UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Building a Neighborhood Resource Map for IoT and Cyber-Physical systems in Resource-Constrained Environments

Sindre Sønvisen

INF-3990 Master's Thesis in Computer Science
15 May 2022

In memory of Aicko

Abstract

Creating and maintaining a shared resource map between observation nodes that have a behavior where they are mostly sleeping, and have a wake up schedule that are determined at each node locally is challenging. This thesis looks at these challenges, and possible solutions have been proposed to overcome them.

Previous research on the topic of constrained IoT networks have looked at the network, energy, and human constraints separately. But no one has looked at what is needed when all the limitations have to be accounted for simultaneously. Three methods for exchanging resource descriptions are created in this paper.

To evaluate the different exchange methods with different node behaviors, a custom simulator is made. The simulator will simulate communication and resource description exchanges between nodes.

The results show that different node behavior has a drastic affect on when each of the exchange methods work best. The main contribution of this paper is to guide designers of IoT and sensor networks, when they are choosing how the nodes will behave in resource constrained environments.

And the main conclusion are that when there are complete overlap of node behavior, the best method to spread resource descriptions is; to have everyone just sharing description for its own resource. When the nodes are not guaranteed to overlap, some other techniques for exchanging information must be used, like SLM or SLMV that are presented in this paper. Also when there is no overlap, the node behavior becomes even more important. Structuring the wakeup schedules even a bit can help improve overall time to create the resource map.

Acknowledgements

Thanks to my main supervisor Professor Otto Anshus, and co-supervisor Associate Professor Issam Raïs for the guide and inspiration when working with this project.

I also want to thank my girlfriend Christine Sofie for being supportive throughout this project. And a massive appreciation to my parents Jane and Karl Atle for making this possible.

Thanks to the Research Council of Norway (IKTPLUSS program, grant number 27062) for funding the DAO project, that this thesis got its research context from.

Contents

List of Figures

List of Tables

List of Definitions

List of Listings

1 Introduction

Resource constrained environments like the Arctic tundra, make it difficult to conduct data collection over a long period of time. The constraints of not having access to a power grid, limited or no connection to a backhaul network, and the remote location, makes the process of observation very difficult.

Simultaneously are these locations important to monitor. These areas are some of the coldest places on earth, but have seen the highest temperature increases[13]. The increase in temperature is not just changing the local environment itself, but also the wildlife and biodiversity in it [9].

When the environment change because of rising temperature, it can create a feedback loop. When the permafrost deteriorates methane is released in to the atmosphere, and thereby interfere with the snow buildup in the winter months, and melting in the summer months. When the layer of snow disappear earlier in the summer, more of the energy from the sun is absorbed by the ground rater than reflecting of the snow [9].

Getting early signals on the changes in the Arctic region can help prepare for changes outside the Arctic region as well [11].

To do measurements and data collection small computers with sensors are placed out in the elements, here termed **Observation Node** (ON). These ONs need to be small enough to be easily transported to the tundra, meaning they cannot just be equipped with a large battery and big antennas. They also need

to be non intrusive to the environment they are in, so no large installations on site can be made.

2

Problem statement

When we do observations with ONs in the places like the Arctic tundra (AT) we experience a lack of important resources like: energy, back-haul data network, and humans. These limitations make it difficult to collect observation data for a longer period of time. To mitigate these challenges ONs can utilize resources on other ONs, and to do this they need to build up a map of available resources and services other ONs in the neighborhood are willing to offer to them.

The challenges that must be overcome to use resources on other ONs are; First they need to exchange information between them about the resources offered from other ONs and build up a resource map (RM). When we take into account that power is a constraint, and therefore the ONs follow duty cycles. The duty cycle result in that the ONs sleep for most of the time. In addition the duty cycles may or may not overlap. We have a problem with how we can setup the ONs cycles, and based on the setup when we can expect to have the complete RM present at each ON.

Some research have been done on the topics of power constraints, use of duty cycle, network limitations, sharing resources, and deployment on remote places. But they do not look at all constraints at the same time. Many of the solutions also assume that there is a cheap broadcast or multicast functionality available on the ONs, something we do not have.

3

Related Work

Research on the topic of resource discovery and data dissemination have been done previously. In this chapter I will briefly describe some of the state of the art papers written on topics, that this paper have derived inspiration from.

3.1 Telecommunication Strategies

Looking at the telecommunication industry can be a guide in which direction to take. The telecommunication industry is moving away from a proprietary hardware and software, and is changing from Service oriented architecture (SOA) to a microservice architecture. [14]

In the paper "Telecom Strategies for Service Discovery in Microservice Environments" [14] they categorize different service discovery frameworks. They present the notion of structured directories where a single directory store the information, or structure less directories where there is no single directory, and hierarchy or peer-to-peer need to be used. The requests can be static where the clients or the designer have full control of the ONs behavior. It can also be dynamic, meaning the client or designer do not have control of the exact ON behavior.

3.2 Data Storage

Data dissemination problems is not a unique problem for just IoT. Everybody that lives in the informational age and have embraced computers in their professional and or personal life generates data, and wants to keep track of where data is stored.

One solution to tackle the problem of multiple devices that only holds a chunk of the users data each, is presented in a paper by Strauss et al. *Eyo: Device-Transparent Personal Storage* [16]. They used what they called generation-vectors (also known as version vectors) to spread data.

They saw the problems users had keeping track of which device held the data they were looking for. So they wanted to create a more device transparent solution where the same view of the data should be seen from any device.

A naive approach would be to store all the data on a centralized server, and have all the devices make a connection to the server when they want to retrieve or update data. This would introduce other problems like devices not being connected all the time and storage capacity on each device may be limited.

The solution was to not sync all the data but create metadata that can be synced with every device. They manage this with the use of vectors. Devices pull for changes whenever connectivity changes and push when a local object is changed. They send the generation vectors which can be a group of many updates. If a device receives a vector which is higher then it has seen before, it then requests the update.

The authors claim that this solution solves the device transparency in disconnected devices. That it can sync between devices on any network topology and have automatic conflict resolution. The key aspect is that they use an objects metadata as a proxy for the actual object.

3.3 Sensor networks

Neighborhood abstractions in sensor networks have been done in a number of ways before. A similar approach on the topic of what we try to achieve is presented in a paper by Kamin Whitehouse [6].

In their approach each ON can have multiple neighborhoods and each neighborhood holds an array of mirrors. A mirror is a reflection (cached view) of

the ON it represents, and holds annotations about that neighbor. To define a neighborhood the authors write:

> "A neighborhood in Hood is defined by a set of criteria for choosing neighbors and a set of variables to be shared. A node can define multiple neighborhoods with different variables shared over each of them." [6]

When an attribute is shared it is always broadcast to all, and then each ON filter for the attributes they are interested in, and cash them for later use.

3.4 Collaborative Sensing

The idea that multiple IoT devices can work together and collect and process data in a group, instead of everyone doing it themselves is a much discussed topic already and have the possibility to greatly reduce the power consumption in many aspects of society.

Liu et al. presented in their paper [8] a method for IoT devices to collectively sense context. It would act as a layer between the application layer and the hardware. The applications would not know if the sensor reading was done locally or at another device.

3.5 Edge computing

Edge computing is computation done on the edges of the network, so putting smaller devices/servers closer to the end user. This architecture can reduce latency for devices just by the data being closer to the user [3]. When we have devices at the edge we usually call this the edge layer. Mist computing is meant to extend IoT bellow the edge [15]. Sattari et al. present in the paper "Edge-supported Microservice-based Resource Discovery for Mist Computing"[15] a hybrid solution for resource discovery by using directory-based and directory-less. They made the mist devices first try to use a directory for resource discovery. If the central resource directory is not reachable they switch to a multicasting mode, where they advertise and discover resources.

3.6 Update Distribution

Tollefsen et al. wrote in a paper, "Distribution of Updates to IoT Nodes in a Resource-Challenged Environment" [17] about update distribution in resource constrained environments. They assume that one ON in each neighborhood have an backhaul network. In the paper they also experiment with different metrics for ON behavior. They state that the results are sensitive to the ON behavior.

3.7 Peer-to-peer

Distributed hash tables (DHT) is often used in peer-to-peer systems, it is regarded as a scalable, adaptable and fault tolerant in large dynamic environment [18]. DHT can be deployed in many different ways, and traditionally support key-value pairs. But to adapt a traditional DHT to a grid service, the paper present pGridS that organize the services in two virtual organizations. Each service description is split in to attributes and values. Then by using Chord for consensus they can then organize the system in to an attribute overlay network, and a value overlay network.

4

Context

The motivation for this project comes from the Distributed Arctic Observatory (DAO) project. The aim of the project is to improve monitoring systems by making them more efficient and robust. The project also wants to improve the ease of use, longevity and performance [10].

4.1 Characteristics of the arctic tundra

What we consider AT in this paper is characterised to be large and cold, far from civilisation with no power grid or backhaul network. In the winter months there is almost no natural light, and in the summer months it never gets dark.

4.2 Spreading Mechanism

When we later in 11.1 analyze the results, it will be clear that different spreading methods and the other parameters will affect the total amount of time to spread from all to all, and therefore also the energy required. Four different scenarios is therefore created to catch where each method works best, and where they do not work.

- Scenario number 1 is called lazy, this means that we want to spread the

data but it is not essential to do it in a short amount of time. This scenario fits well when there is no to little churn of the resources. It is fine to use more time to reduce the energy consumption. It is fine to use weeks to send the data. So we say this criteria is met as long at the resources finished spreading within the maximum time frame of 30 days.

- Scenario number 2 is called medium, the resource spreading is to happen faster but still not use to much amount of extra energy to do so. We set the max time used for this criteria to two days.

- Scenario number 3 is called eager approach. For the eager approach the resources must be sent fast, some delays are still accepted. For this scenario the resources must spread within an hour.

- Scenario number 4 is called panic. Here we want to use the shortest time possible to spread the data regardless of the energy consumption. The maximum time here is set to 10 minutes. This scenario represent a situation where something happens suddenly that the ONs must react to, or there is a high churn rate.

4.3 Parameters & Factors

In this chapter i will describe the different parameters and factors experimented with in this paper.

4.3.1 Exchange type

Each **exchange type** will be described in section 5.5. The ones that will be explored is **Send Local Resource** (SLR) described in 5.5.1, **Send Local Map** (SLM) described in 5.5.2, and **Send Local Map Vector** described in 5.5.4.

4.3.2 Time Between Wakeup

Time between wake up is from now on referred to as just **wakeup**. It is the duration of time between the start of each wake up period. In each experiment it will be set to the same value for every ON.

4.3.3 Uptime duration

Up time duration, from now on just called **uptime**, is the amount of time each ON is coherently awake. Meaning the time from the ONs wake up, until they go to sleep again. This will also be set to the same value for every ON in each experiment.

4.3.4 Spread

The spread is the amount of deviation in exact wakeup time between ONs, meaning they wake up equally often but the exact point in time do not need to be the same. The different types are illustrated in figure 4.1.

1. **No spread**: every ON wake up at the same time.

2. **Random spread**: The ON wakeup at a random non structured time.

3. **Uniform spread**: The nodes wakeup cycles is structured to have minimum overlap, but guarantee enough overlap to be able to transfer the resource(s).

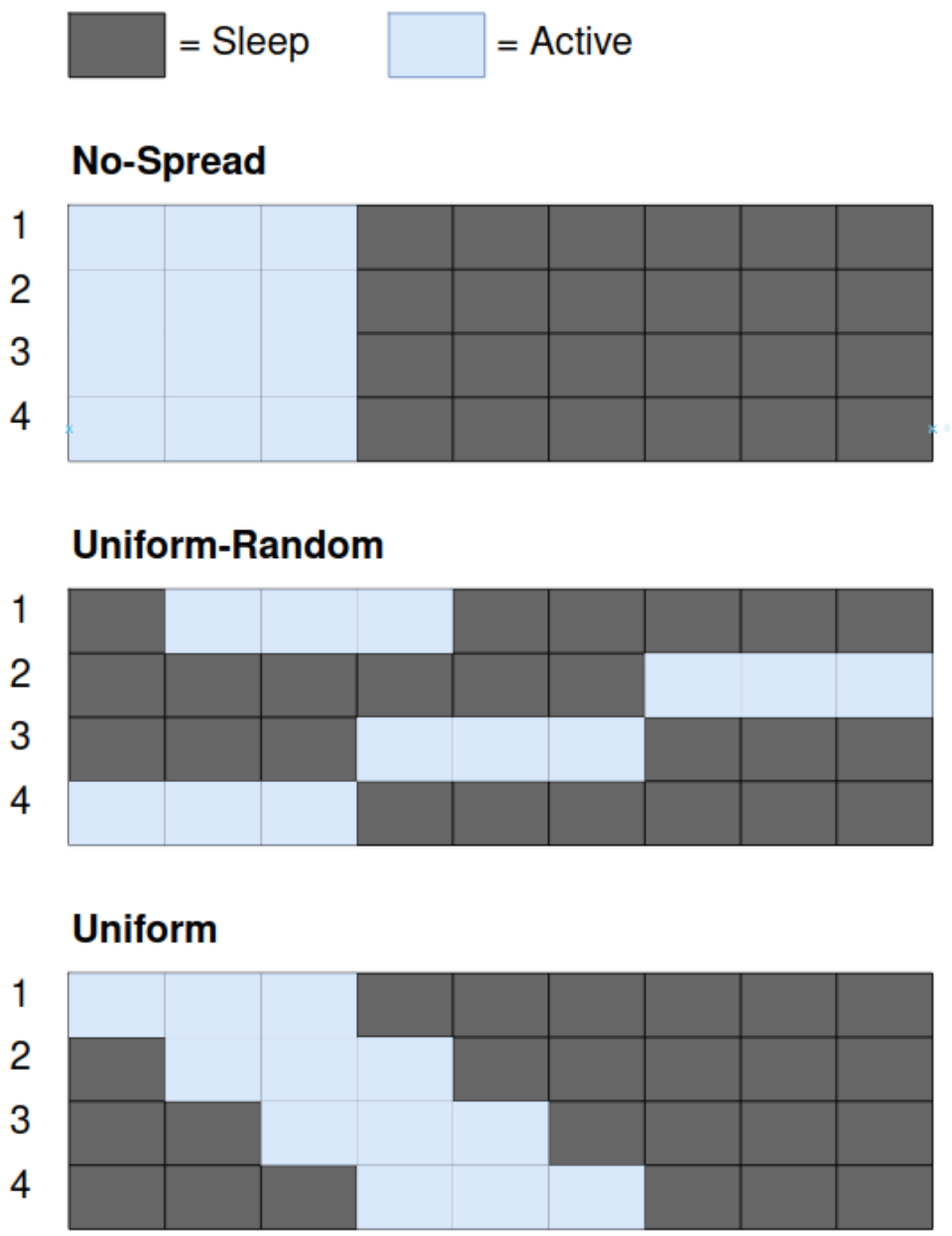


Figure 4.1: Illustrate the different spreading types

4.3.5 Resource Description size

To determine the resource description size a small example resource description was made, counting five int values. An int is 4 bytes [1].

- Type int
- hostID int
- battery int
- latitude int
- longitude int

The example description totals to: $5 * 4byte = 20byte = 160bit$. There is no theoretical max value for the resource description, the size of the description will be based on the number of fields in it. But it is well worth keeping the description size as small as practically possible, because a higher size will require more time to send each description.

4.3.6 Throughput and Latency

Bandwidth is often referred to as pipe size. It is the theoretical volume of data that can be sent over the network [4].

Throughput is the data amount that at any given time leaves the sender [4]. The units we will be using in this paper is bits per second (bit/s), and kilo bits per second (kbit/s).

Latency is the amount of time the data takes from the sender to the receiver. The unit is measured in millisecond (ms), and is the round trip delay [4]. Round trip meaning the time from sender through receiver and back again.

4.3.7 Number of ONs

The number of ONs is the number of individual ONs in a neighborhood. The realistic maximum goal for ON count when everybody is expected to be able to communicate with everybody else, and taking the current network technologies

1. Data Type Ranges `https://docs.microsoft.com/en-us/cpp/cpp/data-type-ranges?view=msvc-170` (Accessed: 07.05.2022)

in to consideration, is probably around 100 ONs. Any more than a 100 and the ONs would need to be put to close to each other [2].

4.4 Requirements

In this section the guidelines for the architecture of the system is presented. They will be based on the constraints and goals outlined chapter 1 and chapter 2.

4.4.1 ON Behavior

The ONs are expected to be mostly sleeping and only wake up in intervals to do measurements and communicate with other ONs or clients. They will be following the same sleep-awake cycle (from now referred to as a **duty cycle**).

4.4.2 Energy

Energy in this sense is the electric power the ONs use when awake. Ideally the ONs should not just wake up to create the maps, meaning they need to mainly use the time the ON is already awake for sensor readings. This may not always be possible or practical, so then the goal will be to find the most suitable configuration for the parameters described in section 4.3. While also keeping in mind the energy consumption and scenarios outlined in section 4.2. The ONs will all be on a fixed energy budget.

4.4.3 Identification

For the ONs to be able to create, and make use of a resource map each ON needs to be uniquely identified in each neighborhood.

4.4.4 Neighbor Detection

The ONs need to have a method to detect other reachable ONs. The use of a non-connectable beacon is an alternative. A beacon is a small device that uses wireless theology to broadcast small peaces of data. The data can be anything.

2. Discussed in a supervisors meeting

When using Bluetooth low energy, beacons can stay active just by using a coin cell battery for years [7].

In this case we specify a non-connectable beacon. This is the best solution for low energy consumption, because the only job is waking up and transmitting data then go to sleep again[7]. This means one device cannot connect and control the beacon on another device. The beacon is just used for advertising that a ON is awake.

Solutions that use Bluetooth Low Energy for neighbor detection already exist. Bluetooth Low Energy neighbor detection (BLEnd) is one such alternative[5], the beacon broadcast a small message/signal that the other ONs can pick up. The BLEnd protocol determines the structure for broadcasting signal, and listening for other signals.

4.4.5 Communication

For the ONs to communicate they need to have at least one network technology present on them. There are many possible solutions for this, but the main goal is to have a long range and a low packet loss. The technology must be robust enough to not fail when the weather and environment change.

Network

The network technology characteristics outlined in this thesis is based on LoRa. LoRa is a long range low power wireless communication solution. These are requirements we need to meet to have a usefully observation cluster. The LoRa specification is extensive and since this is only one solution we only base the communication throughput, latency and packet sizes on it. The sharing of the spectrum is not taken in to consideration, even though this would need to be considered and planned for if it was to be deployed in the real world.

LoRa Physical Layer

Maximum transfer unit or Payload for LoRa physical layer is 2-255 octets (bytes) and theoretical data rate up to 50 Kbps (Kbit/s) [2]. But testing shows much lower speed in real world environment [1]. Therefore a throughput of 320 bit/s and 25 kbit/s is chosen for the experiments, and a latency of 0 ms and 500 ms.

4.4.6 Assumptions

To reduce the scope of this thesis I have made some assumptions:

- Two ONs in range can create a peer to peer connection.
- When an ON wakes up, it has a local network that works.
- Clocks are synchronized.
  - Clock drift synchronization are abstracted away.
- An ON can only communicate with one other ON at a time.
- Each ON have a non connectable beacon.
  - If an ON can see the beacon of another ON they are in range to create a peer to peer connection.
- The ONs are able to use the total bandwidth as actual throughput. Which lead to throughput and bandwidth are used interchangeably in this thesis.

5

Architecture

In this system every ON is equipped with one or more resource(s). All the ONs have the same role in the neighborhood, meaning they all are equal and there is no leader, or ONs controlling other ONs. They all are expected to communicate with each other and build up their neighborhood RM. In figure 5.1 the RM buildup over time is represented for one ON.

