Scheduling Convoys of Nanosatellites

P10 Project
Group Deis108f18
Software
Aalborg University

AALBORG UNIVERSITY
STUDENT REPORT

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Software
Selma Lagerlöfs Vej 300
9220 Aalborg East
https://www.cs.aau.dk

Title:

Scheduling Convoys of Nanostellites

Project:

SW10 project

Project period:

1st February - 1st June

Project group:

Deis108f18

Members:

Oliver Brun Køsted

Anders Lykke Matthiassen

Jacob Nielsen

Supervisor:

René Rydhof Hansen

Isabella Kaufmann

Kim Guldstrand Larsen

Pages: 74
Appendices: 1
Ended: 1st June - 2018

Synopsis:

Increasing interest in observing Earth's climate and air traffic have led companies to explore the possibility of launching multiple satellites into orbit to monitor this. It is desired that these have the capability to communicate with each other and stations located on Earth.
This project aims to explore possibilities for scheduling convoys consisting of multiple satellites. With the use of UPPAAL-CORA and SMC, we are able to produce traces which can be used to create a schedule for multiple satellites with some or all of them sharing the same orbit. The deterministic model, constructed in SMC, implements a preemptive scheduler with a simple dynamic prioritising.
In the process of creating and testing the deterministic model, we explored limitations of scheduling multiple satellites which could be applied in the construction of a non-deterministic model. Such a model was made in CORA. This model facilitates a VBP scheduler for determining the tasks' importance. Finally we evaluated the two models based on their experiments.

Summary

In this project we work with scheduling multiple satellites in convoys, using different models to explore possible options for doing this. The purpose of the convoys is to make the satellites cooperate in sending the most data to some stations on Earth, in a cost efficient manner. This domain introduce different concepts such as;
Orbits — the direction and speed the satellites orbiting Earth. Windows of opportunity — periods of time where a satellite is within range to send data to a station on Earth. Tasks — there are different tasks with different restrictions such as whether or not the satellite is in range of a station. Satellite orientation — the satellite may need to face a specific direction to execute some tasks.

The possibilities explored are, constructing a deterministic model in SMC and a non-deterministic one in CORA. The deterministic model was created first with focus on the satellites and was made with a high level of detail i.e. incorporating the concepts of windows of opportunity, a preemptive scheduler, internal communication between neighbouring satellites, and satellite orientation.
Experimenting on the deterministic model showed which of the defined values and features had the most significant impact on the computation time and schedule effectiveness. Through this we found that slewing prior to performing a task may not be necessary to include, as long as the time taken to do so is incorporated in the task's execution time. Additionally it was discovered that a preemptive scheduler is not necessarily the best option for such a system.
With this information the non-deterministic model was constructed. As it was known the state space for such a model would be significantly larger, compared to that of the deterministic model, the scheduler was not made preemptive, and the concept of orientation was excluded. The non-deterministic model focused more on the convoys than the individual satellites.
Despite of the limitations, the non-deterministic model was not able to find the best trace when attempting to generate one as long as the one the deterministic model was capable of producing. However, through the experiments we found that further restricting the choices that the non-deterministic model could make, it became capable of generating schedules of lengths similar to that of the deterministic model. Restricting the model of course goes against the non-determinism of the model, and makes CORA less effective. Despite of this, it was possible to get close to similar results when relying on the implemented scheduling method.

By constructing these models and experimenting on them, we finally determined that with some modifications both methods may be viable for generating schedules for convoys of satellites. However, we believe a better solution is to integrate the scheduling method from the non-deterministic model into a deterministic model.

Preface

This report documents the work of three 10^{th} semester Software Engineering students during the spring of 2018. This report is part of the specialisation in the field Semantics and Verification.

Citations are made in accordance with the Vancouver style, meaning they are indicated by the use of numbers, and will look like: [1]. A complete list of cites is found in the bibliography and the cites are ordered by when they appear in the report. Lastly, code found in listings may look different from what is found in the implemented code. No functionality has been changed, this is simply to make the code better presentable and easy to read.

Table of Contents

1 Introduction

Increasing interest in observation of Earth's climate and traffic have led to an increase in the number of satellites orbiting Earth. National Aeronautics and Space Administration (NASA) is currently in the process of developing and deploying nanosatellites to monitor Earth's climate, the project is called *Time-Resolved Observations of Precipitation structure and storm Intensity with Constellation of Smallsats (TROPICS)*. NASA plans to launch 12 nanosatellites into three different orbits between 2018 and 2019 to monitor weather. This project span from 2016 with their initial research till 2021 with their final report [1].

