, this approach exists in other systems in form of proxies, CDN's, replicas, and other technologies masking network or remote service full/partial unavailability. Almost any network can be unavailable due to various factors: power outage, physical network line damage, and so on. Reasons for hiding those issues might be (1) entertaining: streaming the media content without interruptions; (2) scientific: having an up-to-date replica with a data backup; (3) others.

For instance, the CAP theorem states that it is impossible to achieve more than two of three following properties in a distributed storage system: availability, consistency, and partition tolerance [17]. In general, a "shadow" layer increases data availability. It makes the data accessible by providing an extra network channel or adding more computational resources.

2.3.4 Observation of observational networks

Monitoring can help to maintain, adjust and fix a network or simulation of a network. Some mechanisms may be intrusive and affect the network workflow bringing extra causalities [14]. As highlighted by Fidge, this phenomenon is sometimes referred to as "The Heisenberg" effect [14]. In computer science, an example is when a profiler alters the original concurrent program behavior. Hence, the intrusive observation method's effects should be taken into account to predict such alterations of behavior. There is also a need for real-time network monitoring. For this purpose, real-time global timestamps can be used. As discussed in [51], this approach may bring causalities, because the system load can be different, so the timestamping in concurrent threads can result in incorrect IO order, despite the global clock [14]. For the flat network, where nodes have private channels and do not use shared channels for data transfer, the discussed drawback is not vital. The role of timestamping in the prototype is to measure the network bandwidth and data transfer rate. Implementation details can be found in chapter 6.

3

Methodology

This chapter describes the research method. Here comes the clarification for the experimental setup and how all the research steps are interconnected. The chapter starts with the description of the scientific research method principles, their connection with the statistical method via analysis, synthesis, and review. The experimental setup is explained from the position of experimental control and randomization.

3.1 Scientific research method

Quantitative research assumes that measurements and experiments are in focus. But as highlighted by Goertzen M. quantitative research has limitations as it is may require longer periods for the data collection [19]. The simulation described in chapter 6 - Prototype has the main goal to save resources and give access to big data massives without long observations.

The scientific research method and the scientific experiment are theoretical frameworks for the simulation from design to execution and evaluation. Generally, basic ideas of the scientific research method and quantitative method are similar: stating the question, defining the hypothesis, making predictions, testing and evaluation.[18]

3.2 Mathematical model

Insights on the step of observation help to identify an open question by virtue of the existing limitations. Understanding is the first step in the mathematical method and correlates with the principle of characterization of observation in the scientific method.

In research conducted by the COAT group [13], the main data collection limitations are the limited energy and connectivity of remote edge nodes. Edge nodes are in the sleep mode most of the time and the back-haul network is not always ready when edge nodes are active. Hence, data availability becomes an issue. That is why the first step was to clearly identify the problem and make the first logical assumption or hypothesis.

The hypothesis is the basic element of the scientific method, while in the mathematical method this step is called analysis. [9] In other words, the hypothetico-deductive method is strongly connected with the mathematical method and sometimes implies the same principles, follows the same steps [9].

The next step assumes making predictions from the hypothesis. In mathematical research, this step is called synthesis. Statistics, as a part of mathematics, is a helpful instrument that can provide the statistical expectation or predicted result with a certain chance of probability. The mathematical synthesis approach that sublimates into the prediction step of the scientific method brings the research further to the experimental stage (hypothesis testing) and the review (evaluation and improvement). In chapter 5 such predictions are listed together with the calculation method.

3.3 Empirical model

When the research question is stated, the hypothesis is formulated and the prediction is ready, comes the testing or the experiment that is aimed at data collection. Analysis of the collected data can validate the hypothesis or require further testing (if the hypothesis is not validated) [18]. The experiment here is the model simulation after the prototype design and implementation as discussed in chapter 6. The prototype is the computer simulation of the network built in accordance with scientific controlled experiments principles but has some limitations as discussed previously in section 1.3.5. A closer look at the design prototyping methods can help to understand the simulation's role as a part of the research method.

3.3.1 Design prototyping methods

There is a number of approaches to building prototypes [7]. The observation unit network simulation, which is the core part of the thesis follows some of those principles as any other prototype.

For instance, the virtual prototyping model is aimed at resource-saving. It may save time and expenditures. Natural shortcomings include the limited amount of environmental factors that can be efficiently emulated. But such a model is a tool for observation, testing, and prediction which is dependent on computational resources, not the natural conditions or access to complex equipment for real-life observation. Virtual prototyping is capable of simulating time with a given scale. Months can be simulated in seconds in case of the properly constructed model and enough compute power.

The iterative prototyping approach means that all the design process is done in several iterations [7]. During the project development, it is possible to go back to the previous iteration in order to evaluate it and enhance components that might increase the accuracy of measurements for example.

Another type described in the article and which is relevant for the simulation discussed is the scaled prototype model [7]. Such models compensate for the lack of time by scaling nanoseconds to minutes by employing extra CPU power and RAM for instance. The next possible advantage is connected with the geographical scale. In the real world, observational nodes can be kilometers away from each other in natural conditions with no roads and no infrastructure. In the scaled model, the physical distance and the network complexity can be expressed in a form of functions and methods.

There is no one and only one proper approach that describes the OU network simulation. That is why the three most relevant modeling principles are listed. The mixed model inherits ideas from all those principles.

3.3.2 Experiment methodology

Once the prototype is built, starts the simulation that generates significant data amounts. In order to assure result correctness, the experiment must follow basic scientific controlled experiments principles [18].

The chance of uncertainty that takes place during any experiment influences the results and should be predicted. Such a prediction helps to identify the standard deviation and the confidence interval. Extra data sampling during the experiment provides more data. Sufficient data amounts can increase the de-

gree of result correctness. A homogeneous experimental setup guarantees that changing the runtime environment would not raise the uncertainty level.

Randomization of input data confirms that the result correlates, but is not bound to the input. Usage of random input helps to identify trends and dependencies. Positive control ensures data processing correctness. Under certain circumstances, the result must be exact. For instance, a signal from the motion detector on the remote node must always result in the camera module activation. In a similar way, negative control helps to figure out if the experiment responds properly to the results that will deliberately bring the failure. For example, connection requests from unauthorized devices must be rejected.

Contrast with observational study can be the next step of the experiment. If we know that the simulation behaves unpredictably in a different manner than the real-world network, some error or uncertainty is existing and is currently unknown. If the contrast is high enough, adjustment to the model should be done in the previous iteration.

3.4 Quality assurance

After the data is collected and analyzed, or the experimental stage is complete, the output should be validated and reviewed. The scientific method assumes the entire process is done in several iterations. It means that is possible to reproduce any stage and introduce new elements that may increase the experiment accuracy. As discussed previously, the contrast with the observation, positive and negative control, extra data sampling, and input randomization are possible indicators of results validity.

In general, reproducibility is the key to research transparency. Repetition of the experiment is the form of control for continuous improvement. If others are able to reproduce the experiment, it ensures that results would not be corrected in order to match the hypothesis. The experiment quality is assured both during the development process as discussed in subsection 6.2 - Design and after result collection as showed in section 7 - Evaluation. Subsections 7.2 and 7.3 are examples of the iterative prototyping approach.

4

Related Work

This chapter gives a brief overview of existing solutions for issues discussed in the thesis. The articles and project results listed below provided a starting point for the research. There are 4 points of interest to be discussed: existing observational edge node networks, synchronization in edge node networks, masking effects of edge nodes being unavailable, and the large-scale network simulation.

4.1 Observational edge node networks

Researchers and research groups all over the world are constantly improving their toolboxes for environmental monitoring. The network of sensors is now an essential part of the large-scale data collection process. Nevertheless, statistical prediction, data analysis, processing, storage, and the search for the perfect balance between the cost and throughput make such networks a subject for research, enhancement, and discussion.

SATURN[1] Observation Network has a system distributed on the Columbia River in order to collect physical and biogeochemical data [37]. The project is a part of The Center for Coastal Margin Observation Prediction (CMOP).

1. SATURN Observation Network: `http://www.stccmop.org/datamart/observation_network` (accessed: 2021-11-10)

Project members introduce own mechanisms for the adaptive data sampling as described by Thanh Dang et al. in [12]. There is a need for a periodical cruise that would go by the river and collect the extra data in addition to the data received from the observational nodes. Such a cruise is expensive and rare, so the data collection must be precise and efficient. In their article, the data collection algorithm for the accuracy improvement is discussed.

The Marine Observation Network uses "Underwater Vehicles" [31]. There is a lack of cheap and robust wireless network infrastructure underwater, so there is a need for specific communication protocols. A set of several autonomous underwater vehicles together with a mobile surface platform can be efficiently used for oceanographic data collection. Vehicles are able to collect the temperature data accurately in both "autonomous" and "connected" modes.

Not only the wild nature is the subject for the data collection via networks of edge nodes. Cities are the everyday environment for millions of people. An effective way for the real-time traffic intensity estimation can be done using vehicles dashboard cameras [27]. Vehicles serve as cyber-physical systems - edge nodes that generate huge data amounts. All the computation is done on the edge node by the modified dashboard camera software. The method of the dashboard camera data extraction shows over 90% precision and exceeds the GPS-based method accuracy.

Raspberry Pi Zero W[2] based network prototype developed by United States Military Academy [10]. Wildlife monitoring is done via a set of Raspberry Pi serving as edge nodes. The sensor network is designed and evaluated from the power-saving perspective and shows 80 hours of working time with 22,000 mAh battery capacity and 5.8A current output. The system collects the data via an infrared motion sensor and a camera with a microphone. If there is no data from the motion sensor for 20 minutes, the device enters sleep mode. Nodes have 30 meters range and can communicate with a master node, which is able to connect to the end user's Android-based device. Project results are especially relevant for the research described in the thesis.

4.2 Synchronization in edge node networks

Synchronization is one of the key aspects in edge node networks. An unbalanced synchronization mechanism may lead to higher resource consumption and become the reason for unpredictable errors and uncertainties. A stable syn-

2. Raspberry Pi Zero W official documentation: `https://www.raspberrypi.org/products/raspberry-pi-zero-w` (accessed: 2021-11-10)

chronization mechanism would help to employ network resources effectively and deliver data in a consistent manner.

Relational Covariate Adjustment (RCA) is an estimation algorithm for causal effects in relational data [3]. The method is declared as a reliable way to estimate causal effect in both simulations and real-world network structures. A set of experiments showed that the causality estimation matches the observational data. The hypothesis is based on relational causal graphical models and is confirmed by summarizing the marginal individual effect with the marginal peer effect and future comparison to the synthetic network of 36,692 nodes and 183,831 edges. This method can be used for the synchronization mechanisms adjustment during the network design and development process even for networks of edge nodes with only one or two neighbors.

Synchronization in complex networks with uncertainty and time delay is another relevant study [54]. Examples of such systems are multi-sensor earthquake monitoring networks, automated highways as well as low-orbit satellite systems. The main synchronization issue is the inability to send the data precisely due to uncertainty and time delay. According to the research, the edge node synchronization accuracy can be predicted with given probability using the Laplacian L and G, and the graph theory.

The optimization algorithm for directed networks synchronizability assessment gives sufficient stability conditions despite the fact that prediction might not always be precise due to the system complexity [11]. The main benefit is the possibility to identify the impact of additional indirected links on the entire synchronization mechanism. This method is applicable to a range of networks from the brain neural synchrony disruption to power-grid network synchronization. The research shows the possibility to adjust the synchronization primitives in directed networks (where nodes and edges are of the same type).

4.3 Masking node's unavailability

The main issue with layered networks is that edge nodes might be unavailable as discussed previously. This issue has especially high priority in nature observation networks, where energy consumption and network availability determine the overall system design.

Edge-Empowered Graph Convolutional Network model consists of edge nodes aggregating representations of their neighbors in case neighbors might be unavailable later [53]. The study demonstrates the high level of statistically modeled data reliability despite edge node's partial unavailability.

Another study proposes a strategy for network resilience and shows the algorithm for optimization of wireless mesh networks in case of edge node's failure [45]. Main steps such as "Defend", "Detect", "Remediate", "Recover" result in two loops: "Diagnose" and "Refine". The network resilience strategy is explained through the $D^2 * R^2 + DR$ principle, where the first loop is based on the runtime analysis, the second loop might involve machine learning. The theory is aimed at improved network stability and its ability to recover.

The Pexip Project[3] introduces two new types of instances: transcoding nodes and proxying edge nodes [38]. They are suggested to be placed closer to the end-user in order to ease the load balancing in the network and mask possible connectivity issues. This might be used as a variant of the edge-node network with a "shadow" layer and superpeers.

4.4 Large scale network simulation

The idea to create a synthetic network is not new. Here some relevant examples of large-scale network simulations and simulation analysis are presented.

Simulation of the network of more than 1500 LTE cells on approximately 1500 CPU cores showed that big-scale network simulation requires the optimization of parallelization mechanisms [49]. The simulation on the high-performance computing cluster is done, results are collected and evaluated (parallelization approach on shared memory called Horizon).

Parallelism potentials in distributed simulations are analyzed [2] by the example of the peer-to-peer distributed hash table-based network called Kademlia [34]. The study considers synchronization waiting time and the physical message exchange between logical processes as primary overhead sources. The main outcome is that the partitioning identical on the host machine and the simulated nodes routing tables reduces the inter-processors communication up to a factor of 6. At the same time, location-based partitioning reduces the synchronization cost, increases the spatial distance between remote nodes.

A study on large-scale network simulators is the overview of large-scale network simulators and their main characteristics [6]. The authors list some parallel discrete event simulators as well as systems based on discrete simulation principles. This is a good starting point for designing own network simulator for a large network.

3. Pexip Infinity technical documentation: `https://docs.pexip.com/admin/distributed_edge.htm` (accessed: 2021-11-10)

5

Statistical expectation

The current chapter illustrates the statistical expectation calculation algorithm and shows the reason for the prediction making based on the mathematical method. The assumption based on calculations below provides the fundament for further research and prototyping.

5.1 Calculation method

The chosen calculation method is taken from the statistical field of science and is based on the theory of probability. The resulting values may have uncertainties due to the relative simplicity of the chosen approximation method. Researches in graph theory and general statistics show that complex models can visualize and predict the behavior of the network after collection of the relational data on the characteristics[29, p.51]. But the thesis doesn't pursue the goal of extensive mathematical research as defined by scope and limitations in section 1.3.5. The motivation is to identify the general trend of single success probabilities dependent on the back-haul network availability chance and the chance of edge node being awake. The statistically derived result will be compared with the simulation results.

5.1.1 Two events probability

From the statistical point of view, two independent events can be defined as events that don't have an influence on each other in terms of occurring order and result dependency [46]. In the observation unit network, there are two types of events with given probabilities:

- P(N) - the chance of the back-haul network being available
- P(A) - the chance that the observational node will be awake and available for the synchronization

The chance of network the back-haul network being available varies in range $0 \leq P(N) \leq 1$. The chance of the node being awake has the same range $0 \leq P(A) \leq 1$ and is applicable to every single node wake-up - a moment when compute node is not sleeping and is ready for the connection. It means that $P(A) = 1$ will result in successful synchronization every single time, while $P(A) = 0.5$ reduces the chance of successful synchronization to 50%. $P(A) = 0.05$ means that the chances of synchronization are about 5%, so for 20 wake-ups, we can expect approximately one synchronization.

These two events P(A) and P(N) are independent, they do not trigger each other and do not have any impact on each other's results. It means that the chance of two events happening together can be calculated using the equation 5.1 [46].

$$P(A) \cap P(N) = P(A) * P(N) \tag{5.1}$$

If the probability P(N) is 50% and the probability P(A) is 5%, using formula 5.1, we get: $P(A) \cap P(N) = P(A) * P(N) = 0.5 * 0.05 = 0.025$. Both elements influence the overall probability chance for one single node wake-up (trial).

5.1.2 Binomial pmf

When one-time probability is given, binomial PMF distribution can estimate the probability that there will be s successful synchronization(s) after n trials

[47]. It is possible to estimate the binomial PMF for a range of trials if only the expected number of successful synchronizations is defined. Given the number of trials, binomial PMF can be estimated for a range of successful synchronizations. The calculation is performed using equation 5.2 [47].

$$P(s;p;n) = \sum P(\{(e_1,\ldots,e_n)\}) = \binom{n}{s} p^s (1-p)^{n-s} \tag{5.2}$$

For instance, using the probability $P(A) \cap P(N) = 0.025$ from the previous example and given that we await at least one successful synchronization after 100 trials, binomial PMF value will be:

$$P(s;p;n) = \binom{100}{1} 0.025^1 (1-0.025)^{100-1} = 0.20389$$

In other words, $P = 0.20398$ corresponds around 20% chance of at least one successful synchronization after 100 trials. But the function should be read from charts, as it will decline after its peak value [47]. The chart shows the direction: either the number of trials should be increased or the number of successes expected should be reduced to get a higher probability P.

To sum up, the binomial PMF is an instrument for prediction of the peak value of at least *s* synchronization chance after n trials. The value in one point can't be exclusively used without the graph as it doesn't show whether the probability goes up or down after that point.

5.2 Results & Expectation

This section shows how the binomial PMF can help to predict a declining/increasing trend in a success chance probability in the simulation. Results derived from the statistical model would be tested in the simulation.

5.2.1 Number of successes

For the first statistical experiment, the probability of the back-haul network availability P(N) is 20%, the probability of the node being awake P(N) is 10%. It

means that the connection will be available with a ratio 1 to 5, and one node will be ready for the synchronization approximately once in 10 trials. The overall chance of two independent events happening together is $P(A) \cap P(N) = 0.025$ as calculated using equation 5.1 and reflected in table 5.1. Let's assume that the simulation period is one week or 168 hours. It means 168 node weak-ups or 168 trials n. More details on time scaling in section 6.2.2 - Scale. Let's calculate the binomial PMF value for a range of successes from 1 to 50 and reflect some of them in table 5.1.

Awake %	Network %	$P(A) \cap P(N)$	n	s	Binomial PMF
10	25	0,025	168	1	0,06123788
				2	0,13111186
				3	0,18602196
				4	0,19675399
				5	0,16547515
				12	0,00081045
				25	1,0046E-12

Table 5.1: Binomial PMF for $P(A) \cap P(N) = 0.025$

Judging from results in table 5.1, after 168 trials, the expected synchronization probability is around 19% for approximately 4 successful synchronizations (s). 5 successful synchronizations can be expected with a probability of around 16%. Let's plot all 50 values to the chart.

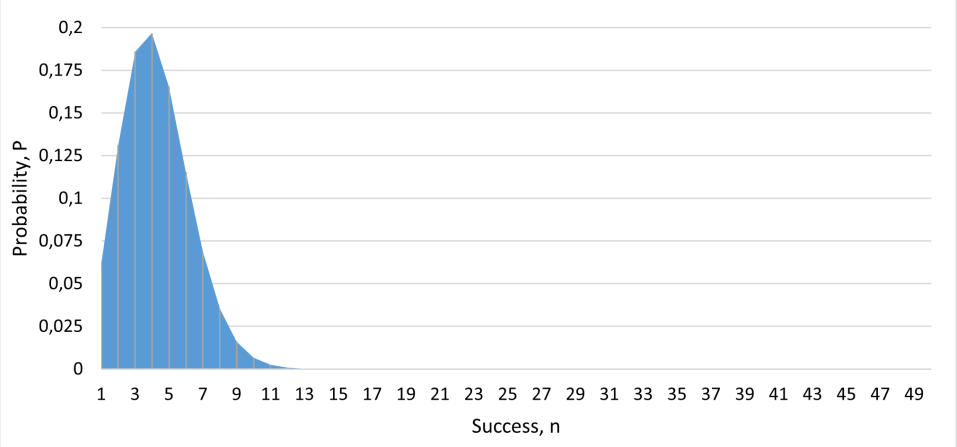
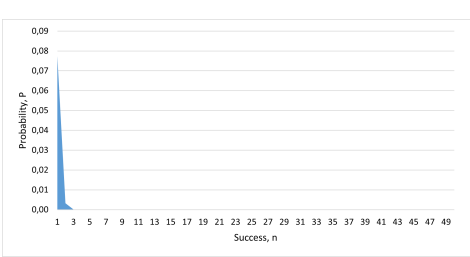


Figure 5.1: Binomial PMF for $P(A) \cap P(N) = 0.025$

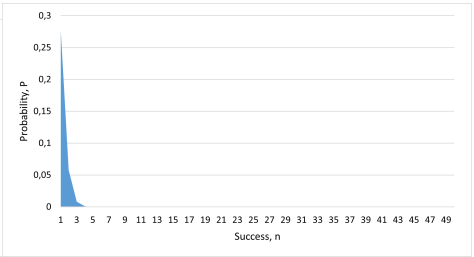
The chart illustrated in figure 5.1 confirms that for 168 trials, the probability of up to 5 successful synchronizations is around 16%. After that point, the synchronization chance is reduced. A relatively low value before this point doesn't mean that at least 5 successful synchronizations are more probable than at least one. Such value distribution means that exactly 5 synchronizations have

a higher probability to occur than exactly one successful synchronization.

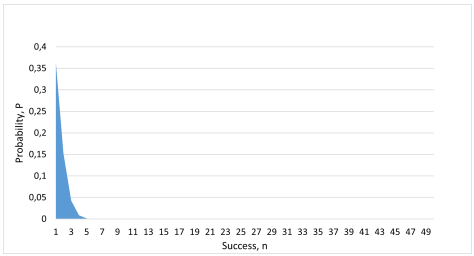
Here, the probability $P(A) \cap P(N) = 0.025$ is relatively high and doesn't cover worst-case scenarios. The next step is to make tables and plot charts for other $P(A) \cap P(N)$ cases to include more realistic chances. Tables are not listed here, but all 6 resulting plots can be found in figure 5.2.



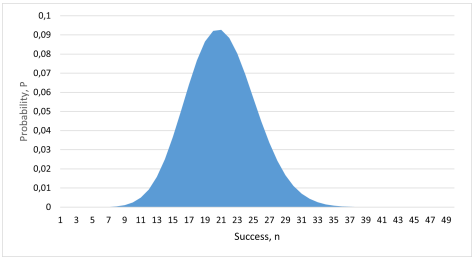
(a) Binomial PMF for $P(A) \cap P(N) = 0.0005$



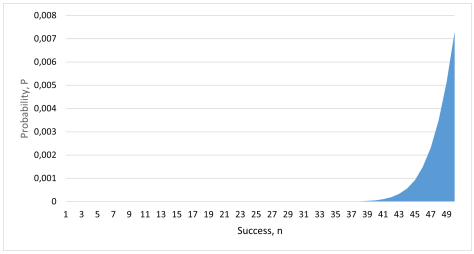
(b) Binomial PMF for $P(A) \cap P(N) = 0.0025$



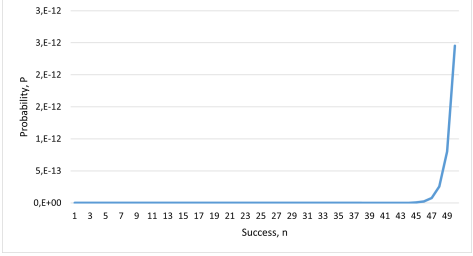
(c) Binomial PMF for $P(A) \cap P(N) = 0.005$



(d) Binomial PMF for $P(A) \cap P(N) = 0.125$



(e) Binomial PMF for $P(A) \cap P(N) = 0.375$



(f) Binomial PMF for $P(A) \cap P(N) = 0.5625$

Figure 5.2: Binomial PMF for $0.00025 \leq P(A) \cap P(N) \leq 0.5625$

Individual charts in figure 5.2 are showing the estimated chance of s after 168 trials for $P = 0.0005$, $P = 0.0025$, $P = 0.005$, $P = 0.125$, $P = 0.375$, $P = 0.5625$ in figures 5.2a, 5.2b, 5.2c, 5.2d, 5.2e, 5.2f respectively. For the first two cases, the function values are relatively low. It means that exactly one synchronization chance is only 7.5% for $P = 0.0005$ as shown in figure 5.2a. Binomial PMF chart for $P = 0.0025$ in figure 5.2b promises up to 3 successful synchronizations with a chance of 25%. In the case of $P = 0.5625$, the function has not even begun to show its peak after the point of 49 trials as shown in figure 5.2f. It can be assumed that the function peak is much more far away to

the right and for such a high P value, so we can expect more than 50 successful synchronizations after 168 trials.

Figure 5.3 below combines a chart from figure 5.1 and all 6 individual charts from the figure 5.2. The chart a secondary axis. The primary axis (on the left) is for the area diagrams which represent $P < 0.1$, the secondary axis contains values for curves of binomial PMF for $P > 0.1$. The three cases with the highest total probability of two independent events show that:

- for $P = 0.125$ we can expect around 21 successes with probability of 10%;
- for $P = 0.375$ there will be over 49 successful synchronizations, probability only starts to go up after the point of 49;
- for $P = 0.5625$ there will be much more than 49 successful synchronizations, probability starting point is not even visible after 50 trials.

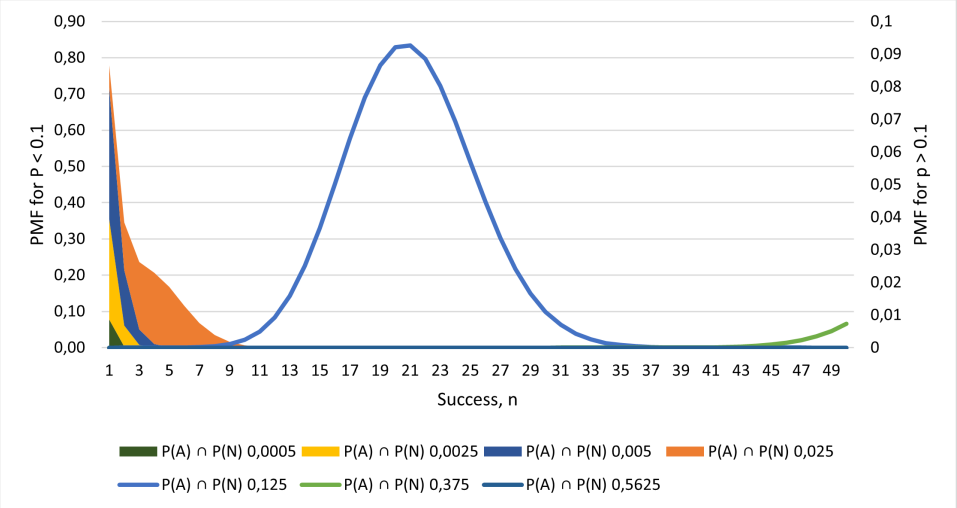


Figure 5.3: Combined graph for binomial PMF for $1 \leq s \leq 50$

The four cases with the lowest total probability of two independent events show that:

- for $P = 0.0005$ there will be maximum one synchronization and the probability is 7%;
- for $P = 0.0025$ there can be one synchronization with probability 20%;
- $P = 0.005$ shows up to 3 successful synchronizations at 20% chance;

- for $P = 0.025$ the number of successful synchronizations is 5 at 20% probability.

To sum up the section, *n=1* (one successful synchronization) reflects the reality, because it is applicable to all the range of probabilities. The simulation will examine worst-case scenarios with low $P(A) \cap P(N)$ probabilities when at most one synchronization during the period of time can be expected. This value will be used for the next statistical experiment and the empirical experiments from sections 7.5, 7.6, 7.7, 7.8, 7.9.

5.2.2 Number of trials

This statistical approximation is opposite to the previous one as it aims to find out a number of trials needed to reach at least one success at various probability ranges.

The Definition 6. Single Synchronization Success Chance

Single Synchronization Success Chance is the probability that the edge node would synchronize at least once during the given period of time at a given one-time synchronization probability $P(A) \cap P(N)$.

As in the previous calculation, for the first example, the probability of the back-haul network availability P(N) is 20%, the probability of the node being awake P(A) is 10%, $P(A) \cap P(N) = 0.0005$, $s = 1$. Table 5.2 shows values of binomial PMF for trials (wake-ups) from 1 to 200 (calculated using equation 5.2).

Awake %	Network %	$P(A) \cap P(N)$	s	n	Binomial PMF
1	5	0,0005	1	1	0,0005
				2	0,0009995
				3	0,0014985
				100	0,047584669
				168	0,077269238
				199	0,09011917
				200	0,09052674

Table 5.2: Binomial PMF for $P(A) \cap P(N) = 0.0005$

The first value set derived from the equation 5.2 is for $P(A) \cap P(N) = 0.0005$. Let's plot the range of PMF values into the chart. The chart is illustrated in figure 5.4. It shows that for such a low synchronization success chance, the number of trials needed for at least one synchronization is over 200. It means

that for this P value, the more trials will be, the higher chance we get. Even if there will be 200 trials, the chance is only 10%.

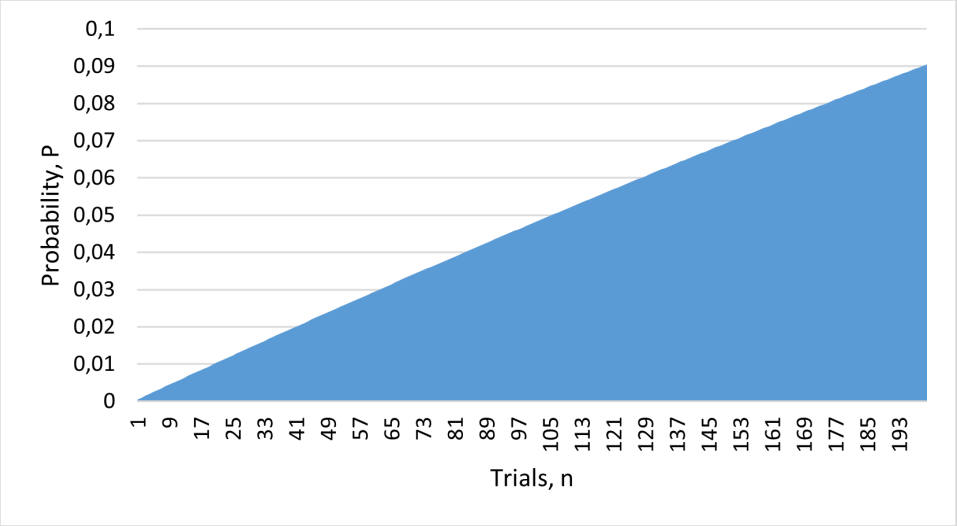


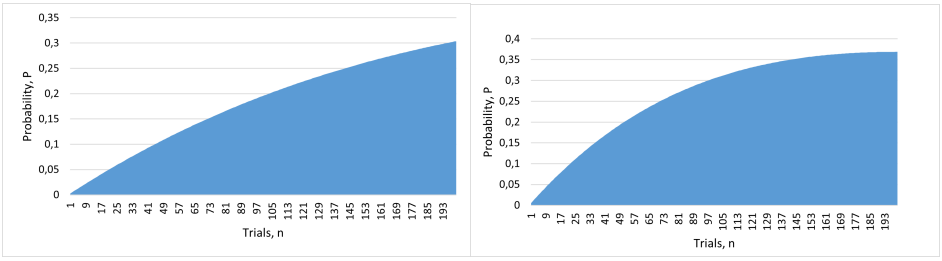
Figure 5.4: Binomial PMF for $P(A) \cap P(N) = 0.0005$; $s = 1$

The next step is to make tables and charts for other $P(A) \cap P(N)$ values. Tables are not listed here, but charts are plotted into the figure 5.5 which contains PMF values for at least one synchronization for $P = 0.0005$, $P = 0.0025$, $P = 0.005$, $P = 0.125$, $P = 0.375$, $P = 0.5625$ after 1 to 200 trials.

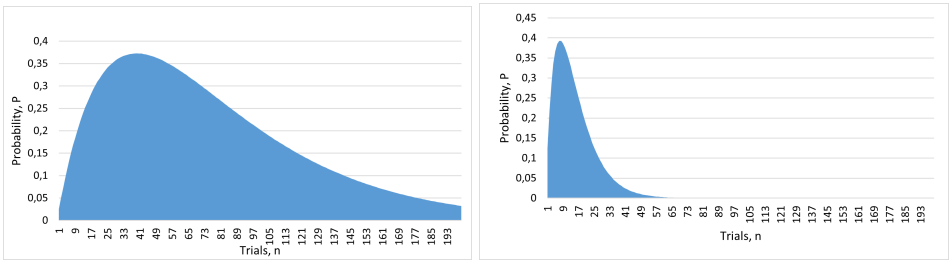
Figure 5.5 shows that the single synchronization chance is growing in the direct correlation with the number of trials and the one-time synchronization probability $P(A) \cap P(N)$. In cases 5.5a and 5.5b the number of trials needed is especially high (over 200 trials to reach the peak PMF value). For the most "positive" scenarios 5.5e and 5.5f the number of trials needed is 2 with 40% probability and only 1 with 55% probability respectively. All 7 cases from figures 5.4 and 5.5 are combined into a single chart illustrated in figure 5.6.

Figure 5.6 combines a chart from figure 5.4 and all 6 individual charts from the figure 5.5. The chart has a secondary axis. The primary axis contains values for curves of binomial PMF for $P > 0.1$, the secondary axis (on the left) is for bars which represent $P < 0.1$. The four cases with the lowest total probability of two independent events show that:

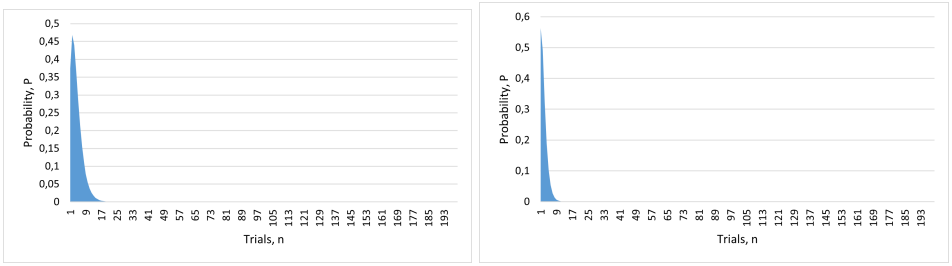
- for $P = 0.0005$ more than 200 trial are needed to get at least 10% single synchronization success chance,
- for $P = 0.0025$ after 195 trials, the chance of to get at least one synchronization is 30%,



(a) Binomial PMF for $P(A) \cap P(N) = 0.0025$ **(b)** Binomial PMF for $P(A) \cap P(N) = 0.005$



(c) Binomial PMF for $P(A) \cap P(N) = 0.025$ **(d)** Binomial PMF for $P(A) \cap P(N) = 0.125$



(e) Binomial PMF for $P(A) \cap P(N) = 0.375$ **(f)** Binomial PMF for $P(A) \cap P(N) = 0.5625$

Figure 5.5: Binomial PMF for $0.00025 \leq P(A) \cap P(N) \leq 0.5625$

- for $P = 0.005$ after 195 trials, the chance of to get at least one synchronization is 36%,
- for $P = 0.025$ the chance to get one synchronization accomplished is 36% after 40 trials.

The three cases with the highest total probability of two independent events show that:

- for $P = 0.125$ 40% synchronization chance is hit after 7 trials,
- for $P = 0.375$ after one-two trials, the chance of getting 1 success is 45%,
- for $P = 0.5625$ even after the first trial, success chance is over 50%.

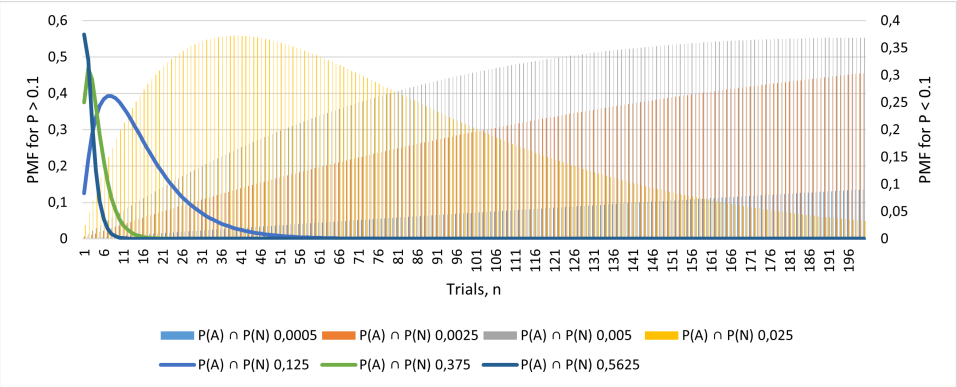


Figure 5.6: Combined graph for binomial PMF for $1 \leq n \leq 200$

It is reasonable to set the simulation period to in the interval between 100 and 200 trials to cover even worst-case scenarios with a low probability $P(A) \cap P(N)$. Too short simulation time would cut off monitoring of nodes that didn't get enough trials. Too long simulation time should not bring any advantage. After this statistical experiment, the simulation period is set to 168 trials which correspond to 7 weeks of simulation where an observation node wakes up once an hour. For more details on the time scaling principle, refer to section 6.2.2.

Taking everything into consideration, the statistical model gives the prediction for the simulation and the simulation gives the prediction for the real-world cluster. The goal here is to investigate the possibility to use the statistical expectation as input for the simulation, to examine the accuracy level of calculated values. The open question list is: is it sufficient with the statistical prediction only, is it accurate enough? If yes, in which cases? Should probability values be used for the simulation planning only or does it work for the real cluster as well? Some of these questions are raised in chapter 8 - Results, some questions transformed into plans for the future work (chapter 10)

6 Prototype

This chapter provides details on the prototype's architecture, design, and implementation.

6.1 Architecture

There are several types of simulation instances that reflect simplified real-world observational network architecture. As described in the introduction (chapter 1), the OU is the edge node and the key element of the simulation. It collects, stores, and sends the data from sensors over the network (definition 1).

SU contains a copy of the OU. To mask possible edge node unavailability, SU is fronted to the user (from definition 4). The shadow (SU) is assumed to be always up and running. It is constantly trying to obtain the latest data from the observation unit.

Another term that is used to describe a specific type of observation unit is neighbor.

The Definition 7. Neighbour

Neighbor is an OU that has a direct link (communication channel for data transfer) to another OU as illustrated in figures 6.4 and 6.5.

Neighbor is used by an OU as a reserve solution for the data transfer in case of network connection unavailability (if there is no link to the router). Every single OU is a neighbor of another OU. The data from OU's goes via the network to shadow units. Devices for data packets forwarding on the network level of abstraction are routers.

The Definition 8. Unit Router

Unit Router (UR) is the access point that brings the data from Observation Units to the next router as illustrated in figures 6.1 and 6.2.

The Definition 9. Shadow Router

Shadow Router (SR) is the access point that brings the data from UR to the end-user - the shadow unit, SU that corresponds the observation unit as illustrated in figures 6.1 and 6.2.

As in definition 3, all Unit Routers and all Observation Units compose the observation unit network, while all Shadow Routers and all shadows represent the shadow network. At the same time, there is one more level of abstraction that will be referred later - a cluster, so an extra definition is required:

The Definition 10. Cluster

A cluster is a group of nodes connected to the same router, alternatively called *island*, illustrated in figure 6.2.

The main difference between a neighborhood (definition 2) and a cluster is that a neighborhood may include several clusters.

The network topology is designed to strengthen one of three desirable properties of the distributed data store: availability (from the CAP theorem [17]). The goal is to create a "mask" that can hide the temporary absent connection to the remote nodes.

The Definition 11. Mask

The mask is the abstraction that includes the shadow network and the set of activities performed from the perspective of increasing observational data availability level.

To sum up, the cluster includes units and one router. All unit clusters form the observation network, while shadow clusters are replicas that form the shadow network. Everything together, all types of units and routers, is the observation unit network simulation.

The Definition 12. Simulation

The simulation is an application that emulates the network of Observation Units, Shadow Units, Unit Routers and Shadow Routers and uses them in a controlled way for a period of time.

6.1.1 Interconnection

A closer look at the network components is now followed by the demonstration of the general interconnection pattern. Figure 6.2 illustrates the connection principle. The global observation unit network simulation scheme is illustrated in figure 6.1.

Figure 6.1 shows that Unit Routers have side connections. Those are planned to enable global indexing in the future. The example illustrates that the network is designed to be scalable, a possible number of clusters and number of a nodes inside of clusters are adjustable. Figure 6.2 gives a closer look on a separate cluster.