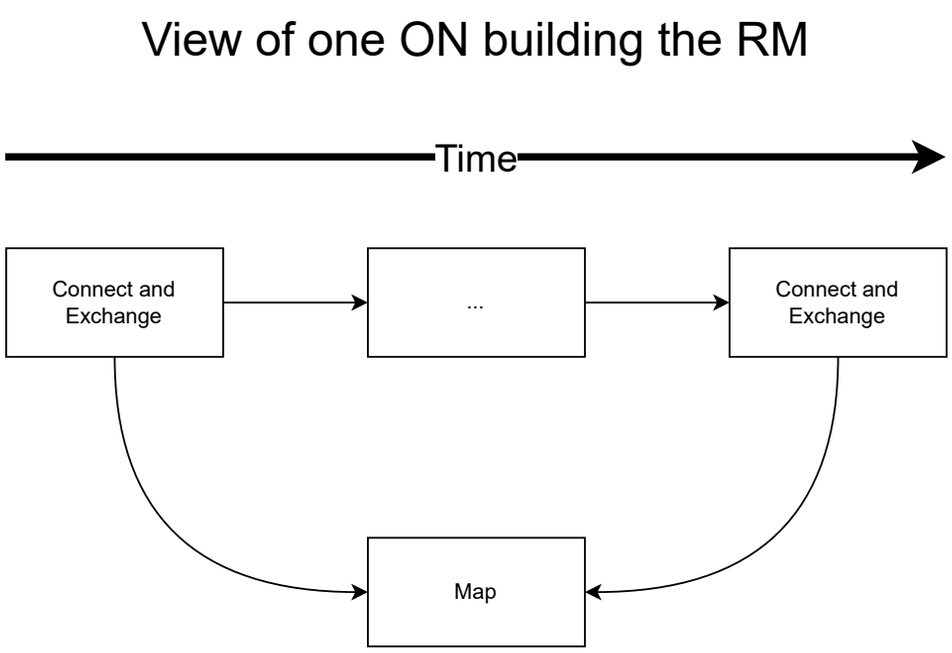


Table 5.1: Illustration of one ON building its RM over time

5.1 Neighbor

The definition of a **neighbor** to an ON is an ON which beacon signal is seen, and adhere to some defined criteria. In this case the criteria is: "The ability to create a peer to peer connection", assuming both are active and not occupied with communication with someone else. In other words connection between two ONs happen without hops.

5.2 Neighborhood

A **neighborhood** is a set of ONs which beacon are seen by all in the set, and that adhere to the criteria of being able to create a peer to peer connection. Each node has defined three neighborhoods: Past, Present, and Future. Each of them have their purpose but they define a neighbor in the same way as defined in 5.1.

5.2.1 Neighbor detection

When ONs wake up they broadcast a small message over long range Bluetooth, that other ONs then can pick up to know that the ON is on. In the rest of this paper this will be called a beacon.

Information in this neighborhood can be used to derive when another ON is likely to be awake and the probability of a successful connection.

Figure 5.1: The different neighborhoods

5.2.2 Past

The past neighborhood is a collection of the previously seen neighbors. This is built up over time and holds information on other ONs schedule as seen by one ON. Beacon signals, failed connection attempts and successful connections is tracked.

5.2.3 Present

On the start of each wakeup period the Present neighborhood is empty. The ON then immediately starts logging beacon signals observed, and connection attempts. Both failed and successful connections are logged. Before the ON goes to sleep the information in the current present neighborhood is inserted in to the past neighborhood.

5.2.4 Future

Future neighborhoods is prediction made using the information in the Past neighborhood. The information in the future neighborhood can be used to estimate the probability of another ON being reachable in the future, and if alteration in its own wakeup schedule needs to be made to reach that specific ON.

5.3 Resource map

The resource map consists of information about resources on other ONs. The information needs to indicate the type of the resource, estimated lifetime of the resource, and the cost of using the resource. Here costs will be measured in energy consumption for both the consumer and provider of the resource.

5.4 Map vector

The map vector is an array of numbers. Each index in the vector indicates the version number of the resource description last seen for the ON corresponding to that index. If the value equals to zero this means that the resource description has never been seen. A higher value indicates a later update of that description.

5.5 Resource spreading

When two ONs have created a peer to peer connection, they need a structure to send data, determining what they send and when they send it. This paper will test three alternatives:

Method number 1) Only send the local resource(s)

Method number 2) Send the entire map view

Method number 3) First send the version vector of the current map, and then receive the missing information.

Number one and two is a push system, the information is just pushed to the connecting ON without further negotiation. The third is a pull system, where the connecting ON first sends a vector of its map. The vector is then compared to the connected ONs vector. The connecting ON then gets sent back what was missing from its own map and present in the connected ONs map. If the connected ON notices that the connecting ON has updates not seen yet the process is repeated the other way around. The granularity of each position in the vector is a single value.

5.5.1 Send local resource(s) with push

In the resource push configuration ONs create peer to peer connections with other ONs, when the connection is setup the ON being connected to immediately start sending the information on its own resource to the connecting ON. When it is finished the ON that initialized the connection sends its resource information back, and then end the connection.

5.5.2 Send map view with push

The send map with a push configuration is similar to the "send resource with push" but instead of just sending information about the resource present locally, it also sends the information it has collected from other ONs. This can make it possible for ONs that do not have overlapping schedules to be able to get information about each other.

5.5.3 Send local resource(s) with push pull

If resource information is likely to be updated after deployment a push/pull system can be more efficient and reduce overhead on the network. Instead of just pushing the complete resource information it first pushes a version vector and then the receiving ON can request the updates it is interested in. This may or may not be more efficient, if the resource information have many fields and each field consists of a notable size of data this can be more efficient. Most likely the information does not consist of mb's or kb's of data, but more likely bytes. Therefore the time used sending the vector could rather be used sending the actual information.

5.5.4 Send map view with push pull

Instead of sending the complete map view an ON has collected every time it connects to a new ON, it can be beneficial to send a version vector first, and by doing so requesting the information correlating to the positions missing in the vector. This approach is similar to what they do in the Eyo paper [16]. This can get the benefit of not needing to overlap with an ON to receive information about it like in 5.5.2, but also reduce the network overhead because the entire map view is not sent every time.

6 Methodology

The methodology used for any research is highly dependent on the nature of that specific research question. Choosing the right methodology is fundamental for the methods used to elaborate a scientific paper [12] We often split the methodology in two main categories, Qualitative and Quantitative, both have its pros and cons. Choosing the methodology impacts every aspect of the research, from design to experiment and result analysis.

6.1 Architecture Exploration

Exploration of the different resource spreading mechanisms defined in 5.5, and the effect of changes in the different parameters described in 4.3 can be done in different ways.

One solution is to create an artifact and load software in real ONs and connect multiple measuring sensors and running test on a real system. This would give pretty accurate results, but have the disadvantage of being costly in labor, money, and time.

Solution number two is to create an emulation that run on one machine and try to mimic real ONs. This would be cheaper than an artifact but still would require a lot of time to create the emulation. And the time to actually run the emulations would still be considerable.

The final solution is to create a simulation of what is happening. This require a much smaller code base then the emulation and will require much less time to actually run. The disadvantage is that the result is not as accurate as we would get with an artifact. Considering that the amount of parameters that can be tested in a shorter amount of time is higher, the results combined will still be able to tell something about how the architecture will behave in the real world.

6.2 Quantitative method

Quantitative research is based on objective data collection where the researcher is external to the result. The data gathering is often well structured and systematically executed.

The implementation described in chapter 9 is of a simulation. A simulation is good for researching complex questions in a compressed time frame, but have the disadvantage of often requiring deeper knowledge of the specific topic at hand [12].

By creating a simulation from scratch it will be possible to collect a huge amount of data points in a relatively short amount of time.

After the data points is collected, they will be analysed and represented using Python3. The key aspects found in the data will be presented in a well documented manner.

6.3 Reliability

The reliability of a study is recognized as high if a researcher at a later point in time can recreate the experiments and achieve the same results. This tells something about how accurate the results are. The fact that my results are based on a simulation tells that if the code does what is described in chapter 7, another implementation based on the same design would reach the same results.

6.4 Validity

Validity tries to say something about how representative the result is. Given small variation in results based on random factors each test scenario is ran five times with the same parameters. The average result is then given in addition to the standard deviation. The variations can be due to witch neighbor an ON decides to connect to first.

7

Simulator Design

The simulator described in this chapter is custom made. The overall structure of the simulation is a table, where each row represents an ON. Each column is given a time representation.

7.1 Overall Structure

The simulation is setup as a table where each row represents an ON and each column represents a step in time for each ON. The cells can only have one of two values: active or sleeping. If two ONs have cells in the same column set to active, means they are overlapping active and is able to start a connection unless one of them already is connected to another one.

7.2 Running the Simulation

The simulation moves through the table one column at a time. At each step one active ON is chosen at random. If no ONs are active, the simulation moves to the next step. If one is active, then another active ON that is not yet connected to anyone is chosen at random. If there is another ON also active, a connection is made. This is done until all connections possible are established. Then continuing to the next step.

7.2.1 Transferring of Resources

When a connection is found, the simulation checks if both the ONs are active for long enough to send the resource(s) one way. Taking in to consideration the latency, time to send the resource(s), and depending on the method also the time to send the vector. If there is not enough time we still assume they tried to send the resource and mark each node as busy until the first ON goes to sleep. If there was enough time the resource(s) is marked as sent and each ON is marked as busy for the time it is estimated to establish connection, send the resource, and depending on the method the vector as well. Then the same thing is done the other way around.

7.2.2 End of Simulation

The simulation run until everyone has received the resources from all the other ONs, checking for it after each step. When the simulation finishes, the number of steps needed for everyone to receive information about every resource in the neighborhood is returned. If the simulation uses too many steps a negative one value is returned.

7.3 Design issues

Knowing when the simulation finishes was something that was not as trivial as it seams. Testing if everyone has received all resources is easy enough. But since the simulation marks the resources as sent immediately, we cannot return until all ONs are out of their busy state. Just continuing the simulation can mean other ONs get in the busy state again, and then possibly ending in a never ending loop, or skewing the results. The solution was to return the last step of the longest busy ON when the simulation notice that all ONs have received all resources.

8 Limitations

This chapter will look at some of the shortcomings of this paper. This paper only looks at time each ON is awake, and uses that as a basis for energy consummation. The findings here do not consider other operations the ONs want to do. In that regard, here are some other parameters that could be useful to look at:

8.1 CPU-usage

The ONs do not only need to send their resources, they also simultaneously need to read sensor data, communicate with other ONs and clients, process incoming data form the sensors, calculate how to behave in the future, just to mention a few. Therefore CPU usage could impact the up time or energy usage of the CPU needed to actually have time to do all of these tasks in addition to the resource spreading.

8.2 Disk-usage

Disk is another aspect of the total system. While the resource map is being created some disk usage is to be expected. Not only need the resource information to be persistently stored, but also the neighborhoods described in 5.2.

We are used to think that in our desktops, laptops and even our phones this is not a problem since they have write speeds of up to 600 MB/s with ssd or up to 2000 MB/s with NVMe [1]. But depending on what type of ON we may need to use SD cards and then the read/write speed could be as little as 2 MB/s but more realistically 6 MB/s or 10 MB/s, witch most modern SD cards support [2].

8.3 RAM-usage

RAM usage could also be a factor. But this is connected to the disk usage, how often do the ONs write to disk? Is everything stored in RAM until it goes to sleep, or do they write to persistent storage right away? These two questions could be another aspect that would be interesting to look at.

8.4 Network bandwidth

Network bandwidth in this paper is only based on LoRa technology. But we assume a LoRa gateway on each ON, and assume that they do not have to share the bandwidth. In the real world the LoRa frequency is limited to 125 KHz or 500 KHz for down link and 500 Khz for uplink [3]. Specification requires devices that use the spectrum to be on a duty cycle so that the network bandwidth can be shared. This will most likely have an impact on the results, but this is just what is available today, in the future the network speed will probably increase and the sharing a non issue.

8.5 Collisions

For the experiments conducted here I have not looked at collisions when establishing connections. In reality it could happen that two ONs try to connect to the same neighbor. Resulting in that one ON waits on a response it will never get. This is also something that can affect the result, but to catch this

1. NVMe vs SSD: Speed, Storage Mistakes to Avoid :`https://www.promax.com/blog/nvme-vs-ssd-speed-storage-mistakes-to-avoid` (accessed: 27.04.2022)
2. A Guide to SD Card Speed & Other Specs: `https://www.focuscamera.com/wavelength/a-guide-to-sd-card-speed-other-specs` (accessed: 27.04.2022)
3. What are LoRa® and LoRaWAN®?: `https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/` (accessed: 27.04.2022)

an emulation or artifact must be made. Or first conduct experiments on how often this happen and then incorporating it in to a simulation.

9

Simulation Implementation

To evaluate the different approaches outlined in 5.5 simulation was chosen. The implementation of the simulation is done in the programming language C. C was chosen for its speed. Due to the relative simplicity of the code, a slower simpler language was less desirable. The simulation setup and result collection is done in Python3. Python3 was chosen here for its simplicity of working with data and json. Each experiment is executed 5 times in separate sub processes so that the CPU of the machine running the simulation is utilized better.

9.1 Setup

The setup is done i python3. The input for the setup is "spreading-mechanism" (SLR, SLM or SLMV), step size, latency steps, and directory where the results should be stored. The values for wakeup, uptime, and spread is placed in lists. Both wakeup and uptime are in seconds, these values are translated in to steps based on the step size given before the simulation starts. Each simulation is to test for a given wakeup, uptime, spread, throughput, and latency. The experiments to run is then set up to test for every permutation of wakeup, uptime, and spread.

9.2 Python3-code

The python3 code starts by taking the parameter and translating the wakeup, uptime and max-simulation steps to the desired step size specified by the step size parameter.

Every permutation of wakeup, uptime, and spread is found and put in a list. After the values for each simulation that should be done is found and placed in a list, the simulations can start. The list is iterated through, every permutation in the list is ran for every ON count. Then every permutation with an ON count is ran five times in different processes. After each of the five processes is finished, the result is collected and written to a temporary result file. When every test for every permutation is done, all the results are written to a final result file in json format.

9.3 C-code

The C program does the actual allocation and execution for each simulation scenario. It takes in the values for a specific setup scenario: number of ONs, max-steps, wakeup, uptime, spread, transfer time, vector transfer time, latency, and exchange type.

9.3.1 Initialization

The initialization of a simulation scenario starts by allocating the memory needed, and then fills in the values for the simulation struct. It saves the right "check and exchange" function to use in a variable based on the simulation type. The simulation struct holds an array of ON structs that are also allocated.

The initialization process also fills the behavior bitmap of each node. How the behavior is structured is based on the spread type indicated. With no spread they all have the same, with random spread they each gets a random starting position from beginning of wakeup to (wakeup - uptime). For uniform spread the start step increase for every ON, the increase in start step from the previous to the next ON is calculated by the equation 9.1.

$$S = \begin{cases} \frac{W-U}{Nr}, & \text{if } \frac{W-U}{Nr} < (U-T) \\ (U-T), & \text{otherwise} \end{cases} \qquad O = \begin{cases} 1, & \text{if } S > 0 \\ \frac{Nr}{(W-U)}, & \text{otherwise} \end{cases}$$

Figure 9.1: Equation for the separation of start step of each node for uniform spread. S is the increase in start step for each ON, O is how many ON should have the same schedule, T = minimum overlap to previous ON, Nr = Number of ONs, W = wakeup, and U = uptime

9.3.2 ON In the simulation

An ON in the simulation is just a struct. The values it holds are: ID, behavior bitmap, connections, resources in map, and last-connection. The ID is just a number and range from zero to the number of ONs in the simulation. The behavior bitmap holds the simulation row for that specific ON. Connections is the vector list of ONs that the ON has received information about. Resources in map is a number indicating the number of resource descriptions in the connections list. Last-connection is the last step the node is occupied to.

9.3.3 Simulation execution

When the simulation starts the simulation steps through each column in the table, from zero to max simulation step. At each step the simulation checks the behavior bitmap of each ON in pseudo random ordering.

When an ON's bitmap indicates that it is active, and that the last-connection is lower then the current step, the simulation tries to find another ON that also is indicated as active at the same step. The other ON also needs to have a lower value for last-connection than the current step, if not another ON is chosen.

For SLM and SLMV the simulation first tries to find an ON match out of the active ONs with last-connection lower then the current step, that it has not yet had a connection to. If there are no ONs active that it has not yet connected to, one active ON is just chosen at random. This is done because they might have received more information since last time.

When a match is found, the ONs are checked if they are awake long enough. They need to be active long enough for the latency and transfer time of the resource(s). If the exchange method is SLMV, the time to send the resource vector is also taken into consideration.

If they are not awake long enough they both will be marked as occupied until the first of them goes to sleep. This will simulate a shutdown of one ON in the middle of a transfer. If they both are long enough awake the resource(s) will be marked as sent. In addition both ONs will be marked as occupied for the time it would have taken to transfer the resource/s. After a send is completed one way, the transfer stages is repeated the other way around.

When all the ON have received all the descriptions from all the other ONs, then the simulation ends, and the step counter is returned. If the simulation did not finish before the step counter hit the max number of steps, the simulation is aborted and a minus one value is returned. A minus one value represent a did not finish (dnf).

9.4 Check and Exchange

One function for each exchange type are created as the check and exchange function. The exchange type specified decide which of these function to use for the simulation. All the other aspects of the simulation is the same for all the exchange types. The exchange method is built up equally for all types, but have small differences.

First they all find the number of steps needed to send the resources. SLR takes the transfer time and add the latency, SLM use (transfer time * number of descriptions in the map of the sender) and add the latency. SLMV find (transfer time * descriptions in the map of the sender, that is not in the receiver's map) then add latency and vector transfer time.

After finding the steps required, then check if each of the ON are active for at least that many steps. If they are not active that long, we find how many steps until the ON that goes to sleep first, then set both as occupied for so many steps, and returning 0. If they are active long enough, the resource description(s) is marked as reserved in the "Connections" list, and mark each of the ONs as occupied for the amount of steps, and return a positive value.

9.4.1 Pseudo Random Order

To choose the ordering for iteration through the ONs in the simulation, a method for generating a pseudo random ordering for the slice of the indexes in the ON array. The algorithm presented here is inspired by a function in the GO

1.17.7 standard library [1]. My implementation can be seen in listing 9.1.

Listing 9.1: My C implementation for the Perm function found in GO 1.17.7 standard library.

```
int *psudo_perm(int n) {
    int *arr = (int*)malloc(sizeof(int)*n);
    int j;
    for (int i = 0; i < n; i++){
            j = rand() % (i+1);
            arr[i] = arr[j];
            arr[j] = i;
    }
    return arr;
}
```

9.5 Optimization

The choice of using the C programming language in it of itself is actually an optimization decision. There would be no problem just using Python3 for the simulation execution too. But C is a lot faster than Python3. The only downside is that C is objectively a harder language to write in.

A bitmap was created to hold the table rows, this reduces the size of the table allot. Now each cell only uses one bit of memory. Instead of four bytes like an Int array would use. This is perfect use of a bitmap since each cell only can be set to active or sleeping.

To reduce the memory needed to hold each simulation table, the length is set to the wakeup. To find the position in the table, the current step the simulation is on modulo wakeup is done. This can be done because each node has the same wakeup spacing, and do not change their wakeup schedules for the whole simulation period.

1. GO standard library Perm function documentation `https://pkg.go.dev/math/rand@go1.17.7#Perm`

9.6 Network Characteristics

The step size is used to infer the throughput of the network. Since each resource is estimated to be 20 bytes or 160 bits, and we set the steps to send a resource statically to one, when we then change the time value each step represents, we can simulate different throughputs. For example if we set each step to represent one second, we have effectively a transfer speed of 160 bit/s. If each step is half a second we have effectively 320 bit/s, this is done by setting the step size parameter to two. Set it to 1/156 part of a second, and we get $160 * 156$, or about 25 kbit/s, achieved by setting the step size parameter to 156.

10 Evaluation

This chapter will describe the experiments and evaluation conducted.

10.1 Metric

The metric used in the evaluation is; steps to spread resource description from all to all ONs. The mean value for all runs that finished are stored combined with the standard deviation. The steps are then given a time value in seconds, in the graphs the results are presented in minutes. The values seen in the graphs are therefore the average total time we can expect the neighborhood to use before the RM is built at every ON, for that specific configuration of parameters and ON count.

10.2 Software environment

OS: Ubuntu-server 20.04.1 LTS
Python: 3.9.5
C-compiler: GCC 9.4.0

10.3 Hardware environment

CPU: Intel Core i7-3820 @3.60 GHz
RAM: 2 x 8G DDR3 @1600 MHz

10.4 Experiment Design

All possible permutations of the parameters are conducted. Some of the permutations will result in the same behavior, they are combined and ran one time. Each permutation is executed five times. This is done because variations in the random selection of who to connect to first, can lead to variation in the final result.

After the results are in, they are plotted in graphs to compare different factors. Some of these graphs will be chosen in chapter 11 to show the key findings.

10.5 Parameter values

The parameters have been described in section 4.3, here are the values for each parameter I want to conduct experiments on:

- Wakeup: 10 min, 20 min, 40 min, 1 hour, 3 hour.
- Uptime: infinite, 10 min, 5 min, 2 min, and 1 min.
- Spread: No spread, random, and uniform.
- Latency: 0 and 500ms.
- Throughput: 320 byte/s and 25 kbyte/s (assuming the ability to utilize the total bandwidth).
- Exchange type: SLR, SLM, and SLMV.
- Number of ONs: 2, 4, 8, 16, 32, 64, 100, 200, and 400.