We have previously explored a method for automating the generation and construction of schedules for nanosatellites in collaboration with GomSpace [2]. In the method, a single satellite was modelled with a considerable focus on optimising the use of the satellite's battery. After the model was finished and the method was presented to the collaborators of the project, a new problem was presented. It has been expressed by GomSpace that the problem of scheduling multiple satellites that are working in conjunction is of significant interest as it is what they are currently working with [3].
The direction for this project will be to present multiple methods that utilises model checking in order to construct schedules for groups or convoys of satellites. The satellites will need to be able to communicate with other satellites within a convoy and with other convoys. They will also be able to communicate with some stations that are located on Earth. When the satellites communicate with the stations there is an associated cost with sending data, meaning that the satellite owners will need to pay some amount per Mb transferred [3]. It will be relevant to include this when making a schedule in order to try and minimise the expenses associated with data transfer wile still being able to send the collected data.
Some other factors that are relevant for a satellite will also be considered in this project:

1. Satellite storage capacity
2. Some inclusion of a non perfect orbit
3. The satellite's orientation

Because we were able to conclude, in our previous work, that the risk of draining the battery completely was not of concern, the satellites' batteries will not be monitored in this project [2].

To the best of our and our collaborators knowledge, there is no tool for scheduling multiple satellites that are communicating internally, using the same orbit, and are communicating with other convoys of satellites in other orbits [3].
In this project, with the help of model checking, we will further explore the domain of automated scheduling for such systems. We will do so by constructing multiple models

that utilises different strategies in order to solve the problem.

Problem statement *How can different forms of model checking be applied to generating a schedule for multiple satellites in a convoy, and how does such methods compare against each other?*

Analysis 2

We have studied various terms and properties related to satellites in order to identify what is important to take into consideration when creating a model that represents convoys of satellites. This chapter will cover common terminology and introduce concepts that will be relevant for the modelling.

2.1 Terminology

In this section we will define some of the therms introduced previously. This includes a definition of what a satellite is, what a satellites orientation is, and what a station is.

2.1.1 Convoy

A convoy in this context is a set of satellites that follows the same orbit and are able to communicate internally. It is also possible for one convoy to communicate with another convoy whenever two or more satellites from the convoys are close enough. We call this a cross convoy communication opportunity and the most common case that allows for this communication is when two or more satellites' orbits cross each other.

2.1.2 SmallSats

A satellite is a physical object revolving around a celestial body, in this scenario Earth. Man made satellites can be categorised by their weight and/or by their hardware specification. This section serves to briefly detail what type of satellite/hardware that will be modelled.

Types of Satellites

SmallSats can be divided into five types of categories depending on their weight. They range from 180 kg down to 0.001 kg. The first category *minisatellite* houses satellites weighting between 100–180 kg, following that is *microsatellite* weighting 10-100 kg and *nanosatellite* ranging from 1–10 kg. Finally we have *picosatellite* and *femtosatellite* which ranges from 0.01–1 kg and 0.001–0.01 kg respectively [4].

The satellites used in TROPICS and GomX-3 case study [5] both operate under the nanosatellite category. These are restricted in their resources such as storage capacity and processors, and this should be reflected in the model.

Hardware specification

Given TROPICS relative detailed hardware description it will serve as our hardware specification in order to make a realistic model. MicroMAS-2 CubeSat is the name of the nanosatellites used in TROPICS, they are equipped with solar panels, radio, battery, and an antenna [1]. Given that it has a radio and an antenna indicate that a satellite will be able to receive and send data concurrently.

2.1.3 Orientation

The satellite's orientation is necessary to consider when executing different tasks, and is therefore a relevant factor to include when constructing schedules for satellites. The orientation is important as the radio or antenna must be pointed in the direction of where another communication partner is at [6]. In addition, correcting the orientation will require the satellite to slew, prior to performing the planed task. As performing the slew may take some time, this should also be accounted for.

2.1.4 Station

A station is defined by a set of properties: location, antenna/signal strength, and taxation. Location consists of a longitude and latitude coordinates to determine where on Earth's surface the station is located, this is measured in relation to Earth's sea level. Antenna strength indicates how far the station is able to pick up transmissions, it will not take mountains or any other obstacles into consideration meaning that if the property is set to 10 km radius it is perfectly circular coverage with no interference.
Section 2.2 describes how a satellite may communicate with a station in more details. Taxation is a price that describes the bandwidth cost when data is either send to or from the station. The taxation property makes it so that some stations may be more attractive to use than others, due to their lower cost of usage [3].

2.1.5 Offset

We assume that a convoy consist of multiple satellites, following the same orbit, but with a different offset. This can be imagined as the first satellite being deployed one minute before the next. The consequence of offsets is that it will cause the satellites to reach stations and other windows at different times.

2.1.6 Communication

Internal communication is in this context defined as a transfer of data occurring between two or more satellites within the same convoy.
External communication is defined as communication to anything outside of the convoy, involving at least one satellite from within the convoy. External communication ca be made with stations, other convoys, or even some 3rd party satellites.