6.1.2 Direct communication

In figure 6.2, a separate island of Observation Units is connected to its shadow representation. As discussed earlier, Observation Units get the data from sensors and send it further to the UR. The UR gets the data from private unit channels and sends the data on separate channels further to the SR. The SR delivers the data via the private channel to the final user - shadow nodes.

When the message reaches the final destination, it would be checked and saved to the local disk. If the disk mode is not enabled, the data consistency would only be checked without disk output (disk mode is described in section 6.3.10). Let's take a closer look at the data flow in the simulation.

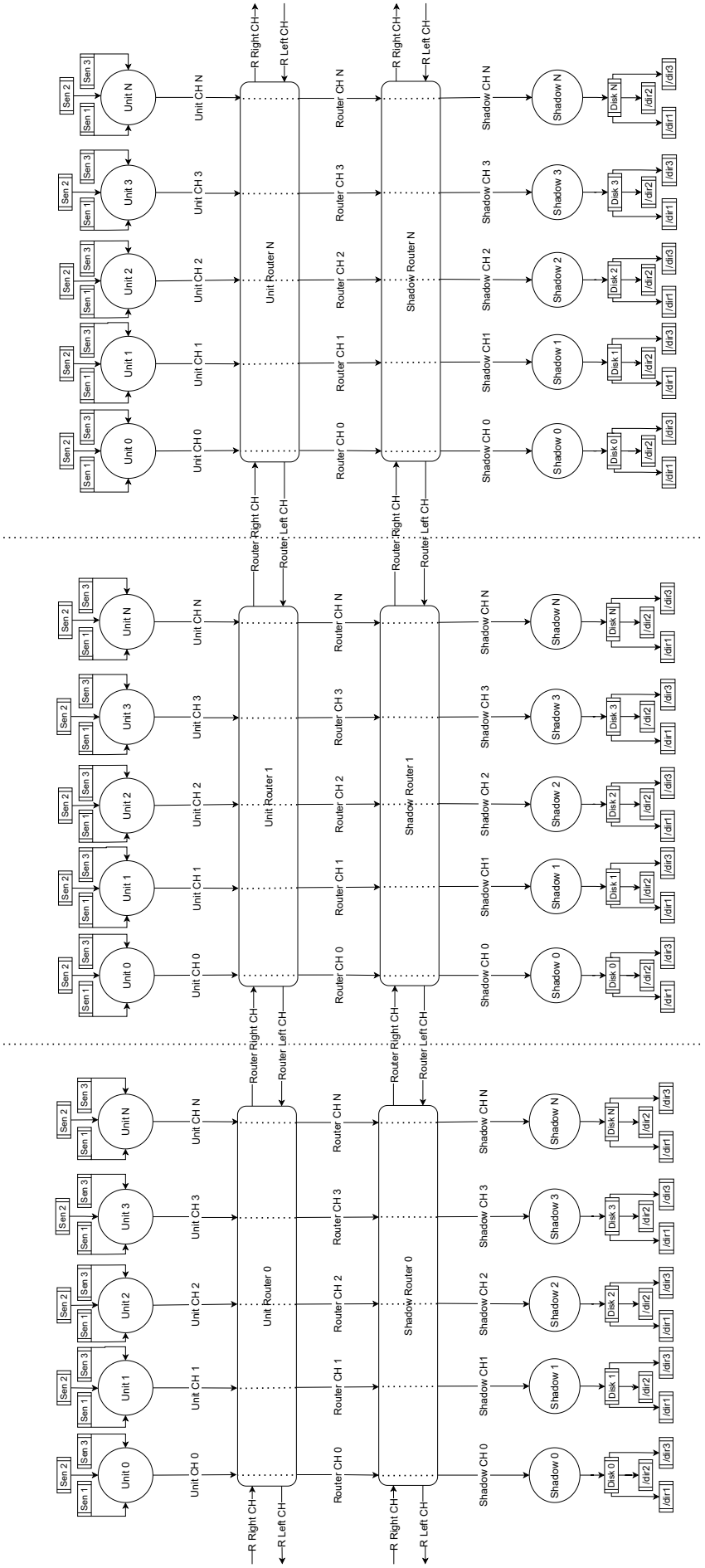


Figure 6.1: The global simulation architecture.

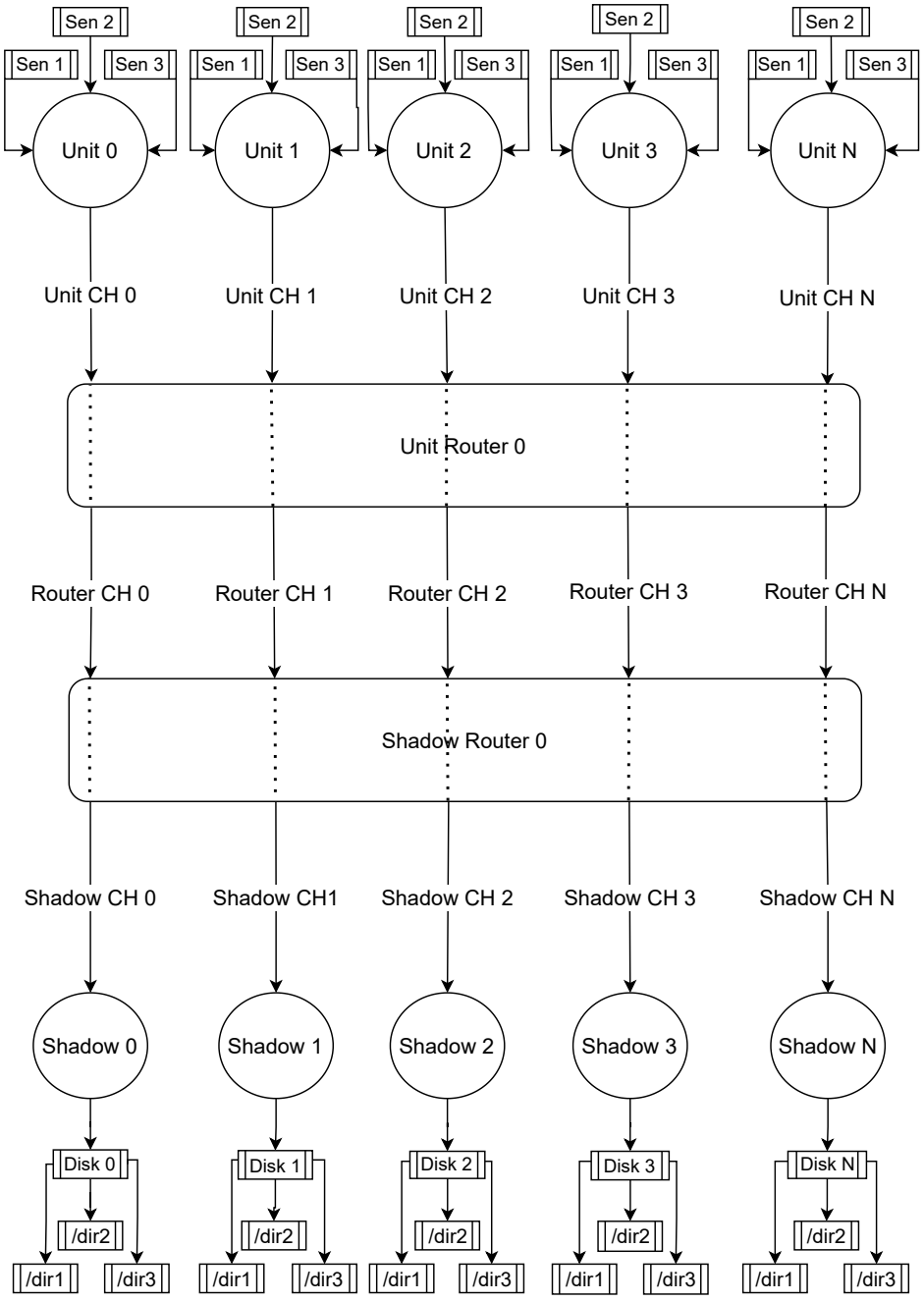


Figure 6.2: The cluster architecture

6.1.3 Direct data flow

There are two main operation modes that would be discussed later - direct synchronization and synchronization via the right neighbor. The direct type of communication is illustrated in figure 6.3. The scheme is the simplified representation of figure 6.2 highlighting the data path. The use of private channels is marked in the figure. Such an organization is an attempt to save simulation clock cycles by avoiding complicated routing.

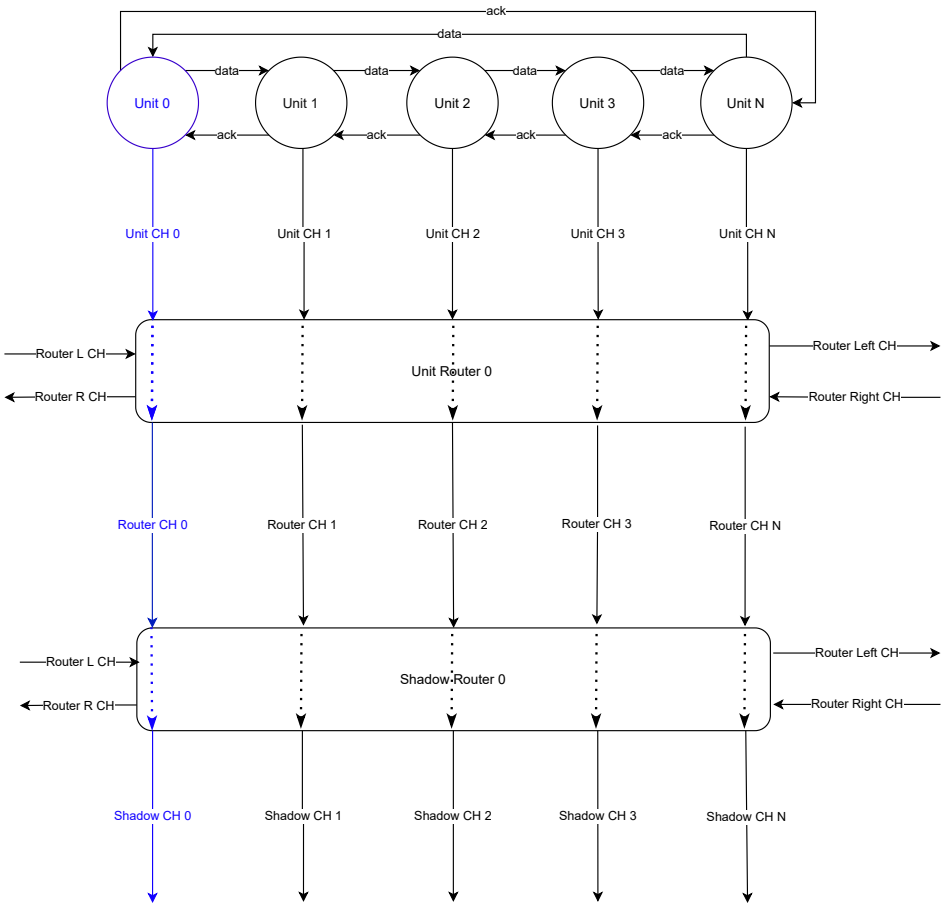


Figure 6.3: Direct data path

6.1.4 Neighbor communication

The next communication pattern designed for the simulation is the synchronization via the right neighbor. Any kind of communication requires a communication channel with communication primitives. Despite the right and left channel availability, only one is suited for the observational data transition. The

right channel serves as a data transfer corridor, while the left one is made for responses. Such an approach makes the orchestration easier. Figure 6.4 is the modification of figure 6.3 with focus on a single observational cluster.

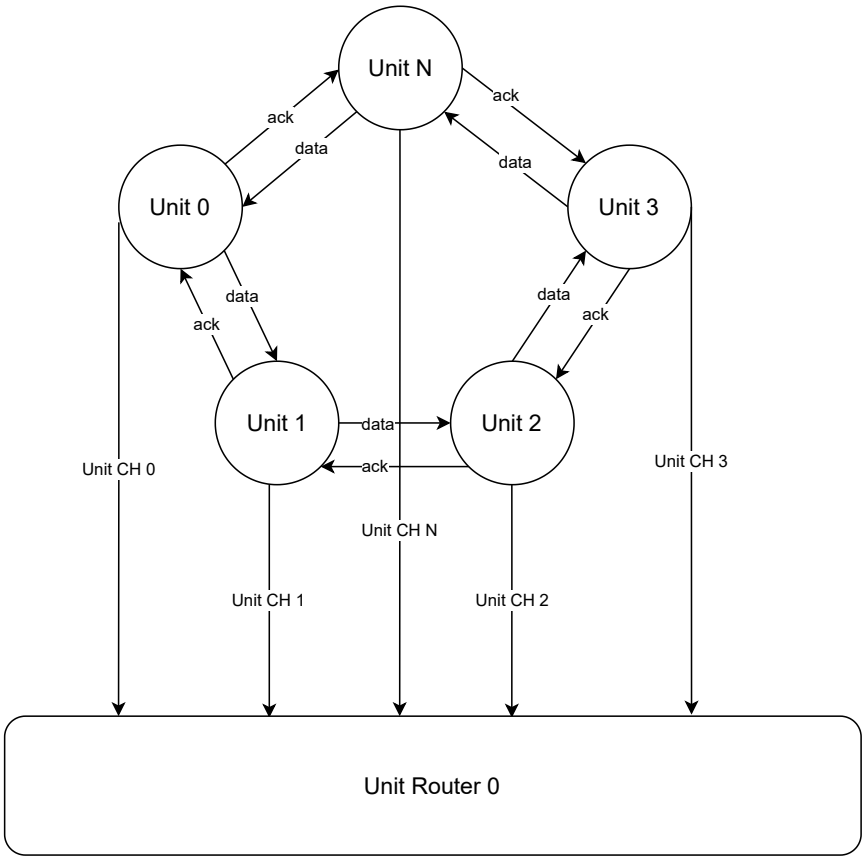


Figure 6.4: The simulation architecture with NEIB

There are four channel-pointers associated with each OU. Every OU supports an incoming data connection via its left channel and has a data connection to the right neighbor via the right channel. The incoming right connection is used to receive simple responses from the right neighbor. The outgoing left connection transfers simple communication primitives back to the left neighbor. Thus, every observation unit has two roles:

- An observation unit, OU in case of the direct data transfer.
- A neighbor that is used as an additional data transfer channel

Such a model is referred to as NEIB in the thesis and corresponds to definition 13.

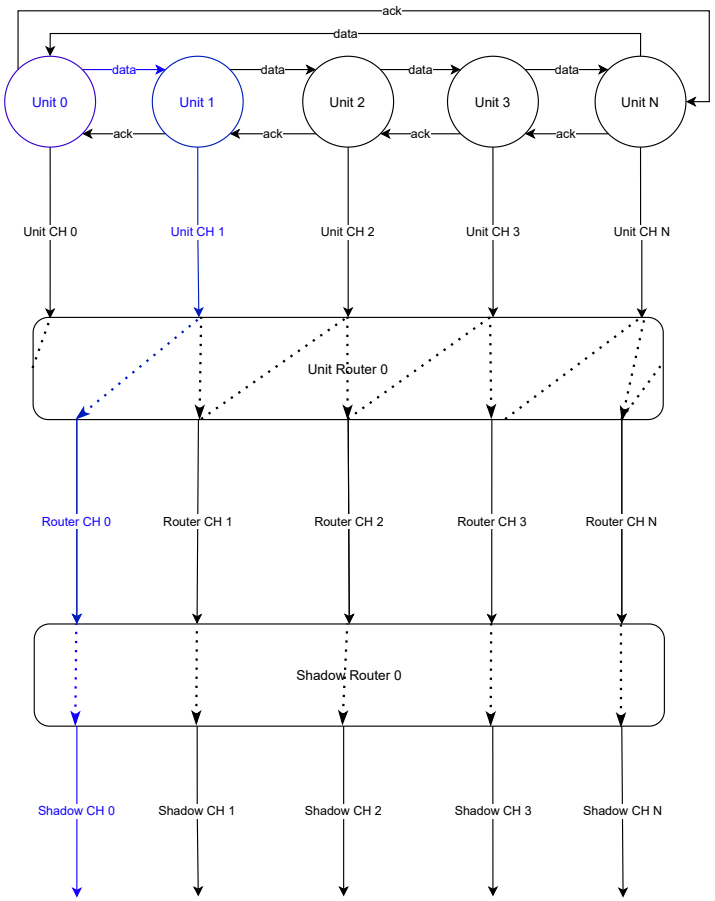


Figure 6.5: NEIB data path

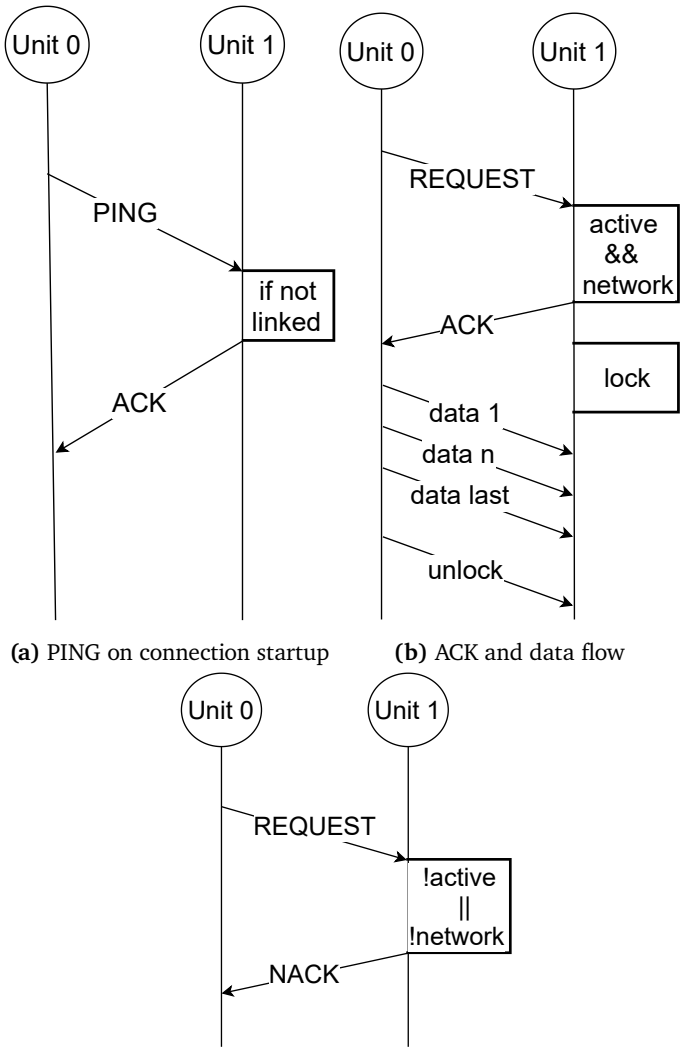
The Definition 13. Neighbor Mode (NEIB)

Neighbor protocol, neighbor mode or NEIB is the data transfer model where the right neighbor serves an additional channel for the data transfer.

6.1.5 NEIB data flow

The data path that is followed when the NEIB mode is activated has similar principles as in the case of direct communication. The difference is that the data can be transferred not only via the node's private channel as illustrated in figure 6.3. The alternative solution in case of network's unavailability is the data path illustrated in figure 6.5.

As reflected in figure 6.5, the data goes first to the right neighbor, then to the UR via neighbors private channel and, finally, routes back to node's private channel on the step of router-router communication. But what are the sufficient criteria for the data transfer via neighbor? If the node has no network connection or is not supposed to try to connect at the moment, it can still ask the neighbor to take the data stream. The entire node-to-node communication process is illustrated in figure 6.6.



(a) PING on connection startup **(b)** ACK and data flow

(c) NACK to REQUEST

Figure 6.6: NEIB communication patterns

As shown in figure 6.6a, the node initially checks if the neighbor exists and is available. Such a sanity check happens only once in the very beginning.

In order to have the synchronization in place, every node must first send a handshake - "ping" message to the right neighbor and respond affirmatively to incoming handshake request by "ack" message. When the handshake procedure is done, the node can request a synchronization via the neighbor - figure 6.6b. If the neighbor has no network or is not supposed to transfer the data now, it would reply negatively with "nack" and synchronization would not happen - figure 6.6c. If there is a back-haul network connection available and the node is awake, the reply must be positive. The neighbor "locks" until the data from the origin is transferred. When the sender issues EOF - "release" message, the neighbor can unlock and send its own data - figure 6.6b.

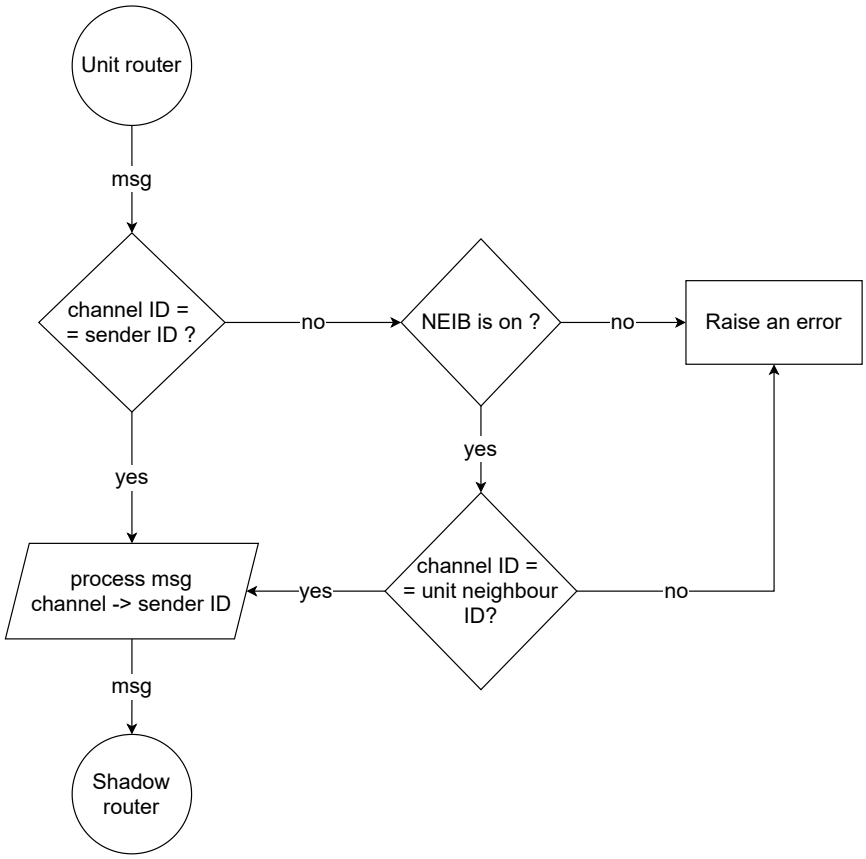


Figure 6.7: Routing principle via neighbor

The question is how to distinguish between the direct communication and the NEIB communication for the UR. Routers have an in-built sanity check procedure as well assertion methods (discussed in section 6.3). Figure 6.7 reflects the general algorithm that is followed by Unit Routers. When a new message comes to the router, it is checked whether the sender's Identificator (ID) corresponds to the channel and to the destination. Only if the NEIB mode enabled, it is allowed that the message origin can be on the left side of the unit

connected to this particular channel. If it is the case, the message would be then re-routed to the private channel of the original sender. The current section is about the architecture and shows the main patterns, while implementation details are covered in section 6.3.

6.2 Design

The prototype is built with respect to design principles discussed in section 3.3.1. This section covers design issues related to distributed systems. There are numerous approaches to design issue identification. Challenge classification proposed by Dan Nesset [36] is taken as a framework for the prototype application design.

6.2.1 Design basic principles

General design issues are in the following list [36]:

Scale is an issue because scaling from a few nodes to millions require a design that is adaptive, can change dependent on network size.

Heterogeneity means that nodes of different types need interfaces to become a single system.

Objects (representation, encoding, and translation) or big data streams require significant computation resources.

Resource management includes workload balancing and the productivity/resource-economy balancing.

Protection (security and privacy) in this particular simulation assumes the use of private data channels.

Naming principle requires unique Identificators for nodes and files to set ownership and keep proper addressing.

Error Control issue can be resolved using runtime monitoring and correctness testing.

Synchronization aspect is connected with naming, protection and is natural to distributed systems because data flows require node synchrony and communication orchestration.

Measurement, testing and debugging is the continual improvement principle, which starts on the development phase and lasts during the simulation lifetime.

The principles discussed in the next subsections address the issues listed above.

6.2.2 Scale: a second for an hour

The scaling of the simulation is not only the question of size and dynamic resource allocation, though it will be discussed in section 6.3. From the design perspective, scaling has more aspects. In the network prototype, the time is scaled. The idea is to simulate weeks of execution. There is not enough time to observe the cluster with a one-to-one time proportion. Hence, if Observation Units wake up every X minutes, we can skip the time when the node is idle. In the real world, edge nodes wake up every hour or half an hour, that is why the simulation lets nodes wake up every second and count it as an hour. In other words, the design principle can be described as "idle time skip".

While the heterogeneity issue is discussed in section 6.1, object and resource management are issues for the section 6.3. The next design issue is protection, security, and privacy.

6.2.3 Randomization

Another principle that enables scaling is the randomization of incoming data. In real world, there are always differences in the environment in two different moments of time. The simulation is limited and does not concern possible external effects. In fact, with totally the same experimental setup, the experiment results would be identical. The reason is the use of pseudo-random values. Some prototype functions (for instance, package *rand* [20], see 6.3) use pseudorandom approach. It doesn't provide truly randomized input but is sufficient for simulation purposes. In the case of future development, more attention should be paid to randomization, because it makes the simulation more similar to the real network.

6.2.4 Dynamic adaptation

A new setup of the number of nodes, workload, execution time, and network type would result in new, sufficiently random input. Such an approach helps to enable scaling from 1 to several thousand edge nodes and make sure that the

result is not bound to the hard-coded values. The prototype is developed with as few hard-coded values as possible. Every time a new cluster size or number of edge nodes is applied, counters, etc. are generated using supplementary functions.

6.2.5 Atomic variables

The issue of privacy and protection in distributed systems assumes that there shouldn't be unauthorized and unexpected access to the data [36]. In this simulation, the protection is related to parallel data IO. Processes and threads in the shared memory can do the simultaneous modification of the same data chunk. Several variables are designed from this point of view. For instance, some global counters are modified by several goroutines simultaneously. Such counters have "atomic" characteristics as illustrated in listing 6.1. It provides a guarantee that only one goroutine would modify the value at one particular moment. This design approach is addressing the data protection during the parallel IO. Every time when a goroutine needs to modify a global variable, it enters the critical region.

Listing 6.1: Atomic variables

```
type counter int32

func (c *counter) increment() int32 {
return atomic.AddInt32((*int32)(c), 1)
}

func (c *counter) decrement() int32 {
return atomic.AddInt32((*int32)(c), -1)
}

func (c *counter) load() int32 {
return atomic.LoadInt32((*int32)(c))
}
```

6.2.6 Naming convention

Every instance in the network simulation, that is wrapped into a goroutine, has a unique name. It helps to identify the sender and recipient of the data. Since there are only a few files used as input in the simulation, there is an issue with file naming. Unique UUID [26] in the data disk mode guarantees

that even similar samples would receive unique identifiers. Unique ID enables data consistency check using md5 checksum as discussed in section 6.3.

6.2.7 Runtime observation

The simulation needs observation for the adjustment and resource monitoring. As stated by Fidge in [14] and discussed previously in 2.3.4 the probe effect may bring extra workload to the system. However, the monitoring of both correctness and stability is vital. One of the principles of the simulation is graceful finalization. The prototype is designed with an in-built observer function: *app_wathcman()*, discussed in the next section - see 6.3. The function does the error check and forces the application shutdown in case of simulation is unresponsive. The secondary role of the function is the resource monitoring by printing the Operating System (OS) memory data and instance runtime data out to the terminal.

6.2.8 Finalization delay

The synchronization issue has a very high priority in this prototype. Due to physical computational resource limitations, parallel processes running with a global clock may have different execution orders. Only the OS decides which parallel threads would receive more resources than others. That is why the global finalization delay principle is introduced. Sometimes Observation Units do not have enough time to complete the data transfer and report the graceful shutdown. To address this problem, an extra delay is introduced before the application termination would be forced. This mechanism adds more reliability to synchronization orchestration assuring that the transition channel would not be cut while data transfer is underway.

6.2.9 Development & Testing

In chapter 7 (Evaluation), module testing, and profiling are discussed closely. From the design perspective, continuous development and iterative modular testing help to measure the prototype characteristics and identify errors. The project is developed with tools and libraries which are not platform-bound and can be executed on most Operating Systems and program environments. There is no specific effort put into portability. The prototype is developed in a single codebase without UNIX or NT add-ons. The development process is iterative, which allows isolating several steps from each other for the purpose of testing and adjustment. The benefits of iterative prototyping are discussed in chapter 3 - Research Method.

6.3 Implementation

The implementation section covers technical details of the prototype. It starts with the technical background which lists the equipment, instruments, and tools employed for the project development. The overview below makes the experimental setup reproducible because it contains exact technical characteristics (physical parameters, versioning) of the hardware and software. The reasoning for choice made in favor of concrete technologies provided.

6.3.1 Development environment

The prototype (simulation of the observational network) is developed in form of a source code written in Golang [20]. As discussed in chapter 1 - Introduction, the simulation is designed for the execution on a High Performance Computing (HPC) cluster.

GoLang vs Message Passing Interface (mpi)

The natural question that comes up is: why not to use MPI with an Intel compiler? It might seem natural to use MPI for the HPC applications. In principle, there are two approaches of distributed application design: shared-memory system architecture and the distributed memory system architecture, according to Tanenbaum [51, p.25]. Practically, in shared-memory applications, processes and threads are operating in the same memory space and can share global variables. In the case of the distributed memory system architecture, different simulation elements are running on remote and isolated memory spaces without globally shared variables. Message passing is designed for distributed memory applications to enable interaction between processes running in different memory spaces. Shared-memory applications require coordination of processes, load balancing, and thread-safety [41, p.49] because shared variables can be accessed simultaneously by several processes. MPI provides a resource inexpensive method for inter-process communication but requires extra complexity in the message organization. For instance, *MPI_Receive* may freeze the process forever because of no matching *MPI_Send* [41, p.94].

Another potential pitfall is connected with the parallel IO. The simulation of the observational network does the parallel processing of global variables in the shared memory. At the same time, the disk data is read and written in a concurrent manner. Several threads representing Observation Units read the input media from the shared disk. Several threads perform the concurrent write to the same file system of the shared storage as illustrated in figure 6.8. In the case of the message passing approach, every node would write to its

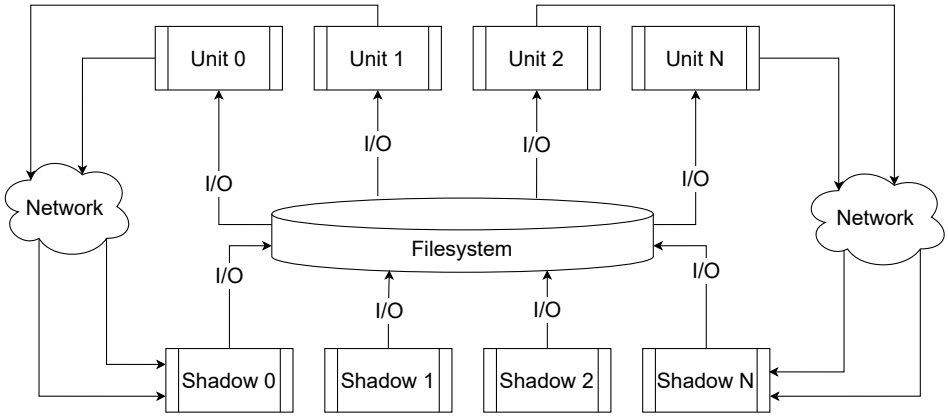


Figure 6.8: Parallel IO on the local filesystem

own isolated file system. That may ease the workload for a single disk but may bring the overhead to the joint file system, where such an application would be emulated. Another point is an extra communication cost during the simulation due to the absence of shared variables.

Go programming language is a popular way to build shared memory applications due to concurrent programming syntax [52]. There is an advantage of global variable's availability without *send* routines. Goroutine, a core element of the GO environment, is a function, lightweight thread that independently of other goroutines, but concurrently runs in the same address space [21]. Concurrency aids parallelism because if the data chunks are processed by independent goroutines (concurrently), it can be done in parallel, simultaneously. The limitation of the shared-memory programming is the total memory limit (both physical and virtual). An HPC cluster has enough memory resources to support more simultaneously running goroutines than an ordinary workstation is able to handle (see section 6.3.2). Go language has in-built primitives for the OS resource utilization such as spreading the functions among CPUs and cores [23]. That can improve the scaling ability on a simple workstation, but HPC cluster resources provide even more scaling potential.

Go channels

Goroutines emulate edge nodes that send and receive data to/from neighbors and through the network. Channels are "data pipes" which can take the input from one goroutine and deliver the output to another goroutine. [21] Such pipes are used for the node communication in the simulation as illustrated in figure 6.9.

Figure 6.9: Go channel working principle

Goroutines receive pointers to a named channel. Using those pointers, goroutines can send a data piece. Message passing over a named channel helps to assure that the exact sender is using the exact named channel to reach the exact receiver. Channels have support for internal buffers [21]. A buffered channel can take several messages before it would be full and blocked. A non-buffered channel can take only one message. In both cases, messages must be received before new messages can be sent from the origin.

Golang additional libraries

The number of cores employed can be altered using *runtime.NumCPU()*. In the simulation, the runtime package is used to force utilization of maximum cores available. Golang *runtime.MemStats* helps to utilize memory, the in-built garbage collector is constantly trying to free unused memory allocations. It keeps track of the activity in the *MemStats* structure [23].

Pprof shows memory leaks, top memory-expensive goroutines and more [22]. A profiler is a lightweight server that has a graphical user interface for the visual function dependency representation. The server responds to API requests and performs stress tests, returning runtime profiling data. Results can be found in section 7.3 - Profiling.

UUID package [26] used for the unique file ID generation. The package *GoNum* [24] has in-built functionality for calculating statistical and mathematical functions: factorial and binomial Probability Mass Function.

Crypto/MD5 package [25] used for the file hash generation. It calculates and returns the MD5 checksum of the data.

6.3.2 Hardware platform

Hardware development platform

Before porting to the HPC cluster, the project was assembled and tested in a form of the local run on the ordinary workstation. The workstation's characteristics are:
Intel® Core™ i7-9750H CPU (2666 MHz, 12 MB cache, 6 Cores), 32 GB DDR4-2667 ECC SDRAM

Software development environment

The main OS was Ubuntu 20.04 LTS with Go programming language compiler:
go version go1.13.8 linux/amd64
The executable file assembly has also been checked on Windows 11 Pro 21H2 with Go compiler: *go version go1.16.5 windows/amd64*

In both cases the application startup may be done either by calling:
go run ounet 50 168
or by passing arguments to the compiled Linux binary as:
./ounet 50 168
or to the Windows executable:
ounet.exe 50 168

Before the HPC run, the binary must be compiled using make *make build* on Linux and *go build* on Windows. The reason is the lack of internet connection on the HPC nodes, so binary with all the external libraries must be built before propagation to compute nodes.

Hardware runtime platform

The simulation is done on Fram[1] supercomputer owned by NRIS - Norwegian Research Infrastructure Services[2] and located at the University of Tromsø (UiT). Table 6.1 shows the main hardware parameters. The theoretical peak performance is 1.1 PFLOP/s [50]. The datacenter consists of Lenovo NeXtScale nx360 machines with Intel E5 / Intel E7 CPU. Most of the nodes have 64 GB memory, 8 nodes have 512 GB RAM ("big memory" nodes), 2 nodes have 6 TB RAM ("huge memory" nodes) [50].

1. Fram official documentation: `https://documentation.sigma2.no/hpc_machines/fram.html` (accessed: 2021-11-10)
2. NRIS home page: `https://www.sigma2.no/nris` (accessed: 2021-11-10)

Software runtime platform

The entire Fram cluster has CentOS[3] 7.9 build 2009.0.el7.centos.x86_64 kernel 3.10.0-1160 as the main OS at the moment of simulation execution. The queuing system installed and used for the job submission is Slurm[4], version 20.11.8. Go is loaded using: *Modules based on Lua[5]: Version 7.7.2*;
module load Go/1.11.2 in the Slurm batch script results in:
go version go1.11.2 linux/amd64

Details	Fram
System	Lenovo NeXtScale nx360
Number of Cores	32256
Number of nodes	1006
CPU type	Intel E5-2683v4 2.1 GHz Intel E7-4850v4 2.1 GHz (hugemem)
Max Floating point performance, double	1.1 Petaflop/s
Total memory	78 TiB
Total disc capacity	2.5 PB

Table 6.1: Fram supercomputer characteristics [50].

3. The CentOS Project: `https://www.centos.org/` (accessed: 2021-11-10)
4. Slurm Workload Manager documentation: `https://slurm.schedmd.com/documentation.html` (accessed: 2021-11-10)
5. Lmod documentation: `https://lmod.readthedocs.io/en/latest/` (accessed: 2021-11-10)

6.3.3 File system organization

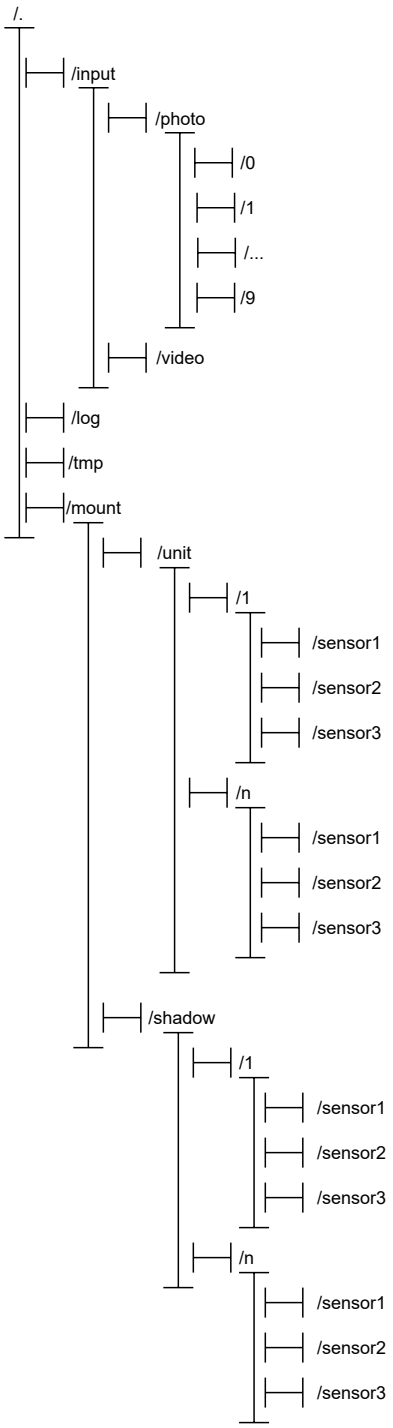


Figure 6.10: Simulation directory tree

The simulation is operating in the shared memory and on a shared disk as discussed previously in section 6.3.1. The shared disk hosts the directory structure with input and output subdirectories. The general directory tree is illustrated in figure 6.10:

- */log* keeps the terminal output and csv tables with instance runtime statistics. Every single node and router sends statistical data by the end of simulation to the printer function which produces such tables (example in table 6.2-6.3).
- */tmp* keeps temporary data for debugging, and logs from previous runs
- */input* has two subdirectories:
  - */video* where the input video can be found (for the testing purposes, the UiT promotional video is used [40]);
  - */photo* has 10 subdirectories with images taken from real OUs, example in figure 6.11;
- */mount* has subdirectories for OUs and SUs in accordance with their IDs. Every subdirectory has another set of three subdirectories:
  - */sensor1* for the text data from the temperature sensor;
  - */sensor2* for images from the OU's camera;
  - */sensor3* for videos from the OU's camera.