11

Results

In this chapter the results from the simulation are described. The graphs presented are bar charts. The height of each bar is the mean value of the runs that did finish within the max time frame, out of the five times each configuration was ran. The black line is the standard deviation, if the result from all the runs that finished are equal, there is no standard deviation. Each ON count is given a color representation, this color is the same for all graphs.

A table is placed under the graphs that display the exact values. Each column represent the bar directly above. The rows represent a specific ON count, following the same color code as in the bar graph. The cells is tables is color coded after the number runs that did not finish (dnf). A dnf of five means that specific configuration did not finish for any of the five runs. A dnf of zero means all finished. The colors move from white for dnf zero to red for dnf five. The darker the color of the cell, the fewer runs managed to finished within the maximum time frame of the simulation.

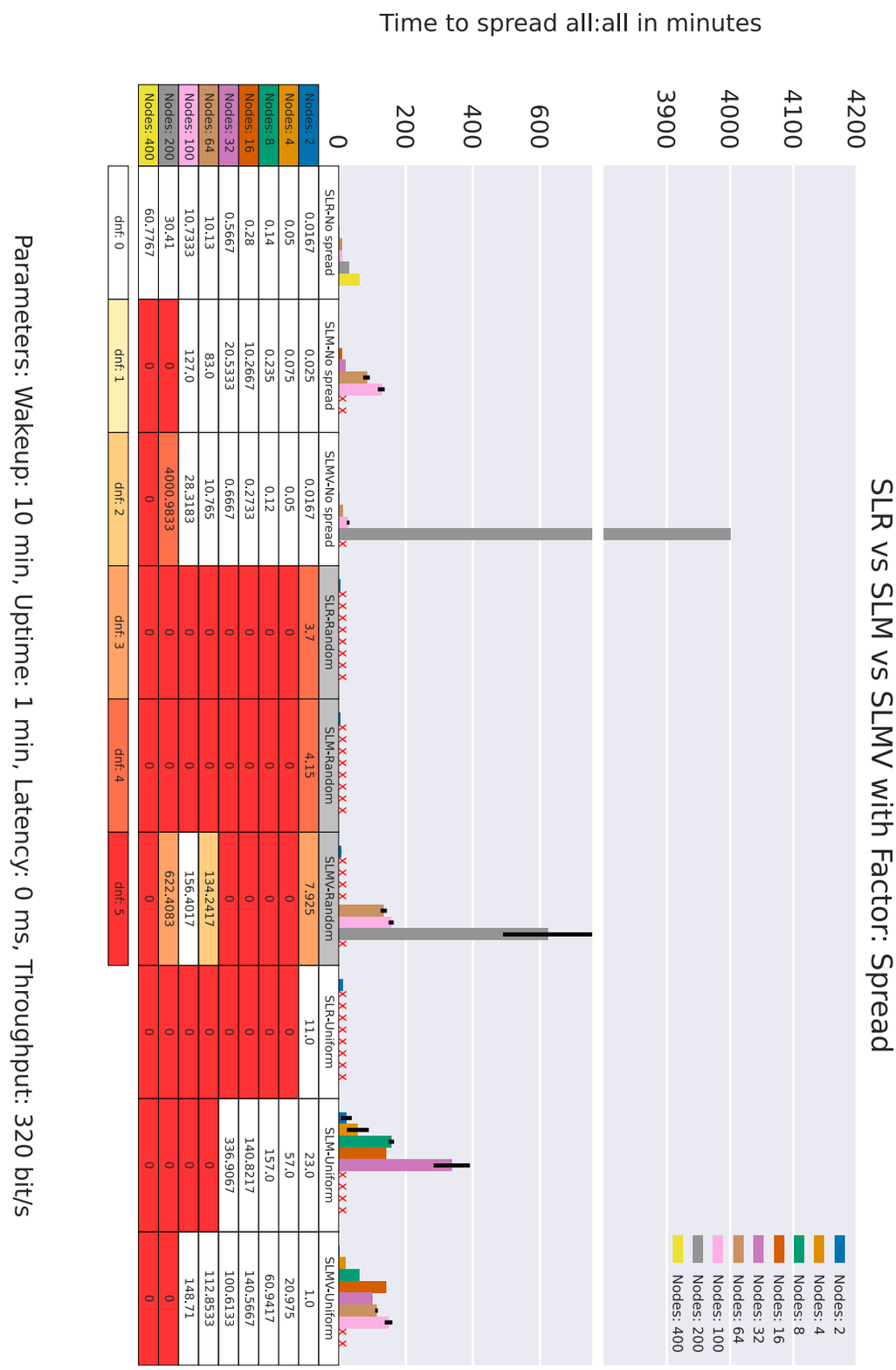
11.1 Results Throughput 320 bit/s and Latency 0 ms

First we will look at the result for the experiments with 320 bit/s throughput and zero ms latency. Some selected graphs have been chosen to highlight the

effect from each parameter: Spread, Uptime and wakeup. At the end the results will be combined to four tables that can be used to estimate the uptime needed based on the wakeup and spread, to reach one of the desired max converging time targets outlined in section 4.2.

11.1.1 SLR vs SLM vs SLMV with different spread

This section will focus at the different exchange methods described in section 5.5. A selection of graphs is chosen, each with a different wakeup. The two first graphs will just be looked at individually. The next three will also be viewed individually first, but they will also be compared to each other. This can be done because the only parameter changed between them is the wakeup.



	SLR-No spread	SLM-No spread	SLMV-No spread	SLR-Random	SLM-Random	SLMV-Random	SLR-Uniform	SLM-Uniform	SLMV-Uniform
Nodes: 2	0.0167	0.025	0.0167	3.7	4.15	7.925	11.0	23.0	1.0
Nodes: 4	0.05	0.075	0.05	0	0	0	0	57.0	20.975
Nodes: 8	0.14	0.235	0.12	0	0	0	0	157.0	60.9417
Nodes: 16	0.28	10.2667	0.2733	0	0	0	0	140.8217	140.5667
Nodes: 32	0.5667	20.5333	0.6667	0	0	0	0	336.9067	100.6133
Nodes: 64	10.13	83.0	10.765	0	0	134.2417	0	0	112.8533
Nodes: 100	10.7333	127.0	28.3183	0	0	156.4017	0	0	148.71
Nodes: 200	30.41	0	4000.9833	0	0	622.4083	0	0	0
Nodes: 400	60.7767	0	0	0	0	0	0	0	0

Figure 11.1: Comparing exchange types with Factor spread, Wakeup 10 minutes, Uptime 1 minute

Graph 11.1 shows the effect of the spread. With the lowest wakeup at ten minutes, and the lowest uptime at one minute. If we first focus at the SLR with no spread in figure 11.1 it is very efficient with a steady growth that show near doubling in time when the ON count is doubled, up to 32 ONs. We then see a jump at 64 ONs with a twenty times increase, from 0.57 to 10.13 minutes. From 64 ONs a steeper increase is shown. This indicates that at 64 the one uptime period was not enough to spread from all to all. That is why we see it jumps to just over 10 minutes, which is the wakeup period. For 100 ONs the second wakeup period was enough, but for 200 ONs four wakeup periods was necessary, and seven periods for 400 ONs.

For the SLM with no spread in graph 11.1 we can see a similar growth as for SLR with no spread for the lower ON counts, but the jump happens sooner at just 16 ONs. The SLM does not even finish with 200, and 400 ONs. The reason is probably because the amount of time it takes to send information about 200 resources at 320 bit/s is $200 * 0.5s = 100s$. This in turn means that they go to sleep before the transfer has time to finish transferring, because of the 60 minute uptime.

The same can be said about the SLMV method with no spread, as we did for SLR and SLM in graph 11.1, but we see the first jump is at 64 ONs. We also can see another jump at 200 ONs, but it only finishes one out of the five times it was ran. This is probably due to the fact that for the one time it finished, they was able to spread out the sending enough. Meaning no ON needed to ask for information about more than $\frac{60}{0.5} = 120$ ONs.

For SLR with random spread in graph 11.1, it fails on almost every ON count. It only finishes some of the time with 2 ONs. This is expected because the probability for overlap is low. We see the same failed result for SLM with random spread, but the reasoning is different. The reason for the lower ON counts is probably due to not overlapping. While for the higher ON counts (over 32) it is probably because of not enough time to send before sleep, just like with no spread.

SLMV with random spread in graph 11.1 is not much better than SLM, resulting in some failed runs for most of the ON count except with 64, 100, and 200 ONs. The reason for the failed attempts is probably for the same reason SLM didn't do so well with random spread.

The final spread type in graph 11.1 uniform shows that the SLR only finishes with two ONs, this is because this spread guarantee overlap with the ON with higher and lower ID. SLM with uniform spread shows more promise, and succeed for the lower ON counts up to 32. SLMV with uniform spread shows lower times than SLM, and for a higher ON count. Indicating that the idea of

reducing the network overhead has an effect. We still see the 200 and 400 ON count fail every time, probably because of trying to send too much information each transfer.

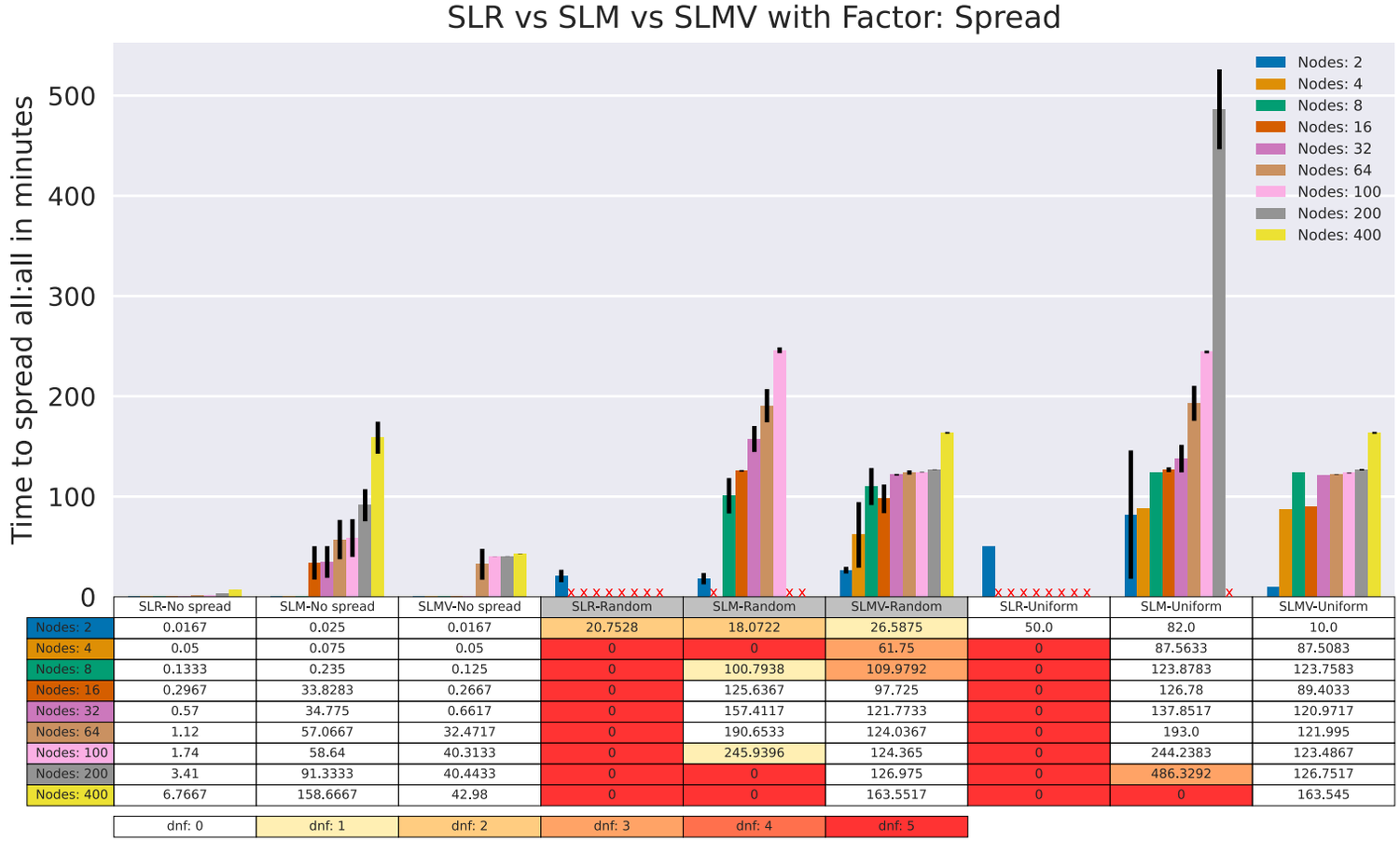
Combining the information in figure 11.1 we can say that this configuration when wakeup is set to ten minutes and time awake is one minute, the solution is ether to guarantee that everyone wake up at the same time, and use the SLR method. If we cannot guarantee that, the only consistent solution is to go with a uniform spread and use the SLMV approach, at least up to 100 ONs.

Figure 11.2: Comparing exchange types with Factor spread, Wakeup 20 minutes, Uptime 5 minutes

In figure 11.2 we can see that all the ON counts finish for all three exchange methods with no spread. SLR is considerably faster with a steady growth up until 200 ONs. SLR with 400 ONs uses over 6 times the amount of time 200 ONs did, up to just over 20 minutes. Indicating that the time needed for converging with 200 ONs, was more than five minutes. SLM seams to have the jump at 16 , then a steady growth up until 200, and then a bigger jump to 400 ONs. SLMV jumps at 64 ONs, and has faster times than SLM. Again we can see at with point the different exchange methods needed to use more than just one uptime period.

With random spread in figure 11.2 SLR does not perform well, as expected. SLM finishes three out of five runs when the ON count is two, four, eight, and 64, indicated by the did not finish (dnf) representation in the table. SLMV seams to finish more constantly for ON count eight and up. SLMV has a big jump at 400 ONs, that also shows a big standard deviation. Here we clearly see the effect that reducing the network overhead reduce the time each ON needs to overlap to successfully transfer the descriptions.

Uniform spread seams to have roughly the same result as random spread for SLM, but more consistently up to 64 ONs. SLMV with uniform spread has some higher and some lower times than with random spread, but also here we see that all of the runs converge successfully. The result shows that structuring the ONs to ensure overlap helps the ONs to be able to spread from all to all consistently. For SLMV the lower ON counts has the best effect of uniform spread, while the higher has the best effect for SLM. This indicate that for SLMV it helped the lower counts to ensure an overlapping trace. For SLM it helps the higher counts to overlap equally so that there is enough overlap to send the descriptions.



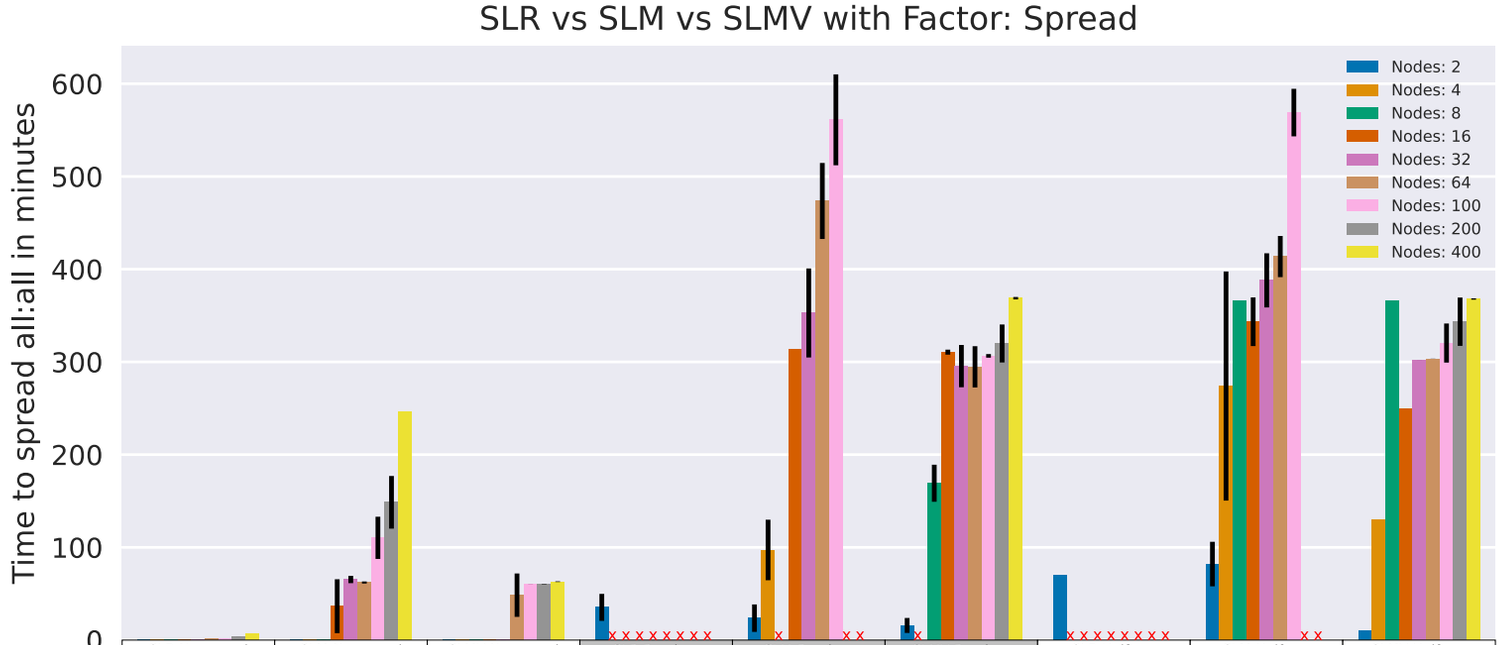
	SLR-No spread	SLM-No spread	SLMV-No spread	SLR-Random	SLM-Random	SLMV-Random	SLR-Uniform	SLM-Uniform	SLMV-Uniform
Nodes: 2	0.0167	0.025	0.0167	20.7528	18.0722	26.5875	50.0	82.0	10.0
Nodes: 4	0.05	0.075	0.05	0	0	61.75	0	87.5633	87.5083
Nodes: 8	0.1333	0.235	0.125	0	100.7938	109.9792	0	123.8783	123.7583
Nodes: 16	0.2967	33.8283	0.2667	0	125.6367	97.725	0	126.78	89.4033
Nodes: 32	0.57	34.775	0.6617	0	157.4117	121.7733	0	137.8517	120.9717
Nodes: 64	1.12	57.0667	32.4717	0	190.6533	124.0367	0	193.0	121.995
Nodes: 100	1.74	58.64	40.3133	0	245.9396	124.365	0	244.2383	123.4867
Nodes: 200	3.41	91.3333	40.4433	0	0	126.975	0	486.3292	126.7517
Nodes: 400	6.7667	158.6667	42.98	0	0	163.5517	0	0	163.545

dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Figure 11.3: Comparing exchange types with Factor spread, Wakeup 40 minutes, Uptime 10 minutes

Looking at graph 11.3 shows almost perfect growth for SLR with no spread, indicating that all finish spreading to all within the first uptime period. SLM with no spread has a jump at 16 ONs, and seam to grow much faster than the other methods. The SLMV with no spread grows slowly but have a jump at 64 ONs.

With random spread the time used is higher, and they do not finish for all of the runs. Uniform for SLM and SLMV seams to have roughly the same results as for random but more consultant, finishing more often and having a lower standard deviation.



	SLR-No spread	SLM-No spread	SLMV-No spread	SLR-Random	SLM-Random	SLMV-Random	SLR-Uniform	SLM-Uniform	SLMV-Uniform
Nodes: 2	0.0167	0.025	0.0167	35.2875	23.65	15.7833	70.0	82.0	10.0
Nodes: 4	0.05	0.075	0.05	0	97.1875	0	0	274.0	129.975
Nodes: 8	0.1333	0.215	0.12	0	0	169.1625	0	366.3767	366.2583
Nodes: 16	0.2867	36.53	0.2633	0	313.9167	310.5333	0	343.3267	249.4
Nodes: 32	0.5667	65.3333	0.6733	0	352.8042	295.48	0	387.9633	301.595
Nodes: 64	1.12	62.09	48.435	0	473.5867	294.7117	0	413.6467	303.44
Nodes: 100	1.73	110.2933	60.2683	0	561.1125	306.5283	0	568.9111	320.3617
Nodes: 200	3.4067	148.5967	60.5067	0	0	319.895	0	0	343.3817
Nodes: 400	6.7667	246.6667	62.9967	0	0	368.7167	0	0	367.8683

dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Parameters: Wakeup: 1 h, Uptime: 10 min, Latency: 0 ms, Throughput: 320 bit/s

Figure 11.4: Comparing exchange types with Factor spread, Wakeup 1 hour, Uptime 10 minutes

Moving from wakeup 40 minutes to wakeup 1 hour graph 11.4 has the same result as 11.3 for SLR with no spread. Again this is because the uptime is high enough to finish within the first uptime period. We also see same pattern but a slight increase in time for SLM and SLMV with no spread, we see that they jump to around 40 minutes when the wakeup is 40 minutes, and to 60 minutes when the wakeup is 60 minutes.