2.2 Orbit

In prior research, tasks a satellite could execute were restricted by making them dependant on windows which opened and closed according to the satellite's position in its orbit. This has previously been implemented [2], however the implementation was simplistic as the satellite was assumed to follow a perfectly circular orbit and therefore inaccurate compared to realistic orbits. Real orbits change over time as they are oval, to some degree, and the Earth's rotation also affect which parts of the surface that the satellite flies above.
The approach has been changed for this report, with the purpose of making it more realistic. To better capture when a satellite is able to transmit its data to stations, several variables must be known; the trajectory of the satellite, and the stations' location on Earth, and their signal strength.
A satellite is only able to communicate with a station when they are in close proximity of one another. To determine whether they are close enough to communicate, a distance is calculated using the longitude and latitude of the satellite and the station. If this distance between the satellite and the station is within the station's signal radius, the two are considered to be close enough for communication. This is not a flawless approach as there are some inaccuracies, especially with orbits that are oval. The satellites will traverse Earth's surface faster the closer they are, which means that their coordinates will change faster/slower depending on their position in the orbit. This might be a problem if a station's equipment is not strong enough to communicate with the satellites if their altitude is too high. We assume that all of the stations that are defined as applicable for receiving and/or sending data has equipment that is powerful enough to communicate with the satellites, regardless of their altitude.

2.2.1 Orbit Types

There exist different types of orbits which defines how a satellite revolve around its celestial body, in this case Earth. These may differ in distance to the surface, and objective of the orbit.

The common orbit types we will describe is in the list below: [7, 8]

- Low Earth Orbit (LEO) is specified by often having an orbit time of 90 minutes as the velocity of the satellite is very high, which is required as its altitude is lower than most other orbit types, typically resulting in the satellite orbiting 200 – 1200 kilometres above Earth.
- Sun-synchronous orbit revolves about every 90 minutes, and the orbit comes close to the poles. The satellite is always in the sun when it orbits Earth.
- Geosynchronous orbit is characterised with the satellite having an one day length revolution, meaning that the difference in Earth's and the satellite's latitude is close to being constant
  - Geostationary orbit is a sub category, similar to the previous mentioned orbit, and has a revolution equal to a day's length, additionally it maintains the latitude and longitude difference such that it becomes stationary above a point on Earth. Geostationary orbits are only achievable if the orbit is placed at Earth's equator.

2.2.2 Orbit Shapes

Most of the orbit types can use one, or both, of the two orbit shapes, circular or ecliptic. The circular orbit has a constant altitude where the elliptic orbit changes the distance to Earth.

To launch a satellite into any orbit a certain speed and acceleration is required, Figure 2.1 shows a satellite in a circular orbit around Earth along with the necessary factors that impacts the satellite's motion. Equations for calculating the speed and acceleration required to stay in a circular orbit can be seen in Equation (2.1). These calculations uses the gravitational pull (g), Earth's mass (m), and the satellite's orbit radius o) [9].