Unit ID	Last Sync since 2021-10-04 20:28:25	Never Synced	Packets all	Text all	Photo all	Video all	Text ->	Photo ->	Video ->	Text queue	Photo queue	Video queue
19	04.10.2021 20:57	0	4	1801	69	12	1716	0	0	85	69	12
0	04.10.2021 20:54	0	18303	1718	69	17	1452	61	0	266	8	17
7	04.10.2021 20:28	1	1	1801	77	21	0	0	0	1801	77	21
45	04.10.2021 20:51	0	3	1801	98	13	1414	0	0	387	98	13
32	04.10.2021 20:28	1	1	1801	92	28	0	0	0	1801	92	28
2	04.10.2021 20:37	0	2	1801	76	13	535	0	0	1266	76	13
18	04.10.2021 20:53	0	11703	1763	92	16	1459	39	0	304	53	16
1	04.10.2021 20:54	0	3	1801	76	10	1592	0	0	209	76	10

Table 6.2: Example of detailed OU output in the CSV table (beginning)

Unit ID	Text neib	Photo neib	Video neib	Exit Code	Link Wait	Data Rate (max 2500000 Bps)	No sync: !active	No sync: !net	No sync: !neib
19	0	0	0	1	0s	0.000000	1781	1719	0
0	0	0	0	1	13.481µs	346797.791592	1703	1618	0
7	0	0	0	1	13.481µs	0.000000	1788	1700	0
45	0	0	0	1	13.481µs	0.000000	1781	1727	0
32	0	0	0	1	13.481µs	0.000000	1794	1726	0
2	0	0	0	1	13.481µs	0.000000	1779	1704	0
18	0	0	0	1	14.035µs	693515.731597	1735	1672	0
1	0	0	0	1	14.035µs	0.000000	1781	1707	0

Table 6.3: Example of detailed OU output in the CSV table (continuation)

Goroutines have simultaneous access to the disk. In the case of several thousand parallel IO operations, undesirable storage overhead can occur. Consequently, for bigger simulations, disk operations were reduced significantly. Only the critical log data would be saved locally. All the data processing would happen in memory as described below in section 6.3.10.

Figure 6.11: OU camera data examples [13]

6.3.4 Application structure

Application is designed in the form of interconnected functions that are spread among a set of files written in Golang. A simplified application architecture in terms of functions is illustrated in figure 6.12. The simulation starts from the *main()* function which is the entry point. Then the *argument_processor()* is called to process command-line arguments and edit the global configuration accordingly. An additional task of the *argument_processor()* is to display the execution instruction if the argument *help* is detected. After that, the program exits as illustrated in listing 6.2. The order of arguments is not important, only the timeout must be entered in the end.

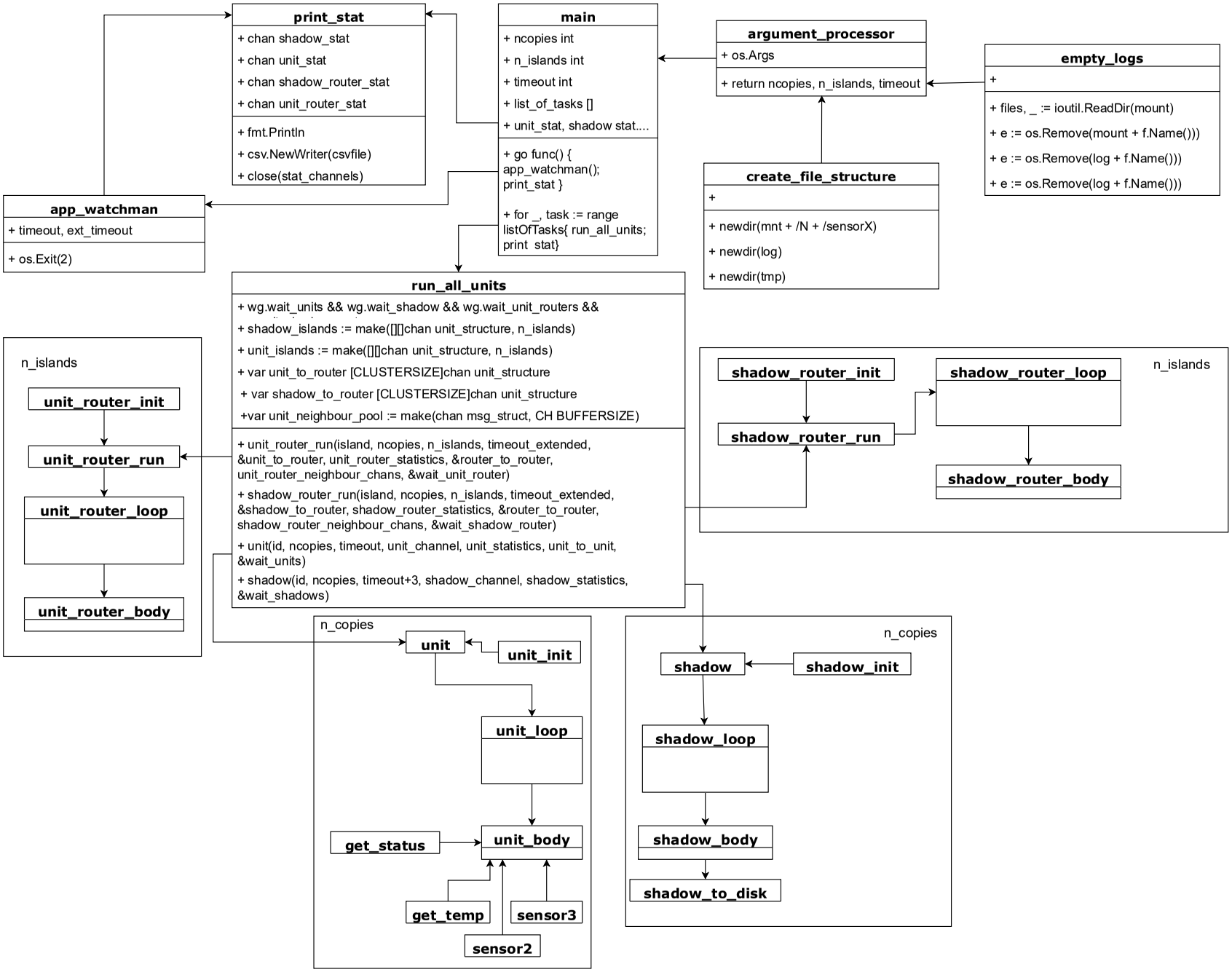


Figure 6.12: Go application scheme.

Listing 6.2: Simulation help meny

```
go run ounet help
_____________________________________________
                Arguments:
_____________________________________________

[ACTIVE5]
[HELP]

_____________________________________________
                Usage:
_____________________________________________

go run ounet (1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (last)

(1) - number of nodes in the simulation [5/50/500/5000...n]
(2) - information level in logs [status/info/debug]
(3) - choose data stream mode [light/medium/heavy]
(4) - network type at unit side [lora/ble/wifi]
(5) - unit availability chance [active1/active5/active10/active25/active50/active75/active100]
(6) - network availability chance [net1/net5/net10/net25/net50/net75/net100]
(7) - sync via neighbours [neib]
(8) - remove data from previous run [cleanup]
(9) - disk write mode [disk]
(10) - log write mode [log]
(11) - memory live info [mem]
(12) - node live info [live]
(13) - enable GPROF server [gprof]
(14) - enable infinite run [infinite]
(15) - enable 100% photo chance
(16) - enable 100% video chance
(last) - timeout in sec [5/15...n]
_____________________________________________

Example 1: ./ounet 25 status heavy wifi active50 net25 neib cleanup disk mem live log 150

Example 2: ./ounet 900 medium wifi infinite

_____________________________________________

exit status 2
```

If disk-related arguments (*disk*, *log* or *cleanup*) were entered, the *argument_processor()* would call functions *empty_logs()* and *create_file_structure()*. Those functions perform the log cleanup and recursively create subdirectories needed for the defined amount of nodes. If no node number is given and no timeout is defined, the default values are 3 nodes and 168 seconds of simulation.

When the command line argument parsing is done, the *main()* function calls *run_all_units()* together with *app_watchman()*. The last one guarantees that the simulation is not frozen after the timeout, the principle described in section 6.3.12. The first one initializes a slice of slices with channels for every type of instance. The function points every unit and router to their private channels. After the channel assignment is done, the function is making wait groups [21] to set a barrier for the simultaneous execution of goroutines representing nodes and routes. More details on unit's and router's runtime principles can be found in the next subsection: 6.3.5.

What is for the initial goroutine execution process, there is a nested loop that allocates the required number of islands based on the node amount. Island allocation happens dynamically. OUs and SUs receive IDs and channel references dynamically as shown in listing 6.3

Listing 6.3: Goroutine startup loop

```
// declare different waitgroups for different hosts
var wait_units sync.WaitGroup
var wait_shadows sync.WaitGroup
var wait_unit_router sync.WaitGroup
var wait_shadow_router sync.WaitGroup

// populate waitgroups dependent on the number of nodes
wait_units.Add(ncopies)
wait_shadows.Add(ncopies)
wait_unit_router.Add(n_islands)
wait_shadow_router.Add(n_islands)

remainder := calculate_remainder(ncopies)

// populate every island with OU's and SU's
for island := 0; island < n_islands; island++ {

  var unit_to_router [CLUSTERSIZE]chan msg_struct
  var shadow_to_router [CLUSTERSIZE]chan msg_struct

  // number of channels inside of the cluster is the same as the number of island members
  channels := CLUSTERSIZE

  if (island == n_islands-1) && (remainder != 0) {
    channels = remainder
  }

  // for every unit inside of this island, create channels for the communication with routers
  for channel := 0; channel < channels; channel++ {

    unit_to_router[channel] = make(chan msg_struct, CH_BUFFERSIZE)
    shadow_to_router[channel] = make(chan msg_struct, CH_BUFFERSIZE)

    id := channel + island*CLUSTERSIZE

    go func(unit_channel chan msg_struct) {
      unit(id, ncopies, timeout, unit_channel, unit_statistics, unit_to_unit, &wait_units)
    }(unit_to_router[channel])

    go func(shadow_channel chan msg_struct) {
      shadow(id, ncopies, timeout_extended, shadow_channel, shadow_statistics, &wait_shadows)
    }(shadow_to_router[channel])
  }
go func(island int) {
  unit_router_run(island, ncopies, n_islands, timeout_extended, &unit_to_router,
    unit_router_statistics, &router_to_router, unit_router_neighbour_chans, &wait_unit_router)
}(island)

go func(island int) {
  shadow_router_run(island, ncopies, n_islands, timeout_extended, &shadow_to_router,
    shadow_router_statistics, &router_to_router, shadow_router_neighbour_chans, &wait_shadow_router)
}(island)
}

// wait all groups
wait_units.Wait()
wait_shadows.Wait()
wait_unit_router.Wait()
wait_shadow_router.Wait()
```

When the timeout is reached, nodes start to report termination. When every instance has reported successful termination or the application shutdown was forced, the *print_stat()* function is called. This function receives pointers to global buffered channels. Those channels are filled by units and routers with simple statistical data that is printed out by the end of the simulation. After that, the application finalizes.

All the main and supplementary functions are spread among several files:

- main.go - initial functions that create the structure and start the simulation;
- config.go - global configuration file;
- unit.go - OU-specific functions;
- shadow.go - SU-specific functions;
- unit_router.go - functions for the Unit Router;
- shadow_router.go - functions for the Shadow Router;
- network.go - network communication functions;
- neighbour.go - primitives needed for synchronization between neighbor edge nodes;
- sensors.go - temperature sensor and camera emulation;
- utils.go - functions for measurement units conversion and statistical expectation;
- disk.go - *os library* based functions for file tree creation and cleanup;
- ounet_test.go - module testing.

The executable can be built using *go build ounet*.

6.3.5 Simulation topology

Functions distributed among files described in the previous subsection have a role to emulate real-world network instances. With respect to the time scaling method, described in section 6.2.2, the piece of Go code from listing 6.4 shows the main loop of any running instance by the example of OU emulation.

Listing 6.4: Unit run loop

```
for x := range time.Tick(d) {
  f(x, i, unit, unit_channel, unit_to_unit)

  // get the current time to calculate if deadline is over
  t := time.Now()
  elapsed := (t.Sub(start)).Seconds()

  // if deadline is now, assign exit code, terminate and return success code
  if int(elapsed) > timeout && !infinite && !unit.locked {
    unit.exit_code = EXIT_TIMEOUT
    return true
  }
}
```

The *for loop* repeats every duration of time *d* which is calculated in the caller function. Duration is set to 1 second but can be random or any fixed period of time. The loop runs the *unit_body()* function, which updates the structure representing the instance. If the time elapsed exceeds the timeout value, the loop breaks and returns successful termination code. The function for structure updating is passed as the argument *f* to the function containing the loop. A closer look at the implementation of specific instances is needed before the data flow principle would be explained in section 6.3.6.

Observation Units

OU is emulated using the loop principle discussed above. The main difference between the OU emulation and other instances here is in the structure that represents the OU. A structure combines several different variables of various types [21].

Every unit gets an associated structure that holds counters, flags, names, and some limited amount of text data from sensors. The OU's structure has information about its ID, cluster, operation and network status, links to associated input directories, lists of sent and pending files by types, network last and best transfer rate, last synchronization time, exit code and some others. This structure is created first by function *unit_init()* and updated by function *unit_body()* which is passed to the *unit_loop()*. Both *unit_init()* and *unit_body()* are called by the function *unit_run()*, which is called directly from the main function and *run_all_units()* as explained in section 6.3.4. The *unit_body()* function communicates to sensors, checks the network availability, and sends the data over the network.

Shadow Units

The SU emulation follows the same principles as OU's one does. The most striking difference is the structure contents: it has some shadow-specific variables like the rate of data downloading, average message travel time on different segments, and so on. *shadow_body()* function is responsible for the communication with associated SR and for the timestamp processing. As the shadow side is expected to have more computational resources, it takes the responsibility of timestamp collecting and aggregating into average values. More details on message timestamping are provided in 6.3.12.

Unit Routers

Unit Routers are initialized and operate in the same manner as OUs and SUs do. The *unit_router_initialize()* returns the structure representing UR. It calculates how many units would be connected, routers ID, status, message counters and so on. When the structure is ready, the *unit_router_body()* function updates it using a function similar to the one in listing 6.4. In addition to this, *unit_router_body()* checks all the channels connected to OUs within the island and sends them further to the SR. Before the sending, the router does the timestamping and checks if the sender ID corresponds to the channel ID as all the units have private channels.

Shadow Routers

In contrast to Unit Router, Shadow Router sends the data to the final destination point - to SUs. The SR double checks that incoming channel ID matches the recipient ID.

6.3.6 Application data flow

Structures of all units are briefly explained in the subsection above. Figure 6.13 illustrates the path of the data stream that every single message has to follow in order to travel from the OU to the SU. All the communication goes via private channels.

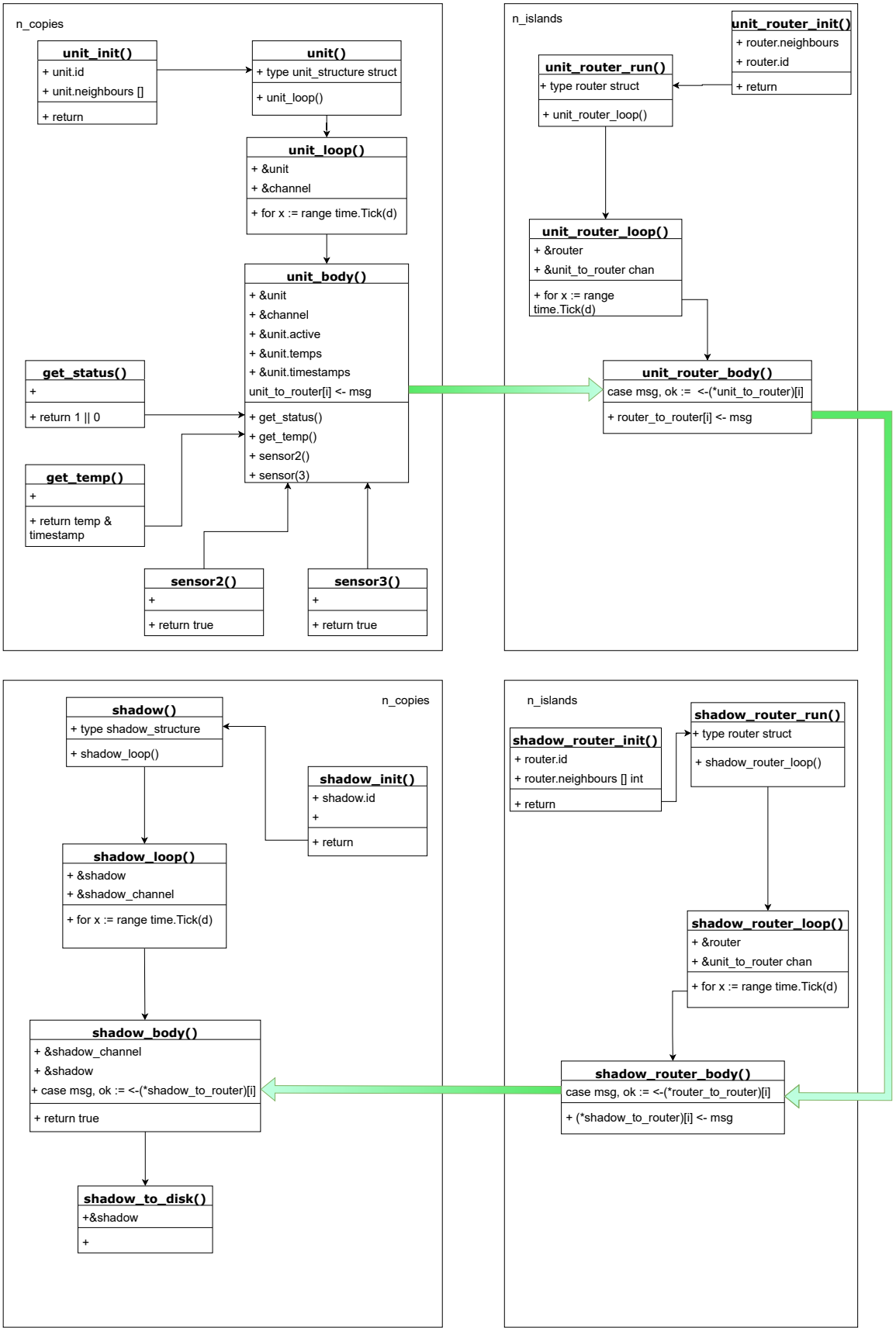


Figure 6.13: Intra-goroutine data flow in the simulation.

Every OUs gets pointers to the channel, that goes to the UR, on the initial call from the *run_all_units()*. Every UR has pointers to unit channels within the cluster and pointers to channels that are connected to the SR on the other side. Shadow Router has pointer channels from UR and pointers to channels to SUs within the cluster. SU has the pointer to the private channel that leads to the SR. As illustrated in figure 6.13, the data is pulled and pushed by the "*_body()*" type of functions.

6.3.7 Network simulation

Every message is extracted from Go channels using *case msg,ok := <-(*channel)[i]* principle. This method is non-blocking. If the channel is closed, the *Boolean* would be *!ok*, but that wouldn't raise a critical problem. The information is only used for debugging and orchestration improvement. If the channel is empty, the *default* rule of *switch* would be applied, but no error appears (listing 6.5). Table 6.4 shows IO result dependent on the channel status.

	NIL	Open	Closed
Send	Blocked	Allowed	Panic
Receive	Blocked	Allowed	Allowed

Table 6.4: Go channel IO in different channel states [20]

Concurrent packet routing

Message extraction happens concurrently as illustrated in listing 6.5. If there is a stream of messages on one channel, the serial approach would stop and wait until the work with the channel is done. As a result, a longer link wait time can be experienced at the OU side. Simultaneous data extraction and processing on the range of channels emulates the higher cumulative bandwidth. Though it might be not viable when only one edge node has data to send, the issue is critical on higher synchronization chances.

Listing 6.5: Concurrent packet routing

```
var from_unit sync.WaitGroup
from_unit.Add(router.n_members)

// go through channels one by one and read messages
for channel := 0; channel < router.n_members; channel++ {

  // but concurrently
  go func(channel int) {
    defer from_unit.Done()

    // check if there is a message
    select {
    case message, ok := <-(*unit_to_router)[channel]:
      message.ur_arrival_time = time.Now()

      // if channel is closed
      if !ok {
        ...
        //router.terminated = true
      } else {
        ...
      }

    default:
        ...
    }
  }(channel)
}
from_unit.Wait()
```

Artificial Bottleneck

The bandwidth can't be unlimited. The simulation supports several types of network protocols that have their limitations. The list of supported networks and their characteristics is in the table 6.5.

The recent version of Wi-Fi protocol IEEE 802.11ax has a maximum theoretical data transfer rate of 2.4 Gbps that is equal to 2400000 Kilobytes per second (Kbps) [44]. Research on a very long-distance Wi-Fi network shows the possibility of a successful connection over WiMAX with a 279-kilometer distance between nodes and transfer rate around 3 Mbps [15]. The article highlights the need for massive non-portable equipment for such connections. For the network of observational edge nodes in the arctic tundra, portability is an important aspect. For ordinary Wi-Fi connections, especially for older standards like IEEE 802.11g, 100 and 200-meter distance between nodes can reduce the network throughput 2.5 times and 10 times accordingly [28]. OUs in the real world have significant distances between each other, so 20000 Kbps is taken as an expected Wi-Fi maximum data transfer rate.

Maximum Transfer Unit (MTU) is another characteristic of networks, for Wi-Fi, it is around 2000 bytes [44] and set to be 2312 bytes in the simulation. BLE has maximum MTU 512 bytes, but various vendors set their limitations. BLE has support for Data Length Extensions (DLE) [1] with 257 bytes MTU and 244 bytes for the payload. The data rate is up to 2 Mbps, but for the simulation with significant distances between nodes 1360 Kbps is chosen as a positive scenario.

For LoRa, the MTU size depends on the data rate used. Some of LoRa family protocols support point-to-point communication. So for LoRa emulation, no exact version is taken, just the general principle. While theoretical rate is 54 Kbps, experiments with an increasing spread show around 12 Kbps [5]. For the simulation, the MTU size of 64 bytes and data transfer rate of 27 Kbps are chosen. The absolute correctness of emulated network characteristics is not vital, approximate values are sufficient to show the difference in simulations behavior in various network environments.

Network	MTU (bytes)	Rate (Kbps)
LoRa	64	27
BLE	244	1360
Wi-Fi	2312	20000

Table 6.5: Network characteristics

A "bottleneck" is needed to reduce the transfer rate in the simulation and create some artificial obstacles. A random run of the simulation shows average 7.3 Gbps get rate which goes up to 49.5 Gbps on a randomly chosen shadow island. The function attached in listing 6.6 illustrated the principle of the artificial bottleneck in the simulation. Any message sequence is monitored during the transfer process, *func bottleneck()* is called to check if the data volume transferred during the period of time is too high. The ratio between the total transferred data amount in this iteration and the cumulative time elapsed in this iteration shows the data rate approximation. If such an approximation is higher than the maximum allowed value from table 6.5, the function enters the "sleep" loop creating the artificial delay. When the estimated transfer rate is under given limits, the function returns.

Listing 6.6: Artificial bottleneck

```
func bottleneck(start time.Time, size_of_processed_data int, id int) {

  // get the time since start of transfer
  elapsed := time.Since(start)

  // convert Kbps To Bps and assign max data rate in accordance with network characteristics
  data_rate_bps := KbpsToBps(data_rate)

  // if time elapsed is > seconds
  if int64(elapsed) == elapsed.Nanoseconds() && (int(elapsed.Seconds()) != 0) {

    // calculate the current rate
    bps := float64(size_of_processed_data) / elapsed.Seconds()

    // if current rate exceed max rate
    if bps > float64(data_rate_bps) {

      // start a ticker function until rate will under the rate from network technical max
      for range time.NewTicker(time.Second).C {
        delayed := float64(size_of_processed_data) / time.Since(start).Seconds()
        if delayed < float64(data_rate_bps) {
          return
        }
      }
    }
  }
}
```

6.3.8 Communication

The main primitive used for communication over the network is a message. The message is a *go structure*. It has a field for the data which is equal to the MTU. MTU in this simulation is always smaller than the maximum size of the message that can go over a Go channel (The Go channel maximum is 2^{16} bytes = 64 kilobytes) [21]. The message structure has pre-defined fields for timestamps because every step of the route is known. There is also information about the sender's ID and the cluster's ID, the receiver must have the same ID as the sender. The type of the message (*PING*, *TERMINATED*, *ACK*, *DATA*...) and the type of the data (*TEXT*, *PHOTO*, *VIDEO*) provided. The first message of the transfer contains the file name, the checksum, and the number of messages in the sequence. The last message contains *message.last_one = true* flag.

Direct communication

When an OU has a connection to the back-haul network and is awake, it would send the data over the network. The first message states the type of the message and the file type. *func send_message()* determines if more messages are needed for the transfer. If there is only one message, it is sent to the router right away. If there is a sequence of messages, those would be sent in the loop using *func cut_and_send()* function. *func send_message()* requires a pointer to the channel to use. An UR can see that the message number is not the same as the total number of messages expected and accepts the data stream until the message with flag *message.last_one = true* is received. The SR does the

same as the UR. SUs receive all the incoming data disregarding any order, but if there is a sequence of messages, an SU starts the receiving loop in function *func get_and_assemble()* until the last message is found. When the last message arrives, the consistency check starts.

Neighbor communication

If communication goes via the neighbor, the message is to be sent in the same way as in the case of direct communication, but the channel would be the right neighbor channel. If a neighbor has a connection to the back-haul network and is awake, it would resend the message sequence via its private channel with the *neib* flag added. It helps the router to reroute the message to the private channel of the original sender.

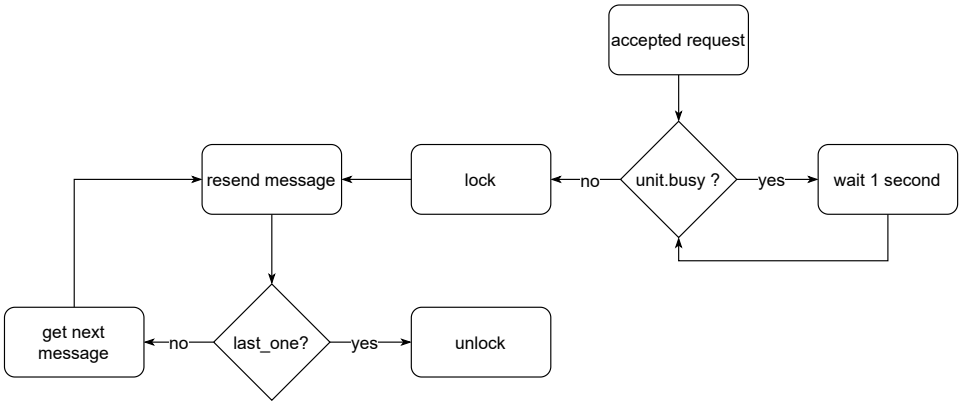


Figure 6.14: The mutex principle

Mutual exclusion

The locking principle is illustrated in figure 6.14. If a neighbor reported the connection availability, it must take the incoming data. It means that the neighbor has to process both its own data, and the data from its left neighbor. An internal lock *unit.busy* is needed and is set for the data transfer time. If there is a transfer of own data, the node would unlock in the end of transfer. If there is a data set from the left neighbor, the sender would issue an MPI-like the message of the type *message.msg_type=UNLOCK*. The neighbor wouldn't take this message further on the network but would cut the transfer channel, unlock and continue with its own data. So if the incoming request would appear during the transfer, it would have to wait until the end of the transfer that is already underway.

6.3.9 Random behavior emulation

The random behavior emulation makes the model closer to real-world behavior. Working conditions may vary in the wildlife environment, that's why the simulation is designed with a certain degree of randomization.

Random delay

The core loop which is the fundament for all instance types in the simulation has support for the random delay. For the time being, units wake up every second. In principle, such a wake-up period can be randomized. There are two variables: *maxdelay := 10* and *mindelay := 1* which are maximum and minimum vales of the delay. The calculation of time duration is done using the *rand* library and is illustrated below:
*duration := time.Duration(int64((rand.Intn(maxdelay-(mindelay)) + mindelay)) * int64(time.Second))*
Such a wakeup delay randomization can enable different behavior of OUs.

Input data

There are three types of data in the simulation: text, photos and videos. Only the input video file [40] is the same for all the units. Photo input directory is assigned on the startup from a random photo directory as illustrated in the directory tree 6.10. Every OU is assigned to one directory, which has up to 10 files. Every time when a photosensor is generating an image, it chooses a random image and assigns a unique name using the *UUID*[26] library.

The same principle works for the text data emulated by the temperature sensor. The sensor returns a timestamp and a random temperature value within a given range.

Sensors

The sensor behavior is randomized and controlled by the global configuration located in *config.go* as discussed in section 6.3.4. Every sensor has a success rate in range $0 < rate < 1$. If the image chance is set to 0.1, the sensor would generate an image approximately once in ten calls. If the OU wakes up every hour, a new image would be generated approximately every 10 hours. The code simply returns *return rand.Float32() < photo_chance*. The video chance and video file naming follow the same pattern.

6.3.10 Disk mode

There are two disk modes: the "no disk" mode and the "disk mode". The disk mode is IO-oriented and is designed for a small-scale simulation which would save files to the local file system. The data is read from the disk as discussed in the previous subsection. The size of the file is divided by the MTU size to get the number of messages to send. On the SU side, the entire message sequence is written to the local disk into the */mount/o/sensor1/*. subdirectory, where *o* is the ID of the SU and the *sensor1* is the sensor type.

	Wi-Fi msg	BLE msg	LoRa msg
Text Size	1	1	1
Photo Size	300	2843	10839
Video Size	3000	28426	108374

Table 6.6: Number of messages required to send one "empty" file by type

No disk mode

In the case of "no disk" mode, the standard message number is set to 1 for the text data, 300 for images, and 3000 for videos as shown in table 6.6. This number corresponds to an approximate number of messages required to transfer an ordinary file, but in this case, the file will be "empty" - filled with nonsense data. The reason for the "no disk" mode is to avoid the simulation IO overhead. Simulation of several thousand nodes would spend too many resources on the parallel read on and the concurrent write. Edge nodes with the shared file system are not the case for the real-world network, it is only a part of the simulation.

For BLE and LoRa, the number of messages to be sent is higher, because MTU sizes are smaller. As illustrated in figure 6.15, LoRa would send 36 times more messages than the Wi-Fi due to the smaller MTU size. In case of connection via BLE, the number of messages required to transfer the same data amount is higher by the factor of 9.0 compared to the Wi-Fi environment.

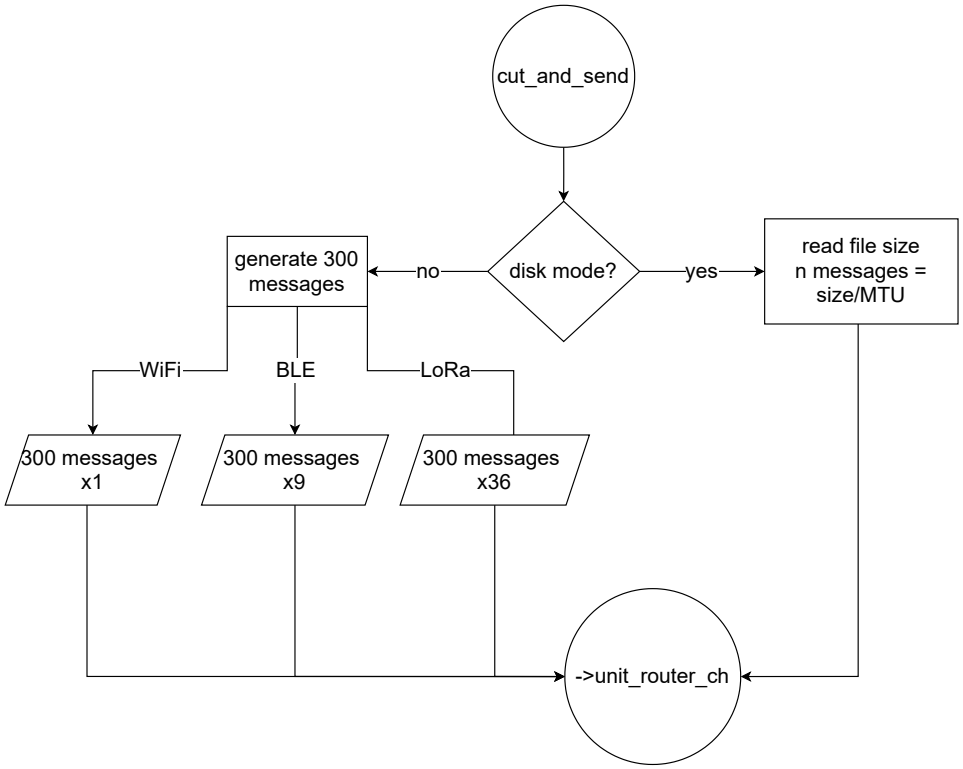


Figure 6.15: Message generation principle in *!disk* mode

6.3.11 Data consistency check

Before any file is sent, its checksum is calculated using *crypto/md5* package. The hash is added to a special field inside the message among with the file name. So the file can be disassembled into smaller messages of MTU size that are sent over the network one by one. The entire process is illustrated in figure 6.16.

When all the messages are received, the SU assembles the file and performs the consistency check as illustrated in the figure. If the new checksum is identical to the hash from the first message - the consistency test is passed. If not, the fatal error would be raised, which means that the simulation needs further data synchronization primitives adjustment and general debugging.

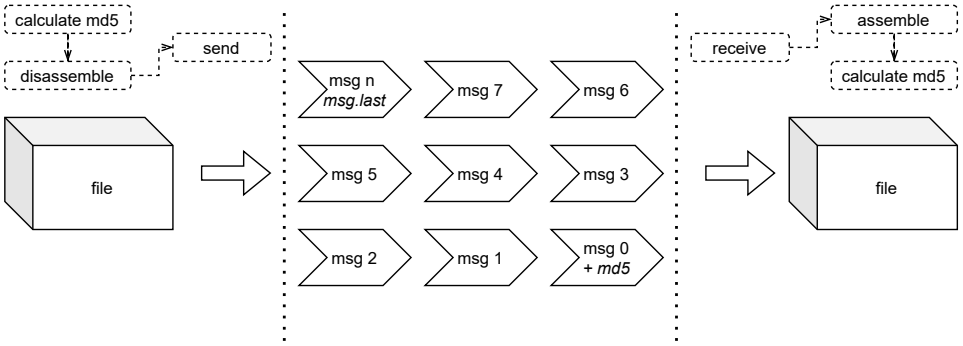


Figure 6.16: Data consistency check process

6.3.12 Runtime observation

The runtime observation helps to monitor the simulation in real-time. There are two techniques described in this subsection: extended timeout and packet time tracking.

Extended timeout

If "*infinite*" command-line argument is provided, the simulation would run as long as it is possible. But when the exact time is given, the main loop of the instance checks if the elapsed time exceeds the timeout. If the timeout is reached, the instance sends a message to the UR with *terminated* flag. The Unit Router brings this message to the SR and, further to the SU. When all units within the cluster have terminated, both routers terminate as well.

In practice, it happens sometime that the OU can not terminate within the given timeout, because it might not be done with the data transfer or still waits for some response from the network. For this particular case, an extended deadline, which is usually, a half simulation time, is given. If nodes can not finalize within extended deadline, the *app_watchman()* function triggers the simulation exit on the OS level by returning *false* to the initial *main()* function (listing 6.7). This function runs concurrently with other goroutines during all the simulation periods. Before exiting, static data is collected over statistical channels.

Listing 6.7: Timeout control function

```
func app_watchman(start time.Time, ncopies int, n_islands int, timeout int) bool {

  // declare MemStats to get \ac{ram} data using runtime package
  var m runtime.MemStats
  var goroutines = 0

  for {
    t := time.Now()
    elapsed := (t.Sub(start_time)).Seconds()

    if int(elapsed) > timeout_killer && !infinite {
      return false
    }
  }
}
```

Packet time tracking

Timestamping is a resource-demanding technique as highlighted by Fidge [14]. The profiling section (7.3) confirms that significant resources are used for the collection of this information. But the value of the monitoring is higher than the cost in this case.

SUs are expected to have more computational resources than OU. That is why the shadow side is responsible for the timestamp processing and aggregating into average values which are kept inside the SU structure in the simulation. The main principle is that at every stage, the message receives a timestamp on any arrival or departure. As a result, the delay between the OU and the UR, the SR and the SU can be estimated.

Another task of the message timestamping is to indicate the delay before the UR is able to handle the incoming data stream. This value can show a degree of asynchrony between OUs and associated routers.

To sum up the chapter, not all the functionality is discussed in detail. Only the basic functions, approaches, and primitives needed for the simulation working principle are explained. The next chapter covers the practical use of the simulation.

7 Evaluation

To keep a homogeneous environment and follow the scientific experiment guidelines (see section 3.3.2), all simulations are done on the cluster. Even those experimental cases without extensive memory requirements are executed on the cluster's compute nodes. Experiments requiring hundreds of gigabytes of shared memory are executed on big memory and huge memory nodes.

7.1 Known shortcomings

The prototype is not perfectly balanced and adjusted. Some uncertainties may appear during the runtime. Typical errors are identified and listed here to be considered and, thus, reduce the overall uncertainty level.

7.1.1 Send on a closed channel error

As discussed in chapter 6.3 Implementation, node-to-node communication happens over gochannels via simple messages of certain types. Some of those messages require a response. In the case of the NEIB synchronization in a big network, the node may finalize and close its neighbor channels too early, because the global clock doesn't guarantee the proper orchestration (see section 2.3.4). As a result, a neighbor may freeze forever waiting for a response or send a message on a closed channel which would result in *panic* (section 6.3.7) and

application's forced termination. In practice, it happens rarely but randomly. The issue is mostly observed for NEIB simulations with > 200000 OUs. In the case of future development, the node communication and orchestration methods should be improved, but such an error is not critical and doesn't influence stated research tasks.

7.1.2 Transfer rate deviation

For a single OU, the transfer rate can be unexpectedly high. It happens because a random small file can be transferred less than in a second. The issue occurred several times during local runs. It can be explained by the 3 GB/s read speed of the local NVMe SSD used for the prototype development and testing. A random file that is small enough can be transferred less than in a second avoiding the bottleneck function (section 6.3.7). Transfer rates with deviation are identified during the runtime. The transfer rate value is set to "-1" in such a case and is excluded from the average transfer rate calculation. These values don't raise the overall uncertainty level. For 100,000 OUs simulation, the number of nodes with transfer rate deviation rarely exceeds 3 or 5.

7.1.3 Time scaling issue

As discussed in section 6.2.2, OUs wake up once a second. That is the timescale of one hour in the real world. But for the network and routers, the time is not scaled, and the data transfer is calculated in actual Bytes per second (BPS). It may result in a problem when units are about to terminate due to timeout, but the data transfer is still underway. Generally, networks with low bandwidth and high data transfer volumes would show some data delivery delay. In the prototype, the issue doesn't affect the link delay or transfer rate calculation in the middle of runtime. But for large clusters emulation and significant data amounts, data delivery delay can be higher than the expected delay in the real cluster. Such an issue doesn't affect the prototype significantly, but the data delay value extracted by the end of the simulation should be used with caution.

7.2 Benchmarks

Benchmarking is implemented in a form of module testing. Module tests are assertions performing positive and negative control as discussed earlier in section 3.3.2. As illustrated in listing 7.1, a go module responsible for project testing runs tests from *ounet_test.go*.

Listing 7.1: Test module output

```
=== RUN   TestConversion
--- PASS: TestConversion (0.00s)
=== RUN   TestNodeIndex
--- PASS: TestNodeIndex (0.00s)
=== RUN   TestStatistics
=== RUN   TestStatistics/30,0.05000,1
=== RUN   TestStatistics/30,0.00500,1
=== RUN   TestStatistics/60,0.02000,1
=== RUN   TestStatistics/60,0.00020,1
=== RUN   TestStatistics/60,0.00002,1
--- PASS: TestStatistics (0.00s)
    --- PASS: TestStatistics/30,0.05000,1 (0.00s)
    --- PASS: TestStatistics/30,0.00500,1 (0.00s)
    --- PASS: TestStatistics/60,0.02000,1 (0.00s)
    --- PASS: TestStatistics/60,0.00020,1 (0.00s)
    --- PASS: TestStatistics/60,0.00002,1 (0.00s)
=== RUN   TestSensors
--- PASS: TestSensors (0.00s)
=== RUN   TestInitUnit
--- PASS: TestInitUnit (0.00s)
=== RUN   TestInitShadow
--- PASS: TestInitShadow (0.00s)
=== RUN   TestInitUnitRouter
--- PASS: TestInitUnitRouter (0.00s)
=== RUN   TestInitShadowRouter
--- PASS: TestInitShadowRouter (0.00s)
=== RUN   TestDiskUtils
--- PASS: TestDiskUtils (0.00s)
=== RUN   TestMemory
--- PASS: TestMemory (0.00s)
PASS
ok      ounet   0.041s
```

Sensor functionality, the correctness of measurement unit conversion, instance initialization, etc. is ensured by benchmarking. Test is run periodically during the prototype development process to ensure that no methods were altered occasionally.