For random we see that the jump at each increase in node count is higher for wakeup 60 minutes, than with 40 minutes. The same can be said for uniform spread. This indicate that increasing the wakeup can increase the number of uptime cycles each ON need to finish.

Interestingly with uniform there is actually a higher value for eight ONs than 16 in both SLM and SLMV. We also see higher jumps moving up to 8 ONs, than we see moving from 16 ONs and up. This can indicate that up to 16 ONs they use more of the time to wait for opportunities to send than actually sending. When there is more ONs to fill the wakeup period there is also more activity.

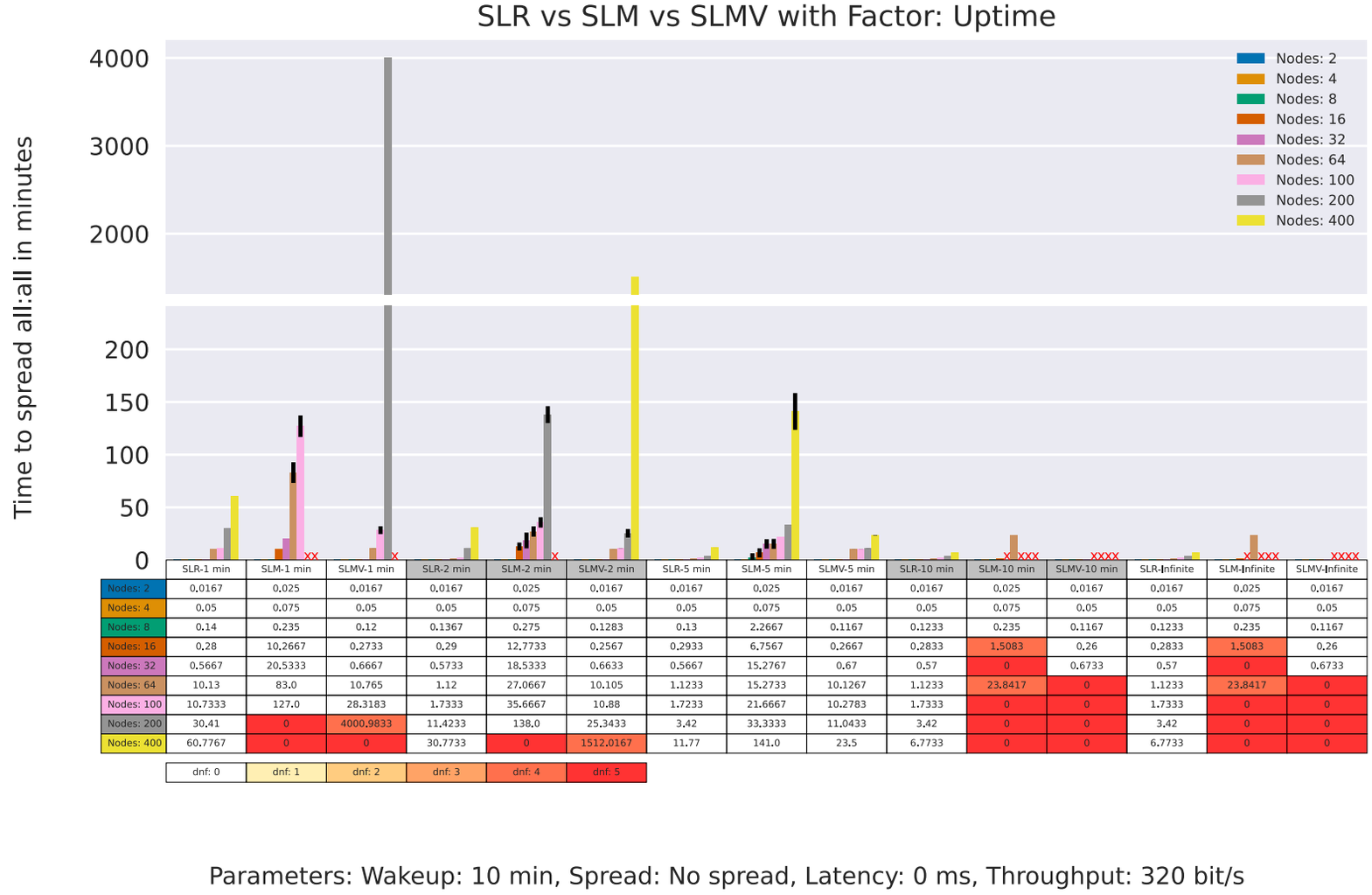
Figure 11.5: Comparing exchange types with Factor spread, Wakeup 3 hours, Uptime 10 minutes

Comparing figure 11.5 to figure 11.4 we see no difference for SLR with no spread, and the same pattern for SLM and SLMV with no spread, just with higher values. Random and uniform now use significantly more time. We can note that the y-axis moves in steps of 1000 in graph 11.5 instead of 100 like in does in 11.4. Interestingly for SLMV with uniform spread there is a decrease in time used from 23 ONs to 200 ONs. Probably due to the lower amount of spread. There is more overlap when the ON count is higher, and therefore using more of the time to send instead of waiting on an opportunity.

To conclude the effect of the spread we can say that it has a huge impact on the result. No spread shows the lowest time to converge for all methods, and especially for SLR. No spread is not always an option, and when we do not have it, the random spread is an alternative. This result shows that structuring the ONs a bit can help on how fast they can spread, and how many ONs are needed compared to random spread. This result also highlights that SLR are the better one with no spread. But the SLM or SLMV can help when the option of no spread is not present. SLMV are almost always better than SLM. Meaning that even though we use more time transferring the vector, the better use of transfer time makes it faster regardless.

11.1.2 SLR vs SLM vs SLMV with different uptime

This section will focus on the difference the uptime makes on the time to create the RM. We will also highlight some of the differences the wakeup have, and also noting the differences in exchange method. There will be selected two wakeup values for each of the spread types.



	SLR-1 min	SLM-1 min	SLMV-1 min	SLR-2 min	SLM-2 min	SLMV-2 min	SLR-5 min	SLM-5 min	SLMV-5 min	SLR-10 min	SLM-10 min	SLMV-10 min	SLR-Infinite	SLM-Infinite	SLMV-Infinite
Nodes: 2	0.0167	0.025	0.0167	0.0167	0.025	0.0167	0.0167	0.025	0.0167	0.0167	0.025	0.0167	0.0167	0.025	0.0167
Nodes: 4	0.05	0.075	0.05	0.05	0.075	0.05	0.05	0.075	0.05	0.05	0.075	0.05	0.05	0.075	0.05
Nodes: 8	0.14	0.235	0.12	0.1367	0.275	0.1283	0.13	2.2667	0.1167	0.1233	0.235	0.1167	0.1233	0.235	0.1167
Nodes: 16	0.28	10.2667	0.2733	0.29	12.7733	0.2567	0.2933	6.7567	0.2667	0.2833	1.5083	0.26	0.2833	1.5083	0.26
Nodes: 32	0.5667	20.5333	0.6667	0.5733	18.5333	0.6633	0.5667	15.2767	0.67	0.57	0	0.6733	0.57	0	0.6733
Nodes: 64	10.13	83.0	10.765	1.12	27.0667	10.105	1.1233	15.2733	10.1267	1.1233	23.8417	0	1.1233	23.8417	0
Nodes: 100	10.7333	127.0	28.3183	1.7333	35.6667	10.88	1.7233	21.6667	10.2783	1.7333	0	0	1.7333	0	0
Nodes: 200	30.41	0	4000.9833	11.4233	138.0	25.3433	3.42	33.3333	11.0433	3.42	0	0	3.42	0	0
Nodes: 400	60.7767	0	0	30.7733	0	1512.0167	11.77	141.0	23.5	6.7733	0	0	6.7733	0	0

dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Figure 11.6: Comparing exchange types with Factor Uptime, Wakeup 10 minutes, No spread

Looking at graph 11.6 the result for one minute uptime shows that SLR and SLMV actually is not that different up to 64 ONs. But at 100 ONs SLR outperform SLMV. It seams like SLMV has the jump earlier than SLR. SLM has it even earlier at just 16 ONs, and worse result for the lower ON counts as well.

Moving to uptime of two minutes, the SLR and SLMV result is close only up to 32 ONs. The result for SLMV is actually improved for 100, 200 and 400 ONs, but SLR has also improved.

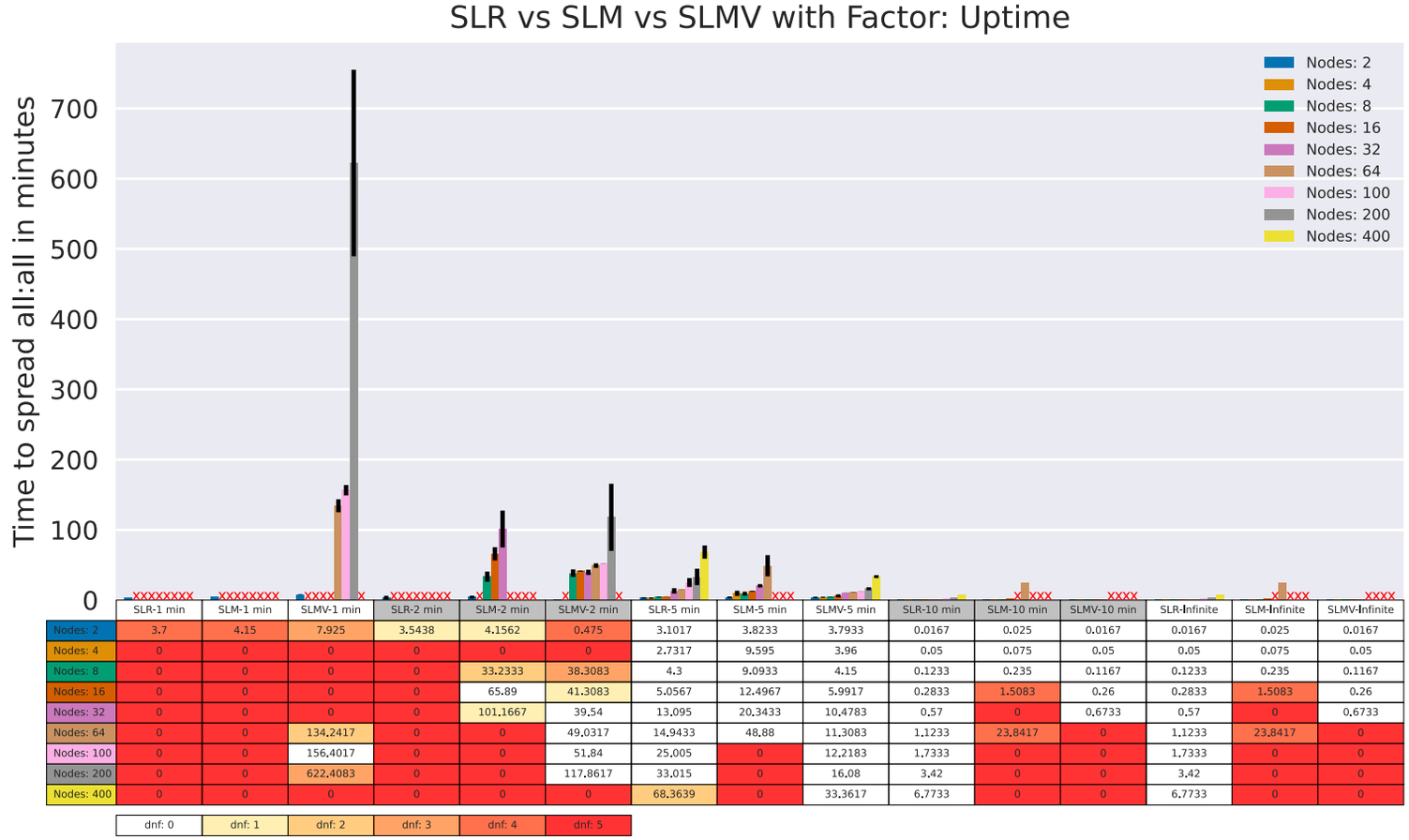
The same happens when the uptime increases to 5 minutes. All the methods improve, but the difference also increases. When we look at the uptime resulting in the ONs being always on, the SLM and SLMV actually fail often for the higher ON counts. This is actually unexpected, I believe this happens because when the ONs finish, they continue to connect to previously connected ONs in case they have an update. This results in a few ONs not being able to receive all the information, because the ON they want to connect to is always occupied.

Figure 11.7: Comparing exchange types with Factor Uptime, Wakeup 1 hour, No spread

Parameters: Wakeup: 1 h, Spread: No spread, Latency: 0 ms, Throughput: 320 bit/s

The graph 11.7 shows almost the same story as figure 11.6, in that the timings improve for SLR, SLM and SLMV when the uptime increases, but the difference also increases. A longer uptime intuitively means more ONs can transfer and receive in each uptime period, and therefore when the uptime increases, the jump happens on an higher ON count. Because the initial jump at one minute uptime happens much sooner for SLM, the improvements are not as big compared to SLMV and SLR. Meaning that SLR has the highest gain of increasing the uptime, followed by SLMV and then SLM.

Comparing graph 11.6 to graph 11.7 highlights that the difference between the exchange methods will increase as the wakeup increases. If we look at SLMV with one minute uptime, the one with 10 minute wakeup jumps to just over 10 minutes at 64 ONs, but the one with a wakeup 1 hour jumps to just over 60 minutes at 64 ONs. This shows us the point which the one uptime period is not enough to spread all the resource information. Also demonstrating that the difference will increase as the neighborhood becomes larger.



	SLR-1 min	SLM-1 min	SLMV-1 min	SLR-2 min	SLM-2 min	SLMV-2 min	SLR-5 min	SLM-5 min	SLMV-5 min	SLR-10 min	SLM-10 min	SLMV-10 min	SLR-Infinite	SLM-Infinite	SLMV-Infinite
Nodes: 2	3.7	4.15	7.925	3.5438	4.1562	0.475	3.1017	3.8233	3.7933	0.0167	0.025	0.0167	0.0167	0.025	0.0167
Nodes: 4	0	0	0	0	0	0	2.7317	9.595	3.96	0.05	0.075	0.05	0.05	0.075	0.05
Nodes: 8	0	0	0	0	33.2333	38.3083	4.3	9.0933	4.15	0.1233	0.235	0.1167	0.1233	0.235	0.1167
Nodes: 16	0	0	0	0	65.89	41.3083	5.0567	12.4967	5.9917	0.2833	1.5083	0.26	0.2833	1.5083	0.26
Nodes: 32	0	0	0	0	101.1667	39.54	13.095	20.3433	10.4783	0.57	0	0.6733	0.57	0	0.6733
Nodes: 64	0	0	134.2417	0	0	49.0317	14.9433	48.88	11.3083	1.1233	23.8417	0	1.1233	23.8417	0
Nodes: 100	0	0	156.4017	0	0	51.84	25.005	0	12.2183	1.7333	0	0	1.7333	0	0
Nodes: 200	0	0	622.4083	0	0	117.8617	33.015	0	16.08	3.42	0	0	3.42	0	0
Nodes: 400	0	0	0	0	0	0	68.3639	0	33.3617	6.7733	0	0	6.7733	0	0

dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Figure 11.8: Comparing exchange types with Factor Uptime, Wakeup 10 minutes, Random spread

In graph 11.8 interestingly SLR actually finishes most of the time when the uptime is five minutes. Meaning the ONs sleep 50% of the time. This shows that if all the ONs overlap at least one slot, that in this case is the time it takes to transfer one resource. The only time they will not be able to converge when sleeping 50% of the time is when one ON wake up at the start of the wakeup period, and one at the very end of the period. The probability of this can be calculated with equation 11.9. The result of which is a 0.25% chance with 16 ONs, 23.7% chance with 200 ONs, and 0.54% chance with 400 ONs. This explains the result shown for SLR with random spread well.

$$
\begin{aligned}
A &= \text{At least one wakeup at beginning.} \\
B &= \text{At least one wakeup at end.} \\
P(\overline{A} \text{ and } \overline{A}) &= 1 - (\frac{U-2}{U})^x \\
P(A) = P(B) &= 1 - (\frac{U-1}{U})^x \\
P(A \text{ and } B) &= 2(1 - (\frac{U-1}{U})^x) - (1 - (\frac{U-2}{U})^x)
\end{aligned}
$$

Figure 11.9: Equation for probability that at least one ON starts at beginning of the wakeup, and at least one starts at (wakup - uptime)[1], when they are asleep 50% of the time. Where U = uptime, and x is number of ONs.

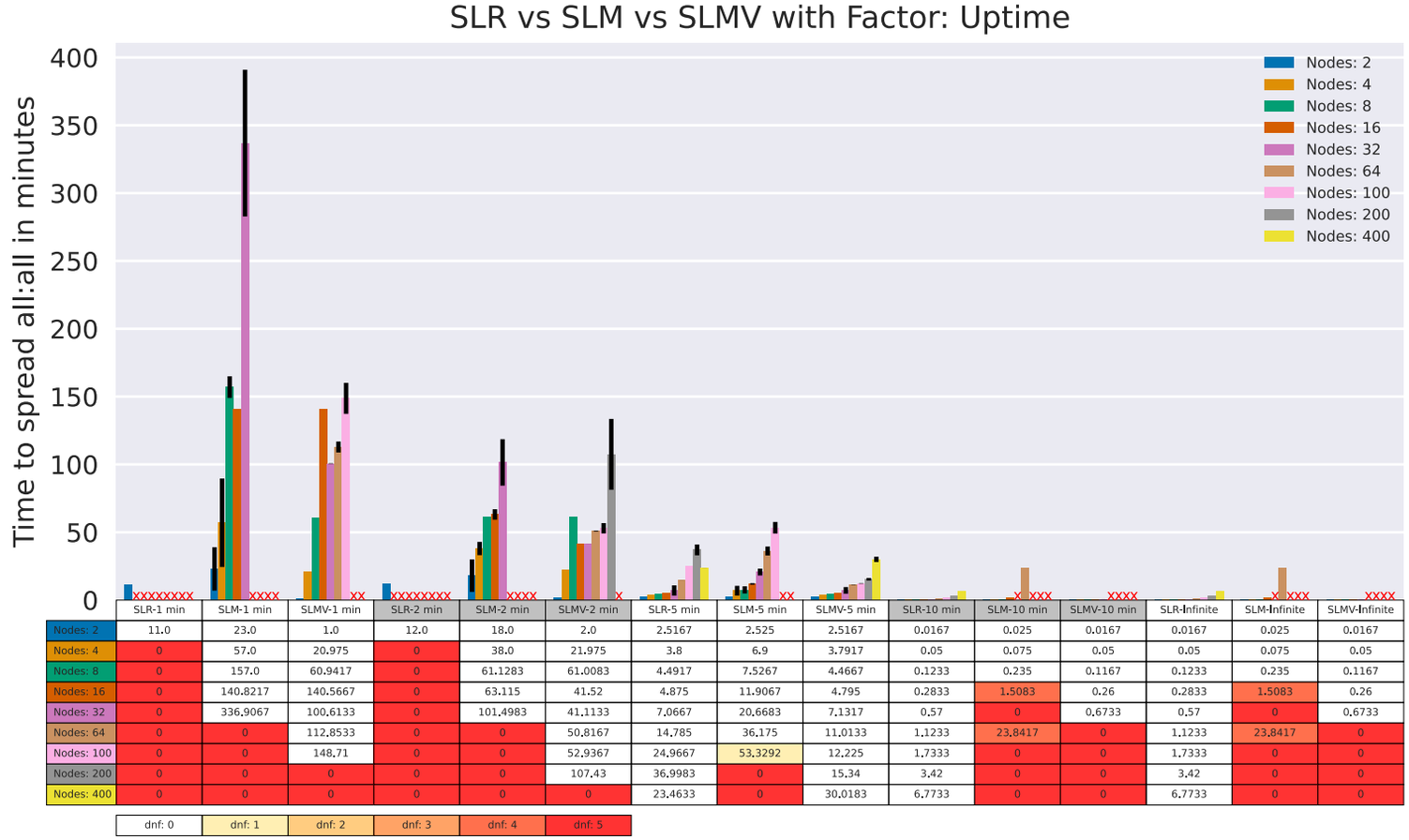
Graph 11.8 shows that SLMV is the only one that finishes for more than just 2 ONs for one minute. Moving to two minutes helps, but we actually must up to 5 minutes to see SLM and SLMV finish consistently.