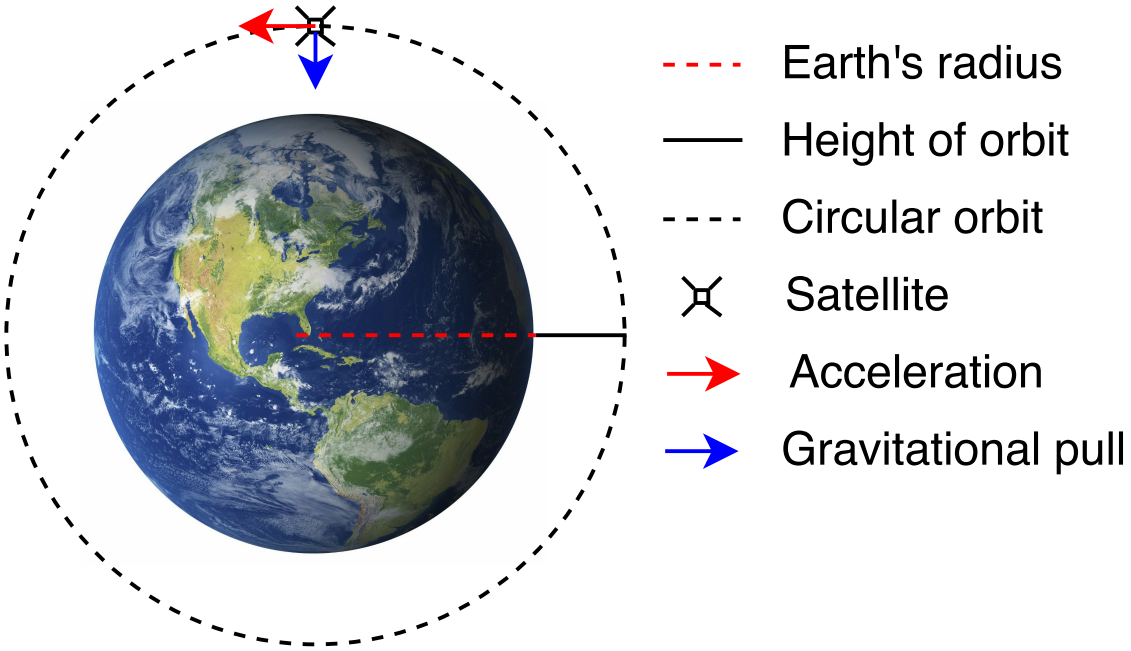

Figure 2.1: Satellite with a circular orbit [10]

An example calculation of a LEO with an orbit height of 160 kilometres can be seen in Equation (2.2). g is Newton's universal gravitation constant ($6.673 * 10^{-11}$), m is $5.98 * 10^{24}$ and o is Earth's radius plus the height of the orbit from Earth's surface in metres ($(6.37 * 10^6) + 160000$).

$$\begin{aligned} speed &= \sqrt{\frac{g \cdot m}{o}} \\ acceleration &= \frac{g \cdot m}{o^2} \end{aligned} \tag{2.1}$$

$$\begin{aligned} speed &= \sqrt{\frac{(6.673 \cdot 10^{-11}) \cdot (5.98 \cdot 10^{24})}{(6.37 \cdot 10^6) + 160000}} = 7817.26 m/s \\ acceleration &= \frac{(6.673 \cdot 10^{-11}) \cdot (5.98 \cdot 10^{24})}{((6.37 \cdot 10^6) + 160000)^2} = 9.358 m/s \end{aligned} \tag{2.2}$$

Equation (2.3) shows the calculation for finding the time it takes to make a revolution in a circular orbit, this equation uses orbital height, gravitational pull and Earth's mass along

with some other constants. In Equation (2.4) the calculation for the previous example is used to calculate one revolution at the height of 160 kilometres, which results in a duration of 87.47 minutes.

$$revolution = \sqrt{\frac{4 \cdot \pi^2 \cdot o^3}{g \cdot m}} \tag{2.3}$$

$$revolution = \sqrt{\frac{4 \cdot (3.1415)^2 \cdot ((6.37 \cdot 10^6) + 160000)^3}{(6.673 \cdot 10^{-11}) \cdot (6.37 \cdot 10^6)}} = 5248.54 \; seconds = \; 87.47 minutes \tag{2.4}$$

A problem with circular orbits is that they are difficult to maintain. For example, other celestial bodies than Earth will have a gravitational pull on the satellite which will influence its orbit. Elliptic orbits are more common and an orbit of this type can be seen in Figure 2.2. Two new terms are introduced with this orbit, periapsis and apoapsis. Periapsis is a point in the orbit where the satellite is closest to the Earth, where apoapsis denote the otbit's furthest point from Earth. The new terms can also be used to define a circular orbit, by having apoapsis equal to periapsis.

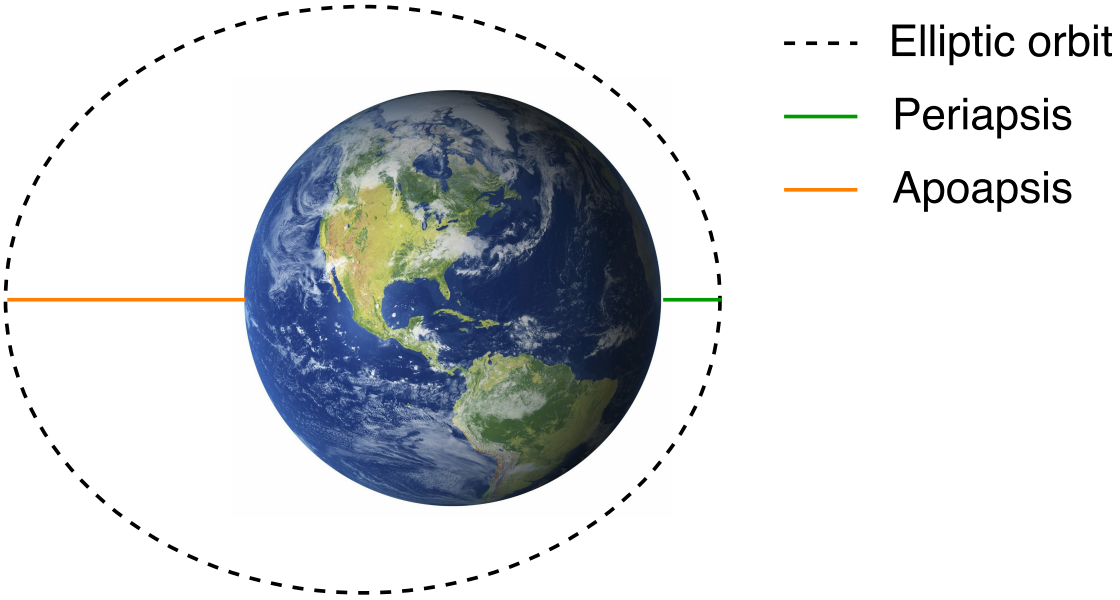

Figure 2.2: Satellite in an elliptic orbit around Earth [10]

This conclude the most common types of orbit shapes that satellites follows. For the rest of this report the elliptic LEO will be used as we find it the most relevant to work with.