7.3 Profiling

Profiling is the code adjustment to reduce undesirable memory leaks and computational resource wasting. The entire profiling process is done using the *PPROF* [22] package. The package supports several profiling modes aimed at various application architectures. For the prototype, the total amount of allocated memory, memory in use, goroutines, and CPU statistics were evaluated. Profiling results provided for the simulation of 10000 OUs running in Wi-Fi mode with 5% back-haul network availability chance and 10% chance of node being awake. Listing 7.2 shows the top of functions that require memory allocation. Most of the memory is allocated for message processing in function *cut_and_transfer()*. Much memory is allocated for the OU functionality in the *unit_body()*. This is the expected behavior, excepting *get_shadow_disk_path* which shouldn't use many resources to assign exclusive paths to SUs.

Listing 7.2: PPROF alloc profiling *-top* representation

```
Type: alloc_space
Time: Oct 23, 2021 at 2:29pm (CEST)
Showing nodes accounting for 656.02MB, 97.84% of 670.49MB total
Dropped 39 nodes (cum <= 3.35MB)
      flat  flat%   sum%        cum   cum%
  494.25MB 73.71% 73.71%   494.25MB 73.71%  main.cut_and_transfer
   42.53MB  6.34% 80.06%   536.78MB 80.06%  main.unit_body
   27.01MB  4.03% 84.09%    27.01MB  4.03%  main.run_all_units
   26.50MB  3.95% 88.04%       29MB  4.33%  main.get_shadow_disk_path
   10.56MB  1.58% 89.61%    10.56MB  1.58%  main.get_message_timetrack
    9.50MB  1.42% 91.03%     9.50MB  1.42%  os.newFileStatFromWin32finddata (inline)
    6.50MB  0.97% 92.00%     6.50MB  0.97%  runtime.malg
    5.50MB  0.82% 92.82%     5.50MB  0.82%  os.newFile
       5MB  0.75% 93.57%    15.51MB  2.31%  os.openDir
    4.50MB  0.67% 94.24%    15.06MB  2.25%  main.process_received_msg
    4.50MB  0.67% 94.91%       16MB  2.39%  os.(*File).readdir
    3.65MB  0.54% 95.46%    30.66MB  4.57%  main.main
    3.50MB  0.52% 95.98%   578.30MB 86.25%  main.unit
    3.50MB  0.52% 96.50%     3.50MB  0.52%  strconv.formatBits
    2.50MB  0.37% 96.87%   541.29MB 80.73%  main.unit_loop
       2MB   0.3% 97.17%    47.57MB  7.09%  main.shadow
       2MB   0.3% 97.47%       4MB  0.6%  syscall.FullPath
    1.50MB  0.22% 97.69%     3.50MB  0.52%  syscall.UTF16ToString
   0.50MB 0.075% 97.77%    45.56MB  6.80%  main.shadow_loop
   0.50MB 0.075% 97.84%    32.51MB  4.85%  main.get_random_input_photo_dir
```

Listing 7.3 shows the heap memory. The central function - *run_all_units()* which starts all the units together with the function for unit body update take most of the memory resources.

Listing 7.3: PPROF heap profiling *-top* representation

```
Type: inuse_space
Time: Oct 23, 2021 at 2:29pm (CEST)
Showing nodes accounting for 73.31MB, 100% of 73.31MB total
      flat  flat%   sum%        cum   cum%
   27.01MB 36.84% 36.84%    27.01MB 36.84%  main.run_all_units
   18.02MB 24.58% 61.43%    19.53MB 26.64%  main.unit_body
    6.50MB  8.87% 70.30%     6.50MB  8.87%  runtime.malg
    3.65MB  4.98% 75.28%    30.66MB 41.83%  main.main
    3.50MB  4.78% 80.06%    28.04MB 38.24%  main.unit
    2.50MB  3.41% 83.47%     3.51MB  4.79%  main.process_received_msg
    2.50MB  3.41% 86.88%    23.53MB 32.10%  main.unit_loop
       2MB  2.73% 89.61%     7.01MB  9.56%  main.shadow
       2MB  2.73% 92.34%     2.51MB  3.42%  time.NewTicker
    1.50MB  2.05% 94.39%     1.50MB  2.05%  main.cut_and_transfer
    1.01MB  1.37% 95.76%     1.01MB  1.37%  main.get_message_timetrack
       1MB  1.36% 97.12%        1MB  1.36%  strconv.formatBits
    0.60MB  0.82% 97.94%     0.60MB  0.82%  runtime.allgadd
    0.51MB  0.69% 98.64%     0.51MB  0.69%  time.startTimer
    0.50MB  0.68% 99.32%     0.50MB  0.68%  main.shadow_router_run
    0.50MB  0.68%   100%     5.01MB  6.83%  main.shadow_loop
```

Listing 7.4 confirms the intrusive nature of chosen runtime observation technique (section 2.3.4). The goroutine checks if the global deadline is over. It takes more than half of the CPU resources consumed by the simulation. Probably bottleneck and timestamping require significant CPU resources for monitoring and message tracking. Functions that update the main body of the OU take some CPU resources but are supposed to have a much higher share.

Listing 7.4: PPROF cpu profiling *-top* representation

```
Type: cpu
Time: Oct 23, 2021 at 2:28pm (CEST)
Duration: 30.12s, Total samples = 37.16s (123.36%)
Showing nodes accounting for 33.38s, 89.83% of 37.16s total
Dropped 200 nodes (cum <= 0.19s)
      flat  flat%   sum%        cum   cum%
     5.74s 15.45% 15.45%     18.76s 50.48%  main.app_watchman
     4.72s 12.70% 28.15%      4.72s 12.70%  time.Duration.Seconds
     3.37s  9.07% 37.22%      3.37s  9.07%  time.now
     2.59s  6.97% 44.19%      2.60s  7.00%  time.Time.Sub
     2.47s  6.65% 50.83%      5.85s 15.74%  time.Now
     1.30s  3.50% 54.33%      1.30s  3.50%  runtime.memmove
     1.23s  3.31% 57.64%      1.25s  3.36%  runtime.stdcall6
     1.09s  2.93% 60.58%      1.09s  2.93%  runtime.siftdownTimer
     0.97s  2.61% 63.19%      0.97s  2.61%  runtime.stdcall1
     0.66s  1.78% 64.96%      3.09s  8.32%  main.unit_body
     0.64s  1.72% 66.68%      0.64s  1.72%  runtime.procyield
     0.54s  1.45% 68.14%      0.54s  1.45%  runtime.gopark
     0.42s  1.13% 69.27%      1.32s  3.55%  runtime.lock2
     0.42s  1.13% 70.40%      0.43s  1.16%  runtime.stdcall2
     0.39s  1.05% 71.45%      0.39s  1.05%  runtime.casgstatus
     0.39s  1.05% 72.50%      1.35s  3.63%  runtime.chanrecv
     0.36s  0.97% 73.47%      1.67s  4.49%  math/rand.(*lockedSource).Int63
     0.33s  0.89% 74.35%      0.33s  0.89%  runtime.(*gList).pop
     0.30s  0.81% 75.16%      3.76s 10.12%  main.shadow_loop
     0.30s  0.81% 75.97%      0.32s  0.86%  runtime.unlock2
```

Listing 7.5 shows that goroutines executed inside the simulation are spread evenly. As expected, for 10000 OUs there are 10000 goroutines. For SUs there are 10000 more goroutines. Additionally, there are 2*157 goroutines for URs and SRs, and 1 for the *app_watchman* running concurrently. The output from the listing is represented graphically in figure 7.1.

Listing 7.5: PPROF goroutine profiling *-top* representation

```
Type: goroutine
Time: Oct 23, 2021 at 2:29pm (CEST)
Showing nodes accounting for 20335, 100% of 20337 total
Dropped 51 nodes (cum <= 101)
      flat  flat%   sum%        cum   cum%
     20335   100%   100%      20335   100%  runtime.gopark
         0     0%   100%      10000 49.17%  main.run_all_units.func1
         0     0%   100%      10000 49.17%  main.run_all_units.func2
         0     0%   100%        157  0.77%  main.run_all_units.func3
         0     0%   100%        157  0.77%  main.run_all_units.func4
         0     0%   100%      10000 49.17%  main.shadow
         0     0%   100%      10000 49.17%  main.shadow_loop
         0     0%   100%        157  0.77%  main.shadow_router_loop
         0     0%   100%        157  0.77%  main.shadow_router_run
         0     0%   100%      10000 49.17%  main.unit
         0     0%   100%      10000 49.17%  main.unit_loop
         0     0%   100%        157  0.77%  main.unit_router_loop
         0     0%   100%        157  0.77%  main.unit_router_run
         0     0%   100%      20286 99.75%  runtime.chanrecv
         0     0%   100%      20283 99.73%  runtime.chanrecv2
```

When it comes to simulations with NEIB enabled, listing 7.6 shows that most memory is consumed by checking neighbor channels for incoming synchronization requests. In the case of future development, this function requires a redesign.

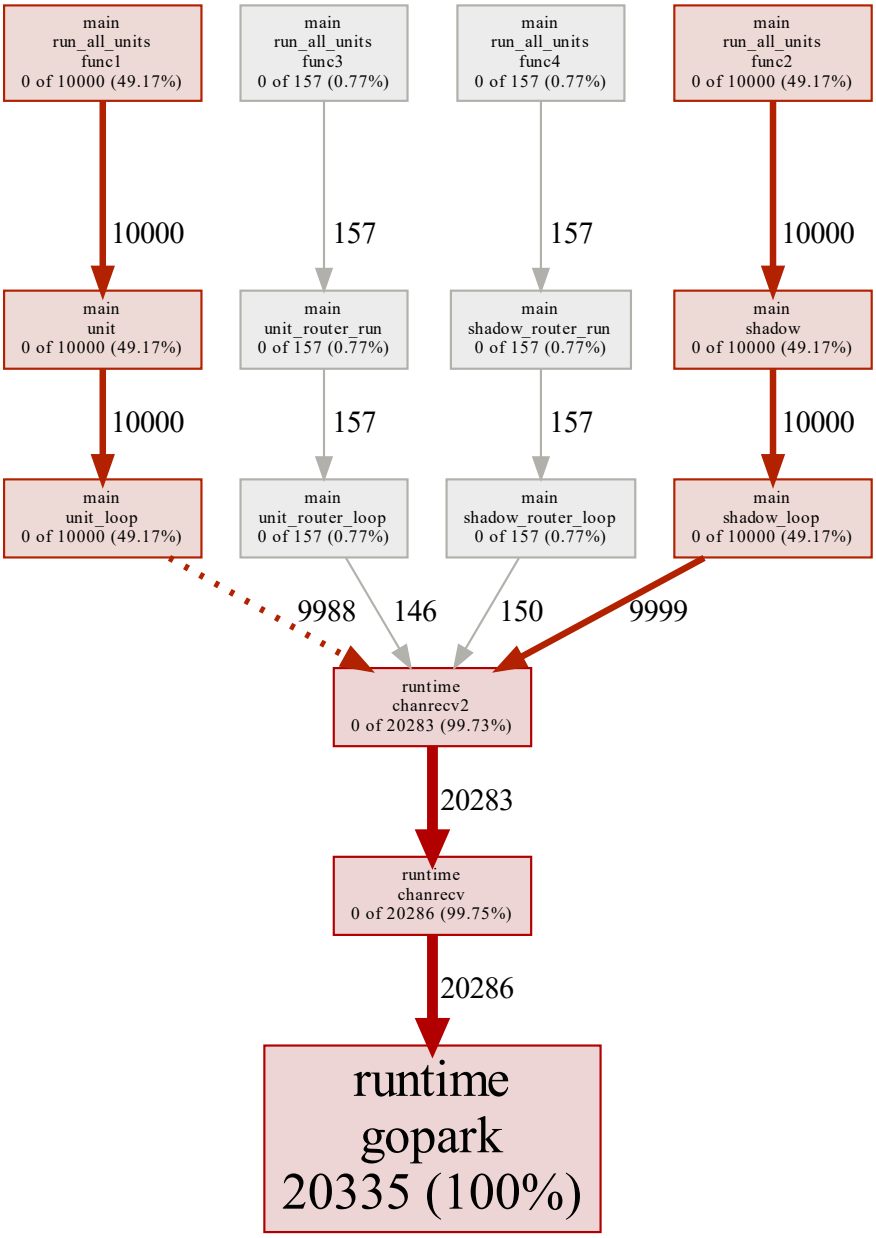


Figure 7.1: PPROF goroutine profiling graphical representation

Listing 7.6: PPROF alloc profiling *-top* representation for NEIB

```
Type: alloc_space
Time: Oct 23, 2021 at 2:27pm (CEST)
Showing nodes accounting for 16065.73MB, 99.30% of 16178.46MB total
Dropped 72 nodes (cum <= 80.89MB)
      flat  flat%   sum%        cum   cum%
10533.16MB 65.11% 65.11% 10533.66MB 65.11%  main.check_neib_channels
 5109.46MB 31.58% 96.69%  5109.46MB 31.58%  main.cut_and_transfer
  157.35MB  0.97% 97.66%  5266.81MB 32.55%  main.unit_body
  104.54MB  0.65% 98.31%   104.54MB  0.65%  main.get_message_timetrack
   78.71MB  0.49% 98.79%   183.25MB  1.13%  main.process_received_msg
   70.50MB  0.44% 99.23%    88.50MB  0.55%  main.get_shadow_disk_path
       7MB 0.043% 99.27%   281.77MB  1.74%  main.shadow
    2.50MB 0.015% 99.29%  5313.32MB 32.84%  main.unit
    1.50MB 0.0093
```

Another striking difference is in listing 7.7. There are 10000 more concurrent goroutines allocated for neighbor channels check. The synchronization via the right neighbor is only an additional task, so this memory leak would not be improved in the current version of the prototype.

Listing 7.7: PPROF goroutine profiling *-top* representation for NEIB

```
Type: goroutine
Time: Oct 23, 2021 at 2:27pm (CEST)
Showing nodes accounting for 30471, 80.01% of 38084 total
Dropped 54 nodes (cum <= 190)
Showing top 10 nodes out of 36
flat  flat%   sum%        cum   cum%
30463 92.95% 92.95%      30463 92.95%  runtime.gopark
    0     0% 92.95%      10000 30.51%  main.check_neib_channels
    0     0% 92.95%        873  2.66%  main.get_shadow_disk_path
    0     0% 92.95%       7128 21.75%  main.request_neib_sync
    0     0% 92.95%      10000 30.51%  main.run_all_units.func1
    0     0% 92.95%      10000 30.51%  main.run_all_units.func2
    0     0% 92.95%        157  0.48%  main.run_all_units.func3
    0     0% 92.95%        157  0.48%  main.run_all_units.func4
    0     0% 92.95%      10000 30.51%  main.shadow
    0     0% 92.95%        906  2.76%  main.shadow_body
    0     0% 92.95%      10000 30.51%  main.shadow_loop
    0     0% 92.95%        157  0.48%  main.shadow_router_loop
    0     0% 92.95%        157  0.48%  main.shadow_router_run
    0     0% 92.95%      10000 30.51%  main.unit
    0     0% 92.95%       7264 22.17%  main.unit_body
    0     0% 92.95%      10000 30.51%  main.unit_loop
    0     0% 92.95%      10000 30.51%  main.unit_loop.func1
    0     0% 92.95%        157  0.48%  main.unit_router_loop
    0     0% 92.95%        157  0.48%  main.unit_router_run
    0     0% 92.95%      19290 58.86%  runtime.chanrecv
    0     0% 92.95%      19197 58.58%  runtime.chanrecv2
    0     0% 92.95%       8573 26.16%  runtime.gcAssistAlloc
    0     0% 92.95%       8573 26.16%  runtime.gcMarkDone
    0     0% 92.95%      11032 33.66%  runtime.goparkunlock (inline)
    0     0% 92.95%       8573 26.16%  runtime.mallocgc
```

PPROF was actively used during the prototype development. There are some shortcomings indicated in the final versions of profiling results. The current implementation is sufficient for the stated goals. In the case of future development, some functions are subjects for improvement.

7.4 Performance metrics

There are performance metrics that are chosen for the evaluation of the simulated network and data processing. In contrast to the system testing and benchmarking, they help to describe the behavior of the emulated observational network, not only the characteristics of the application running in the OS.

The Definition 14. Fresh node

An OU which has been updated during the last 24 hours is referenced as a fresh one.

The Definition 15. Throughput

Throughput is the volume of transferred data during simulation execution time, MB/s

The Definition 16. Generated data amount

Generated Data Amount (GDA) is the total data amount generated by the simulation per second of execution, MB/s

The Definition 17. Mask cost

Mask cost coefficient is calculated as a ratio between the throughput and the OS RAM used by the simulation.

The Definition 18. Simulation efficiency

Simulation efficiency coefficient is calculated as a ratio between the Generated Data Amount and the OS RAM used by the simulation.

The Definition 19. Node resource demand

Node Resource Demand (NRD) is the estimate of how much RAM is used to emulate one instance in KB according to equation 7.1, referred to as RAM per node.

$$\mathrm{NRD} = \frac{\mathrm{RAM}}{\mathrm{OU} * 2 + 2 * \left\lceil \frac{\mathrm{OU}}{clustersize} \right\rceil}$$

(7.1)

The Definition 20. Time to sync one node

Time to sync one node is the average time required to get the latest data for a single edge node (OU → SU), calculated according to equation 7.2, where t is the total execution time, n is the total number of edge nodes, $!s$ is the share of never synchronized edge nodes, s is the number of synchronized edge nodes.

$$\text{time to sync one node} = \frac{1}{\left(\frac{n-n*!s}{t}\right)} = \frac{t}{s} \tag{7.2}$$

The Definition 21. Time to sync all nodes

Time to sync all nodes is the average time required to get the latest data from all edge nodes (OU → SU), calculated according to equation 7.3, where t is the total execution time, n is the total number of edge nodes, $!s$ is the share of never synchronized edge nodes, s is the number of synchronized edge nodes.

$$\text{time to sync all nodes} = \left(\frac{1}{\frac{n-n*!s}{t}}\right) * n = \frac{t * n}{s} \tag{7.3}$$

7.5 Experiment 1: Time Influence

Experiment one tests the possible effects of the execution time on the behavior of the simulation. As discussed in statistical experiments (chapter 5) and the design guidelines (section 6.2), 168 seconds is the standard runtime for experiments. This set of runs would show if the time scaling principle can be used or there is a need for longer runtime.

7.5.1 Setup

Variables used in the experiment are listed in the table 7.1. The number of OUs is 10000, cluster size is 256 nodes per router, the main network type is BLE. For some cases, LoRa and Wi-Fi network environments are tested as well, every run is done both for direct synchronization and NEIB. Execution time values are 60, 120, 168, 240, 480, 960, 1800, 2700 and 3600 seconds. The chance of the back-haul network availability is set to 5%, the chance of an OU being awake is 1%. The $P(A) \cap P(N)$ is 0.0005 which represent one of the worst-case scenarios. For the NEIB mode, the single synchronization chance doubles and is equal to 0.001. There are 24 simulations in total.

7.5.2 Simulation results

Results are partially reflected in the table 7.1. The original table with simulation values spans 30 more columns and is not listed here. Only the most relevant data is reflected in the table, but the general rule is that results are plotted on charts. Values reflected in the table have the following meanings:

- *trans* (means *transferred*) is the share of data transferred from OUs to SUs during the simulation.
- *synced* is the share of nodes synced at least once.
- *fresh* is the share of fresh nodes by the end of simulation.
- *!synced* is the share of never synced nodes.
- *1-sync* is *transferred* multiplied by *synced*. This value reflects the chance that the user will request a random data piece and this data would be available.

Variables							Statistical Data			Empirical Basic Data				
Nodes	Cluster	Time = n	Conn	P(A)	P(N)	NEIB	P(A) ∩ P(N)	s	PMF	Trans	Synced	Fresh	!Synced	1-sync
10000	256	60	Wi-Fi	1	5	No	0.0005	1	0.029	1.36 %	2.86 %	0.90 %	97.1 %	0.04 %
			LoRa			No				1.47 %	2.94 %	1.00 %	97.1 %	0.04 %
			BLE			No				1.40 %	2.85 %	1.00 %	97.2 %	0.04 %
						Yes	0.001		0.107	2.48 %	5.24 %	1.80 %	94.8 %	0.18 %
		120				No	0.0005		0.057	2.80 %	5.63 %	1.20 %	94.4 %	0.16 %
						Yes	0.001		0.107	4.85 %	10.01 %	1.80 %	90.0 %	0.49 %
		168				No	0.0005		0.077	4.11 %	8.24 %	1.40 %	91.8 %	0.34 %
						Yes	0.001		0.142	6.89 %	14.28 %	2.20 %	85.7 %	0.98 %
		240				No	0.0005		0.106	5.48 %	11.21 %	1.10 %	88.8 %	0.61 %
						Yes	0.001		0.189	9.70 %	20.01 %	1.70 %	80.0 %	1.94 %
		480				No	0.0005		0.189	10.65 %	21.43 %	1.20 %	78.6 %	2.28 %
						Yes	0.001		0.297	18.56 %	36.84 %	2.20 %	63.2 %	6.84 %
		960	Wi-Fi			No	0.0005			18.93 %	37.49 %	1.30 %	62.5 %	7.10 %
			LoRa			No				18.73 %	38.06 %	1.20 %	61.9 %	7.13 %
			BLE			No			0.297	18.90 %	37.01 %	1.40 %	63.0 %	7.00 %
						Yes	0.001		0.368	30.90 %	57.89 %	2.30 %	42.1 %	17.89 %
		1800	Wi-Fi			No	0.0005		0.366	32.24 %	59.94 %	1.20 %	40.6 %	19.14 %
			LoRa			No				31.85 %	60.12 %	1.04 %	39.9 %	19.16 %
			BLE			No				31.49 %	58.48 %	1.10 %	41.5 %	19.42 %
						Yes	0.001		0.298	47.11 %	80.36 %	2.40 %	19.6 %	37.86 %
		2700				No	0.0005		0.350	42.20 %	73.47 %	1.60 %	26.5 %	31.00 %
						Yes	0.001		0.181	59.09 %	90.82 %	2.20 %	9.2 %	53.66 %
		3600				No	0.0005		0.298	50.59 %	83.23 %	1.60 %	16.8 %	42.11 %
						Yes	0.001		0.098	66.92 %	96.20 %	3.60 %	3.8 %	64.37 %

Table 7.1: Experiment 1: Variables and general data transfer results

For instance, statistical expectation shows that the PMF would increase together with the runtime. It is confirmed by experiment as illustrated in figure 7.2, primary axis (left), PMF reaches its peak after 960 seconds. It means that with a given single-time probability P=0.0005 for direct synchronization and P=0.001 for NEIB, 1 successful synchronization can be expected after 960 trials with a probability around 0.3 (≈ 0.36 for the NEIB mode).

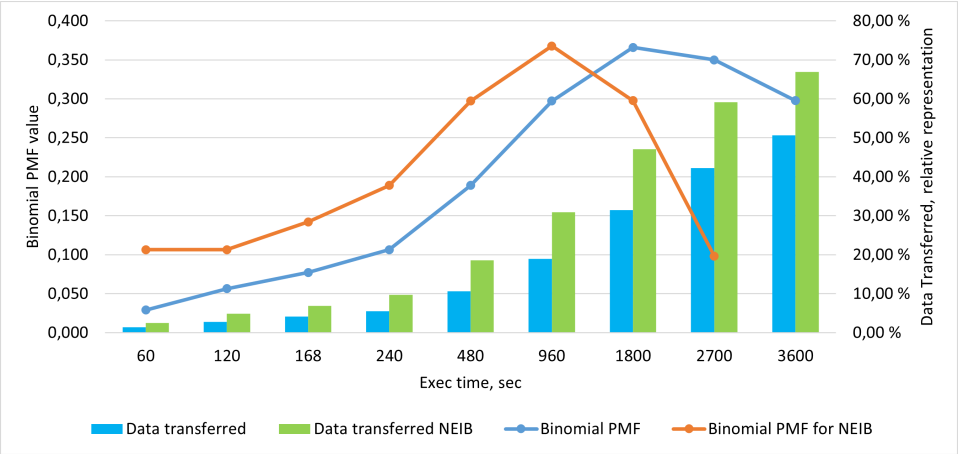


Figure 7.2: E1: Binomial PMF values

The total share of data transferred in comparison to all data generated is growing (secondary, right axis). So the execution time has an influence on the total amount of data transferred both in accordance with statistical expectation and practically. The chance that one single OU would be synchronized at least

once is growing as well, for the NEIB mode the function peak is reached earlier than after 960 trials, because of the higher chance of synchronization via the right node. For the NEIB mode, the amount of transferred data is approximately 1.5-2 times higher.

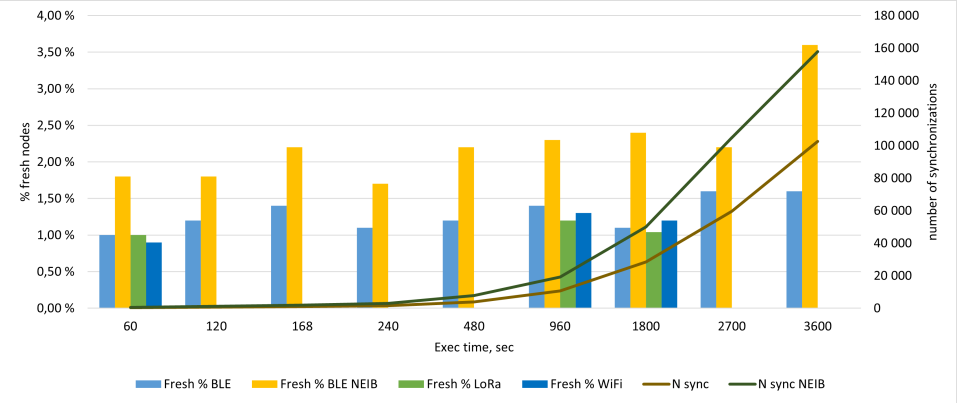


Figure 7.3: E1: Fresh nodes share and global synchronization counter

Figure 7.3 illustrates the share of fresh nodes and the global synchronization counter. The total number of synchronizations grows linearly, for the NEIB mode the number of successful synchronization is always higher. That means the at any point the number of nodes synchronized during the last 24 hours is approximately the same, overall execution time has no impact on the share of fresh nodes. Samples were taken for LoRa and Wi-Fi on 60, 960, and 1800 seconds to demonstrate similar shares of fresh nodes.

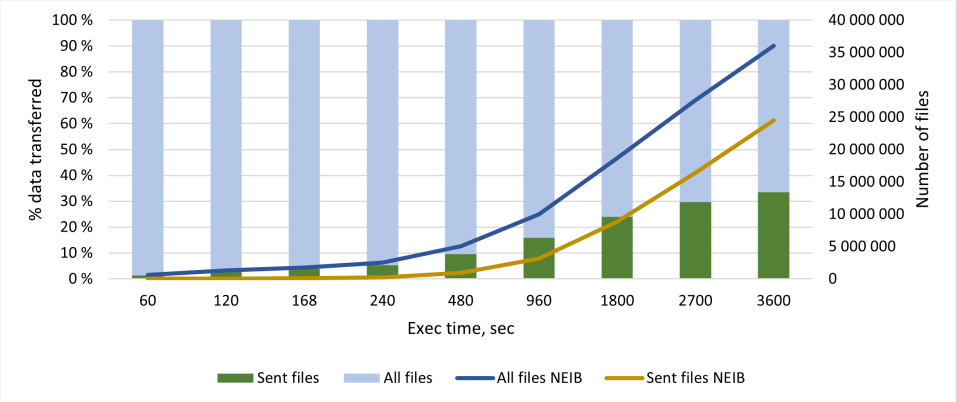


Figure 7.4: E1: The share of data generated in model transferred to shadow nodes

The share of data generated in the model transferred to shadow nodes is illustrated in figure 7.4. The relative share of sent files in the total data amount is growing as shown in the primary axis (left). For the NEIB mode, the secondary axis (right) shows that both the total number of generated files and the number

of sent files increasing over the time. That is the expected behavior, longer simulation time results in more files generated and transferred.

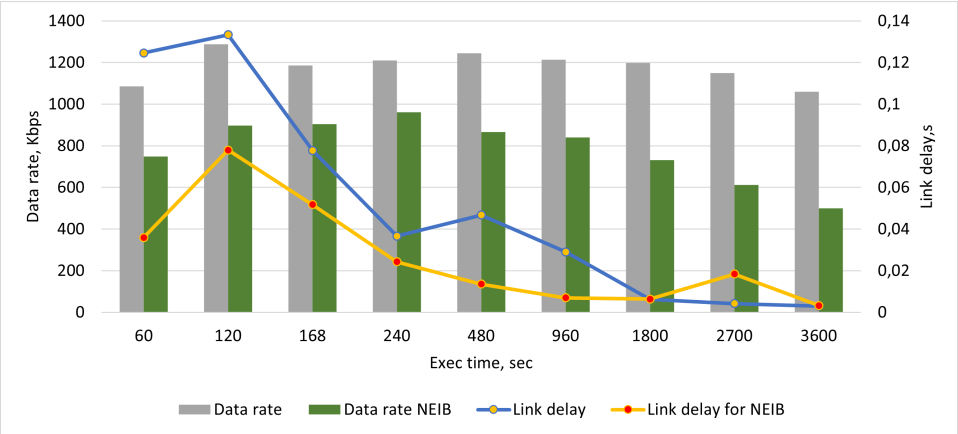


Figure 7.5: E1: Data transfer rate and link delay

Data transfer rate is illustrated in primary (left) axis of figure 7.5. Transfer rate has declining trend probably because of general performance decrease, would be discussed in the next subsection. Link delay is decreasing (secondary, right axis). The reason might be in longer transfer periods. OUs that accumulated significant media data package may use transfer channel for a longer time and, hence, have a lower average link wait time.

7.5.3 Computed performance metrics

Performance and physical characteristics are evaluated using metrics from section 7.4. Simulation's RAM demand during the runtime is in figure 7.6.

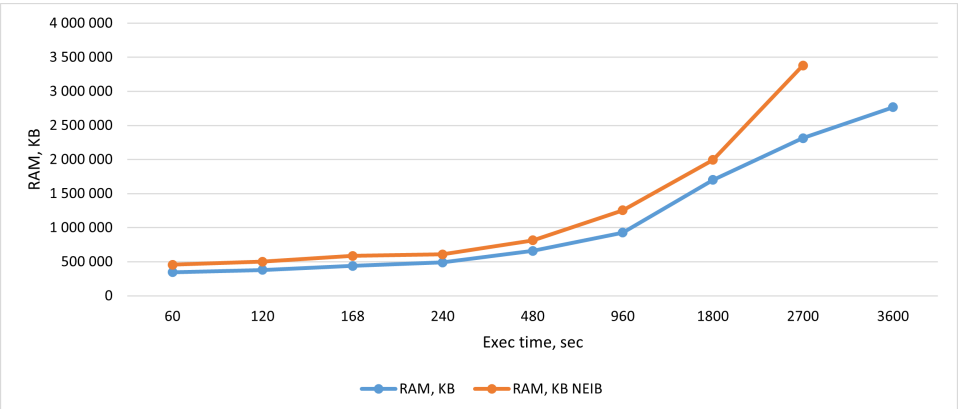


Figure 7.6: E1: RAM usage for BLE with and without NEIB

There is a trend of increasing demand for memory for longer execution periods. The NEIB mode always consumes more memory. It means that instances accumulate the data over time, and hence the simulation would become inefficient after a while. OUs and SUs are not designed to hold huge amounts of data all the time.

RAM use per node (Node Resource Demand) is illustrated in figure 7.7. Go structures that represent nodes become bigger over the runtime as observed previously in section 7.3 on profiling. It is the reason why nodes tend to consume more RAM as the execution time increases (≈ 20 KB for BLE after 120 seconds and ≈ 100 KB for BLE after 1800 seconds). Samples taken for LoRa and Wi-Fi on 60, 960, and 1800 seconds demonstrate that in the LoRa environment less RAM per node is needed. The reason might be that emulated nodes in LoRa network do not have enough time to update structures and all resources are spent on the networking. More runs are needed to come to definite conclusions.

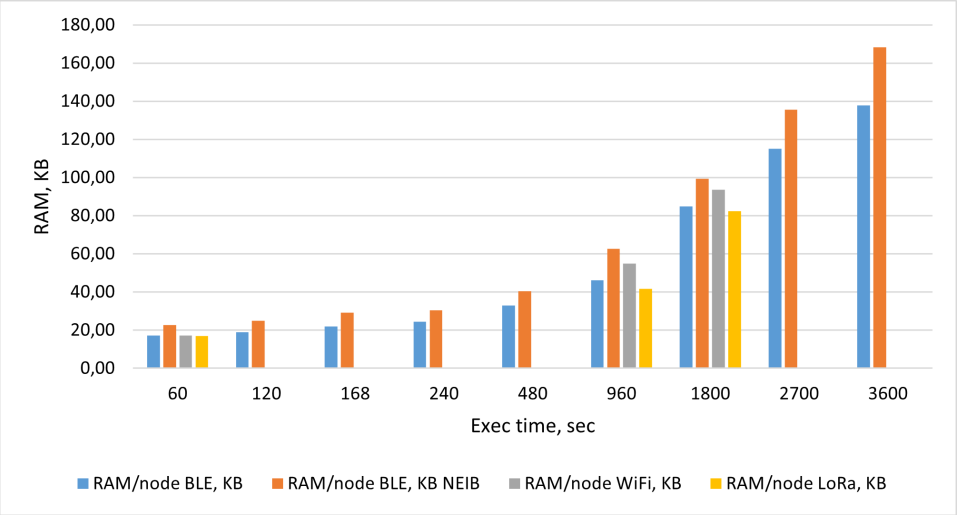


Figure 7.7: E1: RAM use per node, KB

The data amount generated in the simulation per second on average during the execution time is the Generated Data Amount (definition 16). All the data transferred inside the model during the execution is throughput (definition 15). Both metrics are plotted in figure 7.8. The GDA is reflected on the primary axis (left), throughput on the secondary axis to the right.

The decreasing GDA for both modes means that less data generated send per second if the simulation runs for a longer period. For the NEIB mode there is a 10% GDA difference between simulations for 60 seconds and for 3600 seconds. That can be explained by the increased chance of synchronization and

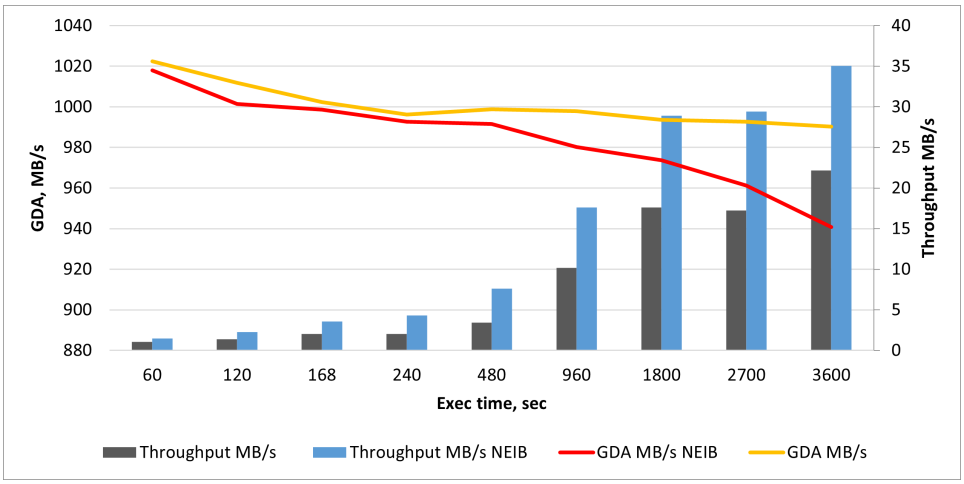


Figure 7.8: E1: Throughput and Generated Data Amount for BLE with NEIB

probably longer queues because the runtime can alternatively be considered as the "number of trials". The throughput goes up, especially for the NEIB mode, if the simulation runs for a longer period. That can be explained by the increased chance of photos and video. The next step is to plot the throughput and the GDA with RAM dependency - see figure 7.9

As illustrated in figure 7.9, the overall simulation efficiency (definition 18) decreases, especially for the NEIB mode. It means that the RAM usage increases much faster than the model produces new data - reflected on the primary axis (left). The mask cost (definition 17) increases until 1800 seconds of simulation, which corresponds 1800 hours or 75 days of simulation (section 6.2.2). That is the peak point of ≈ 3.2 mask cost value. After that point, the coefficient's value goes down. This trend is reflected on the secondary axis (right) and means that after 1800 seconds the simulation uses a significant amount of RAM but the amounts of data produced and sent remain on their levels.

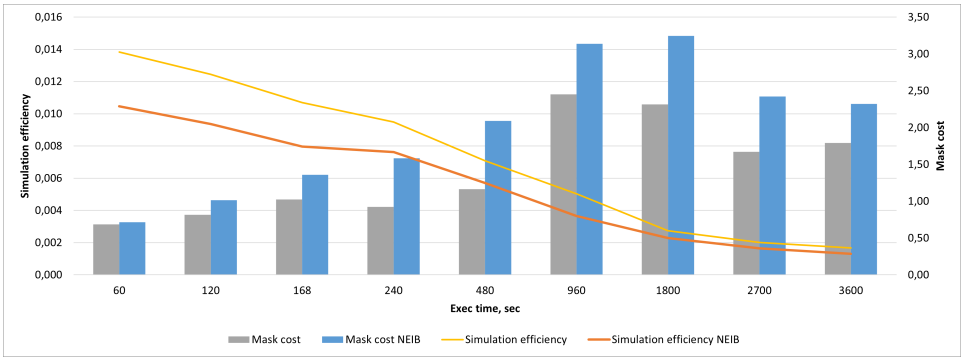


Figure 7.9: E1: Mask cost and simulation efficiency for BLE with NEIB

The next step is to identify the time expected to get the latest data. Time-related results of the simulation are in the table 7.2. Values reflected in the table have the following meanings:

- *Synced nodes* is the number of nodes synchronized by the end of simulation, calculated as: $(1-!sync) * n$, where *!sync* is the share of never synchronized nodes, n is the total number of OUs emulated.
- *Synced nodes per sec* is the number of nodes with the latest data appearing in the shadow network every second.
- *One node sync, sec* is the approximate average time required to get the latest data from a single random node, calculated using equation 7.2.
- *All nodes sync, sec* is the approximate average time required to get the latest data from all edge nodes, calculated using equation 7.3.
- *NEIB benefit* is the ratio between *all nodes sync* time with and without NEIB. This value reflects the speedup of time required to get the latest data if the NEIB mode is activated.

Exec Time	NEIB	Network	Synced nodes	Synced nodes per sec	One node sync, sec	All nodes sync, sec	NEIB benefit
60	No	Wi-Fi	290	4.83	0.21	2068.97	1.857
	No	LoRa	290	4.83	0.21	2068.97	
	No	BLE	280	4.67	0.21	2142.86	
	Yes		520	8.67	0.12	1153.85	
120	No		560	4.67	0.21	2142.86	1.785
	Yes		1000	8.33	0.12	1200.00	
168	No		820	4.88	0.20	2048.78	1.743
	Yes		1430	8.51	0.12	1174.83	
240	No		1120	4.67	0.21	2142.86	1.785
	Yes		2000	8.33	0.12	1200.00	
480	No		2140	4.46	0.22	2242.99	1.719
	Yes		3680	7.67	0.13	1304.35	
960	No	Wi-Fi	3750	3.91	0.26	2560.00	1.564
	No	LoRa	3810	3.97	0.25	2519.69	
	No	BLE	3700	3.85	0.26	2594.59	
	Yes		5790	6.03	0.17	1658.03	
1800	No	Wi-Fi	5940	3.30	0.30	3030.30	1.011
	No	LoRa	6010	3.34	0.30	2995.01	
	No	BLE	5850	3.25	0.31	3076.92	1.374
	Yes		8040	4.47	0.22	2238.81	
2700	No		7350	2.72	0.37	3673.47	1.235
	Yes		9080	3.36	0.30	2973.57	
3600	No		8320	2.31	0.43	4326.92	1.156
	Yes		9620	2.67	0.37	3742.20	

Table 7.2: Experiment 1: Time to get the latest data

Results from table 7.2 are illustrated in figure 7.10. Seconds are scaled to hours. It means that if time to get the latest data from a single node is expressed in 0.21 seconds, it corresponds 0.21 hours ($\approx$ 12.5 minutes). Lines represent time to get the latest data from all edge nodes with and without NEIB on the primary axis (left). Bars represent time to get the latest data for a single edge node with and without NEIB on the secondary axis (right).