1. Inspired by comment from Terry Moore on this question: `https://www.quora.com/Suppose-you-roll-five-dice-what-is-the-probability-that-at-least-one-1-and-at-least-one-6-will-appear` (accessed: 02.05.2022)

Figure 11.10: Comparing exchange types with Factor Uptime, Wakeup 1 hour, Random spread

In figure 11.10 the time between wakeup is increased compared to graph 11.8. The increased wakeup results in slower time to spread and more failed results. Even though we see many failed runs, there is a time decrease to spread when we move from five to ten minutes for the uptime. Increasing the uptime seams to help. The number of failed runs can be explained by: Even though all the ONs does not need to overlap with all the other ONs for SLM and SLMV to work, there still need to be a trace of overlapping ONs.

Increasing the wakeup seams to make the time to spread higher, as we would expect. Regardless increasing uptime of each ON helps on the time to build the RM. We also need to note that with a random spread and the uptime is low; ether the wakeup needs to happen often, or there need to be more ONs in the neighborhood.



	SLR-1 min	SLM-1 min	SLMV-1 min	SLR-2 min	SLM-2 min	SLMV-2 min	SLR-5 min	SLM-5 min	SLMV-5 min	SLR-10 min	SLM-10 min	SLMV-10 min	SLR-Infinite	SLM-Infinite	SLMV-Infinite
Nodes: 2	11.0	23.0	1.0	12.0	18.0	2.0	2.5167	2.525	2.5167	0.0167	0.025	0.0167	0.0167	0.025	0.0167
Nodes: 4	0	57.0	20.975	0	38.0	21.975	3.8	6.9	3.7917	0.05	0.075	0.05	0.05	0.075	0.05
Nodes: 8	0	157.0	60.9417	0	61.1283	61.0083	4.4917	7.5267	4.4667	0.1233	0.235	0.1167	0.1233	0.235	0.1167
Nodes: 16	0	140.8217	140.5667	0	63.115	41.52	4.875	11.9067	4.795	0.2833	1.5083	0.26	0.2833	1.5083	0.26
Nodes: 32	0	336.9067	100.6133	0	101.4983	41.1133	7.0667	20.6683	7.1317	0.57	0	0.6733	0.57	0	0.6733
Nodes: 64	0	0	112.8533	0	0	50.8167	14.785	36.175	11.0133	1.1233	23.8417	0	1.1233	23.8417	0
Nodes: 100	0	0	148.71	0	0	52.9367	24.9667	53.3292	12.225	1.7333	0	0	1.7333	0	0
Nodes: 200	0	0	0	0	0	107.43	36.9983	0	15.34	3.42	0	0	3.42	0	0
Nodes: 400	0	0	0	0	0	0	23.4633	0	30.0183	6.7733	0	0	6.7733	0	0

dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Figure 11.11: Comparing exchange types with Factor Uptime, Wakeup 10 minutes, Uniform spread

Figure 11.11 shows how the different uptime affects the time to spread from all to all with wakeup at ten minutes, and spread type uniform.

The SLR method does not finish for this spread type before the wakeup reaches five minutes. Then the ON is awake half of the time. At five minutes uptime the SLR actually finishes for all ON counts with a random spread. This is due to the spreading the uniform spread does. With half of the time awake, the only time it will fail is when we have a perfect spread. Uniform here just trys to have a perfect spread, but for most ON counts this does not happen (see explanation for uniform spread in 5.5).

For SLM uptime one minute only finishes constantly for ON count up to 32, with times of two hours and 20 minutes for 16 ONs, it uses five hours for 32 . The times decrease drastically when we increase the uptime to two minutes with a 55% decrease in time for 16 and a 67% decrease in time for 32 . In addition there is almost a 50% decrease in ON count 8, 4, and 2. Increasing uptime to five minutes, all but the two highest ON counts finishes. The run with 32 have an 70% decrease from uptime 2 minutes. Interestingly increasing to ten minutes uptime it starts to fail for the higher ON counts.

SLMV starts off well at the uptime at one minute finishing for all the ON counts except the last two. Increasing the uptime to two minutes does actually not have much effect on the ON counts two, four and eight. But the graph shows a drastic decrease for the higher counts, showing a decrease of more than 50%. Moving to uptime of five minutes decreases the time used by more than 50% from uptime of two minutes, for all except 200 ONs. The SLMV seams to have the same problem as SLM when they are always awake.

	SLR-1 min	SLM-1 min	SLMV-1 min	SLR-2 min	SLM-2 min	SLMV-2 min	SLR-3 min	SLM-5 min	SLMV-5 min	SLR-10 min	SLM-10 min	SLMV-10 min	SLR-Infinite	SLM-Infinite	SLMV-Infinite
Nodes: 2	61.0	181.0	1.0	62.0	158.0	2.0	65.0	149.0	5.0	70.0	82.0	10.0	0.0167	0.025	0.0167
Nodes: 4	0	361.0	120.975	0	290.0	121.975	0	342.9867	124.975	0	274.0	129.975	0.05	0.075	0.05
Nodes: 8	0	901.0	360.9417	0	614.0	361.9417	0	857.0	364.9417	0	366.3767	366.2583	0.1233	0.235	0.1167
Nodes: 16	0	1789.0	840.875	0	1682.0	841.875	0	843.6967	843.4417	0	343.3267	249.4	0.2833	1.5083	0.26
Nodes: 32	0	3829.0	1824.7567	0	3590.0	1837.7633	0	1095.3733	901.765	0	387.9633	301.595	0.57	0	0.6733
Nodes: 64	0	0	2724.675	0	7477.9567	2221.4583	0	1370.9467	782.5967	0	413.6467	303.44	1.1233	23.8417	0
Nodes: 100	0	0	1296.715	0	0	2473.0967	0	0	723.7717	0	568.9111	320.3617	1.7333	0	0
Nodes: 200	0	0	0	0	0	2593.5317	0	0	819.4817	0	0	343.3817	3.42	0	0
Nodes: 400	0	0	0	0	0	0	0	0	939.2033	0	0	367.8683	6.7733	0	0

Figure 11.12: Comparing exchange types with Factor Uptime, Wakeup 1 hour, Uniform spread

Comparing 11.12 to graph 11.11 shows that they follow the same trends, but the time to build the RM increases significantly. Again showing that the impact of wakeup is mostly the amount they sleep. Therefore by increasing the sleep will have an effect on how long the total amount of time the RM takes to build.

To conclude from these graphs we can say that when we increase the uptime, the time to spread from all to all decrease in a similar manner regardless of the wakeup. The spread also has a huge impact on the result. Showing that with no spread the effect of the different exchange methods do not make as much effect for lower ON counts. We see that the jump happens at higher ON counts when we increase the uptime. For random spread the uptime combined with the wakeup still needs to be likely to make an overlapping trace. When we force an overlapping trace with uniform spread more of the tests finishes.

11.1.3 SLR vs SLM vs SLMV with different wakeup

In this section there will be displayed graphs with the factor being different wakeup values, the graphs will have the uptime set to five minutes and one graph from each spread.

	SLR-10 min	SLM-10 min	SLMV-10 min	SLR-20 min	SLM-20 min	SLMV-20 min	SLR-40 min	SLM-40 min	SLMV-40 min	SLR-1 h	SLM-1 h	SLMV-1 h	SLR-3 h	SLM-3 h	SLMV-3 h
Nodes: 2	0,0167	0,025	0,0167	0,0167	0,025	0,0167	0,0167	0,025	0,0167	0,0167	0,025	0,0167	0,0167	0,025	0,0167
Nodes: 4	0,05	0,075	0,05	0,05	0,075	0,05	0,05	0,075	0,05	0,05	0,075	0,05	0,05	0,075	0,05
Nodes: 8	0,13	2,2667	0,1167	0,1367	0,195	0,1167	0,13	0,255	0,12	0,1233	0,2817	0,1167	0,1367	0,215	0,1167
Nodes: 16	0,2933	6,7567	0,2667	0,29	16,335	0,2667	0,29	32,8017	0,27	0,29	13,6067	0,27	0,2833	180,2667	0,2633
Nodes: 32	0,5667	15,2767	0,67	0,5667	25,27	0,6717	0,5667	40,5333	0,6783	0,57	60,5333	0,67	0,58	180,5333	0,68
Nodes: 64	1,1233	15,2733	10,1267	1,1267	37,0667	12,7217	1,1133	81,0667	24,7133	1,12	121,0667	60,1367	1,12	289,48	180,1367
Nodes: 100	1,7233	21,6667	10,2783	1,7267	41,9933	20,3117	1,7333	82,32	48,34	1,73	133,6667	60,285	1,7367	433,9933	180,2333
Nodes: 200	3,42	33,3333	11,0433	3,4133	67,3333	21,05	3,4233	139,3333	40,8533	3,4233	195,3333	60,8733	3,42	579,6667	181,03
Nodes: 400	11,77	141,0	23,5	21,77	297,0	43,47	41,7667	621,0	83,41	61,7733	917,0	123,4733	181,77	2669,0	363,5433

Figure 11.13: Comparing exchange types with Factor Wakeup, Uptime 5 minutes, No spread

Looking at graph 11.13 we can see that all uptime values seam to have closely the same result for ON count up to 8. At 16 ONs we can see that SLM has the first jump, indicating that at 16 ONs the SLM could not spread all the resource information within the first period. For SLMV this happens at 64 ONs, and for SLR not before 400 ONs. When they start to use multiple wakeup periods we see that the increase in time closely follows the wakeup. We can also see that because the jump happens sooner for SLM, the increase is steeper as the wakeup increases.

Figure 11.14: Comparing exchange types with Factor Wakeup, Uptime 5 minutes, Random spread

Graph 11.14 shows many failed runs. SLR only finishes when the sleep to awake ratio is 50% or lower. The SLMV method outperforms SLR in those situations. SLM and SLMV can work sometimes with increased wakeup periods, but increase in time to spread is steeper than we saw in figure 11.13. The minimum ON count also increases as the time between wakeup increases.

Figure 11.15: Comparing exchange types with Factor Wakeup, Uptime 5 minutes, Uniform spread

When there is a uniform spread as we see in figure 11.15, there are fewer failed attempts than with random spread. For the 10 minute wakeup period we see improvements for both SLR and SLM, SLMV is not as affected by the change in spreading method. SLMV actually seams to be less affected in the results for ON counts that finished for both spreading types, but what we can see is that the minimum number of ONs decrease. Meaning that SLMV uniform spread can expected to work with fewer ONs than with a random spread.

From the results for throughput 320 and latency 0 ms (see appendix for more graphs), tables for finding the minimum uptime required, and the expected maximum ON counts for each wakeup value is created. In the tables each row represents one of the different scenarios from 4.2. The tables can be used to meet a specific target when it comes to how fast we want to build the RM. Table 11.1 is for SLM and SLMV with random spread. Table 11.2 is for SLM and SLMV with uniform spread. SLR is not included because it only works for setups where they sleep for less than 50% of the time, or have no spread. For SLR with spread the graphs in the appendix combined with equation 11.9 need to be used to get an accurate answer.

Tables for no spread is not made, this is because they will not be as interesting or useful. SLR will finish regardless the uptime, and will be the same result for every uptime up until 32 . After that the wakeup can just be added to the total time. For SLM and SLMV with no spread, we see the same result for the lower ON counts. For the higher ON counts a higher uptime is needed. So it will not be as useful, or accurate to create a similar table for no spread. When no spread is to be used we have many more options for uptime, so the original graphs are better to use.

Smallest Uptime Needed In Minutes Random Spread

(Random)	Wakeup				
SLM	10 min	20 min	40 min	1 hour	3 hour
Lazy	5 (64)	5 (*64)	5 (*64)	10 (*100)	10 (**64-100)
Medium	5 (64)	5 (*64)	5 (*32)	10 (*100)	-
Eager	5 (32)	5 (*8) or 10 (64)	10 (4)	-	-
Panic	5 (8)	5 (*2) or 10(2)	-	-	-

(Random)	Wakeup				
SLMV	10 min	20 min	40 min	1 hour	3 hour
Lazy	1 (**200) or 2 (*200)	2 (*200)	2 (**400) or 10 (*400)	5 (64 to 400)	5 (just 400) or 10 (*64-400)
Medium	2 (*200)	2 (*200)	2 (**100) or 10 (*400)	5 (64 to 400)	-
Eager	2 (*64)	5 (*8) or 10 (200)	10 (*4)	-	-
Panic	5 (32)	10 (16)	5 (**2)	-	-

(x)	Maximum node count
*	nb. Some node counts dnf
**	nb. Most node counts dnf
Lazy	Just finishes
Medium	Within 2 days
Eager	Within 1 hour
Panic	Within 10 minutes

Table 11.1: Table of the smallest uptime needed for different wakeup values. With Spread: Random, Throughput: 320 bit/s and Latency: 0 ms

Smallest Uptime Needed In Minutes Uniform Spread

(Uniform)	Wakeup				
SLM	10 min	20 min	40 min	1 hour	3 hour
Lazy	2 (*32)	2 (*32)	2 (*64)	5 (*64)	10 (*200)
Medium	5 (100)	2 (*32)	2 (**16) or 5 (*64)	5 (*64)	10 (*8)
Eager	5 (64)	5 (8)	10 (4)	any (2)	any (2)
Panic	5 (32)	5 (2)	any (2)	any (2)	any (2)

(Uniform)	Wakeup				
SLMV	10 min	20 min	40 min	1 hour	3 hour
Lazy	1 (100) or 2 (200)	1 (100) or 2 (200)	1 (100) or 2 (200)	1 (100) or 2 (200)	1 (100) or 2 (200)
Medium	2 (200)	1 (100) or 2 (200)	1 (100) or 2 (200)	1 (100) or 2 (200)	1 (16)
Eager	2 (100)	1 (4) or 5 (32)	1 (4)	any (2)	any (2)
Panic	5 (32)	1 (4) or 10 (32)	any (2)	any (2)	any (2)

(x)	Maximum node count
*	nb. Some node counts dnf
**	nb. Most node counts dnf
Lazy	Just finishes
Medium	Within 2 days
Eager	Within 1 hour
Panic	Within 10 minutes

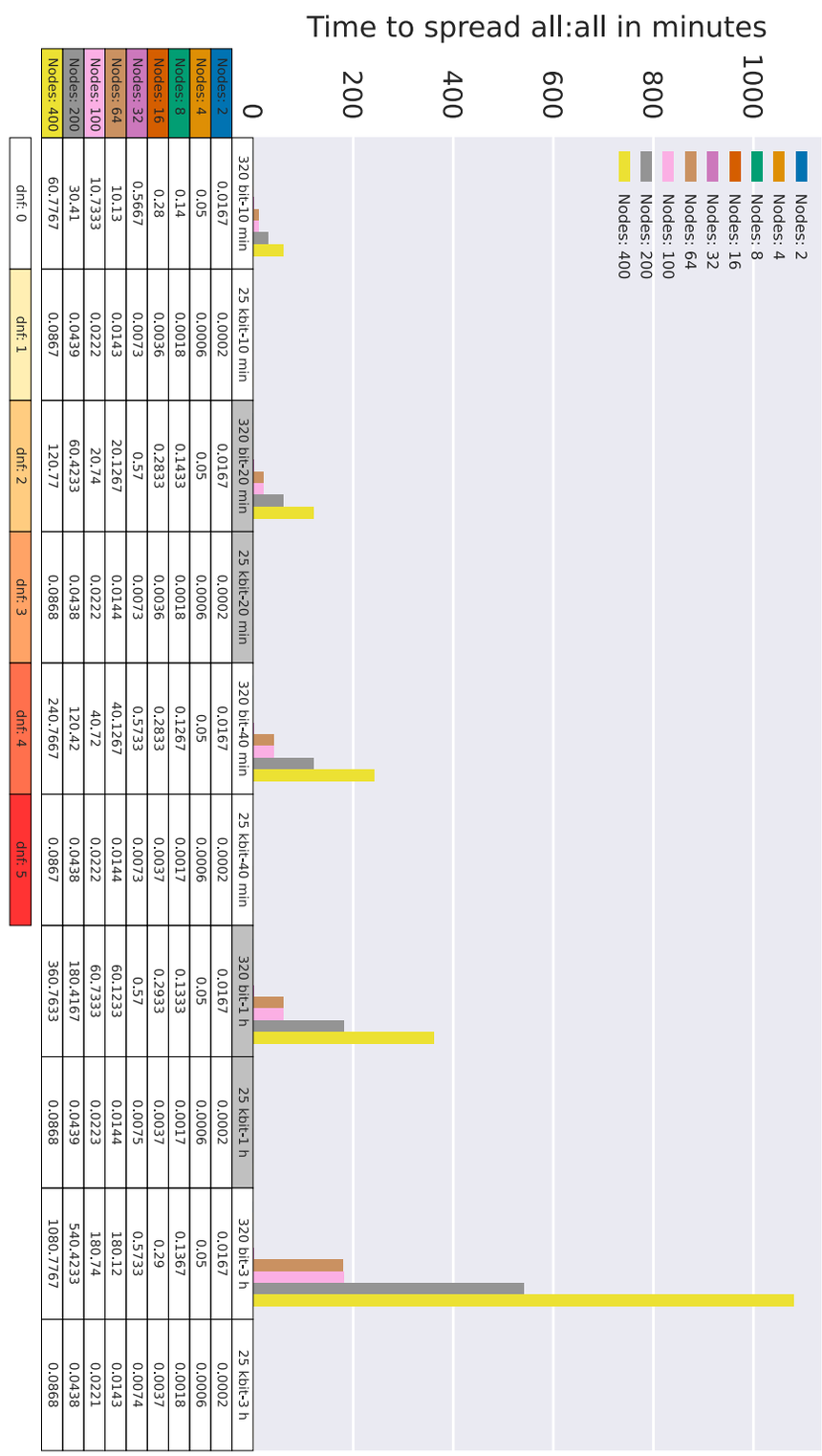
Table 11.2: Table of the smallest uptime needed for different wakeup values. With Spread: Uniform, Throughput: 320 bit/s and Latency: 0 ms

11.2 Results Comparing Throughput

Now that we have looked at the the result for 320 bit/s, we have a baseline of how the different parameters; spread, uptime and wakeup affect the final time to spread from all to all. This section will focus on the effect of throughput, by increasing the throughput to 25 kbit/s (25 000 bit/s) the result is expected to be much faster. The goal here is to see if the pattern still follows the same trends as with a throughput of 320 bit/s.

11.2.1 SLR 320 bit/s vs 25 kbit/s

For comparison between SLR with throughput 320 bit/s and 25 kbit/s two graphs have been chosen. One where the factor is wakeup, and one where the factor is uptime. Only no spread results have been compared. The reason is that 25 kbit/s with spread has the same limitation that 320 bit/s had. The ONs need to be awake at least 50% of the time.



	320 bit-10 min	25 kbit-10 min	320 bit-20 min	25 kbit-20 min	320 bit-40 min	25 kbit-40 min	320 bit-1 h	25 kbit-1 h	320 bit-3 h	25 kbit-3 h
Nodes: 2	0.0167	0.0002	0.0167	0.0002	0.0167	0.0002	0.0167	0.0002	0.0167	0.0002
Nodes: 4	0.05	0.0006	0.05	0.0006	0.05	0.0006	0.05	0.0006	0.05	0.0006
Nodes: 8	0.14	0.0018	0.1433	0.0018	0.1267	0.0017	0.1333	0.0017	0.1367	0.0018
Nodes: 16	0.28	0.0036	0.2833	0.0036	0.2833	0.0037	0.2933	0.0037	0.29	0.0037
Nodes: 32	0.5667	0.0073	0.57	0.0073	0.5733	0.0073	0.57	0.0075	0.5733	0.0074
Nodes: 64	10.13	0.0143	20.1267	0.0144	40.1267	0.0144	60.1233	0.0144	180.12	0.0143
Nodes: 100	10.7333	0.0222	20.74	0.0222	40.72	0.0222	60.7333	0.0223	180.74	0.0221
Nodes: 200	30.41	0.0439	60.4233	0.0438	120.42	0.0438	180.4167	0.0439	540.4233	0.0438
Nodes: 400	60.7767	0.0867	120.77	0.0868	240.7667	0.0867	360.7633	0.0868	1080.7767	0.0868

Figure 11.16: SLR Throughput 320 bit/s vs 25 kbit/s, Factor Wakeup, Uptime 1 minute, No spread

Graph 11.16 illustrates the differences the throughput has for the different uptime values. We can see that throughput 25 kbit/s has the same result regardless of the wakeup. Throughput 320 bit/s on the other hand needs to use multiple awake cycles at 64 ONs, because of this the time increases drastically. We see a bigger difference when the wakeup is high. This probably means that a 25 kbit/s throughput could handle much lower uptime, or a higher ON count.