2.2.3 Coordinates

The satellites in the convoys will gather data which they will need to send to the stations on Earth. Given that a satellite is not always within reach of a station, the concept of windows of opportunities emerges, as a satellite may only perform certain task when it is within a window. To calculate the position of a satellite in correlation to Earth, the orbit for the satellite must be known, along with its starting position and Earth's current

rotation. When these variables are known, it is possible to calculate how far a satellite has moved and how much Earth has rotated after a given duration. This information makes it possible to project the satellite's position down on Earth's surface, thereby obtaining the longitude and latitude coordinates. These coordinates can then be used to calculate the distance to nearby stations of which their longitude and latitude are already known [11].

Two-line element set (TLE) is a data format used to calculate the position of a satellite in orbit projected on Earth. In Table 2.1 and Table 2.2 a description for each of the fields in the data format can be seen. The fields that have an effect on the result are fields 7 ,8 ,11 on TLE line one, and 3–6, 8 on TLE line two [12].

Field	Characters	Description
1	1–1	Line number
2	3–7	Satellite number
3	8–8	Classification
4	10–11	International Designator (Last two digits of launch year)
5	12–14	International Designator (Launch number of the year)
6	15–17	International Designator (piece of the launch)
7	19–20	Epoch Year (last two digits of year)
8	21–32	Epoch (day of the year and fractional portion of the day)
9	34–43	First Time Derivative of the Mean Motion divided by two
10	45–52	Second Time Derivative of Mean Motion divided by six
11	54–61	BSTAR drag term
12	63–63	The number 0
13	65–68	Element set number. Incremented when a new TLE is generated for this object
14	69–69	Checksum

Table 2.1: TLE line one. The important fields are marked in bold.

Field	Characters	Description
1	1–1	Line number
2	3–7	Satellite number
3	9–16	Inclination (degrees)
4	18–25	Right ascension of the ascending node (degrees)
5	27–33	Eccentricity
6	35–42	Argument of perigee (degrees)
7	44–51	Mean Anomaly (degrees)
8	53–63	Mean Motion (revolutions per day)
9	64–68	Revolution number at epoch
10	69–69	Checksum

Table 2.2: TLE line two. The important fields are marked in bold.

As mentioned earlier, it is necessary to know how many degrees Earth have rotated in its current cycle, this information is used as a part of the TLE, specifically in field 7 and 8 in Table 2.1. The rotation us used to define the year, day in this year, and time of the day. Field 11 describes the drag coefficient for the satellite.
TLE line two introduces a few new terms [12]:

- Inclination describes the angle of the orbit, where a 0 degree inclination results in an orbit orbiting Earth's equator and 90 degrees result in an orbit around the poles

- Right ascension of the ascending node (RAAN) is an offset for the orbit in relation to the poles
- Eccentricity describe the orbit's form, the value goes from 0 to 1, where 0 is circular and anything else is elliptic
- Argument of perigee is also an offset in degrees from RAAN
- Mean motion defines how many revolutions the satellite makes per day

By setting these parameters it is possible to calculate the longitude and latitude coordinates for the satellite. Listing 2.1 shows the script we have constructed to generate the longitude and latitude coordinates. Line 9 and 10 in the script constructs the TLE sets with the function *construct_line_one* and *construct_line_two* that is supplied with the relevant parameters as described above. Line 13–14 computes the latitude and longitude based on the supplied parameters, afterwards there is a conditional statement to add the variable *long* correctly, lastly *long* and *lat* is appended as a set to the *LL* list. The list is the final output and contains all the longitude and latitude coordinates for every minute of the schedule.

```
1  def generate_data(schedule_length):
2      LL = []
3      date = datetime.datetime(2018, 3, 7, 9, 40, 39)
4      print(date)
5      for x in range(0, schedule_length, 1):
6          day_of_year = date.timetuple().tm_yday + (date.hour + ((date.minute
               + x) / 60)) / 24
7          date_only = str(date).split(" ")[0].replace("-", "/")
8          date_year = str(date).split("-")[0]
9          line1 = construct_line_one(str(date_year)[2:], day_of_year)
10         line2 = construct_line_two("50.0000", "020.519", "0020247",
               "81.2115", "279.186", "16.30050059", "00000")
11         tle = ephem.readtle("OrbitName", line1, line2)
12         tle.compute(date_only)
13         lat = float(str(tle.sublat).split(":")[-3]) + float(str(tle.sublat)
               .split(":")[-2]) / 60 + float(str(tle.sublat).split(":")[-1]) /
                (60 * 60)
14         long = float(str(tle.sublong).split(":")[-3])
15         if long < 0:
16             long += (float(str(tle.sublong).split(":")[-2]) / 60 + float(
                   str(tle.sublong).split(":")[-1]) / (60 * 60)) * -1
17         else:
18             long += float(str(tle.sublong).split(":")[-2]) / 60 + float(str
                   (tle.sublong).split(":")[-1]) / (60 * 60)
19         LL.append([long, lat])
```