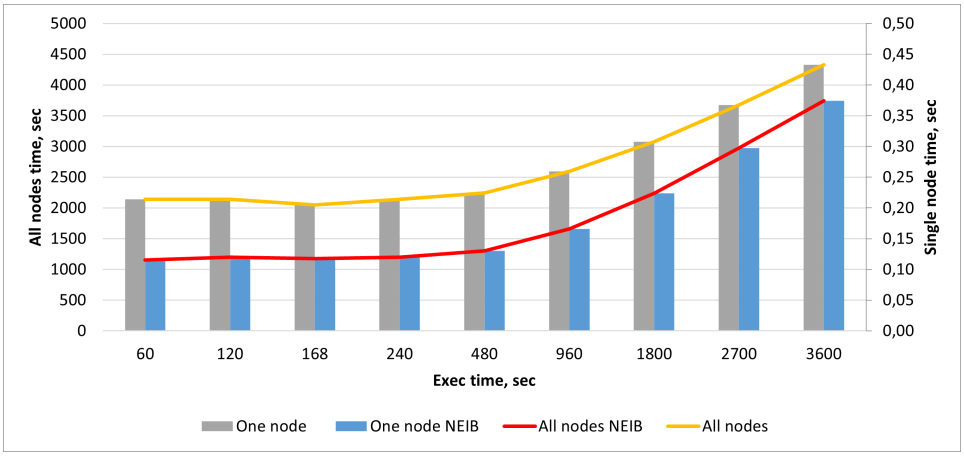


Figure 7.10: E1: Time to get the latest data from one node and from all nodes

Firstly, with NEIB, one node sync time reduces 1.52 times on average. Secondly, there is more time required to get the latest data from one node and all nodes if the simulation runs for a longer period. It means that statistically expected number of nodes that would deliver the latest data is not precise. If the simulation is running for some time and some number of nodes were synchronized, it is assumed that doubling of runtime would double the number of synchronized nodes. In practice, single synchronization chance is not dependent on previous successes. It means that there can be a node that wouldn't synchronize even after several months of simulation at such a low single synchronization success probability $0.0005 \leq P(A) \cap P(N) \leq 0.001$. As a result, the approximation of time increases both for one node and for all nodes.

For the simulation of one week, the expected time to get the latest data from one random node is $\approx$ 5 hours. If the NEIB mode is used, the time reduces to $\approx$ 3 hours. All nodes are expected to synchronize at least once after $\approx$ 90 days but with NEIB the expected time is $\approx$ 50 days.

7.5.4 Interpretation of results

Experiment 1 confirmed that statistically calculated values can be used for further experiments. The execution runtime doesn't affect the simulation in terms of fresh nodes and single success probabilities. The overall performance decrease is noted, adjustment is needed in case of further experiments on longer execution periods. The use of NEIB shortens the time required to get data from OU to SU by 1.52 times on average. Not all the charts are described in details, because numbers are dependent on the $P(A) \cap P(N)$ probability and other variables. It is more important to identify general trends.

7.6 Experiment 2: Scalability

Experiment 2 tests simulation's ability to scale within the range 10 - 1,000,000 OUs for three types of network: Wi-Fi, BLE and LoRa. In accordance with research tasks (section 1.3.2), the prototype supports a network of several thousand OUs. The ability of the simulation to operate normally despite various sizes is the test subject. There are 24 runs, 8 for each network type.

7.6.1 Setup

The experimental setup is the same for three different network types. Number of nodes is 10; 100; 1,000; 10,000; 100,000; 250,000; 500,000; and 1,000,000. Cluster size is 64 units per router. Execution time is 168 seconds, the chance of OU being awake is 50%, the chance of the back-haul network availability is 5%. Variables can be found in table 7.3 for Wi-Fi, table 7.4 for BLE and table 7.5 for LoRa.

Variables						Statistical Data		Empirical Basic Data					
Nodes	Cluster	Time = n	Conn	P(A)	P(N)	P(A) ∩ P(N)	PMF	Trans	Synced	Fresh	!Synced	1-sync	RAM, KB
10	64	168	Wi-Fi	50	5	0.025	0.0612	63.28 %	90.0 %	30.0 %	10.0 %	56.95	21889
100								75.89 %	100.0 %	43.0 %	0.0 %	75.89	53066
1,000								74.25 %	98.1 %	44.6 %	1.9 %	72.84	112246
10,000								73.46 %	98.6 %	43.4 %	1.4 %	72.45	613297
100,000								73.6 %	98.7 %	43.4 %	1.3 %	72.6 %	5735866
250,000								73.4 %	98.6 %	42.9 %	1.4 %	72.3 %	14027043
500,000								56.6 %	88.0 %	25.8 %	12.0 %	49.8 %	20580838
1,000,000								37.6 %	64.3 %	16.0 %	35.7 %	24.2 %	33363774

Table 7.3: Experiment 2: Variables and data for Wi-Fi

Variables						Statistical Data		Empirical Basic Data					
Nodes	Cluster	Time = n	Conn	P(A)	P(N)	P(A) ∩ P(N)	PMF	Trans	Synced	Fresh	!Synced	1-sync	RAM, KB
10	64	168	BLE	50	5	0.025	0.0612	69.8 %	100.0 %	27.0 %	0.0 %	69.8 %	15235
100								74.5 %	90.0 %	32.0 %	1.0 %	73.7 %	42800
1,000								73.6 %	98.3 %	30.0 %	1.7 %	72.3 %	118944
10,000								73.8 %	98.7 %	29.0 %	1.4 %	72.8 %	593536
100,000								73.5 %	98.6 %	29.0 %	1.4 %	72.5 %	5226683
250,000								73.6 %	98.6 %	29.0 %	1.4 %	72.5 %	13215734
500,000								61.5 %	92.1 %	20.0 %	7.9 %	56.6 %	20732040
1,000,000								42.3 %	71.0 %	13.0 %	29.0 %	30.0 %	33535043

Table 7.4: Experiment 2: Variables and data for BLE

Variables						Statistical Data		Empirical Basic Data					
Nodes	Cluster	Time = n	Conn	P(A)	P(N)	P(A) ∩ P(N)	PMF	Trans	Synced	Fresh	!Synced	1-sync	RAM, KB
10	64	168	LoRa	50	5	0.025	0.0612	69.6 %	90.0 %	17.0 %	10.0 %	62.6 %	23595
100								73.8 %	98.0 %	10.0 %	2.0 %	72.3 %	36448
1,000								72.2 %	98.1 %	15.0 %	1.9 %	70.9 %	94647
10,000								73.6 %	98.5 %	8.0 %	1.5 %	72.5 %	511571
100,000								76.0 %	98.6 %	7.8 %	1.4 %	73.5 %	5239528
250,000								73.9 %	98.6 %	8.0 %	1.4 %	72.8 %	11483491
500,000								62.6 %	92.9 %	9.0 %	7.1 %	58.2 %	19452390
1,000,000								42.3 %	71.0 %	8.0 %	29.0 %	30.0 %	31614107

Table 7.5: Experiment 2: Variables and data for LoRa

7.6.2 Simulation results

Some data collected during the simulation is reflected in tables 7.3 for Wi-Fi, 7.4 for BLE, and 7.5 for LoRa. The original table with simulation values spans 30 more columns and is not listed here. Only the most relevant data is reflected in the table, but the general rule is that results are plotted on charts. Data fields in the table have the same meanings as in experiment 1 (section 7.5).

Statistical expectation shows that PMF is 0.0612 which is a value on a declining PMF curve. A case with $P(A) \cap P(N) = 0.025$, at least one success and 168 trials was discussed in chapter 5 and is plotted in figure 5.1. Such low PMF values demonstrate that the function peak was much closer to the beginning of the x (horizontal) axis. According to figure 5.1, ≈ 3 successful synchronizations can be expected with a probability of 20% after 168 trials.

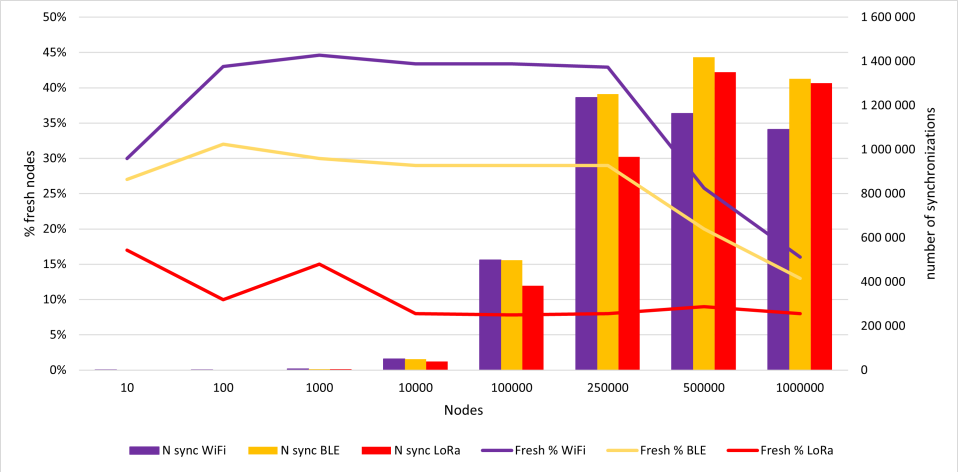


Figure 7.11: E2: Fresh nodes share and global synchronization counter

In figure 7.11 the number of synchronizations is illustrated in bars on the secondary axis (right). Before 250,000 nodes, the number of synchronizations is growing linearly, while after 250,000 OUs, the total number of synchronizations remains almost the same disregarding the network type. That indicates scaling difficulties. The number of fresh nodes is illustrated in the same figure on the primary axis (left) and shows the neutral trend until the point of 250,000 nodes. After that point, the share of nodes synchronized during the last 24 hours before the simulation's termination demonstrates moderate growth continued by the recession. For the LoRa environment, there are 10% – 15% fresh nodes, for BLE $\approx$ 30%, for Wi-Fi $\approx$ 45%. It means that network types that require more packets for the data transfer may cause data delivery delays. The conclusion here is that the network size is growing, but the total number of file transfers goes down after the point of 500,000 nodes.

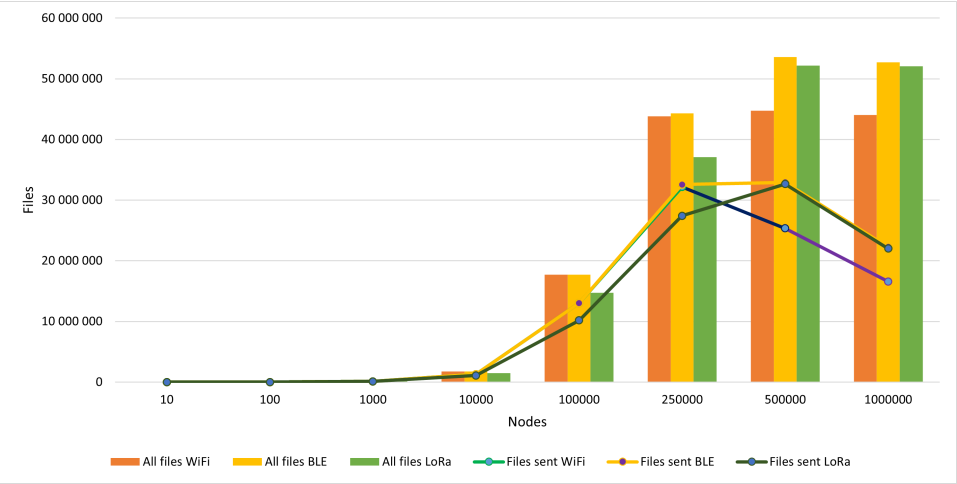


Figure 7.12: E2: All files generated and files sent

Figure 7.12 confirms the trend identified in figure 7.11. Bars illustrate the total number of files generated inside the simulation during the runtime. The network of OUs is not influenced by scaling until 250,000 nodes, because the number of generated files increases proportionally. After 500,000 the scaling doesn't bring benefits because for 1,000,000 OUs the behavior is almost the same. It is confirmed by the line representing the number of sent files. OU doubling should double the number of sent files, but here, this number is going down indicating scaling issues. In all three environments, the behavior is similar, but the number of sent files in LoRa is always lower. It confirms the conclusion from the previous experiment - the simulation of LoRa network requires more packets to send data, and is able to process fewer files.

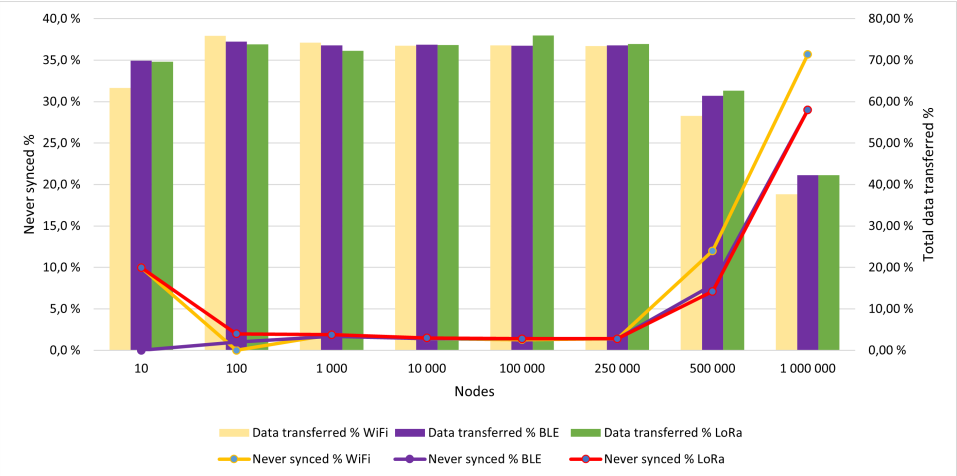
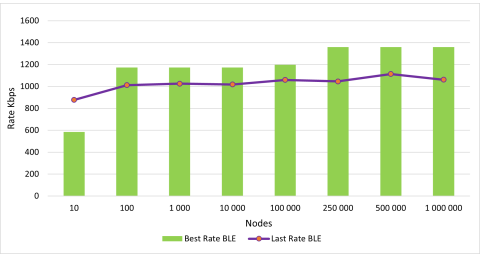


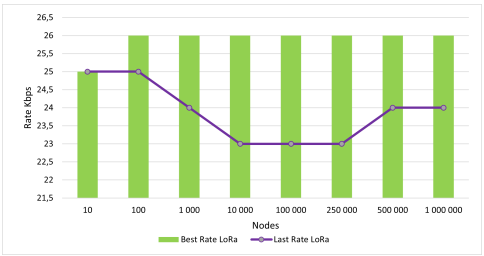
Figure 7.13: E2: The share of never synced nodes and share of total data transferred

The share of the data transferred from OUs to SUs is plotted in the secondary (right) axis of figure 7.13. Relatively high synchronization success probability results in more than 70% of data generated in the model is transferred to the destination. After scaling to 500,000 OUs, all three networks types demonstrate the decrease in the transferred data share. At the same time, the share of nodes that have never been synchronized remains relatively flat for 100 to 250,000 nodes (plotted in the primary axis of figure 7.13). After 250,000 the share of never synchronized nodes increases rapidly. In the network of 1,000,000 nodes, ≈ 36% of nodes didn't manage to synchronize at least once for the Wi-Fi environment and ≈ 29% for the BLE and LoRa.

(a) E2: Data transmission rate for Wi-Fi



(b) E2: Data transmission rate for BLE



(c) E2: Data transmission rate for LoRa

Figure 7.14: E2: Data transmission rate for Wi-Fi, BLE and LoRa

Figure 7.14 provides a closer look at network-specific technical characteristics. The data transmission rate for Wi-Fi is illustrated in figure 7.14a. Both the last synchronization's rate (≈ 5000 Kbps - blue line) and the best rate (≈ 20000 Kbps - green bars) follow neutral trends. They remain within the range defined for the Wi-Fi environment.

BLE in figure 7.14b and LoRa in figure 7.14c do not show any significant deviation and remain within defined limits. The LoRa environment demonstrated 23-25 Kbps last data transfer rate, 27 Kbps best transfer rate. For BLE the last rate is ≈ 1000 Kbps, while the best data transfer rate is ≈ 1200 Kbps. Rates are not affected by the number of OUs emulated.

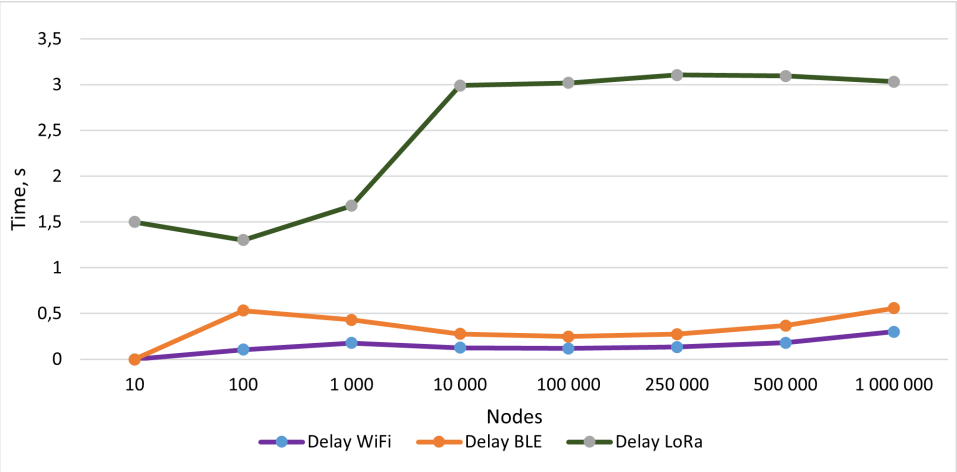


Figure 7.15: E2: Link delay for Wi-Fi, BLE and LoRa

The next figure (7.15) illustrates the average link delay registered for Wi-Fi, BLE, and LoRa on different network sizes. The delay remains on the same level Wi-Fi for different test cases. For 1,000,000 OUs simulation, the delay starts to increase. The same applies to the BLE environment, but the delay level is 10%-30% higher. It means that Wi-Fi routers can handle requests faster, so the time between the first packet and the send one is shorter. LoRa shows a significant link delay up to 3 seconds after the size of 10,000 nodes. LoRa routers use more time to handle incoming requests and the data transfer lasts longer due to the physical limitations of the emulated protocol.

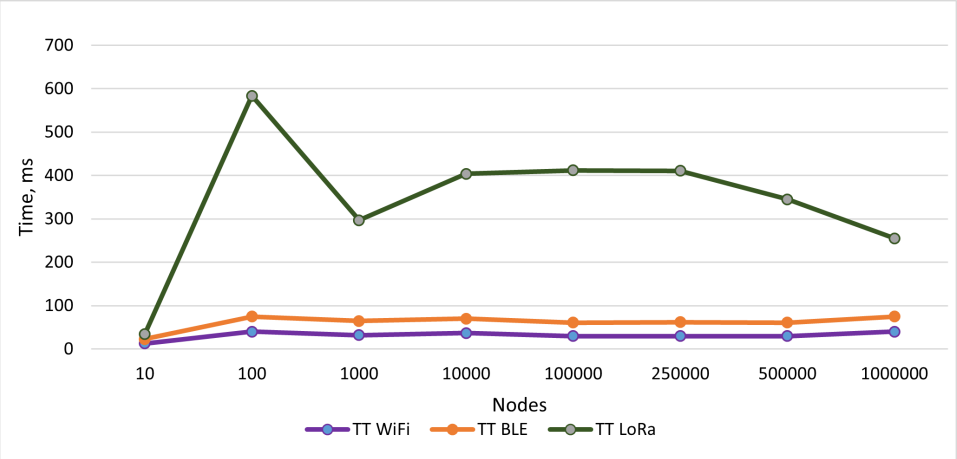
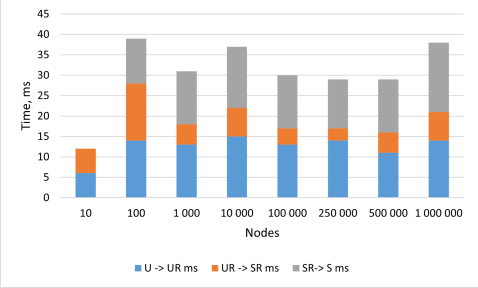


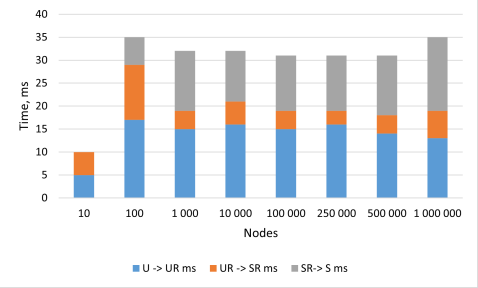
Figure 7.16: E2: Packet Travel Time for Wi-Fi, BLE and LoRa

The Packet Travel Time for all three network environments is demonstrated in figure 7.16, while figure 7.17 shows detailed PTT for Wi-Fi, BLE, and LoRa.

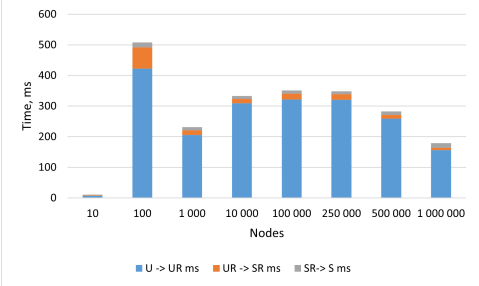
There are no changes in PTT for is Wi-Fi and BLE on different network scales, but LoRa has generally longer PTT, probably because of long delays and the artificial bottleneck. It is interesting that the PTT decreases in LoRa environment in networks of more than 250,000 OUs.



(a) E2: Packet Travel Time for Wi-Fi



(b) E2: Packet Travel Time for BLE



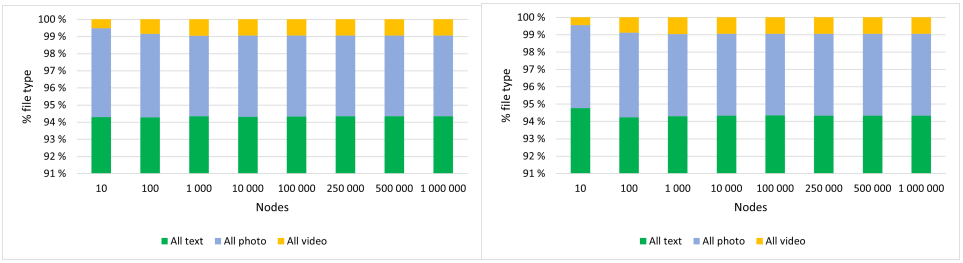
(c) E2: Packet Travel Time for LoRa

Figure 7.17: E2: PTT structure for Wi-Fi, BLE and LoRa

Figure 7.17a contains detailed representation of the PTT trend in Wi-Fi network. There are three segments which every message have to go through:

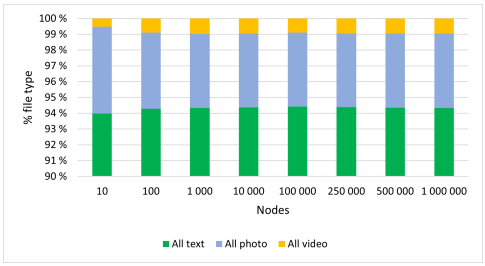
- From the Observation Unit to the Unit Router, marked as U→UR;
- From the Unit Router to the Shadow Router, marked as UR→SR;
- From the Shadow Router to the Shadow Unit, marked as SR→S;

For Wi-Fi and BLE, PTT is mostly spent on endpoints: OUs and SUs as illustrated in figures 7.17a and 7.17b respectively. PTT values for BLE are ≈ 20% higher than for Wi-Fi. For instance, for 100,000 emulated nodes, the PTT between the OU and the UR is ≈ 12 ms for Wi-Fi and ≈ 15 ms for BLE. Figure 7.17c with LoRa data samples confirms that the reason for longer PTT is connected with link delay and communication with the first intermediate destination - the UR. It can be explained through emulated limitations of the LoRa environment (smaller MTU size). The emulation of the LoRa routers requires adjustment.



(a) E2: File types for Wi-Fi

(b) E2: File types for BLE



(c) E2: File types for LoRa

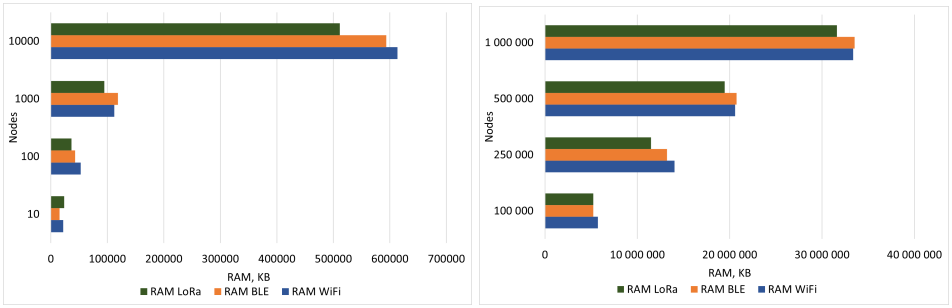
Figure 7.18: E2: File types for Wi-Fi, BLE and LoRa

The next figure (7.18) shows the distribution of file types in three different network environments. Figure 7.18a shows data samples for Wi-Fi, figure 7.18b for BLE and figure 7.18c for LoRa. In all three environments, data types correspond to chosen file probabilities. 94% of data are text messages from temperature sensors, 5% of data are images and approximately 1% of data are video files. The network size and type don't affect this behavior.

7.6.3 Computed performance metrics

The technical performance evaluation starts with RAM monitoring. Figure 7.19 combines two datasets: RAM used by simulations from 10 to 10,000 OUs (figure 7.19b) and ram used by simulations from 100,000 to 1,000,000 OUs (figure 7.19a) The RAM consumption increases linearly as network size increases in all environments. For all three network types, the RAM required to emulate a particular number of nodes is approximately on the same level. RAM amount used for LoRa emulation is always slightly lower. The reason might be that in slow network environments, less data is transferred and, hence, fewer communication primitives are used.

Figure 7.20 shows how much RAM is needed to emulate one node in various network sizes and different protocols. The calculation was performed using equation 7.1.

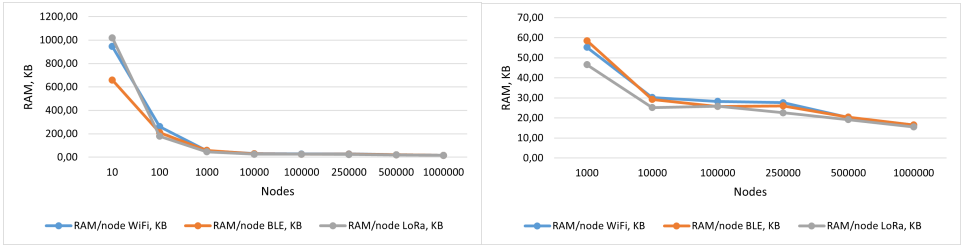


(a) E2: RAM usage for nodes 10 - 10000 for Wi-Fi, BLE and LoRa

(b) E2: RAM usage for nodes 100,000 - 1,000,000 for Wi-Fi, BLE and LoRa

Figure 7.19: E2: RAM usage for Wi-Fi, BLE and LoRa

As in previous figure, 7.20 combines two datasets: NRD in simulations from 10 to 10,000 OUs (figure 7.20a) and NRD in simulations from 100,000 to 1,000,000 OUs (figure 7.20b). It is "cheaper" in terms of RAM per node to emulate bigger networks. For BLE, less than 20 KB RAM needed to emulate one node in the network of 1,000,000 OUs. If there are 1,000 nodes, the NRD is ≈ 60 KB. It means that some part of memory is always used on the emulation of basic OS-related functions, statistics, etc. In bigger networks, most of the memory is spent on OUs. The efficiency of such big simulations is the next question.



(a) E2: RAM for nodes 10 - 1,000,000 (KB)

(b) E2: RAM for nodes 1000 - 1,000,000 (KB)

Figure 7.20: E2: RAM use per node for 10 - 1,000,000 nodes for Wi-Fi, BLE and LoRa (KB)

In figure 7.21, the simulation's GDA is shown for different network environments and different numbers of nodes. The Generated Data Amount is plotted using bars on the primary axis (left). As in figure 7.12 that shows the number of generated files, the amount of data generated per second doesn't increase after the point of 500,000 nodes. The growth happens before the point of 250,000 nodes. There is no clear trend in GDA values if the three environments would be compared, but for all three network types scling issues occur after the point of 250,000 nodes.

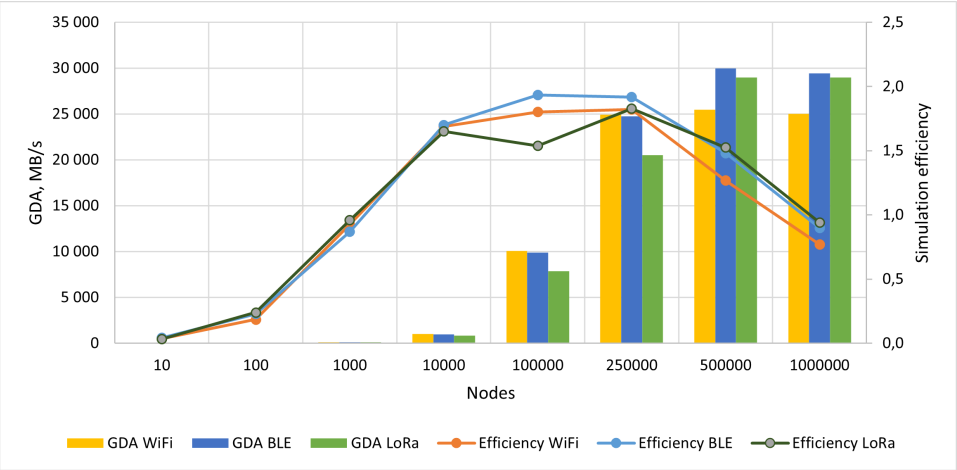


Figure 7.21: E2: Generated Data Amount and simulation efficiency for Wi-Fi, BLE and LoRa

The efficiency, ratio between RAM and GDA is plotted in the secondary axis, represented by lines. The value goes down after the network size of 250,000 OUs. That confirms scaling issues. RAM consumption goes linearly up dependent on the number of OUs, while the GDA is the same or reducing after 500,000 nodes. Efficiency calculation demonstrates especially high RAM consumption per GDA for LoRa. There is a deviation noted at the point of 100,000 edge nodes. The efficiency coefficient is RAM-bound, so there might be a slightly higher RAM requirement at that particular point. Several runs with the same input could show the standard deviation and minimize such anomalies.

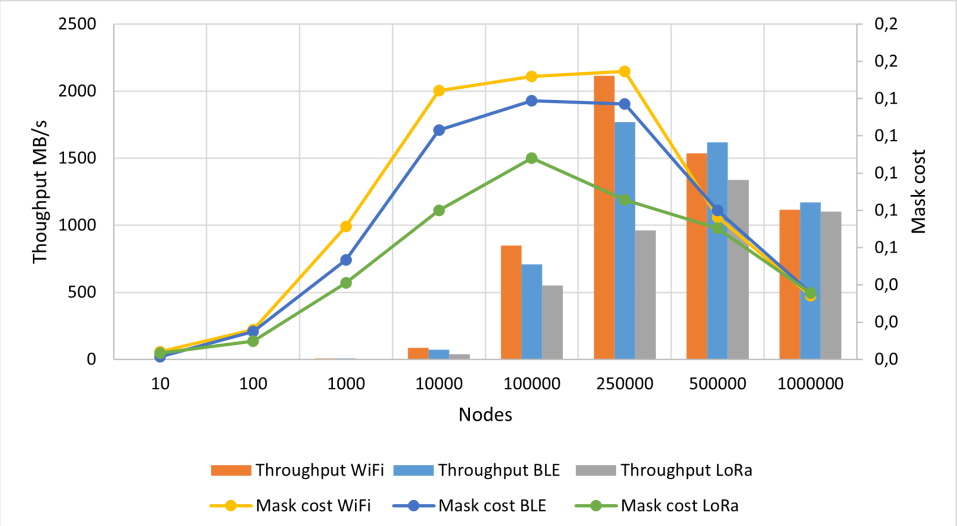


Figure 7.22: E2: Network throughput and the mask cost for Wi-Fi, BLE and LoRa

Figure 7.22 provides the information about network throughput (definition 15) illustrated by bars on the primary axis (left). The throughput is always lower for LoRa and decreases after its peak of ≈ 1300 MB/s with 500,000 OUs. For BLE and Wi-Fi, the decrease starts already after 250,000 nodes. Wi-Fi demonstrates generally higher network throughput, as expected due to higher network data transmission rate. The maximum throughput of ≈ 2100 MB/s is reached in the Wi-Fi environment in the network of 250,000 emulated OUs. When it comes to the mask cost, the ratio between RAM and throughput, it goes down after the size of 250,000 nodes (plotted in the secondary axis and represented by dotted lines). For LoRa the breaking point is 100,000 nodes. The mask cost coefficient is low for simulations with a few nodes, the lowest value is 0.0042 (Wi-Fi network with 10 edge nodes). The maximum throughput observed in the Wi-Fi environment corresponds to the maximum mask cost coefficient equal to 0.1545 meaning that the throughput is the most efficient with respect to the used RAM.

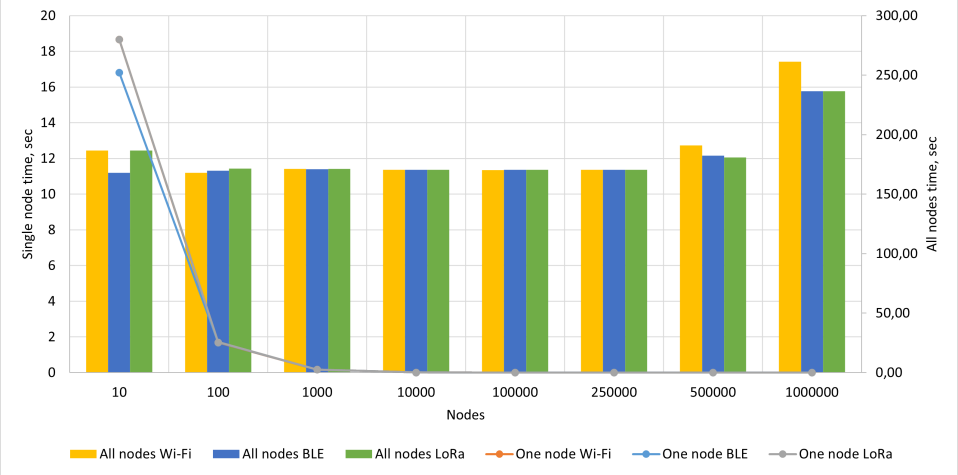


Figure 7.23: E2: Time to get the latest data from one node and from all nodes

Figure 7.23 shows the time estimated to get the latest data from a single edge node and all edge nodes calculated according to equations 7.2 and 7.3. Results for the average time to get the latest data from a single edge node are represented by lines plotted along the primary axis (left). Bars represent the average time to get the latest data from all edge nodes, plotted along the secondary axis (right). With respect to the scaled model, the time to get the latest data from all edge nodes is ≈ 180 hours. In other words, the entire observation network would synchronize the data after approximately one week with given $P(A) \cap P(N) = 0.025$ probability. The time expected to synchronize a random OU is dependent on the number of edge nodes (≈ 2.5 seconds for 1,000,000 edge nodes; ≈ 2 hours for 100 OUs). For networks with 500,000+ OUs more time to get the latest data from all edge nodes is needed.

The decrease in time to get the latest data from a single edge node doesn't assume a correlation with the number of edge nodes. Having more OUs means more trials at a given single synchronization success chance. That's why the number of updates from nodes per second is increasing as the observation network size is growing. But the time expected to get updates from the entire observation network is approximately at the same level until the number of edge nodes is 1,000,000. The 1.5 times increase of the expected time at this point either confirms the scaling issues revealed before. It also highlights that more edge nodes would require many more trials to get the entire network completely synchronized. The average time expected to get the latest data from the entire network at the given single synchronization chance (0.025) is $\approx$ 180 hours, which corresponds to 7.5 days.

7.6.4 Interpretation of results

The experiment showed that the simulation scales from 10 to 1,000,000 Observation Units. As a result, simulation supports up to 2031250 instances emulated by concurrently running goroutines (1,000,000 OUs + 15,625 URs + 15,625 SRs + 1,000,000 SUs). The emulation of the observation network size over 250,000 nodes is possible, but the behavior is not as it was expected. RAM consumption goes significantly up, while the data amount produced and transferred is the same as it was for smaller network sizes or even lower. That is indicated by lower efficiency coefficients. Scaling problems occur already after the point of 100,000 OUs in the LoRa environment. Wi-Fi demonstrated generally higher throughput, GDA values, and lower mask cost coefficient value. It means that there were more data, but if RAM usage is taken into consideration, the data generation and transfer was not so "efficient". The average time expected to get the latest data from all edge nodes is around one week at the given single synchronization chance $P(A) \cap P(N) = 0.025$.

For the next experiment, the observation network size of 100,000 OUs is chosen as a safe point, where scaling issues have no impact on the emulated network behavior.

7.7 Experiment 3: Cluster Size

This experiment tests if the size of the cluster can influence the simulation's behavior. As a result, an optimal cluster size for further experiments would be chosen. There are 7 simulations in total.

7.7.1 Setup

The experimental setup has the same parameters as in the previous experiment: execution time is 168 seconds, the chance of node being awake is 50%, the chance of the back-haul network being available is 5%. The optimal number of OUs is found in the previous experiment (section 7.6) - 100,000. The network environment is BLE. Cluster sizes to be tested are 16, 32, 64, 128, 256, 512 and 1024 units per router. Variables and the statistical expectation are in table 7.6 below.

Variables						Statistical			
Nodes	Cluster	Exec Time	Connection	Awake %	Network %	P(A)∩$P(N)$	n	X	Binom PMF
100,000	16 32 64 128 256 512 1024	168	BLE	50	5	0,025	168	1	0,061237875

Table 7.6: Experiment 3: Variables

7.7.2 Simulation results

In contrast with previous experiments, an extended range of data extracted from simulations is provided in tables. Table 7.7 contains node statistics, information about the number of messages and their types. Most of the fields in the table have the same meanings as in experiment 1 (section 7.5). Additionally, "*N sync*" is the number of synchronizations; "*Packets sent*" is the number of sent packets; "*Files*→" is the total number of sent files followed by columns for file types: "*Text*", "*Photo*", and "*Video*".

Variables	Empirical Basic Data						Network Data					
Cluster	Trans	Synced	Fresh	!Synced	1-sync	RAM, KB	N sync	Packets sent	Files→	Text	Photo	Video
16	73.5 %	98.6 %	43.6 %	1.4 %	72.4 %	5484027	501404	494551131	13044840	12937540	99719	7581
32	73.5 %	98.6 %	43.4 %	1.4 %	72.5 %	5309598	501219	496367998	13041196	12933513	100072	7611
64	73.5 %	98.6 %	43.3 %	1.4 %	72.5 %	5226683	498652	494333893	13022228	12915367	99239	7622
128	73.5 %	98.5 %	43.0 %	1.5 %	72.4 %	5123998	501637	497955015	13030761	12921787	101443	7531
256	73.5 %	98.6 %	42.5 %	1.4 %	72.5 %	5141210	499721	500647114	13012483	12904304	100452	7727
512	73.6 %	98.6 %	42.5 %	1.4 %	72.6 %	4809645	497796	485774991	13004848	12898307	99224	7317
1024	73.6 %	98.6 %	42.3 %	1.4 %	72.6 %	4666470	498521	496495696	12958752	12850527	100668	7557

Table 7.7: Experiment 3: Data

Graphical representation of the total number of files and packets generated for cluster sizes 16 to 1024 from table 7.7 can be found in figure 7.24. The total number of transferred packets is represented by blue bars and has similar values between ≈ 485,000,000 and ≈ 500,000,000 packets for all cluster sizes. The minor difference can be explained through the deviation in value distribution due to data emergence dependency on media data chances and node's awakeness probability. The total number of files generated is approximately 1.4 times bigger than the number of files sent (≈ 13,000,000 files sent; ≈ 18,000,000 files generated). The same trend is identified for all cluster sizes.