	320 bit-1 min	25 kbit-1 min	320 bit-2 min	25 kbit-2 min	320 bit-5 min	25 kbit-5 min	320 bit-10 min	25 kbit-10 min	320 bit-Infinite	25 kbit-Infinite
Nodes: 2	0.0167	0.0002	0.0167	0.0002	0.0167	0.0002	0.0167	0.0002	0.0167	0.0002
Nodes: 4	0.05	0.0006	0.05	0.0006	0.05	0.0006	0.05	0.0006	0.05	0.0006
Nodes: 8	0.14	0.0018	0.1367	0.0018	0.13	0.0018	0.1233	0.0017	0.1233	0.0017
Nodes: 16	0.28	0.0036	0.29	0.0038	0.2933	0.0038	0.2833	0.0035	0.2833	0.0035
Nodes: 32	0.5667	0.0073	0.5733	0.0073	0.5667	0.0073	0.57	0.0074	0.57	0.0074
Nodes: 64	10.13	0.0143	1.12	0.0143	1.1233	0.0144	1.1233	0.0143	1.1233	0.0143
Nodes: 100	10.7333	0.0222	1.7333	0.0223	1.7233	0.0221	1.7333	0.0222	1.7333	0.0222
Nodes: 200	30.41	0.0439	11.4233	0.0438	3.42	0.0438	3.42	0.0438	3.42	0.0438
Nodes: 400	60.7767	0.0867	30.7733	0.0867	11.77	0.0869	6.7733	0.0868	6.7733	0.0868

Figure 11.17: SLR Throughput 320 bit/s vs 25 kbit/s, Factor Uptime, Wakeup 10 minutes, No spread

Graph 11.17 illustrates the differences the throughput has for the different wakeup values. Again we see that with throughput 25 kbit/s the uptime dos not have an effect, because it is so fast that it finishes before even the lowest uptime ends. 320 bit/s we can see improves the result as the uptime increases, like we found in section 11.1.2.

With the information in the figures we can say that for SLR the time to build the RM is reduced a lot, which was not surprising given the 78 times increase in throughput. The difference reduces when the uptime increases or wakeup decreases. But even compering the result where both are always on, we can see that 25 kbit/s can handle a much higher ON count.

11.2.2 SLM 320 bit/s vs 25 kbit/s

To compare the different throughputs for the SLM version, two graphs are chosen with spread as the varying factor.

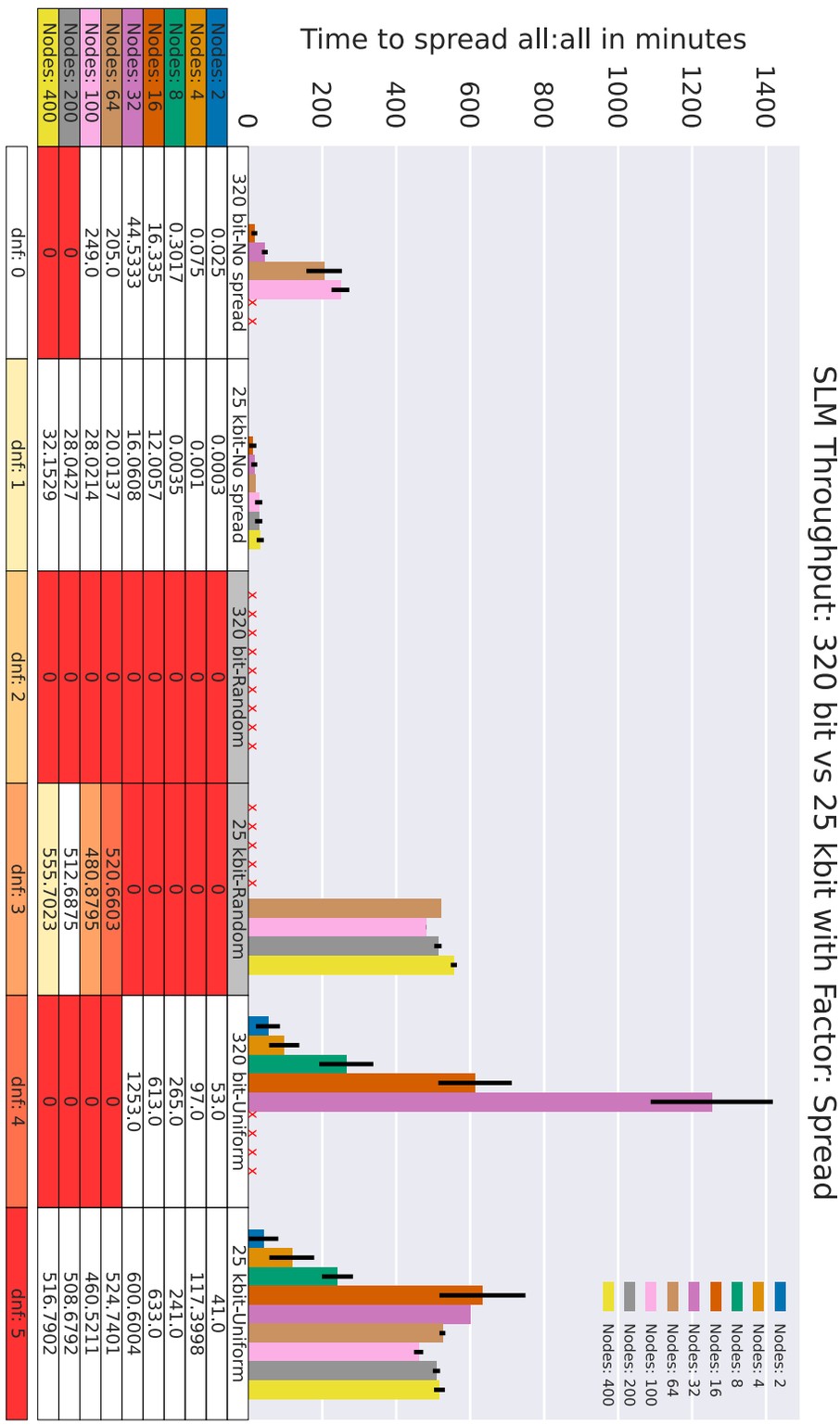


Figure 11.18: SLM Throughput 320 bit/s vs 25 kbit/s, Factor Spread, Wakeup 20 minute, Uptime 1 minute

When we look at graph 11.18 that shows the comparison between 320 bit/s and 25 kbit/s for SLM with different spread, and uptime one minute and wakeup 20 minutes. With no spread they jump at the same ON count, but for the 25 kbit/s it is more a step up, while the 320 bit/s continues to grow. With random spread the 320 bit/s does not finish any of the runs, while the 25 kbit/s actually finishes for 100, 200, and 400 ONs. For uniform spread the distribution seams to be the same up to 100 ONs, but the 25 kbit/s also finishes for 200 and 400 unlike the 320 bit/s.

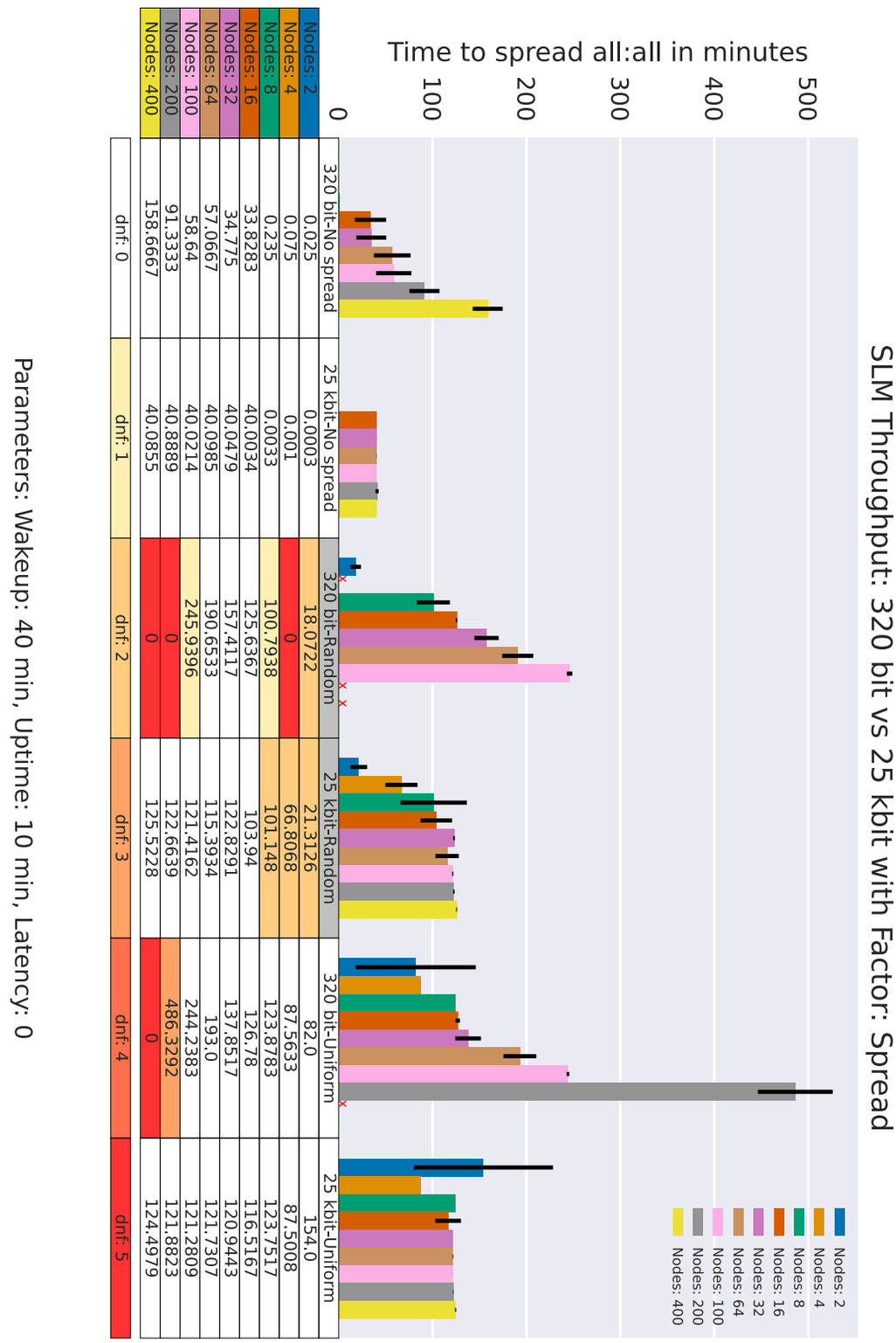


Figure 11.19: SLM Throughput 320 bit/s vs 25 kbit/s, Factor Spread, Wakeup 40 minute, Uptime 10 minutes

Looking at no spread in graph 11.19 shows that 25 kbit/s is faster for most of the ON counts, but interestingly a worse result with 8 and 16 ONs. The biggest difference with no spread seams to be at the higher end of the ON counts. Comparing the result for random spread also shows better results for 25 kbit/s, but within the margin of error up to 16 ONs. Again showing that the biggest difference is in the higher ON count. The same story can be told for SLMV. We can also note in this graph that 25 kbit/s random and uniform spread, is much closer than it is for random and uniform with 320 bit/s.

11.2.3 SLMV 320 bit/s vs 25 kbit/s

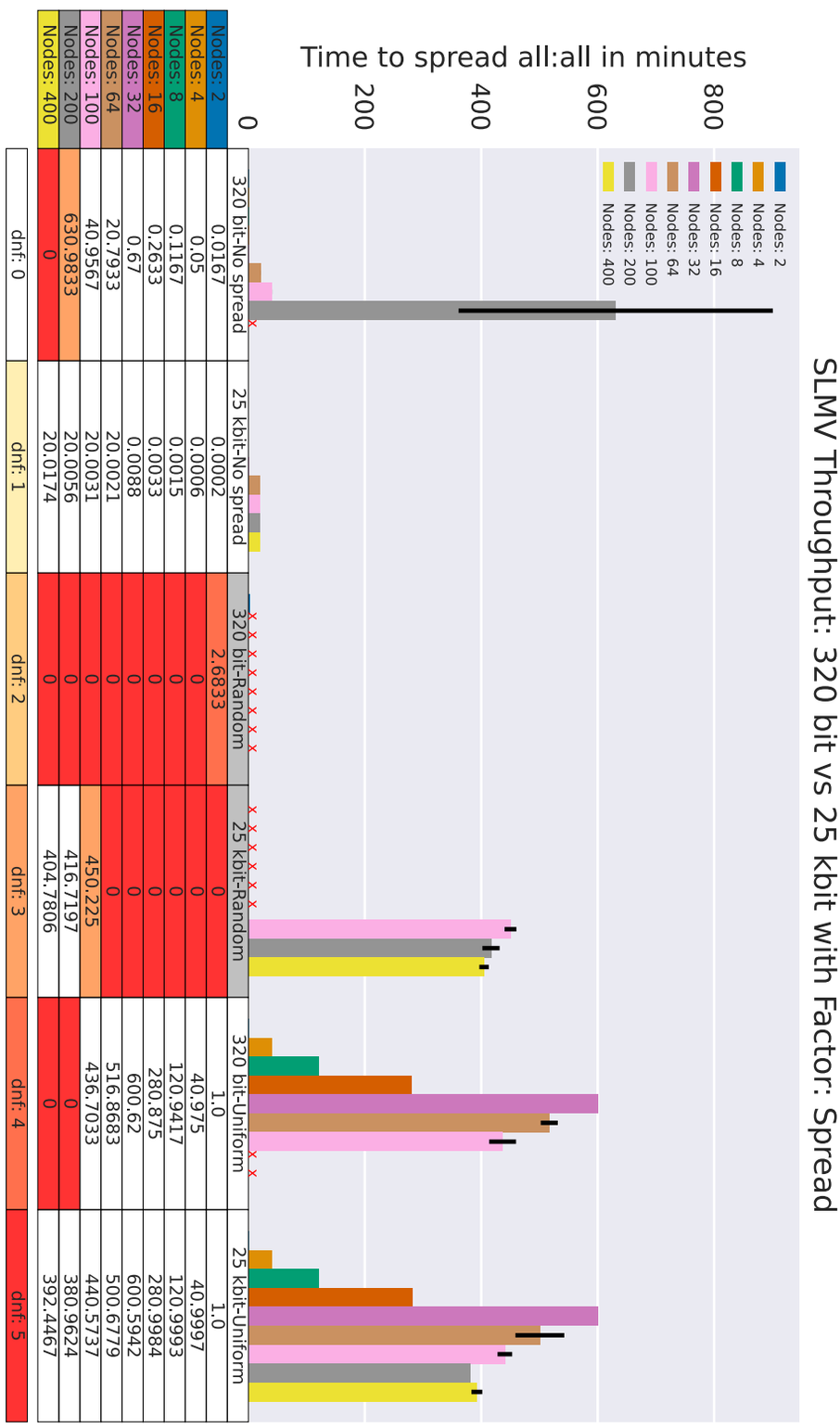
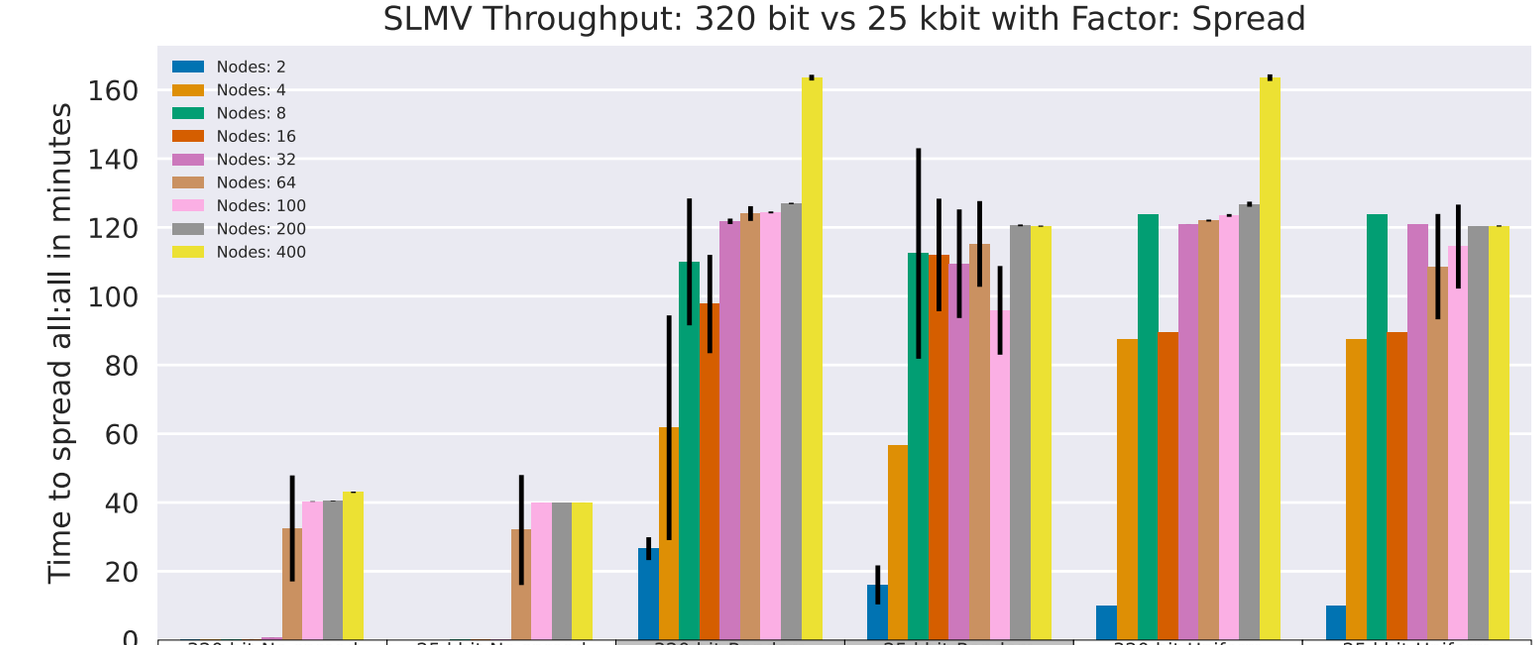


Figure 11.20: SLMV Throughput 320 bit/s vs 25 kbit/s, Factor Spread, Wakeup 20 minute, Uptime 1 minute

Graph 11.20 also shows better results for 25 kbit/s. Looking at no spread shows better results for all ON counts, but most significantly for the higher ON counts. Random spread 320 bit/s does not finish for any ON count. 25 kbit/s random spread finishes for the higher ON counts. This difference is probably that with higher throughput, the system can handle a shorter period of overlap. For SLMV the results are almost the same for 320 bit/s and 25 kbit/s up to 32 ONs. After that 25 kbit/s is significantly faster.



	320 bit-No spread	25 kbit-No spread	320 bit-Random	25 kbit-Random	320 bit-Uniform	25 kbit-Uniform
Nodes: 2	0.0167	0.0002	26.5875	16.0503	10.0	10.0
Nodes: 4	0.05	0.0006	61.75	56.5705	87.5083	87.5001
Nodes: 8	0.125	0.0016	109.9792	112.4241	123.7583	123.7501
Nodes: 16	0.2667	0.0034	97.725	111.9901	89.4033	89.3756
Nodes: 32	0.6617	0.0086	121.7733	109.4478	120.9717	120.9379
Nodes: 64	32.4717	32.0053	124.0367	115.1989	121.995	108.5976
Nodes: 100	40.3133	40.0031	124.365	95.8868	123.4867	114.4251
Nodes: 200	40.4433	40.0071	126.975	120.5969	126.7517	120.1533
Nodes: 400	42.98	40.0126	163.5517	120.3745	163.545	120.4207
	dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Figure 11.21: SLMV Throughput 320 bit/s vs 25 kbit/s, Factor Spread, Wakeup 40 minute, Uptime 10 minutes

The graph 11.21 highlights the difference between 320 bit/s and 25 kbit/s for the different spread types, with 40 minutes wakeup and 10 minutes uptime. No spread seams to have small differences, but 25 kbit/s is faster for all but 200 ONs. Random spread seams to follow the same pattern for both throughputs up to 200 ONs, for 400 ONs the difference is significant. For uniform the same pattern can also be seen between them, but the higher ON counts have a more profound difference.