Listing 2.1: Generating longitude and latitude from TLE set

With the knowledge of where the satellite is at any given time, it is now possible to calculate the distance to the stations for a given point in time. A list for each satellite is produced, and contains information about when the satellites are able to communicate with one or more of the stations [13].

2.3 Scheduling

In this section we will explore different scheduling strategies. First an overview of some scheduling methods then a look at preemptive scheduling along with some of its drawbacks.

The most important responsibility of a scheduler is to avoid starvation i.e. one or more tasks are continuously prevented from finishing. Fairness it is the promise that all tasks will get to run and finish at some point. Considering different scheduling methods will help us to delimit scheduling options when constructing the models.

- First In, First Out (FIFO): The first task to request a resource gets it. If another task requests it before the first finishes, the other task will be queued and run as the next task. This means that FIFO fulfils the fairness property, as all tasks gets to run.
- Round Robin (RR): Cycles through the set of tasks allowing all tasks to be run an equal amount of times. This means that RR is a fair scheduler.
- Priority scheduling is divided into two categories Fixed Priority (FP) and Dynamic Priority (DP). Many priority based scheduling algorithms have the priorities tied to different attribute. A task's priority describes the importance of the task. Priority scheduling in itself does not guarantee fairness.
  - FP: All tasks have a predefined and never-changing priority. When a task finishes the next task to be started is the one with the highest priority that is available and ready.
  - DP: All tasks are prioritised, but unlike FP the priority may change based on the state of the system. This may be tied to some value such as, when the task was last executed.
- Shortest Remaining Time first (SRF): Is a DP scheduling method, where the priority is the remaining execution time. The general idea is that the queue is sorted by remaining execution time, starting with the one that will finish first. A scenario may be that we want to complete as many tasks as possible in the shortest amount of time. In this case SRF is very effective as it will always start the task that has the shortest execution time. However, SRF does not guarantee fairness.

Non-preemptive schedulers do not allow one task to take hold of the CPU if it is already being used by another task. This can be problematic as some tasks take a large amount of time to finish. Because of our earlier research, we have chosen to look into preemptive scheduling as it might be helpful when working with tasks that may only be executed at certain time intervals. [2]

2.3.1 Preemptive Scheduling

A preemptive scheduler is able to pause the active task and have a more urgent or important one take over.Consider the set of tasks presented in Table 2.3.

Task ID	**Warm-up time**	**Execution**	**Interval**	**Priority**
0	3	3	20	3
1	4	6	20	2
2	2	4	20	1
3	0	8	30	0

Table 2.3: Set of tasks used in Table 2.5

Here a task has been described with a number of attributes;

- Task ID — an identifier for the task
- Warm-up time — initial delay for the task i.e. the time before the task becomes ready
- Execution — the time required to complete the task
- Interval — how often the task should be executed
- Priority — describes the importance of a task

Table 2.4 illustrates what would happen without preemption. In this example, task 0 is started at time 0 and is continued until it is complete, even though other tasks of higher priority becomes available. In this case, task 2 does not finish within its interval of 20 time units.

Time	0–8	8–11	11–17	17–21
Task	3	0	1	2

Table 2.4: Scheduler showing which task is being processed by a single core CPU

An example of the use of preemption can be seen in Table 2.5. In this task 3 is the first one to be executed as it has no warm-up time, but is preempted by task 2 after two time units, which then itself gets preempted by task 0 after one additional time unit. After task 0 has finished task 1 is then started and finishes without being preempted further. Thereafter task 2 and then task 3 finishes. Then a small waiting period appears where no task is active.
The example shows how preemption can be very useful as it allows all tasks to continuously finish within their respective interval.