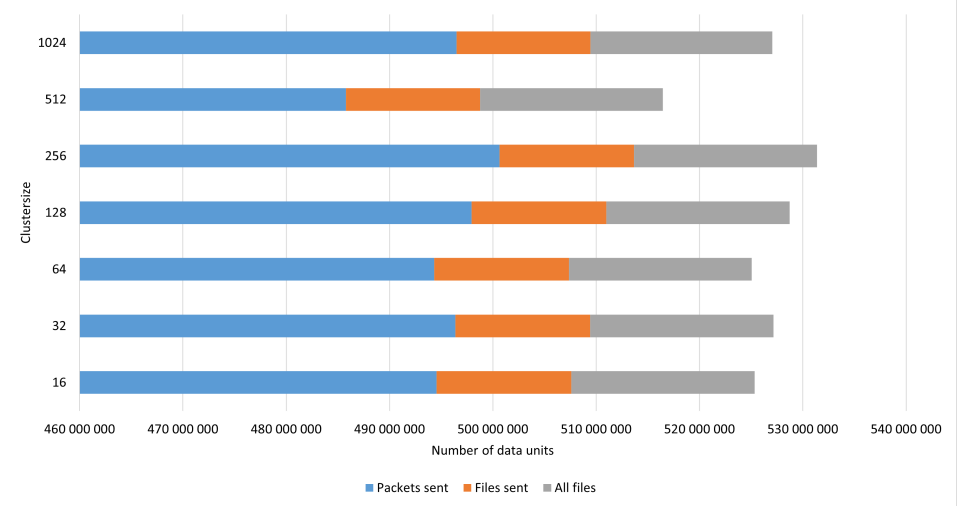


Figure 7.24: E3: Total data generated for cluster sizes 16 to 1024

The diagram in figure 7.25 demonstrates the following:

Red line: The share of synchronized nodes in the total number of nodes.

Blue line: The share of transferred data in the total data amount of data.

Light green line: The probability of single synchronization success - (plotted nearly over the blue line, values are similar).

Dark green line: The share of fresh nodes in the total number of nodes.

Yellow line: The share of never synchronized nodes in the total number of nodes.

All the data-related values are similar for all cluster sizes. It means that the cluster size doesn't affect the simulation's behavior.

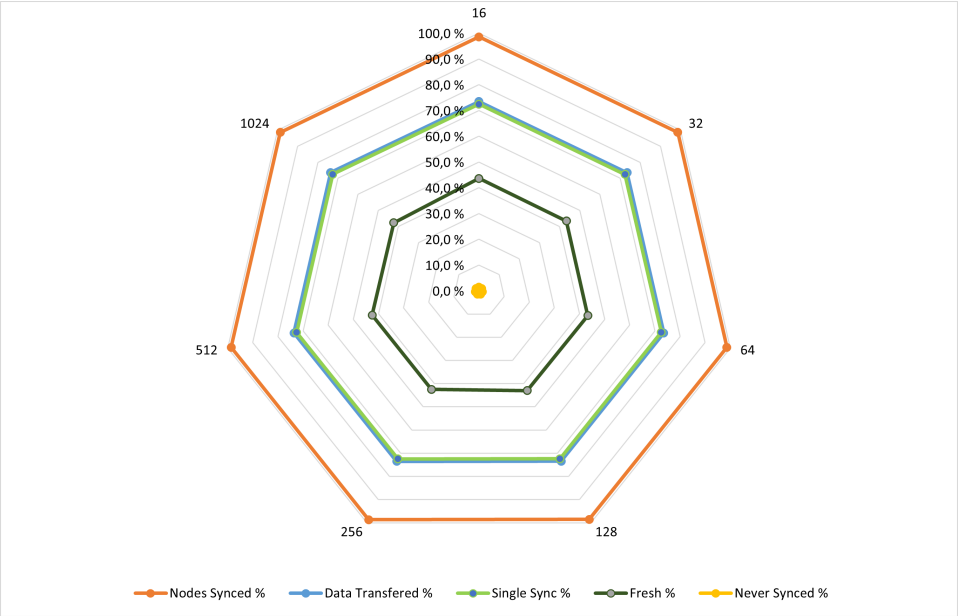


Figure 7.25: E3: Data on shadow nodes for cluster sizes 16 to 1024

Table 7.8 represents network metrics. "*Last Rate*" is the last registered data transfer rate in Kbps (for OUs), while "*Best Rate*" is the highest registered one. "*Get Rate*" is the data rate registered at the SU side. The PTT between instances is shown using the "*X→Y ms*" form. "*Link Delay*" is the time before the OU gets the connection to the UR after the connection was requested. "*!Network*" and "*!Active*" are counters of reasons for no synchronization.

Variables	Network Data									
Cluster	Last Rate Kbps	Best Rate Kbps	Get Rate Kbps	PTT ms	OU → UR	UR → SR ms	SR → SU ms	Link Delay	!Network	!Active
16	1154	1197	620521	29	12	4	11	0,12407	15912546	8375726
32	1126	1200	623977	29	13	4	11	0,12209	15902750	8368381
64	1060	1198	627602	31	15	4	12	0,13132	15874111	8353781
128	1118	1359	606918	33	13	7	12	0,12354	15896081	8311574
256	1064	1359	638541	35	17	5	12	0,15387	15871097	8366790
512	956	1359	699271	41	23	5	13	0,19268	15833448	8332450
1024	834	1359	672130	47	29	5	13	0,23738	15771885	7198937

Table 7.8: Experiment 3: Network

Figure 7.26 contains the data from table 7.8. The red line represents the average of all best data transfer rates on OUs. There is no correlation between the best transfer rate and the size of the cluster, the rate is around ≈ 1300 Kbps on average. The value is almost the same for all cluster sizes and is within technical specifications for the BLE environment. The blue line representing the last registered data transfer rate value decreases by 20% from ≈ 1100 Kbps to ≈ 850. That is because a longer waiting time can apply when one router has to transfer packets from up to 1024 concurrent connections.

Figure 7.26: E3: Network metrics for different cluster sizes 16 to 1024

That is confirmed by the link delay trend represented by the yellow line in the same figure. The link waiting time goes significantly up from ≈ 0.13 seconds to ≈ 0.24 seconds after the cluster size exceeds 128 nodes.

The Packet Travel Time is increasing as the cluster size increases. The PTT represented by the gray line increases by ≈ 20% from 33 seconds at the size of 128 to 47 seconds when there are 1024 OUs in the cluster (PTT specification can be found in table 7.8). The most of the PTT increase is connected with the OU→UR step. For generally higher synchronization probabilities, more data would be generated and sent. The Unit Router may become a bottleneck if it has too many concurrent channels to process, both in simulation and in a real-world network.

7.7.3 Computed performance metrics

Simulation efficiency coefficients and other metrics are listed in table 7.9. This table illustrates the minor difference despite significantly different cluster sizes.

Variables	Metrics				
Cluster	Throughput, MB/S	Mask Cost	Generated data amount, MB/S	Simulation Efficiency	RAM/node, KB
16	709.08	0.132	9920.61	1.852	27.42
32	711.65	0.137	9905.51	1.910	26.55
64	708.78	0.139	9877.44	1.935	26.13
128	713.88	0.143	9889.03	1.976	25.62
256	717.67	0.143	9887.87	1.969	25.71
512	696.68	0.148	9858.48	2.099	24.05
1024	711.75	0.156	9834.03	2.158	23.33

Table 7.9: Experiment 3: Metrics

The graphical representation of RAM usage per node (NRD) is in figure 7.27.

For bigger cluster sizes, fewer simulation resources are needed to allocate all the instances. There are fewer routers to allocate because more units can use the same router. That is why there is a trend of a minor decrease in RAM requirement from 25.5 KB for the size of 256 to 23.5 KB for the size of 1024.

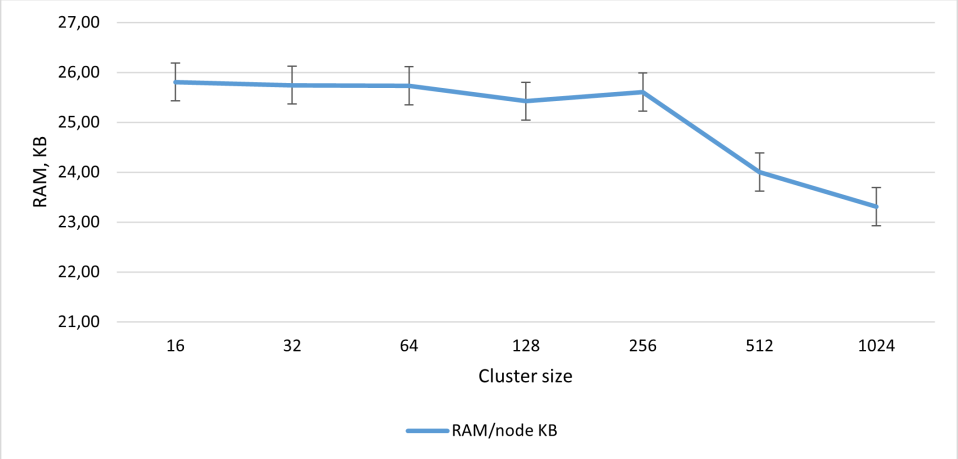


Figure 7.27: E3: RAM use per node, KB

The mask cost (RAM divided by the network throughput) is represented by bars in figure 7.28 on the primary axis. Table 7.9 shows that the throughput is almost the same for different cluster sizes. There is a minor difference because of probabilistic deviation. But there is slightly lower RAM consumption which results in the uprising trend of mask cost coefficient. The coefficient is rising to 2.15 at the point of 1024 edge nodes per router. At the initial point, where the size of the cluster is 16, the value is 15% lower.

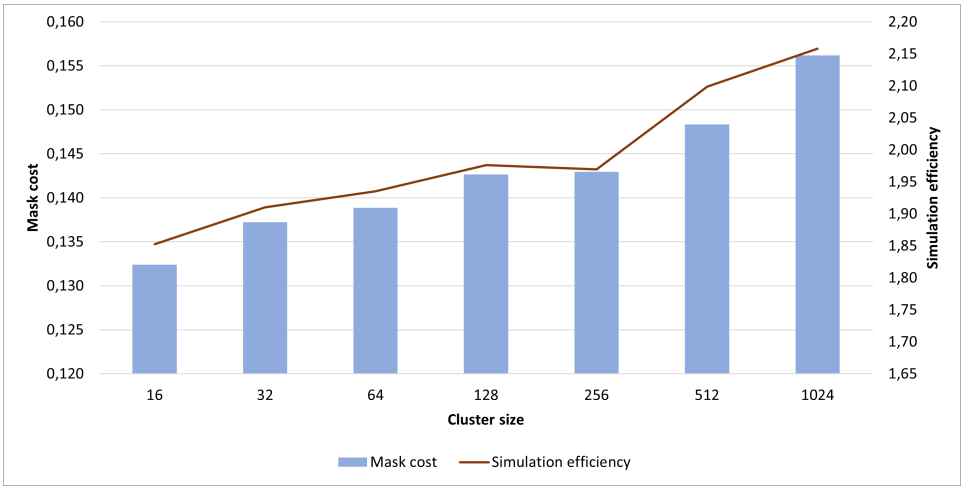


Figure 7.28: E3: The mask cost and simulation efficiency

The simulation efficiency trend is illustrated as the red line plotted along the secondary axis. In the same way, from table 7.9, the amount of data generated by the simulation is approximately the same for all cluster sizes. But there is less RAM needed to generate the same amount of data (GDA) because fewer instances are needed. That is why the simulation efficiency coefficient has an uprising trend that repeats the mask cost trend.

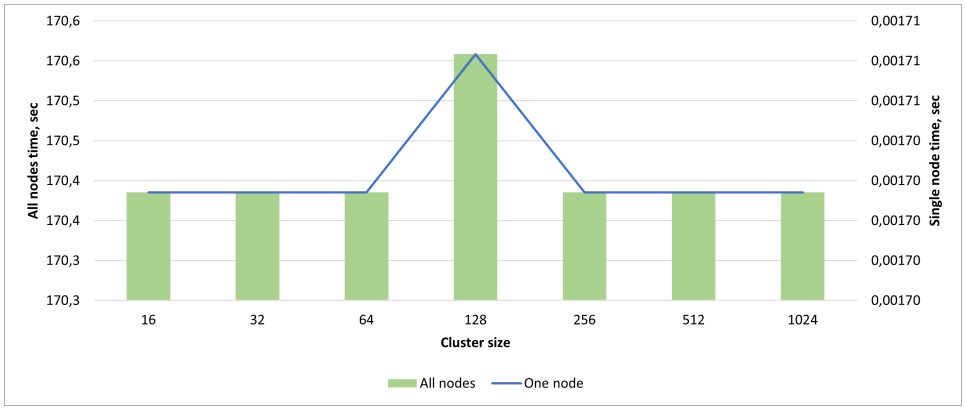


Figure 7.29: E3: Time to get the latest data from one node and from all nodes

Figure 7.29 illustrates the time expected to get the latest data from a single and all edge nodes, calculated using equations 7.2 and 7.3. The figure confirms observations made previously in this experiment. Bars represent the time to get the latest data from all edge nodes (primary axis, the left one). Lines represent the time to get the latest data from the entire observation network (secondary axis, the right one). Generally, time here does not depend on the size of the cluster. Minor deviation at the point of 128 nodes is only 0.00001, which counts for ≈ 0.04 seconds in the scaled model. With respect to the scaled model, the average single node sync time is ≈ 0.002 hours, all nodes sync time is ≈ 170 hours (approximately one week). In other words, a new synchronization of one random edge node is expected every 7 seconds if $P(A) \cap P(N) = 0.025$.

7.7.4 Interpretation of results

From the perspective of a real-world cluster, many concurrent connections may result in a bottleneck. In case of higher synchronization probabilities and, hence, higher data volumes transferred over the network, the bottleneck effect may appear even for smaller cluster sizes. From the perspective of simulation, allocation of fewer routers require less RAM, but the difference is minor. The link delay and PTT can increase, so the RAM economy doesn't bring any positive effect that can be taken into account. The time expected to get the latest data from a single and all edge nodes does not depend on the cluster size.

7.8 Experiment 4: NEIB

Experiment 4 described in the current section tests the scalability of the NEIB protocol. The expected result is the set of values demonstrating the advantages and disadvantages of synchronization via the right neighbor. There are 16 simulations in total, 8 with NEIB and 8 with direct synchronization.

7.8.1 Setup

The experimental setup has the same execution time of 168 seconds as in the previous experiment, the network environment is BLE. The chance of being awake is set to 5%, the chance of the back-haul network being available is 5%. An optimal cluster size of 64 nodes is found in the previous experiment (section 7.7). The number of Observation Units is in the following range: 10; 100; 1000; 10,000; 100,000; 250,000; 500,000 and 1,000,000. For each step, there are two runs: one with NEIB and one without. Variables and statistical expectations can be found in table 7.10.

Variables							Statistical			
Nodes	Cluster size	Exec Time	Conn.	NEIB	Awake %	Network %	P(A) ∩ P(N)	n	X	Binom PMF
10	64	168	BLE	No	5	5	0.0025	168	1	0.276505846
				Yes			0.005			0.363692019
100				No			0.0025			0.276505846
				Yes			0.005			0.363692019
1,000				No			0.0025			0.276505846
				Yes			0.005			0.363692019
10,000				No			0.0025			0.276505846
				Yes			0.005			0.363692019
100,000				No			0.0025			0.276505846
				Yes			0.005			0.363692019
250,000				No			0.0025			0.276505846
				Yes			0.005			0.363692019
500,000				No			0.0025			0.276505846
				Yes			0.005			0.363692019
1,000,000				No			0.0025			0.276505846
				Yes			0.005			0.363692019

Table 7.10: Experiment 4: Variables

7.8.2 Simulation results

An extended range of data extracted from simulations is provided in table 7.11. The table contains node statistics, information about the number of synchronizations, used resources, and the counter of sent messages.

Statistical expectation shows that for $P(A) \cap P(N) = 0.0025$, which is relatively low chance of single synchronization, PMF demonstrates a relatively high value. It means that after 168 trials we can expect at least one synchronization of one OU with probability 28% without NEIB and 36% with NEIB. Use of NEIB doubles the single synchronization chance (according to table 7.10), but it

doesn't guarantee a double chance of success after 168 trials. Does it mean that resources used for the synchronization via the right neighbor would bring 7% additional fresh nodes only? A closer look at the simulation result is needed to answer this question.

Variables	Empirical Data									
Nodes	NEIB	Trans	Nodes Synced	Fresh	!Synced	1-sync	RAM, KB	N sync	NEIB sync	Packets Sent
10	No	11.43 %	20.00 %	0.00 %	80.00 %	2.29 %	17244	4	0	126
	Yes	30.11 %	70.00 %	10.00 %	30.00 %	21.01 %	24216	19	5	35307
100	No	14.15 %	28.00 %	2.00 %	72.00 %	3.96 %	40444	36	0	33615
	Yes	25.68 %	54.00 %	6.00 %	46.00 %	13.87 %	51712	41	34	90576
1,000	No	16.97 %	34.20 %	5.30 %	65.80 %	5.80 %	35450	476	0	396126
	Yes	27.76 %	52.00 %	10.00 %	48.00 %	14.44 %	117060	479	428	906588
10,000	No	17.83 %	34.45 %	6.00 %	65.50 %	6.14 %	503804	5269	0	5366142
	Yes	29.29 %	54.32 %	10.30 %	45.70 %	15.91 %	603756	4982	4519	7714053
100,000	No	17.47 %	34.37 %	5.50 %	65.60 %	6.01 %	4308592	51030	0	51246567
	Yes	15.50 %	8.94 %	3.10 %	91.10 %	1.39 %	3402552	7266	5771	10658952
250,000	No	17.60 %	34.52 %	5.60 %	65.50 %	6.07 %	10321668	129377	0	132698727
	Yes	4.02 %	49.87 %	0.00 %	50.10 %	2.00 %	13992156	11129	3452	5646726
500,000	No	15.98 %	31.48 %	5.00 %	68.50 %	5.03 %	14964809	225407	0	216182979
	Yes	-	-	-	-	-	-	-	-	-
1,000,000	No	4.87 %	8.27 %	2.00 %	91.70 %	0.40 %	24915547	91524	0	60901623
	Yes	-	-	-	-	-	-	-	-	-

Table 7.11: Experiment 4: Data

Figure 7.30 is based on the information from table 7.10 The number of synchronizations is illustrated by lines on the secondary axis (right). Before 500,000 nodes, the counter grows up to the peak value of $\approx 225,000$ synchronizations. At the point of 1,000,000 OUs, the total number of synchronizations goes back to the result observed at the point of 250,000 OUs simulation. That indicates scaling difficulties. For NEIB, the number of synchronization shows the neutral trend until the point of 250,000 nodes. After that point, the synchronization via the right node is not tested due to significant RAM use (over 200 GB) and a decrease in the Generated Data Amount and the network throughput.

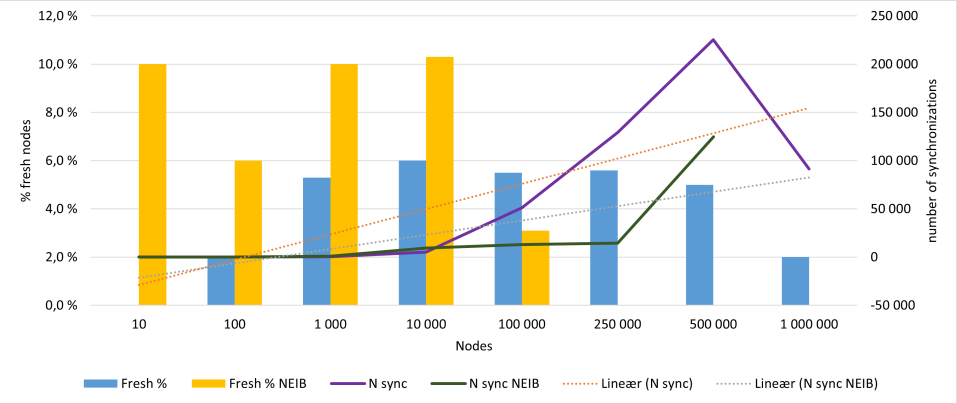


Figure 7.30: E4: The share of fresh nodes and global synchronization counter

In principle, the number of synchronizations in the network is expected to grow together with the number of OUs. The share of fresh nodes is illustrated in the same figure by bars on the primary axis (left). It shows the neutral trend

until the point of 500,000 nodes (≈ 6% fresh OUs). After that point, the share of nodes synchronized during the last 24 hours halves. The NEIB mode can double the number of fresh nodes, but the synchronization issues appear at the point of 100,000 nodes. The conclusion here is that the network size is growing, but the total number of file transfers goes down after the point of 500,000 nodes in case of direct synchronization.

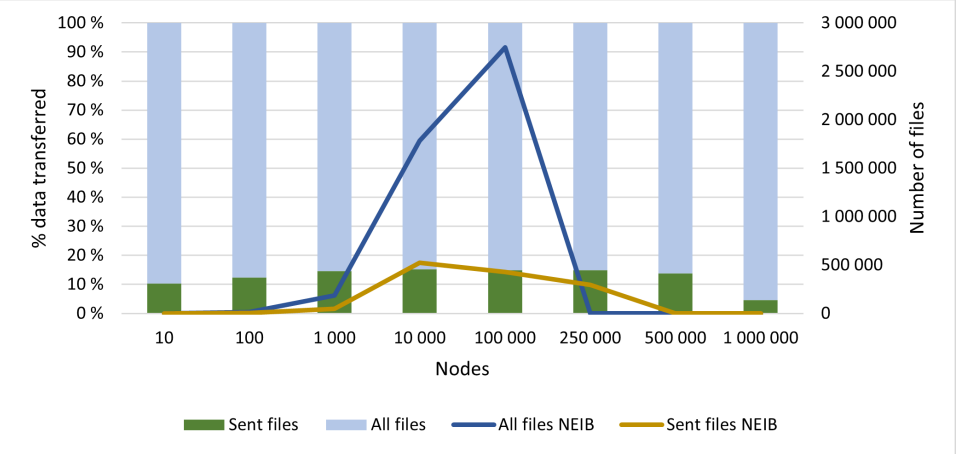


Figure 7.31: E4: The share of data generated in simulation transferred to shadow nodes

Figure 7.31 confirms the trend identified in figure 7.30. The chart combines two different data sets. Stacked bars illustrate the share of sent files in the total number of files without NEIB on the primary axis (left). There are 10% – 15% transferred data on average. It is expected behavior because this value shouldn't be affected by the changing size of the observation network. As in the previous figure, the share of sent files decreases after the point of 500,000 OUs. Such a decrease to ≈ 5% indicates scaling issues.

Lines on the secondary axis (right) represent the total number of sent files and the total number of files generated in the simulation with NEIB. The number of generated files grows rapidly until 100,000 OUs, while the trend of the number of sent files remains almost flat. After 100,000 OUs both the total number of files generated and the number of sent files drops.

The chart for the direct synchronization is represented by relative values, but the NEIB mode is described using absolute numbers. Despite this difference, the dynamics of sent files are similar with similar points of growth and decrease. Hence, the same scaling issues are relevant for both modes, but in the case of NEIB those are observed earlier. It means that more resources are used to keep the NEIB up, but it doesn't bring the desired effect of growing synchronization chance.

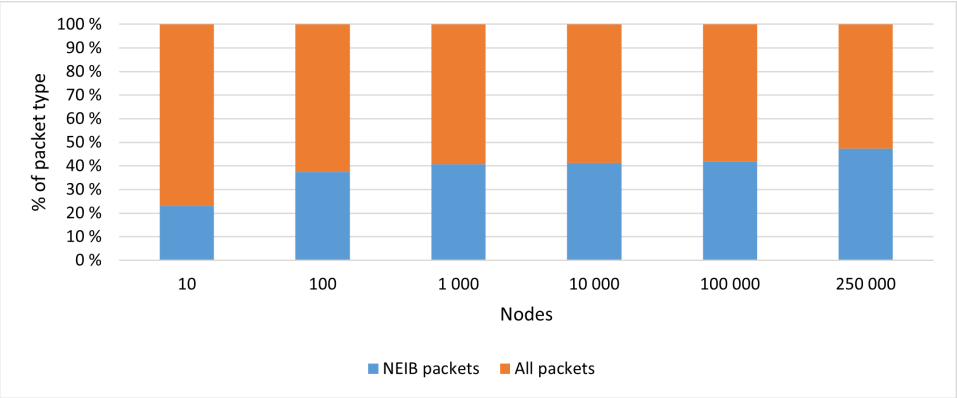


Figure 7.32: E4: The share of NEIB packets in all packets

Figure 7.32 shows the share of NEIB-related messages in the total number of messages. Packets that contain NEIB synchronization data make almost 40% of all the communication inside the model and this share is constantly increasing. It means that while the size of the observation network is growing, the NEIB mechanism becomes more expensive in terms of resources spent on communication.

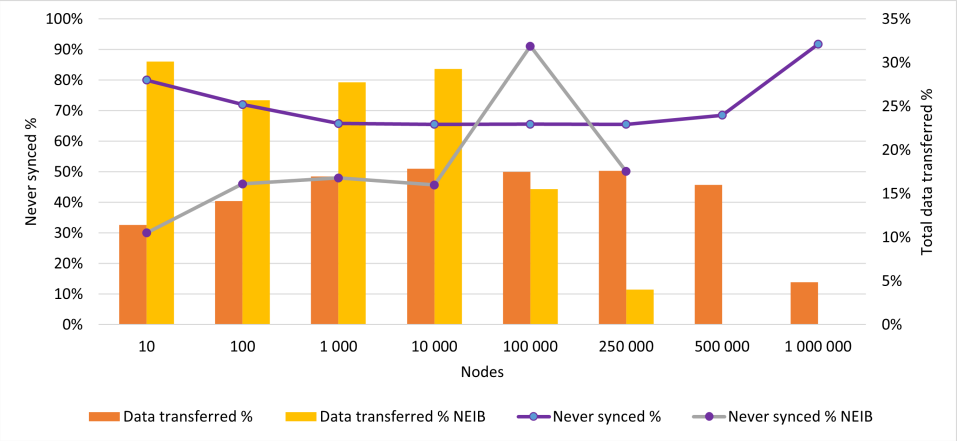


Figure 7.33: E4: The share of never synced nodes and the share of total data transferred

Bars plotted along the secondary axis (right) of the next figure (7.33) illustrate the share of data transferred from OUs to SUs in the total amount of generated data. Without NEIB, the share of transferred data has a neutral trend ($\approx$ 17%) until scaling issues at the point of 1,000,000 OUs. With NEIB, the share of transferred data is 50% – 60% higher until the point of scaling issues at 100,000 OUs. After that, NEIB becomes "inefficient" which would be confirmed by further tests.

Lines on the primary axis of the same figure show the share of never synced nodes. The trend identified before is confirmed here as well. Without NEIB, the share of never synced has a neutral trend (≈ 65%) until scaling issues at the point of 1,000,000 OUs. For the NEIB, the share of never synced is ≈ 25% lower until the point of scaling issues (100,000 OUs). The conclusion is that synchronization via the right neighbor brings higher synchronization success chances, but can be evaluated until the network size of 100,000 OUs.

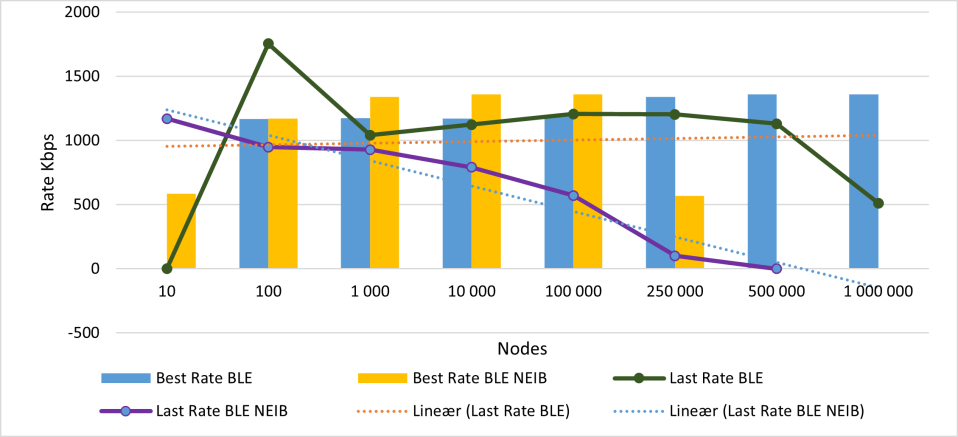


Figure 7.34: E4: Data transmission rate for BLE with direct synchronization and NEIB

The next figure (7.34) demonstrates the data transfer rate in the BLE network with and without NEIB. Bars show the best registered data transfer rate. Without NEIB, the average value is 1250 Kbps. It is expected behavior corresponding the technical limitations of the emulated network environment. For the NEIB mode, the best rate goes down after the network size is 250,000 OUs.

The last synchronization rate represented by the green line indicates some delays at the point of scaling issues (1,000,000 OUs). The last synchronization rate with NEIB represented by the blue line shows the decreasing trend (the reverse dependency on the number of emulated OUs).

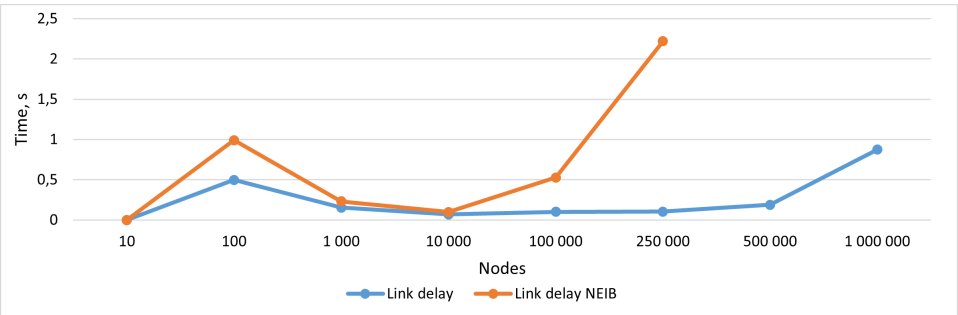


Figure 7.35: E4: Link delay for BLE with direct synchronization and NEIB

When it comes to the delay, figure 7.35 demonstrates how the link wait time changes for various network sizes. The delay increases at critical points: ≈ 2.2s at 100,000 OUs for NEIB and ≈ 0.9s at 1,000,000 OUs without NEIB.

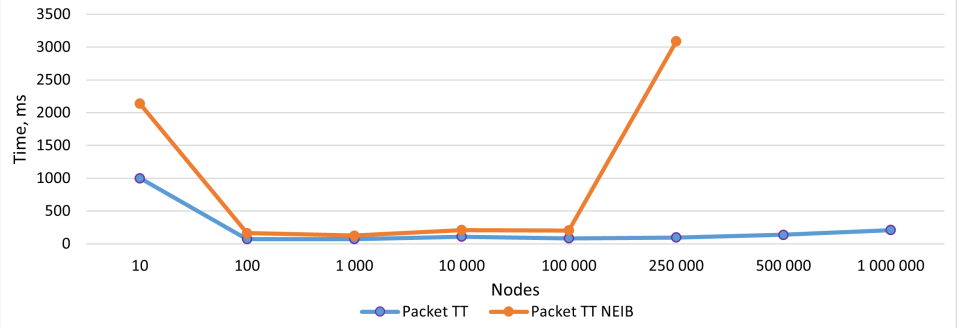


Figure 7.36: E4: Packet Travel Time

The PTT characteristics are illustrated in figures 7.36 and 7.37. Figure 7.36 shows that the PTT repeats the trend identified previously. Lines remain flat until critical points. The PTT increases to ≈ 3s at 100,000 OUs for the NEIB mode and ≈ 0.21s at 1,000,000 OUs without NEIB.

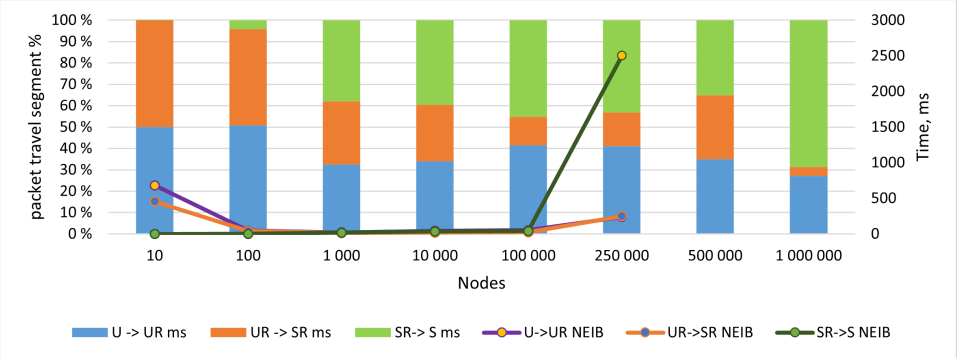


Figure 7.37: E4: Packet Travel Time and structure

Figure 7.37 demonstrates the structure of the PTT. For the direct synchronization, relative values are reflected by bars on the primary axis (left). As the number of nodes is increasing, the share of travel time from the SR to SU increases. An unexpected bottleneck occurs at the side of the shadow network (≈ 70% of the PTT spent here). For the NEIB synchronization, absolute values are represented by lines on the secondary axis (right). The time that message is traveling from the SR to the SU increases rapidly after 100,000 OUs, that is the point of scaling issues. In this case, as well, the point between the SU and the Shadow Router becomes an unexpected bottleneck with ≈ 2.5s PTT.

7.8.3 Computed performance metrics

Computed performance metrics are represented graphically in the current subsection. Figure 7.38 illustrates the simulation's Generated Data Amount (definition 16) and network's throughput (definition 15). The GDA is represented by lines plotted along the primary axis (left).

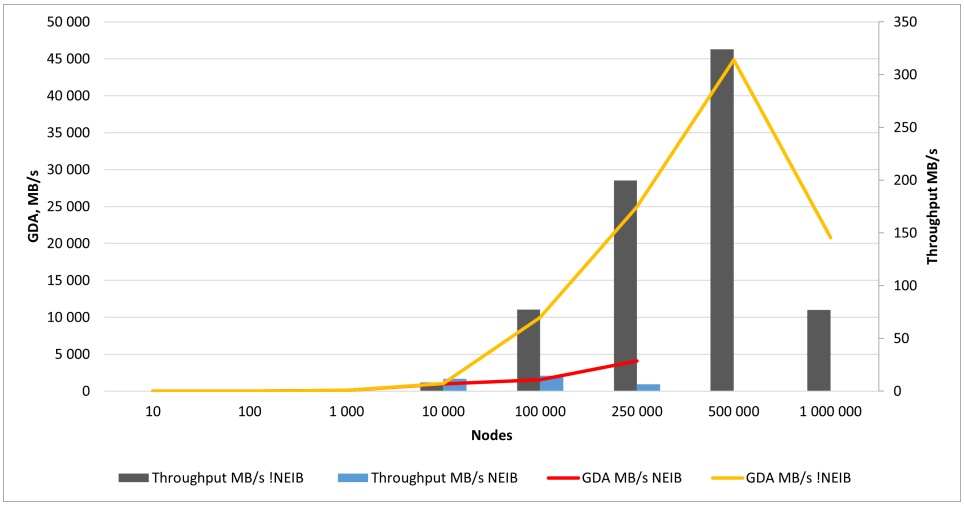


Figure 7.38: E4: Network throughput and simulation's GDA

The maximum GDA value of ≈ 45, 000 MB/s is reached at the point of 500,000 OUs for direct synchronization. The GDA reduces when the number of OUs is 1,000,000. With the NEIB mode, the GDA is 9 times lower and reaches the maximum of the number of ≈ 5, 000 MB/s at the point of 250,000 OUs. It means that most of the resources are spent on communication primitives in the NEIB mode.

Throughput can be found in the same figure (7.38), represented by bars plotted along the secondary axis (right). The data amount transferred per second is following the same pattern as the data amount generated per second does. The maximum throughput of ≈ 320 MB/s is reached at the point of 500,000 OUs for direct synchronization. With the NEIB mode, the throughput is 20 times lower and reaches the maximum of the number of ≈ 15 MB/s at the point of 100,000 OUs. Scaling issues observed previously are confirmed by these metrics.

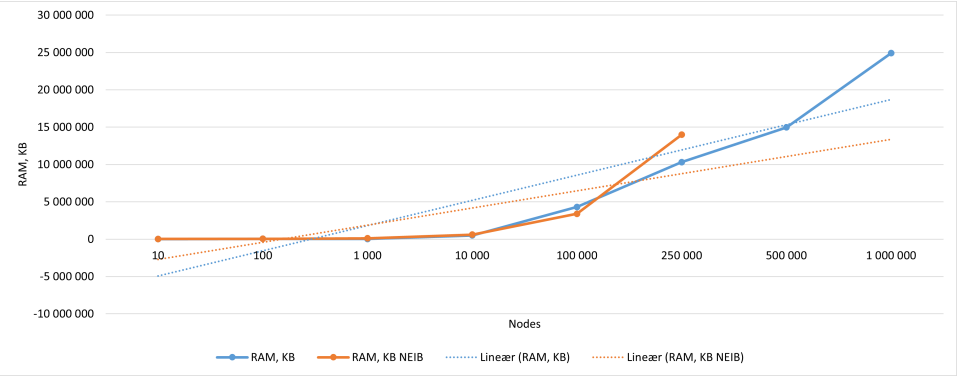
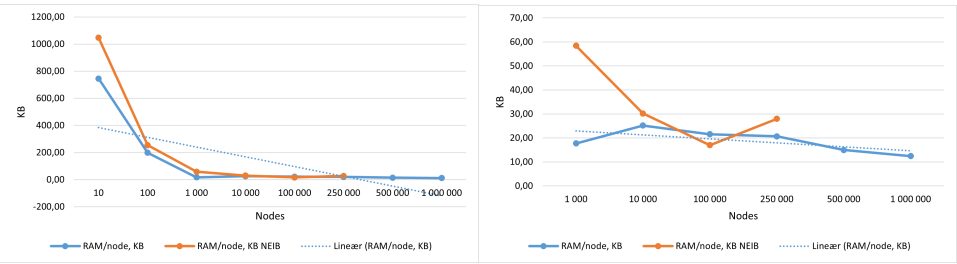


Figure 7.39: E4: RAM usage with and without NEIB for cluster sizes 10 - 1,000,000 (KB)

RAM usage is generally higher for NEIB as illustrated in figure 7.39. But the trend doesn't indicate significant scaling issues, the increase in RAM requirements is expected, because the number of emulated instances grows. The conclusion is that not only the RAM demand triggers the scaling issues.

Figure 7.40 shows the RAM usage per node. Chart 7.40a reflects the general picture, where the NRD is higher for several instances than for several thousand. That is because all the supplementary functions of the simulation have to be started regardless of the network size. For bigger networks, more resources are needed to allocate memory for instances. The NEIB mode requires on average ≈ 30% more RAM per node.



(a) E4: RAM per node, KB **(b)** E4: RAM per node for >1000 nodes, KB

Figure 7.40: E4: NRD for cluster sizes 10 - 1,000,000 for BLE (KB)

Figure 7.40b shows the NRD without points of 10 and 100 OUs (to cut the highest values). Here, 100,000 nodes is the point when a run with NEIB requires less RAM per node (≈ 17 KB) than the direct synchronization does (≈ 22 KB). It means that the simulation with NEIB might be effective on certain network sizes despite the scalability issues identified before. But in general, the NEIB mode is more RAM resource-demanding.