For all exchange methods SLR, SLM, and SLMV the most significant difference can be seen in the higher ON counts. The SLR exchange method seams to have the most benefit of higher throughput. The lower effect on SLM and SLMV for the lower ON counts, can be explained by the more profound effect of the waiting. Even though the actually sending happens much faster, the most time is spent on waiting on someone to connect to.

From the result for throughput 25 kbit/s, latency 0 ms. Tables for finding the minimum uptime needed based on the wakeup, exchange type, and spread are created. The graphs are created and structured the same way the tables for 320 bit/s were. Table 11.3 is for SLM and SLMV with random spread. Table 11.4 is for SLM and SLMV with uniform spread. SLR and tables for no spread are not included, the reason is the same as for 320 bit/s section 11.1.

Smallest Uptime Needed In Minutes Random Spread

(Random)	Wakeup				
SLM	10 min	20 min	40 min	1 hour	3 hour
Lazy	1 (**400) or 2 (*400)	1 ** or 2 ** or 5 (*400)	5 (**400) or 10 (*400)	> 5 (**400)	> 5 (100-400)
Medium	1 (**400) or 2 (*400)	1 ** or 2 ** or 5 (*400)	5 (**400) or 10 (*400)	> 5 (**400)	-
Eager	2 (*400)	5 (*64)	5 (**2) or 10 (**2)	any (*2)	-
Panic	5 (32)	10 (16)	-	-	-

(Random)	Wakeup				
SLMV	10 min	20 min	40 min	1 hour	3 hour
Lazy	1 (**400) or 2 (*400)	1 ** or 2 ** or 5 (*400)	5 (**400) or 10 (*400)	> 5 (**400)	> 5 (100-400)
Medium	1 (**400) or 2 (*400)	1 ** or 2 ** or 5 (*400)	5 (**400) or 10 (*400)	> 5 (**400)	-
Eager	2 (*400)	5 (*100)	10 (*4)	any (*2)	-
Panic	5 (200)	10 (32)	-	-	-

(x)	Maximum node count
*	nb. Some node counts dnf
**	nb. Most node counts dnf
Lazy	Just finishes
Medium	Within 2 days
Eager	Within 1 hour
Panic	Within 10 minutes

Table 11.3: Table of the smallest uptime needed for different wakeup values. With Spread: Random, Throughput: 25 kbit/s and Latency: 0 ms

Smallest Uptime Needed In Minutes Uniform Spread

(Uniform)	Wakeup				
SLM	10 min	20 min	40 min	1 hour	3 hour
Lazy	1 (400)	1 (400)	1 (400)	1 (400)	1 (200)
Medium	1 (400)	1 (400)	1 (400)	1 (400)	any (4)
Eager	1 (4) or 2 (200)	any (2)	> 2 (2)	-	-
Panic	5 (32)	10 (2)	-	-	-

(Uniform)	Wakeup				
SLMV	10 min	20 min	40 min	1 hour	3 hour
Lazy	1 (400)	1 (400)	1 (400)	1 (400)	1 (400)
Medium	1 (400)	1 (400)	1 (400)	1 (400)	any (16)
Eager	1 (4) or 2 (200)	10 (400)	any (2)	any (2)	any (2)
Panic	5 (64)	10 (32)	any (2)	any (2)	any (2)

(x)	Maximum node count
*	nb. Some node counts dnf
**	nb. Most node counts dnf
Lazy	Just finishes
Medium	Within 2 days
Eager	Within 1 hour
Panic	Within 10 minutes

Table 11.4: Table of the smallest uptime needed for different wakeup values. With Spread: Uniform, Throughput: 25 kbit/s and Latency: 0 ms

11.3 Results Comparing Latency

In this final section of the result we will compare and see if the latency affects the result in any meaningful way. Both the throughput of 320 bit/s and 25 kbit/s will be compared to simulations with an additional 500 ms latency.

11.4 320 bit/s 0 ms vs 500 ms

Until now we have not looked at latency. The expectation was that this will drastically alter the affect of the changes in behavior compared to no latency. To illustrate graphs of each spreading type SLR, SLM, and SLMV with both the 0 ms and the 500 ms latency is created. This will be done for both 320 bit/s and 25 kbit/s.

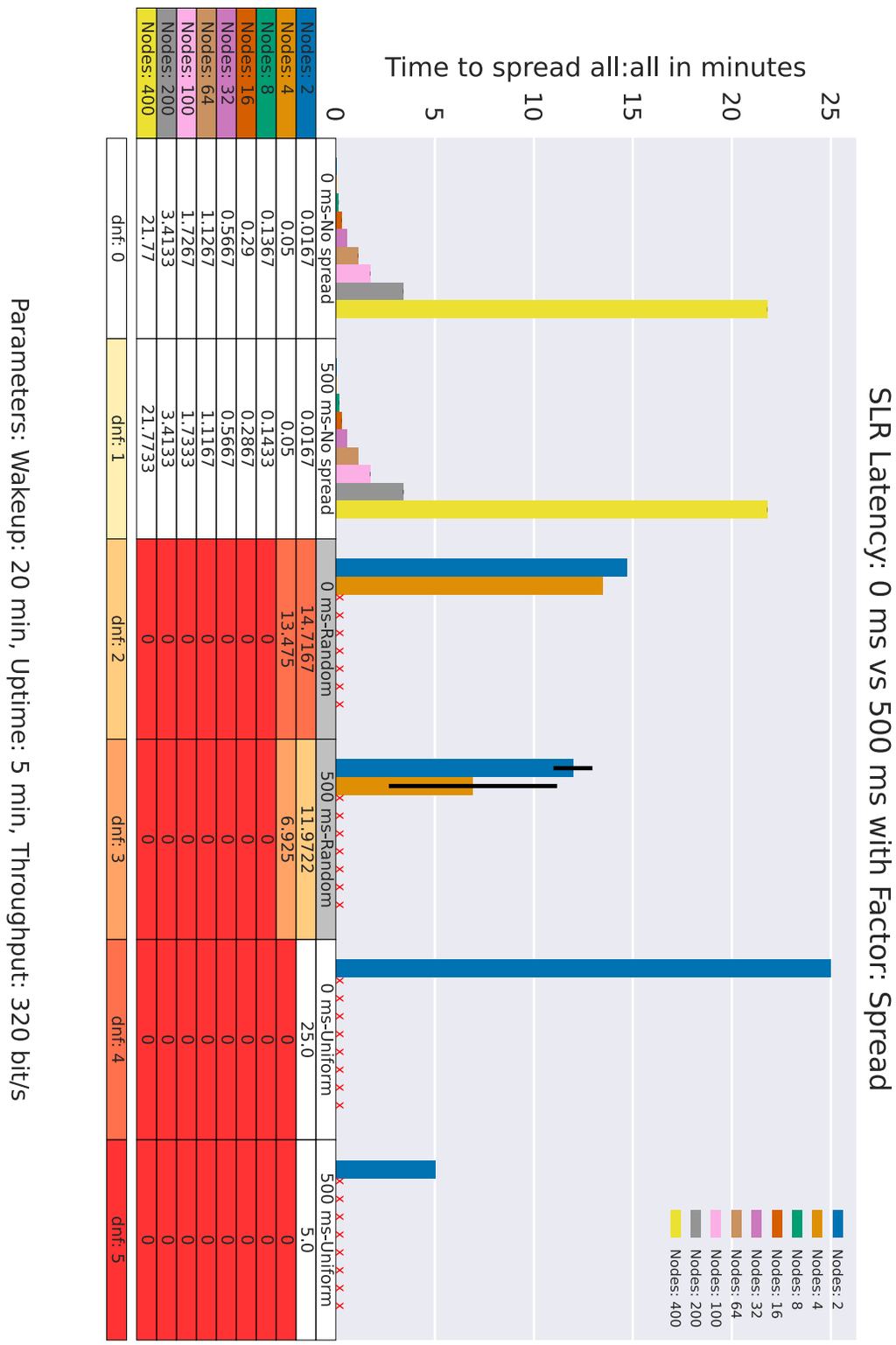


Figure 11.22: SLR Latency 0 ms vs 500 ms, Factor Spread, Throughput 320 bit/s

Graph 11.22 shows the difference the latency has for SLR, with different spread. Whit 320 bit/s, 20 minutes uptime, and five minutes wakeup. We can see that with no spread the results are almost identical. With random spread the results are interestingly lower with 500 ms than with 0 ms. The same interestingly happens for the uniform spread.

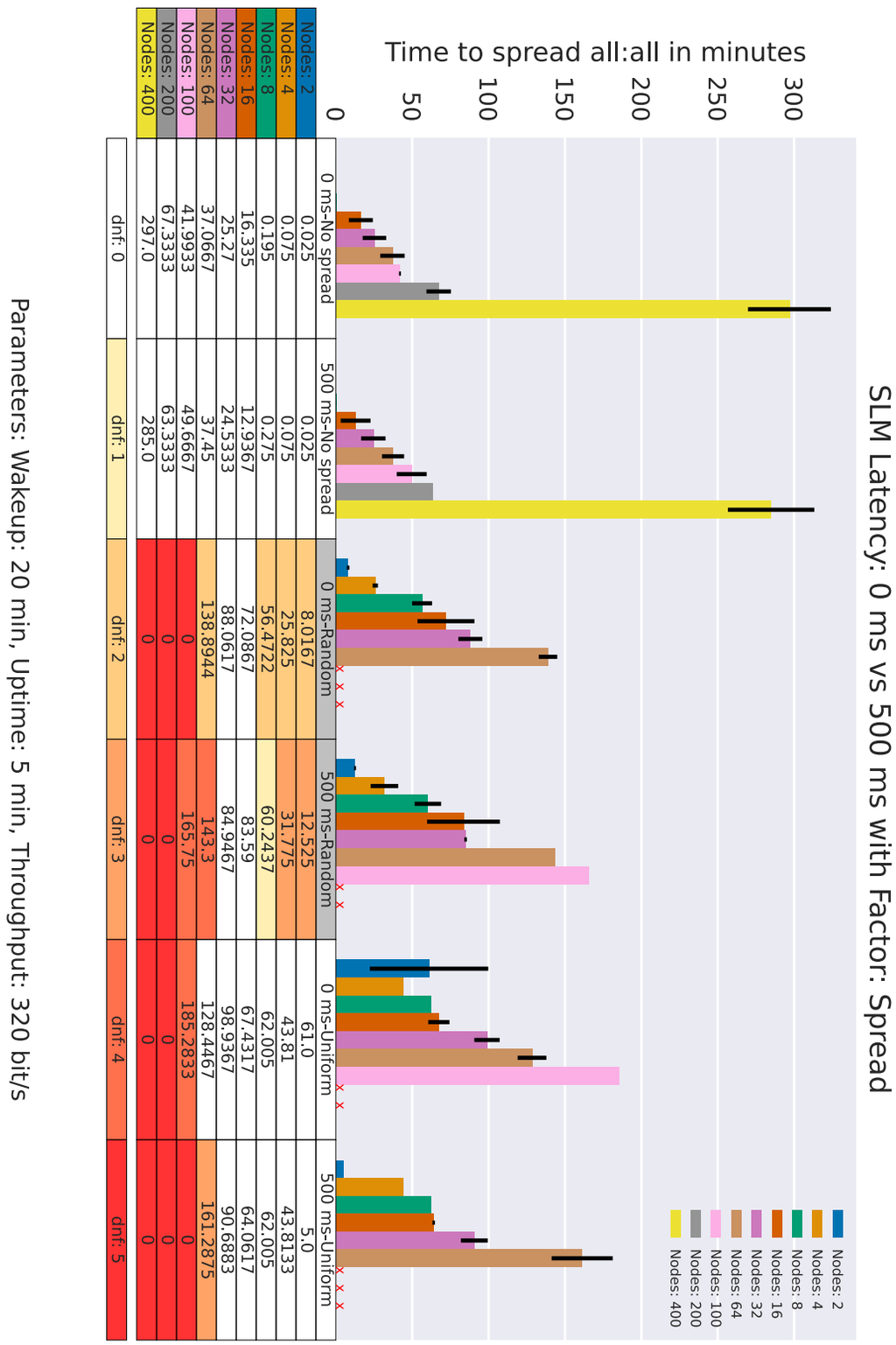
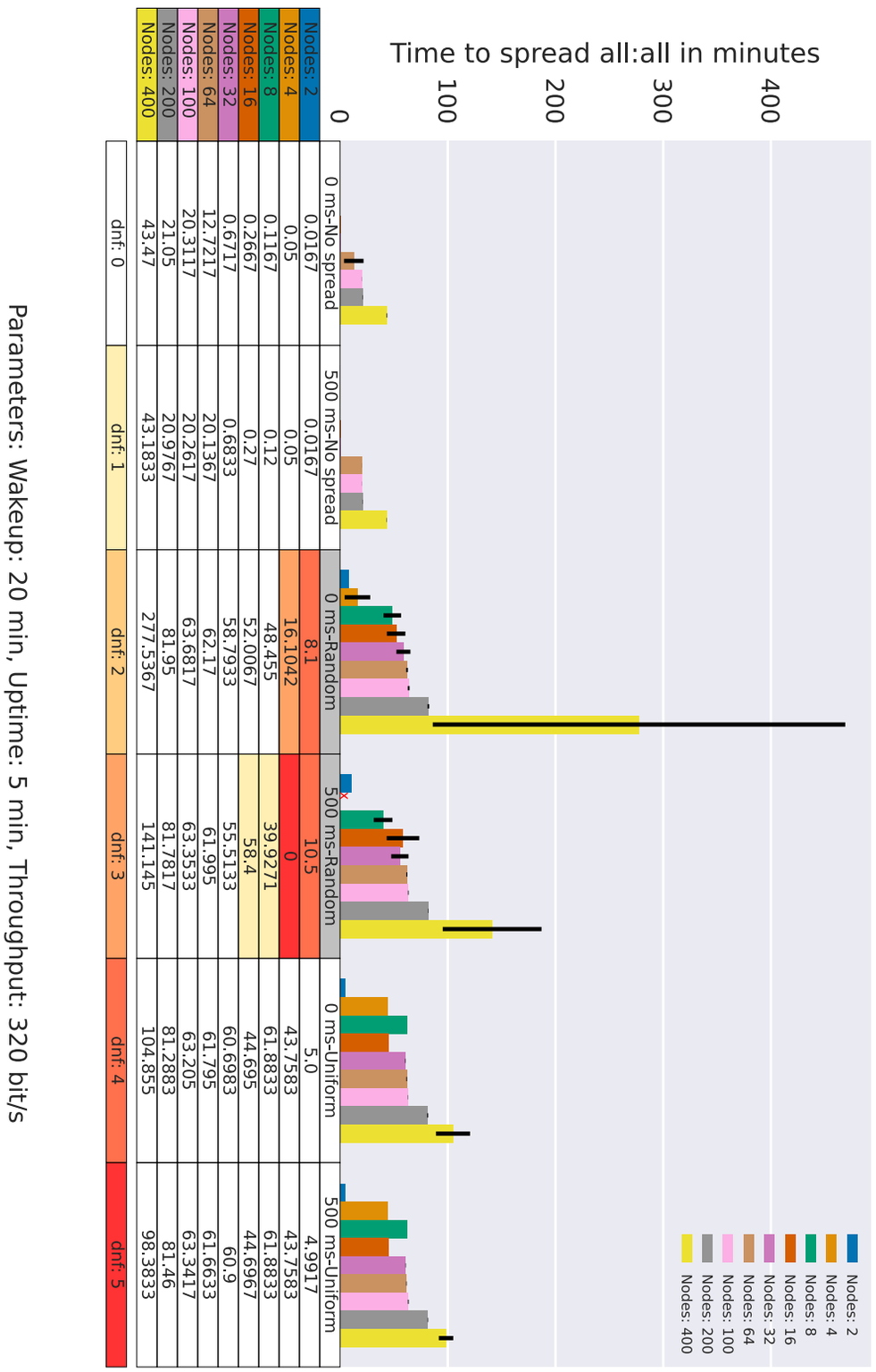


Figure 11.23: SLM Latency 0 ms vs 500 ms, Factor Spread, Throughput 320 bit/s

Looking at graph 11.23 that compares the latency for SLM with 320 bis/s and 20 minutes wakeup and five minutes uptime, shows that the effect with no spread is within margin of error. With random spread the result is similar, but the amount of failed runs increase with 500 ms compared to 0 ms. With uniform spread the 500 ms actually has a significantly better result with two ONs, but the 0 ms has a much higher standard deviation. There are a higher number of failed results for 500 ms than 0 ms on the higher ON counts.



	0 ms-No spread	500 ms-No spread	0 ms-Random	500 ms-Random	0 ms-Uniform	500 ms-Uniform
Nodes: 2	0.0167	0.0167	8.1	10.5	5.0	4.9917
Nodes: 4	0.05	0.05	16.1042	0	43.7583	43.7583
Nodes: 8	0.1167	0.12	48.455	39.9271	61.8833	61.8833
Nodes: 16	0.2667	0.27	52.0067	58.4	44.695	44.6967
Nodes: 32	0.6717	0.6833	58.7933	55.5133	60.6983	60.9
Nodes: 64	12.7217	20.1367	62.17	61.995	61.795	61.6633
Nodes: 100	20.3117	20.2617	63.6817	63.3533	63.205	63.3417
Nodes: 200	21.05	20.9767	81.95	81.7817	81.2883	81.46
Nodes: 400	43.47	43.1833	277.5367	141.145	104.855	98.3833
	dnf: 0	dnf: 1	dnf: 2	dnf: 3	dnf: 4	dnf: 5

Figure 11.24: SLMV Latency 0 ms vs 500 ms, Factor Spread, Throughput 320 bit/s

The graph 11.24 shows the difference of 0 and 500 ms for SLMV with a throughput of 320 bit/s, wakeup 20 minutes and uptime 5 minutes. The graph shows that the effect is within margin of error for all spread types.

The result indicates that the effect of latency with 320 bit/s is marginal for all spreads with both SLR and SLMV. The SLM approach shows an increase in failed results, but the reported times are fairly similar.

11.5 25 kbit/s 0 ms vs 500 ms

Now looking at the effect of latency with throughput 25 kbit/s

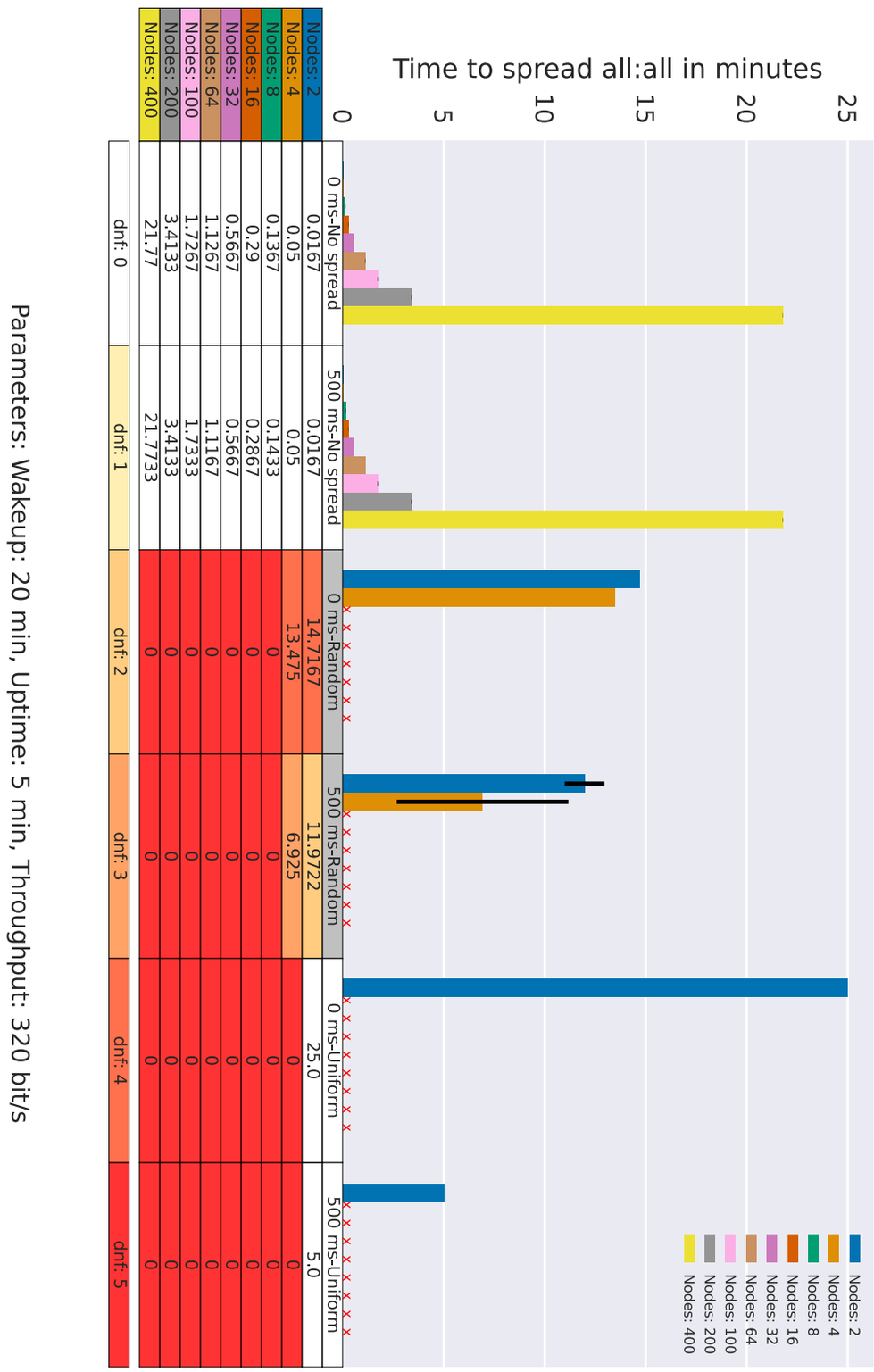


Figure 11.25: SLR Latency 0 ms vs 500 ms, Factor Spread, Throughput 25 kbit/s

By looking at graph 11.25 comparing the effect of latency for SLR, we see no effect with no spread. A small decrease in time for random, but higher standard deviation. For uniform the result also goes down when the latency goes up. It is worth noting that the result with random spread only finished sometimes for the two lowest ON counts. For the uniform only for the ON count two.