Time	[0–2)	[2–3)	[3–6)	[6–12)	[12–15)	[15–21)	[21–22)	[22–23)	[23–26)
Task	3	2	0	1	2	3	-	2	0

Time	[26–32)	[32–35)	[35–42)	[42–43)	[43–46)	[46–52)	[52–55)	[55–56)	[56–62)
Task	1	2	3	2	0	1	2	3	–

Table 2.5: Schedule showing which task is being processed by a single core CPU with preemption

Starvation among a set of tasks can be avoided by tweaking the intervals of each task, e.g. by increasing the interval of all tasks such that all of them can be completed. However, this it is not always possible to just increase or decrease the interval, especially not if they are bound by other resources that requires the task to be run at a specific interval.

Other ways of avoiding starvation is to use priority- boosting or inheritance. Priority inheritance increases the priority of a task when another task tries to preempt it, so that it can finish its critical section before giving up control of the shared resource. Priority boosting works by boosting a task's priority when it takes hold of a shared resource, the priority is sat to the highest priority of all the tasks that share the resource [14].

If a preemptive SRF is used it will keep track of how much time it has already spent on a task if the task gets preempted. This means that if a task with a long execution

time is executed, it will typically be preempted a number of times, but as it progresses its remaining execution time will be shorter thereby increasing its priority which results in it getting preempted less [14].

No strategy is necessarily better then the others even with or without preemption. However, in most cases some tasks needs to be executed more often than others. This often leads to a scheduler that utilises preemption, such that the more important tasks can be executed within their interval.

2.3.2 Choice of Scheduler

For this project we will be using a preemtive scheduler with DPs. The reason for using a preemptive scheduler is that some tasks, such as sending data to Earth, are restricted to being executed only when the satellite is in close proximity to one or more stations that is accepting data. Because of this it should be possible to pause the active task and start the more restricted one.
Using a DP scheduler allows for modified priorities dependant on the state of the system. This may be relevant in situations where there are no more storage available for storing data, as this would make data collection tasks pointless.

2.4 UPPAAL Version

As previously mentioned a model for generating a schedule for a single nanosatellite has been made [2]. For this UPPAAL Cost Optimal Reachability Analysis (CORA) were used, in conjunction with UPPAAL Statistical Model Checking (SMC). In this project we will explore the possibility of generating a schedule for a convoy of multiple satellites at once. We believe that modelling multiple satellites at once will cause a significant increase in the state space and we will therefore need to make choices to accommodate this. For the first part we will make a deterministic model, as it will allow for a larger implementation at limited effect to the state space. The model is deterministic as there will never be more than one choice when the model has to select a transition or delay.

2.4.1 UPPAAL

A UPPAAL model consists of one or more templates and some global decelerations. Each template may have its own private decelerations that is inaccessible for other templates. Figure 2.3 is an example of a template that consists of three locations and two edges.

One of the locations in a template must be initial in order to define the starting state. In addition a location can be marked as urgent or committed, which means that time is not allowed to pass while the location is active i.e. while the location is part of the current state. The difference between urgent and committed is that committed locations are prioritised, meaning that the next transition taken must include taking at least one edge going from a committed location. Locations can be named and have an invariant assigned. A location's invariant must be evaluated to true, before a transition can activate it, and for all time where the location remains active. Invariants are coloured `magenta`.

Besides the locations and local declarations, templates also consists of edges that connects the locations. An edge may have a guard, select, synchronisation, and/or update statement. Guards are boolean repressions that ensure that an edge is not taken prematurely, and are coloured `green`. Selects are used for capturing temporary variables, and are coloured `yellow`. Synchronisations are used for simultaneously activation of edges across multiple templates, and are coloured `light blue`. Synchronisations are suffixed with an exclamation mark if it is used to call the synchronisation, as opposed to a question mark for those that are receiving. Updates are used to update variables and to call either global or local defined functions, and are coloured `dark blue`.

An important part of UPPAAL, is the ability to formulate and ask queries that can test if some properties are upheld, such as; is it possible to reach a certain location, and is it possible to do so within a given time frame.

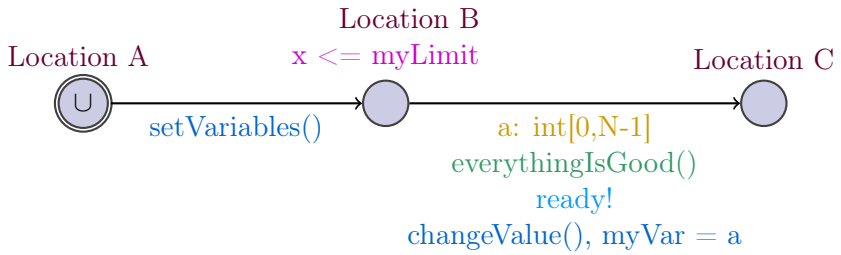

Figure 2.3: Example UPPAAL template

Deterministic Model 3

In this chapter we will explore the scheduling problems that are associated with satellites communicating with other satellites within a convoy and with one or more stations. A possible solution will be presented and designed in the form of a UPPAAL SMC model which will be benchmarked and experimented on in order to test the scalability and feasibility thereof.