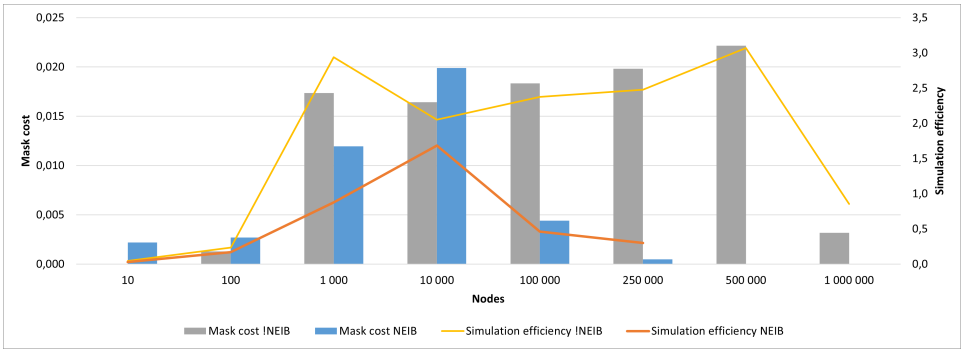


Figure 7.41: E4: The mask cost and simulation efficiency

Figure 7.41 shows the last chart used for the efficiency evaluation in this experiment. Bars represent the mask cost (definition 17) plotted along the primary axis (left). The mask cost coefficient is slightly growing for the direct synchronization until it drops when 1,000,000 OUs are emulated. With the NEIB mode enabled, the mask cost is growing and becomes higher (≈ 0.016) than direct synchronization's mask cost (≈ 0.02) when the network consists of 10000 OUs. That confirms the previous observation, under certain conditions, the NEIB has a 20% higher mask cost efficiency coefficient.

The trend for the simulation efficiency (definition 18) is represented by lines plotted along the secondary axis (right). NEIB mode demonstrates generally lower simulation efficiency because more resources are needed to perform the communication between edge nodes. Without NEIB, the model has a growing efficiency trend until 1,000,000 OUs. The peak of ≈ 3 is reached at the point of 500,000 OUs without NEIB.

The next step is to calculate the time expected to get the latest data. Time-related results of the simulation are in table 7.12. Only the data which can be used to estimate the NEIB mode is presented. The *NEIB benefit* field is calculated as a relation between time to get the latest data from all edge nodes with and without NEIB. The average value indicates 1.75 times speedup.

Nodes	NEIB	Synced nodes	Synced nodes per sec	One node sync, sec	All nodes sync, sec	NEIB benefit
10	No	2	0,01	84,00	840,00	3,5
	Yes	7	0,04	24,00	240,00	
100	No	28	0,17	6,00	600,00	1,928
	Yes	54	0,32	3,11	311,11	
1,000	No	342	2,04	0,49	491,23	1,520
	Yes	520	3,10	0,32	323,08	
10,000	No	3450	20,54	0,05	486,96	1,573
	Yes	5430	32,32	0,03	309,39	
100,000	No	34400	204,76	0,00	488,37	0,258
	Yes	8900	52,98	0,02	1887,64	

Table 7.12: Experiment 4: Time to get the latest data

Results from table 7.12 are illustrated in figure 7.42. Seconds are scaled to hours. It means that if the time to get the latest data from a single node is expressed in 6 seconds, it corresponds to 6 hours. Bars on the secondary axis (right) represent time to get the latest data from all edge nodes with and without NEIB.

Lines represent time to get the latest data from a single node with and without NEIB on the primary axis (left). The time to get the latest data from a single random node is dependent on the size of the observation network. For example, for 10,000 OUs one synchronization is expected every 0.05 hours (≈ 3 minutes) with direct synchronization and 0.03 hours (≈ 1 minute) with the NEIB mode enabled. Before the point of scaling issues, the use of NEIB shortens the time required to get data from the OU to the SU by 1.75 times on average.

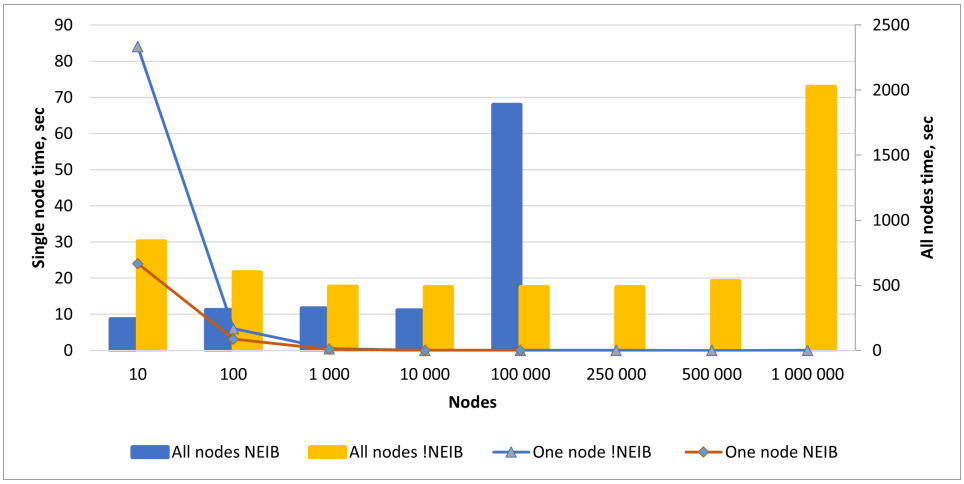


Figure 7.42: E4: Time to get the latest data from one node and from all nodes

Less time is required to get an update from one random node as the number of edge nodes grows. That is because more nodes are able to make more attempts to synchronize but the chance to get data from one particular node would be the same. Statistically, more edge nodes mean more trials are done for the same $P(A) \cap P(N)$ probability. That is confirmed by the flat trend of time required to get the last data from the entire observation network (≈ 500 hours to get the latest data from all edge nodes). With the NEIB mode enabled, the time required to get the last data from the entire observation network is ≈ 300 hours before the point of scaling issues at 100,000 OUs.

7.8.4 Interpretation of results

The simulation demonstrated general scaling issues for 1,000,000 OUs with direct synchronization. Issues appeared for NEIB synchronization already after 100,000 edge nodes. It means that the simulation-based predictions can't be derived when the overall efficiency is low because uncertainty makes results unreliable. In case of future development, the model should be adjusted for the execution of 1,000,000+ Observation Units.

Before points of scaling issues, the results can be used to make some assumptions. Despite the difference in absolute numbers, the observed dynamics of the NEIB mode and direct synchronization is similar with similar points of growth and decrease. Hence, the same scaling issues are relevant for both modes, but in the case of NEIB those are observed at the earlier stage because of the higher workload.

Chosen single synchronization probability (0.25%) represents one of the worst-case scenarios when OUs can synchronize only once in 24 hours and the backhaul network might be available only once in 24 hours at a random point of time. In this case, NEIB synchronization helps to achieve better synchronization chances. RAM cost per unit is around 30% higher, while the number of fresh (definition 14) nodes by the end of synchronization increases by 50%-100%. The time required to get data from the OU to the SU is 1.75 times less on average.

The simulation showed scaling issues after 100,000 OUs because more resources are needed to perform the communication between edge nodes. In real world, such an implementation may bring some degree of communication overhead, because an extra port listening to incoming connections from the neighbor node is needed. That might influence hardware requirements, and, hence, power consumption. Not all the charts are described in details, because numbers are dependent on the current set of variables. It is more important to identify general trends.

7.9 Experiment 5: Success Heat Map

Experiment 5 tests the network characteristics and the synchronization results under various single synchronization success probabilities $P(A) \cap (P(N))$. The goal here is to combine all data derived both statistically and experimentally. Synchronization success heat map creation is the main result. There are 39 runs, 13 for each network environment.

7.9.1 Setup

The experimental setup is the same for all three network environments: Wi-Fi, BLE, and LoRa. Various combinations of chance of node being awake P(A) and chance of the back-haul network availability P(N) give single synchronization success probabilities $(P(A) \cap P(N))$: 0.0001, 0.0005, 0.0025, 0.005, 0.01, 0.025, 0.0625, 0.125, 0.25, 0.375, 0.5625, 0.75 and 1. Execution time is 168 seconds as suggested in section 5, calculated statistically and checked in experiment 7.5. There are 100,000 OUs, such size of the network is chosen to avoid possible scaling issues discussed in experiment 7.6. Cluster size is 64 units to minimize the bottleneck (section 7.7). Variables are in tables 7.13 for LoRa, 7.14 for BLE, and 7.15 for Wi-Fi.

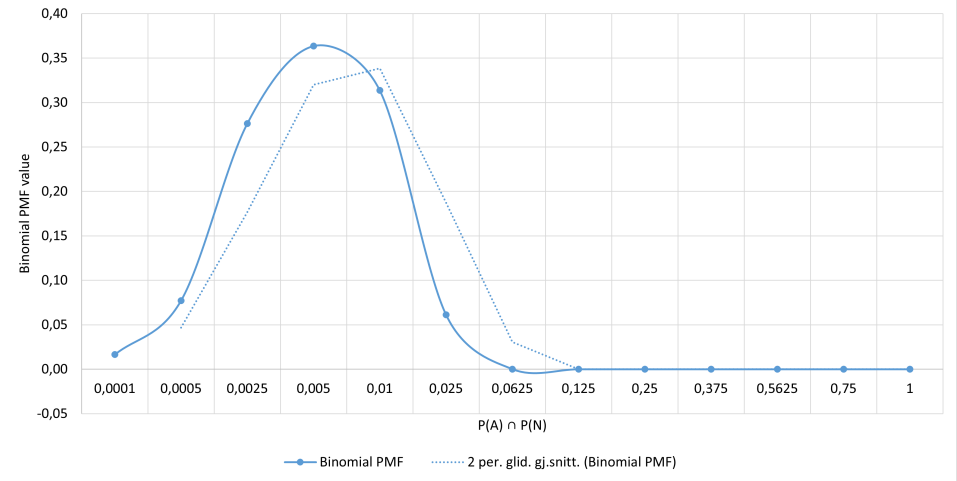


Figure 7.43: E5: Binomial PMF for n = 168 and s = 1

The statistical expectation is provided in tables with variables, values are the same for different network environments. Figure 7.43 shows the Probability Mass Function for 168 trials (n) and at least one expected synchronization success (s). The function reaches its peak at the point of $P(A) \cap P(N) = 0.005$, where the chance that the OU would be synchronized at least once is 30%. Lower probabilities represent "worst-case scenarios".

Variables						Statistical		Empirical Basic					
Nodes	Cluster	Conn	Time = n	P(A)	P(N)	P(A) ∩ P(N)	PMF	Trans	Synced	Fresh	!Synced	1-sync	RAM, KB
100,000	64	LoRa	168	1	1	0.0001	0.0165	0.79 %	1.68 %	0.20 %	98.30 %	0.01 %	5754831
				1	5	0.0005	0.0773	3.89 %	8.14 %	1.10 %	91.90 %	0.32 %	5654827
				5	5	0.0025	0.2765	17.56 %	34.50 %	5.20 %	65.50 %	6.06 %	6141578
				5	10	0.005	0.3637	30.81 %	56.88 %	8.80 %	43.10 %	17.53 %	5835360
				10	10	0.01	0.3136	49.61 %	81.66 %	14.60 %	18.30 %	40.51 %	6308659
				10	25	0.025	0.0612	74.47 %	98.60 %	20.50 %	1.40 %	73.44 %	6681536
				25	25	0.0625	0.0002	88.57 %	100.00 %	15.40 %	0.00 %	88.57 %	7181064
				25	50	0.125	0.0000	93.40 %	100.00 %	12.30 %	0.00 %	93.40 %	6774713
				50	50	0.25	0.0000	96.31 %	100.00 %	7.20 %	0.00 %	96.31 %	6805847
				50	75	0.375	0.0000	97.19 %	100.00 %	4.80 %	0.00 %	97.19 %	6693161
				75	75	0.5625	0.0000	98.06 %	100.00 %	2.80 %	0.00 %	98.06 %	6550555
				75	100	0.75	0.0000	98.91 %	100.00 %	1.40 %	0.00 %	98.91 %	6503276
				100	100	1	0.0000	100.00 %	100.00 %	0.10 %	0.00 %	100.00 %	6464071

Table 7.13: Experiment 5: Variables and data for LoRa

Variables						Statistical		Empirical Basic					
Nodes	Cluster	Conn	Time = n	P(A)	P(N)	$P(A) \cap P(N)$	PMF	Trans	Synced	Fresh	!Synced	1-sync	RAM. KB
100,000	64	BLE	168	1	1	0.0001	0.0165	0.78 %	1.60 %	0.20 %	98.40 %	0.01 %	5680004
				1	5	0.0005	0.0773	3.81 %	8.04 %	1.10 %	92.00 %	0.31 %	5626445
				5	5	0.0025	0.2765	17.47 %	34.37 %	5.50 %	65.60 %	6.01 %	4308592
				5	10	0.005	0.3637	30.77 %	65.80 %	10.50 %	43.20 %	17.48 %	6354914
				10	10	0.01	0.3136	49.27 %	81.57 %	20.00 %	18.40 %	40.19 %	6559335
				10	25	0.025	0.0612	73.60 %	98.62 %	32.33 %	1.40 %	72.58 %	7103293
				25	25	0.0625	0.0002	87.59 %	100.00 %	56.63 %	0.00 %	87.59 %	8105345
				25	50	0.125	0.0000	93.22 %	100.00 %	66.15 %	0.00 %	93.22 %	9110557
				50	50	0.25	0.0000	96.43 %	100.00 %	56.70 %	0.00 %	96.43 %	9130606
				50	75	0.375	0.0000	97.59 %	100.00 %	66.38 %	0.00 %	97.59 %	9365521
				75	75	0.5625	0.0000	98.56 %	100.00 %	72.08 %	0.00 %	98.56 %	9482062
				75	100	0.75	0.0000	99.27 %	100.00 %	73.65 %	0.00 %	99.27 %	9278584
				100	100	1	0.0000	100.00 %	100.00 %	74.63 %	0.00 %	100.00 %	9325042

Table 7.14: Experiment 5: Variables and data for BLE

Variables						Statistical		Empirical Basic					
Nodes	Cluster	Conn	Time = n	P(A)	P(N)	$P(A) \cap P(N)$	PMF	Trans	Synced	Fresh	!Synced	1-sync	RAM. KB
100,000	64	Wi-Fi	168	1	1	0.0001	0.0165	0.82 %	1.73 %	0.20 %	98.30 %	0.01 %	6081871
				1	5	0.0005	0.0773	3.88 %	8.03 %	1.10 %	92.00 %	0.31 %	6112438
				5	5	0.0025	0.2765	17.49 %	34.37 %	5.50 %	65.60 %	6.01 %	6381879
				5	10	0.005	0.3637	30.90 %	57.14 %	10.60 %	42.90 %	17.66 %	6498280
				10	10	0.01	0.3136	49.25 %	87.58 %	20.00 %	18.40 %	4.18 %	6726923
				10	25	0.025	0.0612	73.52 %	98.68 %	43.20 %	1.30 %	72.54 %	7177333
				25	25	0.0625	0.0002	87.60 %	100.00 %	75.60 %	0.00 %	87.60 %	7552031
				25	50	0.125	0.0000	93.45 %	100.00 %	75.50 %	0.00 %	93.45 %	8607589
				50	50	0.25	0.0000	96.49 %	100.00 %	70.50 %	0.00 %	96.49 %	8598620
				50	75	0.375	0.0000	97.52 %	100.00 %	86.30 %	0.00 %	97.52 %	8849556
				75	75	0.5625	0.0000	98.55 %	100.00 %	95.50 %	0.00 %	98.55 %	8947392
				75	100	0.75	0.0000	99.28 %	100.00 %	98.90 %	0.00 %	99.28 %	8382304
				100	100	1	0.0000	100.00 %	100.00 %	99.80 %	0.00 %	100.00 %	8778004

Table 7.15: Experiment 5: Variables and data for Wi-Fi

7.9.2 Simulation results

Simulation results are interpreted both from the model's perspective and the real-world network's perspective. The first figure - 7.44 - shows the success heat map build using the share of fresh nodes from tables 7.13, 7.14, and 7.15 for three network environments. For worst-case scenarios, the share of fresh nodes is almost the same. Rare synchronizations have no impact on network bandwidth, the network has enough resources to deliver the data from Observation Units to Shadow Units.

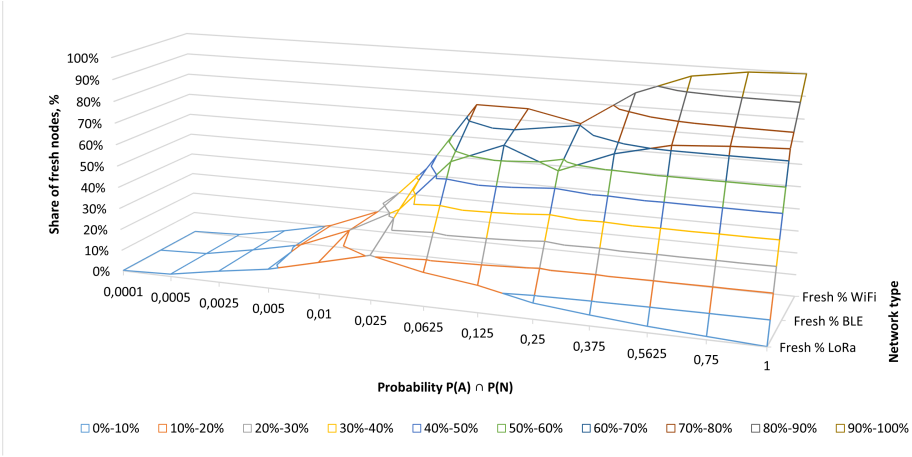


Figure 7.44: E5: The share of fresh nodes for different $P(A) \cap P(N)$ for Wi-Fi, BLE and LoRa

Starting from the point of $P(A) \cap P(N) = 0.025$, differences begin to appear. Such a single synchronization probability corresponds $P(A) = 0.05$; $P(N) = 0.5$ or $P(A) = 0.25$; $P(N) = 0.10$ which are realistic scenarios. In those cases, there would be significant data amount both generated and transferred.

The share of fresh nodes on higher $P(A) \cap P(N)$ probabilities indicate clearly that the data transmission rate becomes a bottleneck. For instance, the Wi-Fi environment is superior to LoRa in terms of the share of fresh nodes by the end of the simulation, but the reason is still undefined. Starting from the point of $P(A) \cap P(N) = 0.0625$, all nodes managed to synchronize during the last 24 hours. In the LoRa environment, there are only 15% fresh nodes at that point. Afterward, the share of fresh nodes in the LoRa environment decreases significantly. The BLE environment demonstrates the flat trend in the share of fresh nodes trend which is almost 2 times lower than for Wi-Fi.

Figure 7.45 shows two data sets on the same axis. The share of never synchronized nodes and the share of all data transferred is plotted dependent on the $P(A) \cap P(N)$. The share of transferred data is not influenced by the network type, remains on the same level disregarding the environment. Only the single synchronization success chance influences the share of transferred data.

The same applies to the share of never synchronized nodes in the total number of nodes. In all three network environments, independent tests showed the same number of nodes that have never synchronized. There is a direct dependency on the single success probability. After the point of $P(A) \cap P(N) = 0.0625$, there are no never synchronized nodes, all the data was successfully transferred to the shadow network.

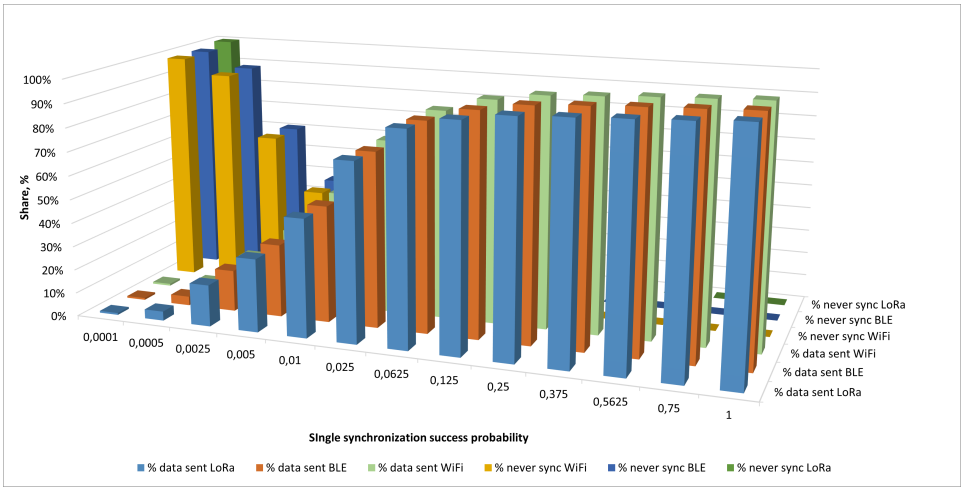


Figure 7.45: E5: The share of data transferred and the share of never synced nodes for different $P(A) \cap P(N)$ for Wi-Fi, BLE and LoRa

Figure 7.46 shows the number of all files generated during the simulation (bars) and the number of files sent (lines) during the simulation. The first trend is that almost all files were transferred in cases of a single synchronization probability higher than 25%. That is the expected behavior of the real-world network.

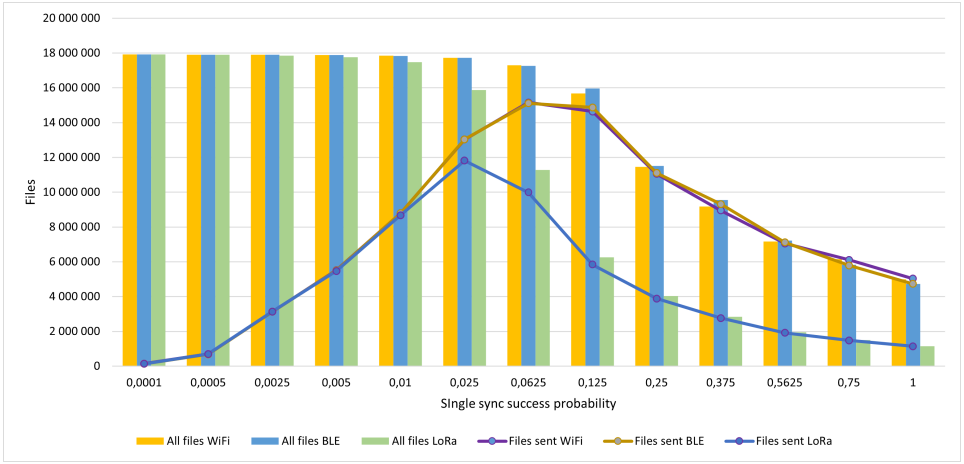


Figure 7.46: E5: All files generated and files sent

The second trend is that the total number of generated files is reducing after the maximum of 18,000,000 files. As the network workload grows, more simulation resources are spent on data transfer, but on the new data generation. The behavior is not expected, but the general trend is correct: on higher synchronization chances, more files are transferred until the point of scaling issues $P(A) \cap P(N) = 0.025$.

The third trend is connected with the LoRa environment. It is generally less data generated, and, therefore, less data sent in this network type. Usage of smaller packets for network communication led to the lack of resources for data generation. It is seen to be an uncertainty, but it doesn't explain the trend of fewer fresh nodes in the LoRa environment.

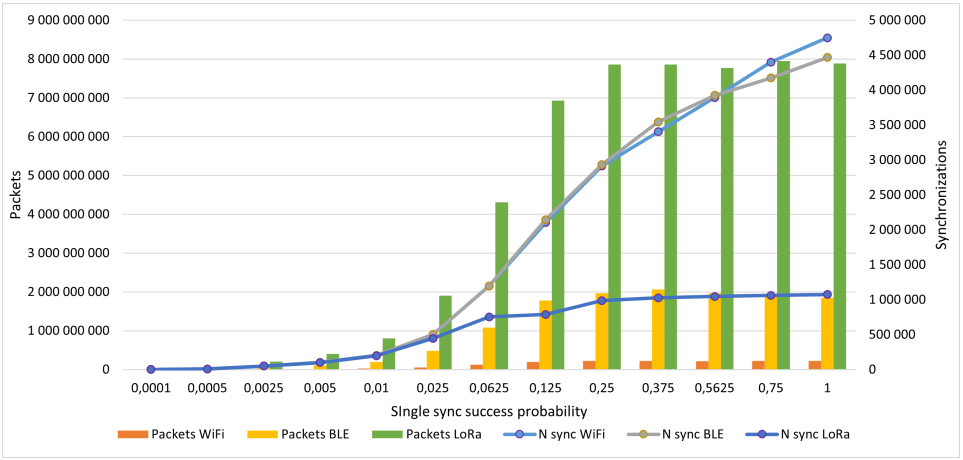


Figure 7.47: E5: The number of synchronizations and number of packets sent

Figure 7.47 has a focus on a number of packets that were sent in different network types (represented by bars). The LoRa environment produced significantly more packets as reflected in the primary axis (left). There are up to $\approx 7,800,000,000$ packets in the LoRa network, $\approx 2,000,000,000$ for BLE, and $\approx 220,000,000$ for Wi-Fi. The difference in numbers corresponds to the technical characteristics of the emulated network types. The secondary axis (right) shows the absolute number of successful synchronizations. For Wi-Fi and BLE the number is very close ($\approx 4,500,000$), but LoRa demonstrates 4.5 times less number ($\approx 1,000,000$). This is the same behavior as discussed previously ("heat map", figure 7.44) but it doesn't explain the reason.

The next figure (7.48) illustrates the average link delay for Wi-Fi, BLE, and LoRa at different $P(A) \cap P(N)$ probabilities. The delay increases when the synchronization probability is high. LoRa shows significant link delay which is up to ≈ 16 seconds at the maximum value of single synchronization success chances. When the $P(A) \cap P(N)$ probability is lower (and the network workload is not so high), all three environments show delays of ≈ 0.2 seconds.

Most likely, LoRa routers use more time to handle incoming requests and the data transfer lasts longer due to the physical limitations of the emulated network protocol. Queues can be the reason for the lower share of fresh nodes in the LoRa environment. The BLE environment demonstrates similar maximum delays of ≈ 3.6 seconds as in the Wi-Fi network (≈ 3.3). The difference is

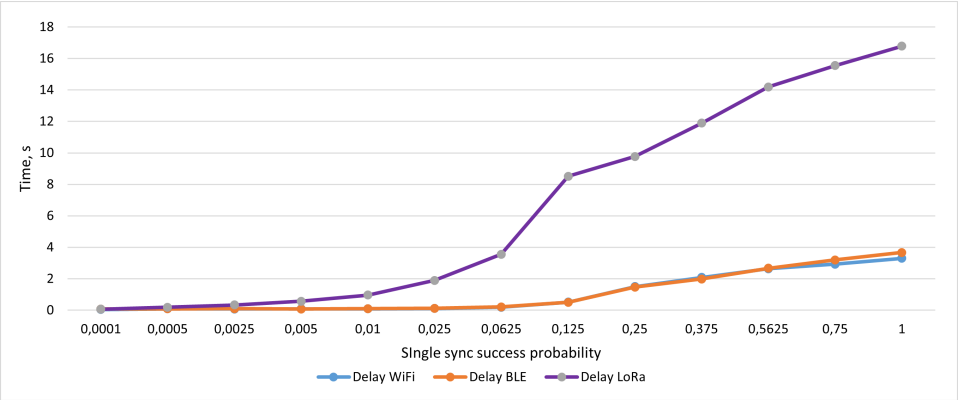


Figure 7.48: E5: Link delay for Wi-Fi, BLE and LoRa

minor and becomes visible only if the $P(A) \cap P(N)$ probability increases and the network throughput is maximal.

The PTT value for Wi-Fi, BLE, and LoRa are demonstrated together in figure 7.49, while figure 7.50 shows details for the different network environments. In "worst-case" scenarios, the LoRa network has longer PTT $\approx$ 350 ms, which reduces to $\approx$ 100 ms as $P(A) \cap P(N)$ probability becomes higher. BLE has longer PTT than Wi-Fi, but the difference is minor. For cases with rare synchronizations, the PTT values are $\approx$ 100 ms. In the case of the high network workload, the Packet Travel Time reduces to $\approx 2 - 10$ ms.

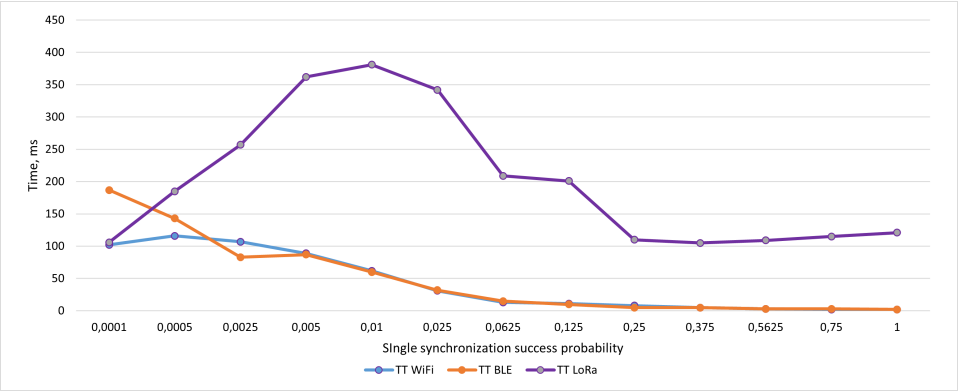
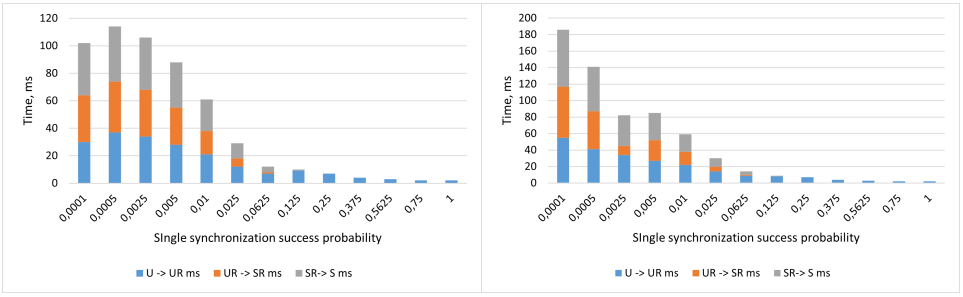


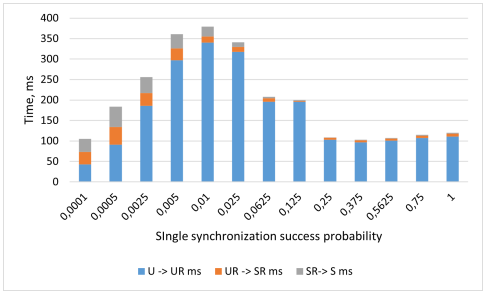
Figure 7.49: E5: Packet Travel Time for Wi-Fi, BLE and LoRa

The assumption here is that when the network is rarely available (P(N)<0.1), the PTT is long because there are orchestration issues between instances. Nodes and routers wake up approximately once a day and follow the global clock principle. The time on edge nodes and Unit Routers is not synchronized deliberately. That is why the time before the packet is received by the Unit Router might be long.



(a) E5: Packet Travel Time for Wi-Fi

(b) E5: Packet Travel Time for BLE

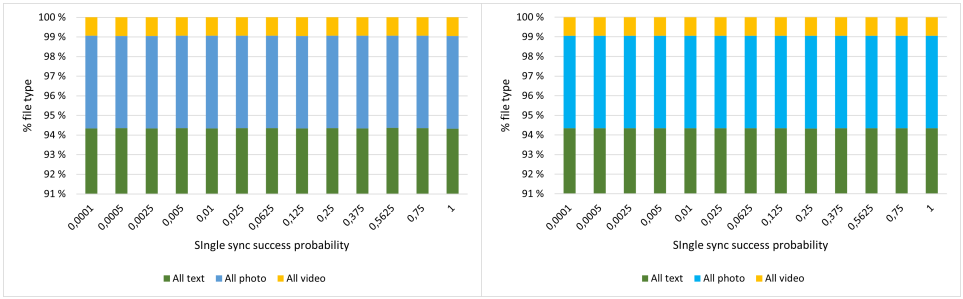


(c) E5: PTT for LoRa

Figure 7.50: E5: PTT structure for Wi-Fi, BLE and LoRa

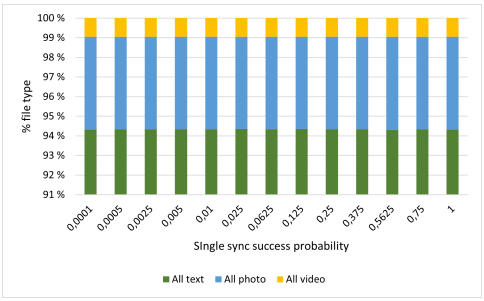
For Wi-Fi and BLE, the PTT is mostly spent on endpoints: Observation Units and Shadow Units. This trend is illustrated in figures 7.50a and 7.50b respectively. The longest PTT occurs in case of a low single synchronization chance. But even in the case of the lowest $P(A) \cap P(N)$ probability, the PTT is distributed evenly. For the Wi-Fi environment, $\approx$ 30 ms is spent on each stage (OU$\rightarrow$UR, UR$\rightarrow$SR, SR$\rightarrow$SU). For the BLE type of network, $\approx$ 40 ms is spent on each stage. As the single synchronization chance becomes higher, most of the PTT is spent between OUs and URs. Thus, Unit Routers become bottlenecks.

Figure 7.50c with data samples from the LoRa environment indicates that the reason for the longer PTT is connected with link delay. There are communication issues with the first intermediate destination: Unit Router. It can be explained through emulated technical limitations of the LoRa network environment and possible orchestration issues between edge nodes. Another trend here is that Wi-Fi and BLE have shorter PTT as the $P(A) \cap P(N)$ probability becomes higher. But for the LoRa environment, the PTT is generally high, and the problematic area is always in the UR. 90% of the Packet Travel Time (which is up to $\approx$ 300 ms) is spent here.



(a) E5: File types for Wi-Fi

(b) E5: File types for BLE



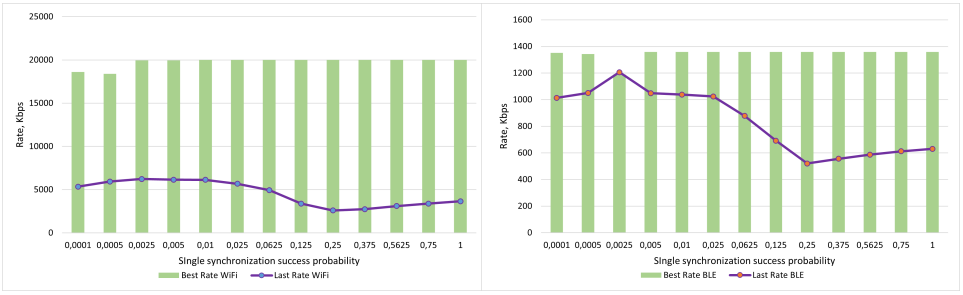
(c) E5: File types for LoRa

Figure 7.51: E5: File types for Wi-Fi, BLE and LoRa

Figure 7.51 combines three charts showing types of files generated in different network environments under various $P(A) \cap P(N)$ probabilities. Figure 7.51a is for Wi-Fi, 7.51b is for BLE, and 7.51c is for LoRa. There are $\approx$ 94% of text data, $\approx$ 5% of images, and $\approx$ 1% of video files. In all three environments, data types correspond to chosen media probabilities. Types of files are not influenced by the emulated network environment or single synchronization probability. Minor differences in samples are results of statistical deviation at given media probability values.

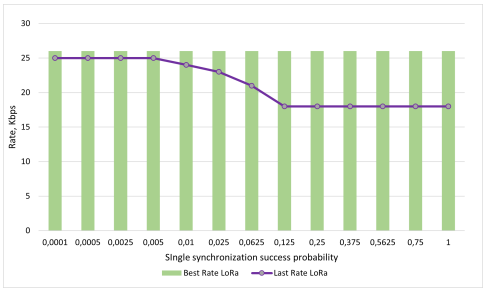
Figure 7.52 provides a closer look at network-specific technical characteristics. The data transmission rate for the Wi-Fi is illustrated in figure 7.52a. Both the last synchronization's rate (blue dotted line, $\approx 5,000$ Kbps) and the best rate (green bars, $\approx 20,000$ Kbps) follow neutral trends. They remain within the range defined for Wi-Fi.

BLE in figure 7.52b and LoRa in figure 7.52c indicate that the last registered data transfer rate becomes lower at higher $P(A) \cap P(N)$ probabilities. The deviation is especially highlighted in the BLE chart (decrease from $\approx$ 1100 Kbps to $\approx$ 600 Kbps). A relatively stable rate trend in the LoRa network can be explained through its generally low max data rate.



(a) E5: Data transmission rate for Wi-Fi

(b) E5: Data transmission rate for BLE



(c) E5: Data transmission rate for LoRa

Figure 7.52: E5: Data transmission rate for Wi-Fi, BLE and LoRa

7.9.3 Computed performance metrics

Performance metrics are represented graphically in the current subsection. The first chart reflects the overall RAM usage for three different network modes in figure 7.53. The total RAM usage is increasing as $P(A) \cap P(N)$ raises.

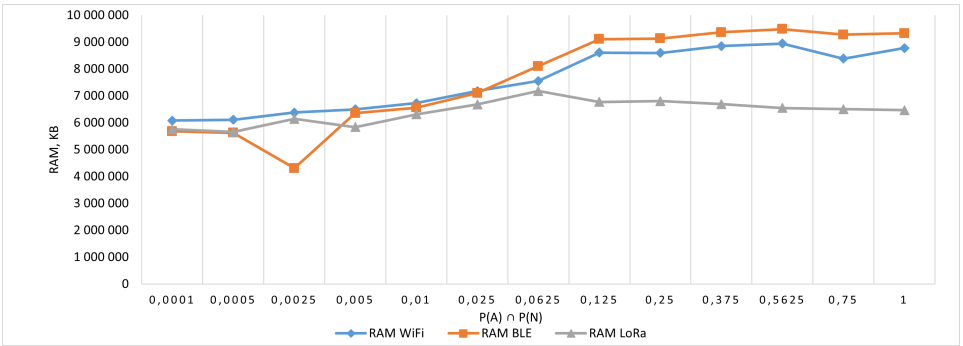


Figure 7.53: E5: RAM usage for Wi-Fi, BLE and LoRa

For Wi-Fi, the range of values is generally higher because there are fewer packets to be sent per message, but more resources are needed for data generation and transfer. The LoRa network demonstrates the lowest RAM requirement despite the smaller single message size and, hence, the higher number of packets.

More resources are used by simulation for data transfer, so there is less data generated. This trend indicates scaling issues. BLE trend is in the middle, but follows the same pattern as Wi-Fi: more synchronizations means more individual packets transfers and more RAM to emulate the network.

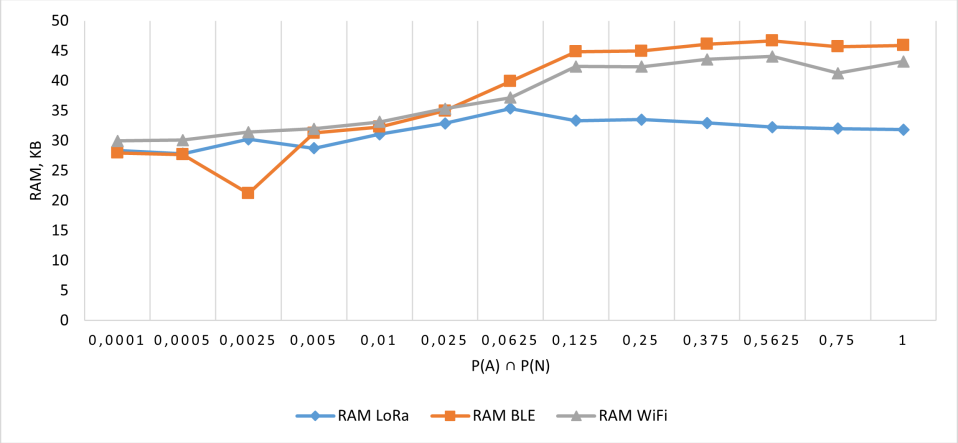


Figure 7.54: E5: RAM usage per node for Wi-Fi, BLE and LoRa

Figure 7.54 reflects the NRD value (equation 7.1). BLE and Wi-Fi demonstrate similar value rows, the minor deviation for the BLE can be explained by statistical uncertainty. The NRD values are generally lower for the LoRa type of network ≈ 30 KB compared to BLE and Wi-Fi which require $\approx 40-45$ KB. The LoRa network environment has a declining trend of RAM per unit requirement as $P(A) \cap P(N)$ becomes higher. It confirms the assumption made for figure 7.53: there is less RAM required for more packets, but some scaling issues take place.