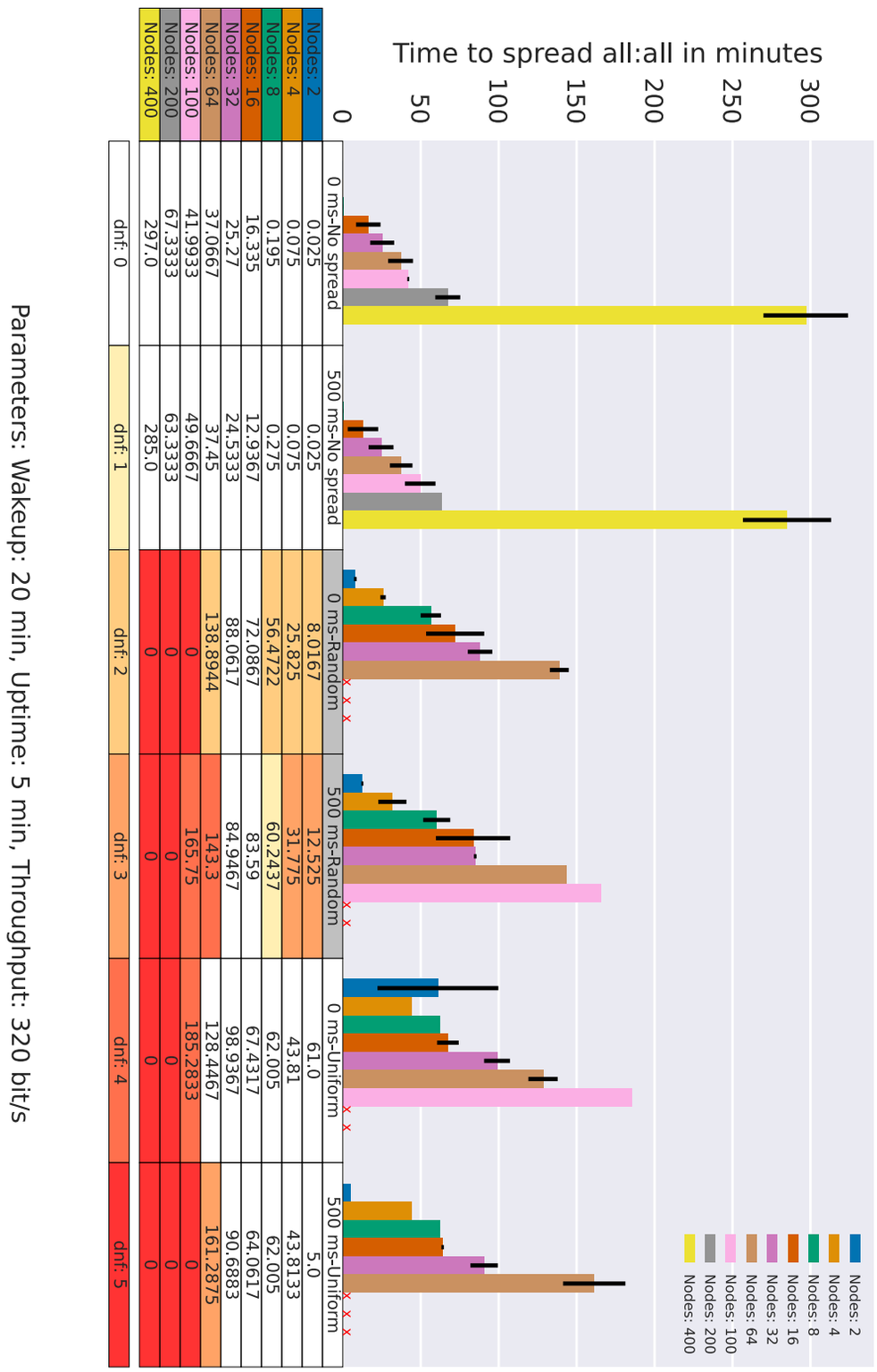


Figure 11.26: SLM Latency 0 ms vs 500 ms, Factor Spread, Throughput 25 kbit/s

Looking at the result comparing latency for SLM in graph 11.26, the change in result from 0 ms latency to 500 ms latency, is within margin of error for both no spread and random spread.

With uniform spread the results are similar for all ON counts except 2, 64, and 100. With 2 ON the result is actually faster with 500 ms, for 64 the result is slower with 500 ms and has three failed runs. When the ON count is 100 it only finishes one time with zero ms latency.

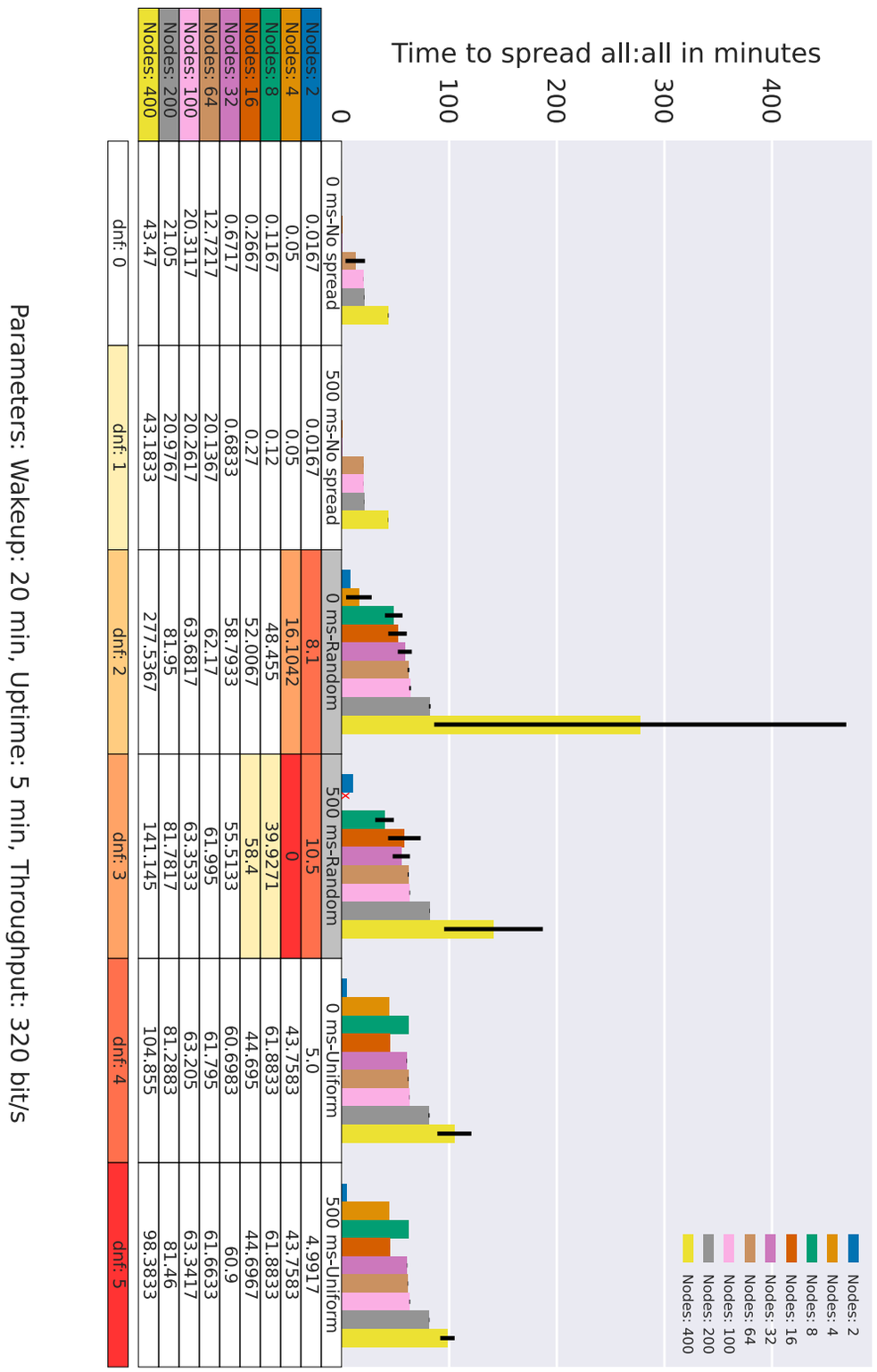


Figure 11.27: SLMV Latency 0 ms vs 500 ms, Factor Spread, Throughput 25 kbit/s

Lastly when looking at the effect of latency for SLMV, the result in graph 11.27 shows all results are within margin of error.

When we have compared the results for 25 kbit/s with zero and 500 ms latency, it is clear that the effect is minimal, only showing small variations.

12 Discussion

12.1 Set Limitations

Because the SLM and SLMV with the lower uptime fail when we send too much, we could set a limit on the amount of data that they could send each time. This would help, but we would need to make a decision on how to select what is sent and not.

12.2 Remember Failed Transactions

Another approach is to remember what is already sent when a transaction brakes, but this would require more negotiation when the ONs connect again after a failed exchange. In addition, it could be hard to guarantee that the same ONs connect again in a reasonable amount of time after a failed exchange.

12.3 Radio Wakeup

Other solutions to the time gap problem exist. Instead of the ONs just randomly waking up at the same time, or scheduling to wake up at the same time, ONs could be equipped with some sort of radio wakeup technology. This would make it possible for one ON to wake another ON up from sleep when it wants

to talk to it.

But the radio wakeup have some problems. First we would need to choose who can wake up who, because if anyone can wakeup anyone we have potentially a security problem, or at least a problem where the system is prone to sabotage. Another issue is the energy issue. If ONs cannot plan their own schedule they cannot preserve the energy left on them in the best way. The designers would have to map out every possible situation and the importance of the situation, to justify each wakeup. If every task is deemed important, the result could be that all ONs are up all the time anyway.

12.4 Use Beacon for Resource Descriptions

We could use the beacon signal to broadcast the resource description itself. But this comes with the drawback of security and vulnerability for sabotage. The other problem is to know when to broadcast, and when to stop. Even if we had the ONs active at the same time and everybody broadcasts and collects resource description, there could be problems. An ON that for some reason did not receive information about one or more ONs, would not be able to ask for it again. An other problem is that ONs would have to overlap and ONs that do not overlap would not be able to get information from each other.

12.5 Hybrid Solution

Because the time to spread increase the most when the neighborhoods become larger, maybe some hybrid solution can be effective. We can assign three waves of action, and split the neighborhoods in to smaller groups. In the first wave each group of overlapping members with no spread exchange their resource information with each other, with exchange method SLR. In the second wave one leader from every group overlaps and shares the group resources with all the other leaders, again with SLR. In the final wave the leader starts to spread the resource information gotten from the other leaders with the initial group again, now using SLM or SLMV.

For smaller neighborhoods this would probably not be efficient, but for bigger neighborhoods this could actually help. From the result it is clear that ether higher throughput is needed, or other solutions to spread must be invented to help with the spread for bigger neighborhoods. This was not the goal for this research, though.

12.6 Strange Behavior When Always On

The fact that SLM and SLMV seam to fail for the test where the ONs are awake all the time is a bit strange. I believe it is because of the design choice to make ONs connect to other ONs they already have been connected to, if no other ON is available. The effect of this is that the ONs get occupied, meaning that some ONs is not able to receive the last information when most of the others are finished.

12.7 Choosing Parameters

If the spread of the ONs is already decided to be ether no spread, random or uniform, we have already narrowed down the choices for the exchange method. If there is no spread, the SLR approach is always better. If there is a random spread, it seams that SLMV comes out better for higher ON counts. But for the lower ON counts, the SLM can also be a solid choices. SLMV will require some more complexity at each node.

Latency seam to have a smaller effect then fist thought. The result shows that with a small latency, we see no changes in the pattern when we make alterations to the other parameters. This could be that the latency is too low to see the effect of it, and it may be that a higher latency would create different patterns for the changes in parameter values.

13

Conclusion

To conclude we can say that most of the parameters have a significant effect on the total time to spread resource information from all to all. Some of the parameters have higher effect than others.

When we look at the effect of the wakeup, uptime, and spread, we can say that spread has the most unpredictable effect. The result for the spread illustrates that having the ONs always on simultaneously makes the building of the RM a lot faster than when spread is introduced. When we have a random spread there need to ether be long enough uptime compared to wakeup, or enough ONs so that an overlapping trace is likely to happen. When there is either a low uptime compared to wakeup, or a low ON count, a more structured spread that can help ensure an overlapping trace can help the RM building.

Wakeup mostly effects the sleep time of the ONs, but also how many ONs are needed in the neighborhood to achieve overlap. When the ratio of uptime compared to wakeup decrease, more ONs is needed to be able to create an overlapping trace with random spreader. The sleep time can effect the total time to build the RM drastically when ONs start to use several uptime periods.

The uptime effects the number of ONs that are able to connect and share information before they go to sleep, and therefore effects the amount of sleep/awake cycles they need before they finish spreading the information.

Throughput has the most effect when the spread is lower. The most profound

effect of a higher throughput is that the transfer times goes down. Therefore when we have no spread, the effect is the highest. When spread is introduced, the effect goes down. This is because most of the time is spent sleeping, or waiting for an opportunity to connect to someone. The higher throughput have the effect of also being able to spread to higher ON counts. A small latency has a low or no impact on the total time to build up the RM.

The results shown in this theses should help designers of observation networks in remote places, that lack essential infrastructure, like the AT. To be able to tune the specifications, and parameters of the ONs behaviour, so that they reach a specific target. This paper and the results can also warn about some pitfalls that might not be as obvious when setting up an observation network.

14

Future work

For future it would be interesting to see experiments where ONs do not have the same wakeup and uptime behaviors. It would also be interesting to see more done with the past and future neighborhoods proposed in chapter 5, and highlighting the effect of letting ONs change their behavior based on the information in their maps and future neighborhood. In addition it would also be interesting to show more static experiments where wakeup and uptime is different for some, or all the ONs.

In chapter 8 some shortcomings of this project is highlighted. Exploring these questions further could have significant importance for a more complete understanding of ON behaviors. Especially CPU usage could be interesting to know more about.

For the parameters tests it could be interesting to see more values for network throughput and latency tested.

Bibliography

[1] Valentin Popa Alexandru Lavric. Performance evaluation of lorawan communication scalability in large-scale wireless sensor networks. In *Wireless Communications and Mobile Computing*, volume vol. 2018, page 9 pages, 2018. URL https://doi.org/10.1155/2018/6730719.

[2] Aloÿs Augustin, Jiazi Yi, Thomas Clausen, and William Mark Townsley. A study of lora: Long range amp; low power networks for the internet of things. *Sensors*, 16(9), 2016. ISSN 1424-8220. doi: 10.3390/s16091466. URL https://www.mdpi.com/1424-8220/16/9/1466.

[3] Red Hat. What is edge architecture?, 2021. URL https://www.redhat.com/en/topics/edge-computing/what-is-edge-architecture?sc_cid=7013a000002pu40AAA&gclid=Cj0KCQjwsdiTBhD5ARIsAIpW8CJgKcSLuhy7sgt2d7xmVdAjiBKZD3S9PIW-r44UfI3lR9xi3qjsUPYaAp9CEALw_wcB&gclsrc=aw.ds.

[4] Alex Hawkes. Understanding network speed and latency, 2021. URL https://blog.consoleconnect.com/understanding-network-speed-and-latency.

[5] Christine Julien, Chenguang Liu, Amy L. Murphy, and Gian Pietro Picco. Blend: Practical continuous neighbor discovery for bluetooth low energy. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 105–116, 2017.

[6] Eric Brewer David Culler Kamin Whitehouse, Cory Sharp. Hood: a neighborhood abstraction for sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, page 99–110, 2004. doi: 10.1145/990064.

[7] Joakim Lindh. Bluetooth® low energy beacons. Technical report, Texas Instruments, 01 2015. URL https://www.ti.com/lit/an/swra475a/swra475a.pdf?ts=1651953502414.

[8] Chenguang Liu, Jie Hua, and Christine Julien. Scents: Collaborative sensing in proximity iot networks. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 189–195, 2019. doi: 10.1109/PERCOMW.2019.8730863.

[9] Christina Nunez. Tundra threats explained, 2020. URL https://www.nationalgeographic.org/article/tundra-threats-explained/.

[10] UiT The Arctic University of Norway. Distributed arctic observatory. URL https://en.uit.no/project/dao.

[11] Norwegian polar institute. Climate change in the arctic. URL https://www.npolar.no/en/themes/climate-change-in-the-arctic/#toggle-id-3.

[12] André Queirós, Daniel Faria, and Fernando Almeida. Strengths and limitations of qualitative and quantitative research methods. *European Journal of Education Studies*, 0(0), 2017. ISSN 25011111. doi: 10.46827/ejes.v0i0.1017. URL https://oapub.org/edu/index.php/ejes/article/view/1017.

[13] Kate Ramsayer. Warming temperatures are driving arctic greening, 2020. URL https://climate.nasa.gov/news/3025/warming-temperatures-are-driving-arctic-greening/.

[14] Csaba Rotter, János Illés, Gábor Nyíri, Lóránt Farkas, Gergely Csatári, and Gergő Huszty. Telecom strategies for service discovery in microservice environments. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 214–218, 2017. doi: 10.1109/ICIN.2017.7899414.

[15] Arash Sattari, Rouhollah Ehsani, Teemu Leppänen, Susanna Pirttikangas, and Jukka Riekki. Edge-supported microservice-based resource discovery for mist computing. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 462–468, 2020. doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00087.

[16] Jacob Strauss, Justin Mazzola Paluska, Chris Lesniewski-Laas, Bryan Ford, Robert Morris, and Frans Kaashoek. Eyo: Device-transparent personal storage. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, Portland, OR, June 2011. USENIX Association. URL https://www.usenix.org/conference/usenixatc11/eyo-device-transparent-personal-storage.

[17] Roberth Tollefsen, Issam Rais, John Markus Bjørndalen, Phuong Hoai Ha,

and Otto Anshus. Distribution of updates to iot nodes in a resource-challenged environment. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 684–689, 2021. doi: 10.1109/CCGrid51090.2021.00082.

[18] Qi Xia, Ruijun Yang, Weinong Wang, and De Yang. Fully decentralized dht based approach to grid service discovery using overlay networks. In *The Fifth International Conference on Computer and Information Technology (CIT'05)*, pages 1140–1144, 2005. doi: 10.1109/CIT.2005.122.

A

Appendix

Extra material for this thesis is placed in a compressed file. In extra material the source code, and the raw results can be found. The source code is placed in a folder called "src", and the results will be placed in a folder called "results".

The the source code folders are: the simulation written in c, experiment setup written in Python3, code for plotting the results, and scripts for automate the process. A README.txt file is created for instructions on how to compile and use the implementation and scripts.

The raw results from the experiments are placed in the folder named "results", combined with all the graphs plotted from the results. In the folder four sub folders can be seen with the naming "throughput-latency" where throughput is one of 25kbs or 320bs and latency is ether oms or 500ms. In each of these folders is the raw result for that configuration, and also a plot folder with all permutations for the factors and parameters. The graphs in the plot folder follow the same naming structure; "what is compared_Factor_F_A_B.svg" where F is the factor used, and is either: Spread, Wakeup, or Uptime. A B is the parameter names followed by the values.