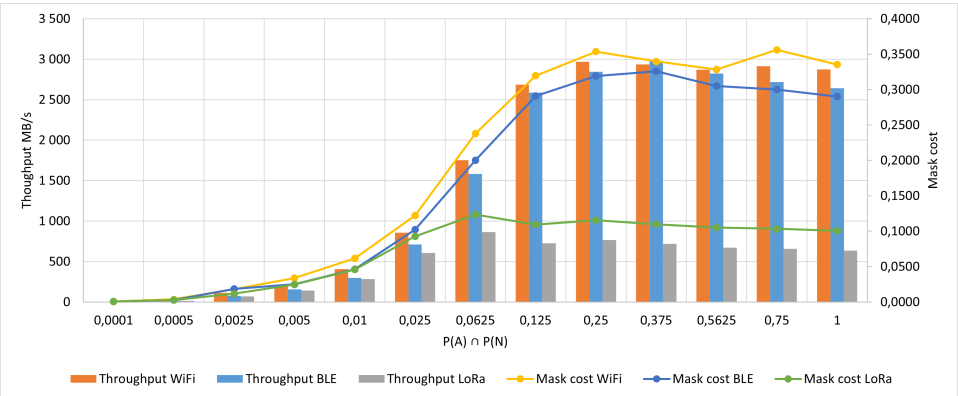


Figure 7.55: E5: Network throughput and the mask cost for Wi-Fi, BLE and LoRa

Figure 7.55 illustrates the network's throughput (definition 15) and the mask cost. Throughput is represented by bars plotted along the primary axis (left).

For the LoRa environment, the maximum network throughput (≈ 800 MB/s) is 3.5 times lower than max for Wi-Fi and BLE (≈ 2900 MB/s). It means there were less data sent per second in the simulation in the LoRa type of network.

The mask cost coefficient values are represented by lines plotted on the secondary axis (right). The Wi-Fi environment has the highest mask cost coefficient (≈ 0.35) which means the most efficient (definition 17) RAM utilization. The values of the mask cost coefficient are expressed by lines plotted along the secondary axis in the same figure 7.55. The LoRa network demonstrated the lowest mask cost coefficient (≈ 0.1) because there is less data transfer for this environment, but the RAM usage is almost the same.

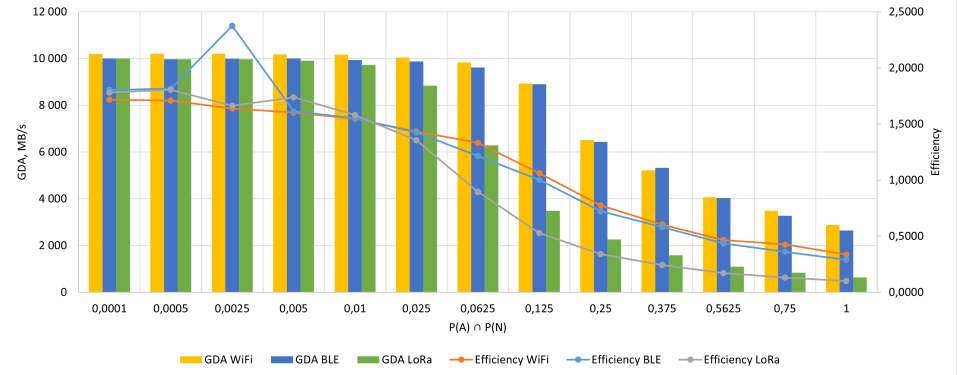


Figure 7.56: E5: The GDA and simulation efficiency for Wi-Fi, BLE and LoRa

Figure 7.56 illustrates models GDA (definition 16) and simulation efficiency (definition 18). The GDA is represented by bars plotted on the primary axis (left). The value is decreasing for all three network types as the single synchronization chance is increasing. It means that the simulation is able to generate less data if there are more resources spent on data transfer. LoRa environment demonstrates extra sensitivity to the synchronization chance, the GDA starts the decrease at the earlier point. In the case of a lower single synchronization chance, the maximum GDA value of ≈ 10,000 MB/s is reached. In the case of high network data traffic, the Generated Data Amount value goes down to ≈ 2,700 MB/s for the Wi-Fi and BLE environments. For the LoRa type of network, the value is 3.6 times lower (≈ 750 MB/s).

The simulation efficiency coefficient (definition 18) is expressed by lines plotted along the secondary axis in the same figure 7.56. The maximum simulation efficiency coefficient is observed at the point of $P(A) \cap P(N) = 0.0025$ for the BLE environment (≈ 2.2). More runs are needed to get the standard deviation because the main trend shows the highest coefficient of ≈ 1.8 for the cases with the low workload of the network. Simulation efficiency coefficients in all three

network environments decrease after the point of $P(A) \cap P(N) = 0.0625$. That confirms scaling issues, the simulation doesn't cope with high data amounts to be generated and sent.

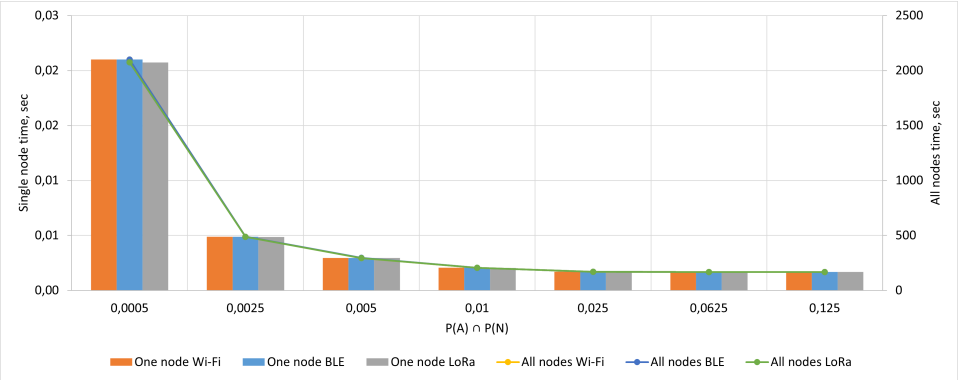


Figure 7.57: E5: Time to get the latest data from one node and from all nodes

Figure 7.57 shows the time estimated to get the latest data from a single edge node and all edge nodes calculated according to equations 7.2 and 7.3. The average time to get the latest data from a single edge node is expressed by bars plotted along the primary axis (left). Lines represent the average time to get the latest data from all edge nodes, plotted along the secondary axis (right). Both time estimates are decreasing inversely to the increasing $P(A) \cap P(N)$ chance, but after $P(A) \cap P(N) = 0.125$, all OUs were synchronized within one week ($\leq$ 168 seconds of simulation). The average time to get the latest data from a single edge node is 0.0012 hours ($\approx$ 6 seconds).

7.9.4 Interpretation of results

Experiment 5 demonstrated that the simulation and, probably the real-world network behave differently under various network environments. For instance, Wi-Fi and BLE require more RAM to emulate networks. Wi-Fi and BLE seem to be suitable for networks with higher throughput because high throughput requires higher bandwidth. The LoRa environment may become a bottleneck in case of many concurrent synchronizations with many large media files.

At the same time, LoRa shows an acceptable level of throughput and mask cost in the case of rare synchronizations, when a single synchronization chance is under 0.025%. LoRa requires less RAM resources and, therefore, less energy. The data rate and link delay are acceptable for networks with fewer synchronizations and smaller files (mainly the text). The time estimates to get the latest data are decreasing inversely to the increasing $P(A) \cap P(N)$ value.

8

Results & Discussion

This chapter summarizes the results described in previous chapters. Statistical expectation validity, prototype design advantages, and disadvantages discovered practically, discussed. The data collected during simulations tests the validity of theoretical assumptions and vice versa.

8.1 Statistical expectation

In chapter 5 statistical expectation was provided based on the theory of probability. It was concluded that 168 seconds of simulation is sufficient to get at least one successful synchronization on all ranges of single synchronization success probabilities $P(A) \cap P(N)$. Statistical expectation formed the basis for the prototype development and initial experimental setup.

In principle, the neighbor node that has the same setup as the Observation Unit would just "give one more synchronization trial" and, therefore, double the chance. The calculation showed that the NEIB protocol would not double the single synchronization success probability, it was expected to gain 30% more synchronization success.

The statistical expectation here is not the exact statistical model representing the real-world network. It doesn't concern causalities, physical limitations, pseudorandom behavior, and so on. Using the equations provided, the model

scalability can't be estimated. For instance, network characteristics and bottleneck effect can be observed in the prototype only due to simplifications of statistical methods in this thesis.

8.2 Prototype functionality

The prototype was developed in order to check statistical assumptions and make predictions for the real-world network behavior. The model was suited for the HPC machine. In the process of application development (section 6.2), parallel data processing (section 6.3.7), time scaling factor (section 6.2.2), and optimization of memory usage via *gprof* (section 7.3) led to a significant decrease in RAM requirement. One of the first versions of the application needed $\approx$ 400 GB RAM to emulate half a million edge nodes. After improvement runs of the simulation required up to $\approx$ 100 GB RAM for all the range of experiments.

Data consistency checks, "graceful finalization", real input data, and various network environments emulation helped to make the prototype closer to the real-world network. At the same time, there were new problems discovered during the simulation evaluation and execution.

8.2.1 Discovered shortcomings

In addition to known errors listed in 7.1, new issues were discovered. A short overview is in the following list:

1. overloaded edge nodes become undesirable and unintended bottlenecks;
2. difficulty with network types comparison under high loads;
3. global clock principle brings delays and orchestration issues;
4. Unit Routers message queues are bigger than expected which result in longer PTT;
5. time scaling may bring measurements distortion at the end of the simulation execution;
6. iterative synchronization approach can result in extra delays in case of future development for the large scale simulation;

7. the "intrusive observation method's" effects were stronger than expected;

8. NEIB required significant communication to emulate bidirectional connection between pairs of several thousand nodes.

Probably there is too much data to be processed and stored inside structures representing edge nodes. Observation Units are generating data and affecting the workflow of the entire simulation. It results in extensive memory demand and extra network delays which are architecture-bound.

Another difficulty is connected with high data traffic in the network. Sometimes network's throughput is especially high because of many concurrent synchronizations with many media files. In that case, most of the simulations resources go to upkeep communication and deliver messages. Emulated network environments are affected by the lack of resources in simulation. As a result, new causalities are distorting the measurement of the network characteristics. In the real world, networks are equipped with suitable routers and synchronization primitives. All networks in the prototype use similar communication paths and mechanisms, Wi-Fi and BLE emulation works better than the simulation of LoRa on very large networks with high $P(A) \cap P(N)$ probability. In a situation when one of the networks is affected more than others, a comparison is not fully credible.

A set of experiments showed the weakness of the global clock approach. Implemented global clock-based way of synchronization was supposed to work without affecting the prototype's functionality. In practice, OUs are emulated by different goroutines and only the OS decides how much RAM would be spent on every single thread. As a result, different instances have a slightly different local clock, so the measurement of the data transfer rate taken between any two instances doesn't reach the desired high level of accuracy.

Message queues are longer than expected, and the router implementation is more sensitive to the size of the buffer than proposed. After an Observation Unit has been unavailable for a long period, it starts the synchronization which may last for a long time. A data transfer can take so much time that synchronization wouldn't finish within a timeout or even an extended timeout. That is why metrics would show fewer recently synchronized nodes.

In the simulation, the time is scaled. In the real world with a similar setup, an OU would wake up every hour, and synchronization would have enough time to finish before the next hour starts. The network would have enough time to finalize the data transfer if the timeout is reached. But what if data transfer in the simulation lasts for several seconds? What if data transfer would not finish by the time timeout is reached? That is why, to be more precise, the time is

not scaled, but the idle time of edge nodes is skipped. As a result, the share of fresh nodes by the end of the simulation can be lower, because the node execution time is scaled from hours to seconds, but the data transfer rate is not. The issue is especially relevant for the LoRa environment with a smaller MTU size and, hence more messages to be generated per file. This moment should be taken into account, and the simulation results should be adjusted accordingly. The drawback is not critical, but an extra effort should be put into router's functionality adjustment in future.

Another aspect is connected with the iterative approach to instance update. Every emulated network element has an internal structure that is updated by a special function once a second. Consequently, all instances are updated once a second. Let's assume that a synchronization lasts for several seconds and the NEIB is on. As a result, a node may "hold" the router by data transfer, while its neighbor can wake up for a second time and request a synchronization. Both the OU port listening to NEIB request and the router would be busy. Theoretically, it can result in unrealistically long link wait time and Packet Travel Time.

Observation of the simulation runtime gave the undesirable effect of extra memory requirement. Profiling (7.3) showed that the function which guarantees the finalization takes up to 20% of all resources in certain periods of time. An HPC cluster orientation lowered the implementation's sensitivity to RAM requirement. In contrast to this, an increasing memory resource demand may result in synchronization and orchestration deviations, because the workload manager has to distribute fewer resources among more instances.

For the NEIB synchronization, most of the memory and network traffic was used for communication between nodes. A significant share of packets was sent only to request a synchronization via the neighbor. It became critical to the simulation at the point of 500,000 emulated nodes. In the case of a real-world network, the effect of efficiency decrease probably wouldn't be so strong because every pair of nodes would be responsible for the communication between each other, there wouldn't be shared memory for the entire cluster.

Despite investigated issues, the simulation demonstrated a certain degree of stability. The issues discussed are undesirable but not critical. Various network environments with their technical characteristics were emulated. The data path in the simulation and its architecture resemble the data path in the real-world network. The current implementation is sufficient for stated goals, so the planned experiments were performed.

8.3 Simulation data collection

After the prototype has been tested and evaluated, experiments provided the data for analysis of the modeled observational network behavior. Results of experiments 7.5, 7.6, 7.7, 7.8 and 7.9 created basement for further theoretical assumptions.

8.3.1 Real world assumptions

There are several predictions about the real-world network behavior listed below:

1. Extra port always listening to incoming connection can be used for "worst-case scenarios".

2. Some environments suited better for the low level of network traffic, some show better results in higher workloads.

3. Edge nodes, endpoints become bottlenecks for the data flows.

4. Cluster size can't be scaled unlimitedly, queues can occur.

5. Failures have a tendency to accumulate over time, the code testing needed before the deployment.

As discussed previously, the NEIB synchronization requires an increase of memory and network coordination resources by approximately 30% for successful emulation. In the real world, such a solution would require an extra port listening to incoming connections. The port should be able to wake up the node or, at least, send a simple negative response. It might be a resource expensive solution from the perspective of power. But should be acceptable for very low synchronization chances, when the back-haul network is unavailable and the node is often offline.

Networks with smaller MTU size and, hence, lower data transmission rates are suited for environments with lower synchronization chances and lower network traffic. If media data transfer is expected, there should be a network protocol with a higher data rate and bigger MTU despite possibly lower energy efficiency. Otherwise, bottleneck effects and longer transfer time would minimize extra gain from choosing energy-saving protocols.

Mainly, bottlenecks occur where edge nodes connect to the network, on the step: OU→UR. The use of superpeers with leader elections or communication

between neighbor routers may ease the workload distribution.

The size of the cluster didn't have any significant effect on the simulation. But the last experiment with higher data volumes showed the possibility of queues and bottlenecks. It can be assumed that unsuitable cluster sizes can have a negative effect on data delivery time, cluster size should be adjusted in accordance with network environment and expected throughput.

It is noted that failures and memory leaks tend to accumulate over time. If there is an issue with the structure of the node, it may become worse after a while. Errors tend to repeat and can crash the node. If longer runtime is expected, generally more attention should be paid to testing before the deployment both in terms of hardware and software.

8.4 Hypothesis validity

The structure of the thesis followed the following logic: The identified problem became the subject of the theoretical research and related work analysis. Methodology showed how statistical expectation can be used for the simulation. But the simulation needed to be first designed and developed so that statistically derived input could be tested. So every step of the project gives the input to the next step.

Experiment 1 showed that the suggested simulation time of 168 seconds was sufficient for stated goals. Experiments 2 and 3 were aimed at the choice of variables for the main experiments. The expectation of at least one successful synchronization ($s = 1$) is dependent on the node being awake and the backhaul network availability with probabilities on the range from 0.005 to 1.0. During simulations, such metrics as the share of fresh nodes, the Packet Travel Time, the Generated Data Amount, and simulation's efficiency were computed. Results collected in experiments 4-5 showed prototypes advantages/disadvantages and expected behavior of the real-world network with the given input and environment.

The main outcome is that there is a cost behind masking the effects of edge nodes being unavailable. Synchronization via the right neighbor can require an increase of resource demand by 30% per unit but may increase the data availability level by 50%-100%. The time required to get data from OU to SU is 1.75 times less on average using NEIB. Energy-efficient networks such as LoRa may save battery but are suited for networks with low data traffic. The next step is a comparison of collected values to the real-world network, so the prediction's accuracy level can be estimated.

9

Contributions

The main contributions of this research is in the Mask artifact, in the experimental results about the Mask, in the simulation system artifact, and in the measured performance of the simulation system.

The main findings are summarized in the current chapter. Those are listed from the edge node's, shadow node's and the client's perspective with respect to the research question: What are the benefit and cost of masking edge node's unavailability?

First of all, the mask provides instant access to the data storage for edge nodes when the back-haul network becomes available. Observation Units do not have to wait for the client connection to push the observational data. The cost of the mask is connected with replicating the edge nodes in the shadow network. The shadow network receives and processes incoming data traffic from edge nodes. It also receives and responds to requests from clients trying to access the edge nodes. Consequently, the needed resources are processing, memory, network bandwidth, and storage.

The shadow network is exposed to the client, it has enough energy to be always-on and has a network connection with sufficient bandwidth. Even if Shadow Units have not received updates for a longer period, the old data would be still available for users. The main value for the client is that the shadow layer is polling the observation network all the time. Clients do not have to wait for edge nodes to become available, in principle they can subscribe for the notification

and access the data even after the edge node became unavailable again. From the client's perspective, the prioritized characteristic of the observation network is the data availability - time to get the latest data from one edge node and the time to get the latest data from all edge nodes. Those are dependent on single synchronization success chances.

For instance, after a week of execution, all the edge nodes in the observation network of 100,000 OUs would synchronize at least once in the case of P(N) = 0.25 and P(A) = 0.25. That corresponds to one node wake-up at least every 6 hours and the back-haul network being available at least every 6 hours. If the cumulative $P(A) \cap P(N)$ probability is lower or the desired time to get the latest data should be minimized, an additional data delivery path should be considered on the edge node's side. Synchronization via the right neighbor principle adds an extra communication channel which increases the data availability level by 50%-100%, but the resource demand grows by 30% per unit. The time required to get data from an OU to a SU is 1.75 times less on average using the NEIB. From the statistical point of a view, a simple approximation can indicate how much time is required to get at least one data update with a given probability $P(A) \cap P(N)$. The expectation of time to get the latest can be adjusted according to the approximation.

The simulation provided predictions and assumptions for the observational network. For instance, the cumulative network throughput of approximately $\approx$ 2100 MB/s and the Generated Data Amount $\approx$ 25000 MB/s can be achieved at the cost of $\approx$ 80 KB Node Resource Demand. That helps to predict the behavior of the observation network with 250,000 edge nodes running for one week. The simulation scales from 10 to 1,000,000 OUs and supports Wi-Fi, BLE and LoRa network environments. Proven simulation's sensitivity to the network type, the number of edge nodes, and networks throughput indicates possible observational network sensitivity to the same list of factors. 5 times more data synchronizations per day may require an increase of resource cost by 50%.

10

Future Work

There is an open list of questions even though actual goals have been achieved. Scope and limitations defined in section 1.3.5 highlight where the current work should be stopped. There is still a list of tasks that is out of the stated scope. Another project development vector is defined by the improvement of known weaknesses. One more aspect of motivation is connected with scientific curiosity. The idea is to test the hypothesis further and try to make sufficiently reliable predictions for the real-world network of Observation Units.

10.1 Model development

Experiments and benchmarking showed opportunities for prototype improvement. Some elements can be implemented as additions, while others require the redesign of existing solutions.

10.1.1 Scaling

There are general scaling issues observed under high network throughput (experiment 7.9) and for large networks (experiment 7.7). A memory leak fix is probably needed. The first step is to do extra profiling and stress tests.

Structures that contain data of network instances (OU, UR, SR, SU) need optimization. One suggestion is to split them into a "static" and a "dynamic" part. All the data that is updated rarely or on startup only can be put into the "cold" storage - the "static" part of the structure. The data that is accessed regularly can reside in "hot" storage and become a part of the "dynamic" structure element. Less IO would result in fewer resources required to allocate structures in memory.

Routers possibly need to operate differently. Start-up every second makes it impossible to invoke synchronization outside the router's uptime. If there will be a development of bidirectional communication between Shadow Units and Observation Units, an extra port always listening to incoming connection is required at the router side.

10.1.2 Superpeers

Superpeers can reduce the workload on the edge nodes. It is a kind of "fog" that can be introduced in terms of "cloud"-"edge" perspective. Superpeers or "fog" nodes can increase data availability because OUs would have an intermediate always available instance located closer than a router. A leader election mechanism would increase the level of resilience to failures. The node with "superpeer" role shouldn't be static in order to enable network recovery in case of node's crash. Any node should be capable of becoming a leader, so the election mechanism has to be dynamic and available on every OU.

10.1.3 Edge computing

Another possible improvement is connected with load balancing. The first step is to find out how much power is required for every wake-up. If there is the time when OUs are up, but idle then more work can be done on nodes, the idle time can be spent on more data processing.

10.1.4 Isle to isle communication

There was an attempt to develop communication between clusters. The prototype has this functionality on the "beta" version, routers can send a handshake and establish connections on the current development stage. The implementation required extra effort but is out of the scope of defined goals. The communication is supposed to go via unit routers as illustrated in figure 10.1. Unit Routers can be connected in the same way as OUs are connected via the right neighbor.

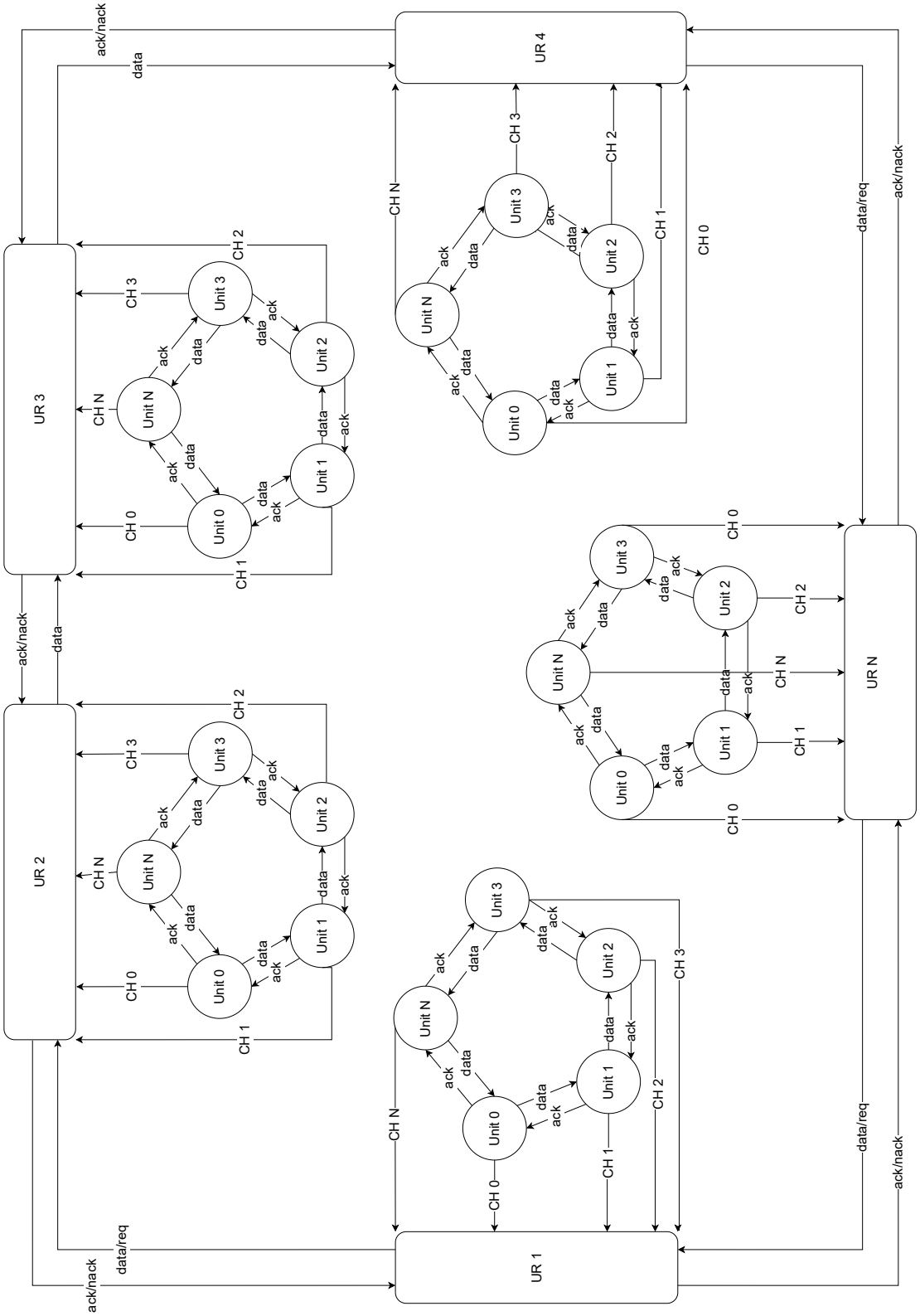


Figure 10.1: Router-router communication scheme.

Superpeers can be assigned to groups of edge nodes in the same way, so the topology can be inherited from the router level. This approach may ease the future development and maintenance process.

Such a connection pattern has two goals:

- to increase data availability, because it would be possible to lookup and synchronize nodes in the nearest cluster;
- to spread the load evenly among neighbor routers, because not all routers have the same number of concurrent connections for the data transfer.

10.1.5 Message Passing Interface add-on

The next suggestion is connected with MPI usage in the simulation. It is impossible to guarantee that the message-passing approach would bring definite benefits. But such an implementation can be compared to the current way of communication via gochannels in future experiments. MPI may help to distribute the workload of the simulation among several compute nodes in the HPC cluster.

10.1.6 Clock synchronization

The global clock approach turned out to be a weak point of the current implementation The next development opportunity is implementation, test, and comparison of logical clocks, vector clocks, and Lamport's clocks [51]. Coordination and synchronization of local clocks require extra communication between nodes. In case of future development, this experiment would include significant prototype redesign, a new set of benchmarks, and experiments.

10.1.7 Pseudorandomization improvement

The pseudorandom approach showed that random values are repeating from time to time in the case of an identical experimental setup. It was sufficient for current goals, but can be improved in case of future work. For example, randomized input can be generated using a hardware-based random number generator which significantly increases the level of input data unpredictability by means of physical equipment instead of programming language functions. Such an addition would make the prototype's behavior closer to the real-world network.

10.1.8 Power mode (*sync_now*)

One more idea is to force an OU to be awake all the time and scan for the network using all available means. That can be done via a remote command, for example, *sync_now*.

$P(N) \cap P(A)$ consists of two independent probabilities. The chance of node being awake would be set to 100%: $P(A) = 1$. In addition to this, NEIB protocol can be enabled if such request (*sync_now*) is received

The prototype would require an adaptive NEIB communication pattern and support for bidirectional communication. As a result, the data availability level can increase significantly, limited only by the back-haul network availability.

10.2 Future experiments

When new features would be implemented, a new set of experiments can be planned. Experiments involve prototype testing and the practical application of results.

10.2.1 Scaling and synchronization

Evaluation of the current implementation has demonstrated several weaknesses. Implementation of previously discussed modifications can improve shortcomings. New experiments can be aimed at scaling and synchronization.

For instance, the network size can be scaled up to 10 million compute nodes for various network environments and both the direct synchronization and synchronization via the right neighbor. The simulation can show network characteristics and data availability expectations in different scenarios, including several types of clock coordination using superpeers or flat topology.

10.2.2 Real cluster comparison

The main result of the enhancement of existing weaknesses should be the model which is closer to the real-world network. Simulation results can provide an expectation of the real-world network behavior. A set of experiments determining the prediction accuracy level should be designed and executed. In the bast case, the simulation would allow predicting the effect of changes in the real-world network.

11

Conclusion

This thesis is an attempt to answer the research question: What are the benefit and cost of masking edge node's unavailability? Scope and limitations do not allow including all the elements of the real-world network. A simplified representation of the network of Observation Units resulted in the prototype implementation.

Statistical approximation provided the initial input for the simulation. Suggested execution time of 168 seconds and the expectation of at least one successful synchronization became basic variables for the simulation.

Execution, observation, profiling, and the benchmarking of the source code showed not only limitations of the implemented applications. It was the first step to identify issues that may be relevant for the real-world network as well. For instance, an undesirable bottleneck between Observation Unit and Unit Router indicated the possible need of extra attention to load balancing. The use of a global clock is relatively simple to implement but may result in a certain degree of asynchrony between edge nodes due to differences in the workload and clock cycles order.

The simulation is conducted on Fram supercomputer owned by NRIS - Norwegian Research Infrastructure Services and located at the University of Tromsø. There was a set of experiments that confirmed the validity of statistically derived variables and that is possible to use those values as input for further experiments. Various cluster sizes and a number of edge nodes were tested

to check the simulations scaling abilities. Simplified versions of network types such as Wi-Fi, Bluetooth 5.2 and LoRa were emulated. Some environments are suited better for the low level of network traffic, some show better results in higher workloads.

The number of emulated edge nodes was between 10 and 1,000,000. The results were assessed for the expected single synchronization success probabilities in the range: $0.0001 \leq P(A) \cap P(N) \leq 1$. Such a range of variables is chosen to cover most of the realistic scenarios. Not only the physical metrics of the simulation were gathered, but also assumptions about the real cluster behavior. For instance, possible benefit from "superpeer" usage was identified. But testing of network topology modification in the simulation requires the prototype redesign. Proposed efficiency coefficients indicated scaling issues. Both Generated Data Amount in the model and network throughput tend to decrease when the number of nodes exceeds 500,000 or the amount of transferred data becomes high at single synchronization success chances over 25%.

Synchronization via the right neighbor is designed and implemented in the simulation. Experiments showed that the increase of resource cost per unit by 30% due to the need of an extra port listening to incoming requests may increase the number of recently synchronized nodes by 50%-100%. The time required to get data from OU to SU is 1.75 times less in this case. Such an assumption is the same for the simulation and the real-world network. This cost may be accepted on very low single synchronization success probabilities when the cumulative chance of the back-haul network availability and node being awake is under 2.5%. Such a single synchronization probability can correspond: (1) $P(A) = 0.05$ and $P(N) = 0.5$; (2) $P(A) = 0.25$ and $P(N) = 0.10$. Those are realistic scenarios. But there is no exact calculation on increasing battery demand and other aspects because a limited range of real-world factors is taken into consideration.

In case of high network workload or network environments with smaller MTU size and, hence, lower transfer rate, the NEIB implementation brings additional communication overhead. Simulation, therefore, demonstrates scaling issues when the number of edge nodes exceeds 250,000.

The current implementation has a list of drawbacks, those are not critical, but should be considered as a potential source of uncertainties. That opens new ways for the future development of the prototype. For now, all the primary and additional research tasks are completed.

Bibliography

[1] Bluetooth 5.2. Core specifications. https://www.bluetooth.com/specifications/bluetooth-core-specification/, accessed: 2021-11-10.

[2] Philipp Andelfinger, Konrad Jünemann, and Hannes Hartenstein. Parallelism potentials in distributed simulations of kademlia-based peer-to-peer networks. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '14, page 41–50, Brussels, BEL, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[3] David Arbour, Dan Garant, and David Jensen. Inferring network effects from observational data. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 715–724, New York, NY, USA, 2016. Association for Computing Machinery.

[4] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. Wiley, 2004.

[5] Aloÿs Augustin, Jiazi Yi, Thomas Heide Clausen, and William Townsley. A study of lora: Long range low power networks for the internet of things. *Sensors*, 16:1466, 10 2016.

[6] Gayatry Borboruah and G. Nandi. A study on large scale network simulators. 2014.

[7] Bradley Camburn, Vimal Viswanathan, Julie Linsey, David Anderson, Daniel Jensen, Richard Crawford, Kevin Otto, and Kristin Wood. Design prototyping methods: State of the art in strategies, techniques, and guidelines. *Design Science*, 3, 01 2017.

[8] Hyun-Jong Cha, Ho-Kyung Yang, and You-Jin Song. A study on the design of fog computing architecture using sensor networks. *Sensors*, 18(11), 2018.

[9] J.H. Conway and G. Polya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton Science Library. Princeton University Press, 2014.

[10] Brian H. Curtin, Rachelle H. David, Emmet D. Dunham, Cullen D. Johnson, Nikhil Shyamkumar, Thomas A. Babbitt, and Suzanne J. Matthews. Designing a raspberry pi sensor network for remote observation of wildlife. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, HotSoS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Kevin Daley, Kun Zhao, and Igor V. Belykh. Synchronizability of directed networks: The power of non-existent ties. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(4):043102, 2020.

[12] Thanh Dang, Nirupama Bulusu, Wu-chi Feng, Sergey Frolov, and Antonio Baptista. Adaptive sampling in the columbia river observation network. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, page 429–430, New York, NY, USA, 2007. Association for Computing Machinery.

[13] COAT. Climate ecological Observatory for Arctic Tundra. `https://www.coat.no`, accessed: 2021-11-10.

[14] C. Fidge. Fundamentals of distributed system observation. *IEEE Softw.*, 13:77–83, 1996.

[15] Rob Flickenger, Steve Okay, Ermanno Pietrosemoli, Marco Zennaro, and Carlo Fonda. Very long distance wi-fi networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Networked Systems for Developing Regions*, NSDR '08, page 1–6, New York, NY, USA, 2008. Association for Computing Machinery.

[16] J. Bullinaria G. Barnett, L. Del Tongo. *Data Structures and Algorithms: Annotated Reference with Examples*. University of Birmingham, 2019.

[17] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[18] P. Godfrey-Smith. *Theory and Reality: An Introduction to the Philosophy of Science*. Science and Its Conceptual Foundations series. University of Chicago Press, 2009.

[19] Melissa J. Goertzen. Introduction to quantitative research and data. *Library technology reports*, 53:12, 2017.

[20] Golang. Go programming language. `https://golang.org/`, accessed: 2021-11-10.

[21] Golang. Golang official documentation. `https://golang.org/doc/effective_go`, accessed: 2021-11-10.

[22] Golang. Golang pprof profiling package. `https://pkg.go.dev/net/http/pprof`, accessed: 2021-11-10.

[23] Golang. Golang runtime package. `https://pkg.go.dev/runtime`, accessed: 2021-11-10.

[24] Golang. Gonum numerical packages. `https://www.gonum.org/`, accessed: 2021-11-10.

[25] Golang. Md5 hash algorithm package. `https://pkg.go.dev/crypto/md5`, accessed: 2021-11-10.

[26] Golang. Uuid package for go language. `https://github.com/satori/go.uuid`, accessed: 2021-11-10.

[27] Gorkem Kar, Shubham Jain, Marco Gruteser, Fan Bai, and Ramesh Govindan. Real-time traffic estimation at vehicular edge nodes. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[28] Rachana Khandurian, S. Rattan, and Arun Uniyal. Understanding the features of ieee 802.11 g in high data rate wireless lans. *International Journal of Computer Applications*, 64:1–5, 02 2013.

[29] E.D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Springer Series in Statistics. Springer New York, 2009.

[30] Guangshun Li, Yonghui Yao, Junhua Wu, Xiaoxiao Liu, Xiaofei Sheng, and Qingyan Lin. A new load balancing strategy by task allocation in edge computing based on intermediary nodes. *EURASIP Journal on Wireless Communications and Networking*, 2020, 01 2020.

[31] Shuo Li, Jiancheng Yu, Yiping Li, and Yu Tian. Marine observation network based on underwater vehicles. In *Proceedings of the International Conference on Underwater Networks amp; Systems*, WUWNET '14, New York, NY, USA, 2014. Association for Computing Machinery.

[32] Yuval Lubowich and Gadi Taubenfeld. On the performance of distributed lock-based synchronization. In *Proceedings of the 12th International Conference on Distributed Computing and Networking*, ICDCN'11, page 131–142, Berlin, Heidelberg, 2011. Springer-Verlag.

[33] Mamoru Maekawa. A n algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.*, 3(2):145–159, May 1985.

[34] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS*, 2002.

[35] Christopher Millard. *Cloud Computing Law*. Oxford University Press, 3rd edition, 2021.

[36] Dan Nessett. Massively distributed systems: Design issues and challenges. In *Workshop on Embedded Systems (Workshop on Embedded Systems)*, Cambridge, MA, March 1999. USENIX Association.

[37] SATURN Observation Network. Endurance stations. `http://www.stccmop.org/datamart/observation_network`, accessed: 2021-11-10.

[38] Pexip. Distributed Proxying Edge Nodes. `https://docs.pexip.com`, accessed: 2021-11-10.

[39] DAO. The Distributed Arctic Observatory. `https://site.uit.no/dao/`, accessed: 2021-11-10.

[40] UiT. The Arctic University of Norway. `https://uit.no/startstudieneinord`, accessed: 2021-11-10.

[41] P. Pacheco. *Parallel Programming with MPI*. Elsevier Science, 1997.

[42] P. Pacheco and M. Malensek. *An Introduction to Parallel Programming*. Elsevier Science, 2021.

[43] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[44] Intel. Different Wi-Fi Protocols and Data Rates. https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html, accessed: 2021-11-10.

[45] J. Rak and D. Hutchison. *Guide to Disaster-Resilient Communication Networks*. Computer Communications and Networks. Springer International Publishing, 2020.

[46] Sheldon M. Ross. In *Introduction to Probability and Statistics for Engineers and Scientists (Fifth Edition)*, page iv. Academic Press, Boston, fifth edition edition, 2014.

[47] Sheldon M. Ross. Random variables. In *Introduction to Probability Models (Eleventh Edition)*, pages 21–91. Academic Press, Boston, eleventh edition edition, 2014.

[48] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, January 2017.

[49] Mirko Stoffers, Sascha Schmerling, Georg Kunz, James Gross, and Klaus Wehrle. Large-scale network simulation: Leveraging the strengths of modern smp-based compute clusters. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '14, page 31–40, Brussels, BEL, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[50] Fram Supercomputer. Official documentation. https://documentation.sigma2.no/hpc_machines/fram.html, accessed: 2021-11-10.

[51] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Createspace Receiving, USA, 3rd edition, 2017.

[52] Cong Wang, Mingrui Zhang, Yu Jiang, Huafeng Zhang, Zhenchang Xing, and Ming Gu. Escape from escape analysis of golang. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '20, page 142–151, New York, NY, USA, 2020. Association for Computing Machinery.

[53] Pengyang Wang, Jiaping Gui, Zhengzhang Chen, Junghwan Rhee, Haifeng Chen, and Yanjie Fu. A generic edge-empowered graph convolutional network via node-edge mutual enhancement. In *Proceedings of The Web Conference 2020*, WWW '20, page 2144–2154, New York, NY, USA, 2020. Association for Computing Machinery.

[54] Yunlong Wu, Qian Zhao, and Hui Li. Synchronization of directed complex networks with uncertainty and time-delay. *International Journal of Distributed Sensor Networks*, 14(5):1550147718778501, 2018.

[55] Dirk Wübben, Peter Rost, Jens Bartelt, Massinissa Lalam, Valentin Savin, Matteo Gorgoglione, Armin Dekorsy, and Gerhard Fettweis. Benefits and impact of cloud computing on 5g signal processing. *IEEE Signal Processing Magazine*, 31:10, 11 2014.