3.1 Scenario

In this section, we will explore a model that modules each satellite within a single convoy individually. The convoy's satellites all follow the same orbit around Earth, and are able to communicate with a specified number of stations, as seen in Figure 3.1. The primary objective of the satellites is to collect and transfer data to the stations on Earth. A satellite can have up to three objectives (*i*) gather data (*ii*) transfer data to another satellite (*iii*) send data to one of the stations. Additionally each satellite has a memory storage unit with a maximum capacity which should be considered along with the satellite's orientation as some tasks may require the satellite to either face Earth or another satellite.

We would like to build a model that can test a simple strategy for collecting and transferring data between satellites within a convoy and the various stations. The model should be constructed such that a schedule can be produced which describes what the individual satellites does for the duration of the schedule.
We will test if it is feasible to generate a schedule for such a system, the feasibility is measured in how many satellites that can be added, and in how long it takes to generate a schedule of a certain duration. In this scenario a convoy is expected to contain $\approx 30 - 40$ satellites [6].

When testing the feasibility, additional information will be collected in order to determine which factors have the largest impact on the performance of the model.

Throughout the rest of this chapter we will describe the elements that goes into this scenario. Thereafter an implementation of the system will be described and lastly a comparison of what factors contributes most significantly in regards to; time taken to generate a schedule, payload efficiency, the amount of data transferred, and the accumulated taxation of transferring data to Earth.

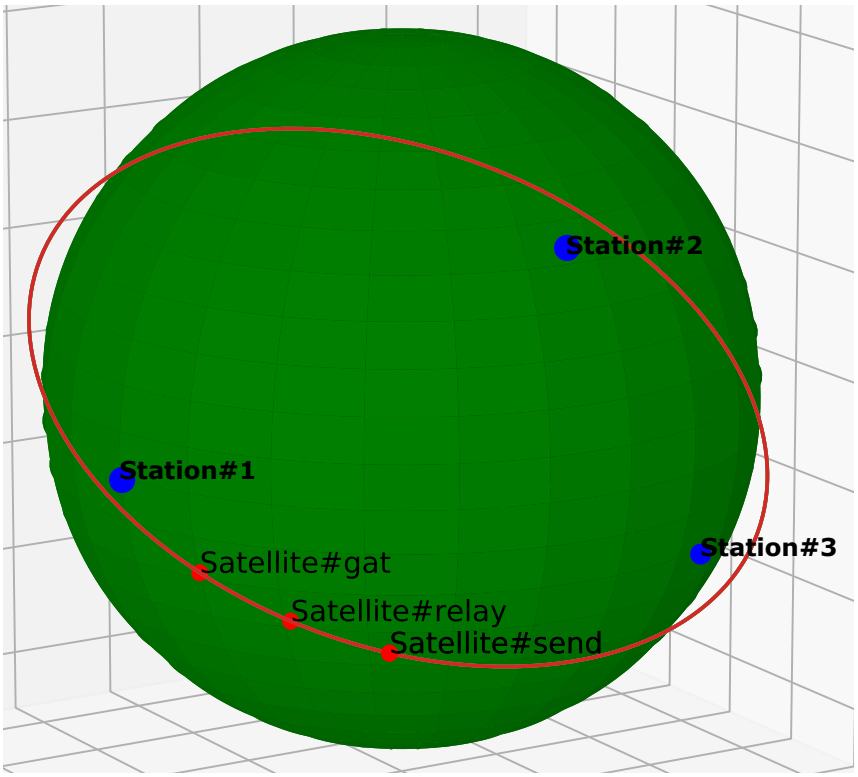

Figure 3.1: Three satellites orbiting Earth with three stations

3.2 Scheduler

As concluded in Section 2.3 we will be using a preemptive scheduler with DP. This section will describe how such a scheduler can be designed.

All tasks have been given a base priority which will change during simulation. The priorities will be tied to the satellite's current storage level. Thresholds that describe when the satellite is at high or low storage capacity have been defined in order to change the priorities.
In scenarios where the satellite is at high storage the priority of all tasks which allow it to remove data is multiplied by two. Likewise when at low storage tasks that allow for the satellite to accumulate data are multiplied by two.
To further change the priorities based on the current storage, tasks that accumulate data will have a priority of zero when the satellite is at maximum storage. Similarly, when the satellite has no data, tasks that remove data will be at zero priority.
When a task is completed its priority is reset to its original value. Additionally, when a task is completed, all other tasks that have a priority of zero is also reset to their original value. This is done in order to regularly re-evaluate the priority of the different tasks.

In order for a task to be preempted, another task of higher priority needs to be available. When a task is preempted, the priority of the preempted task will increase to match that of the preempting task. In this way the preempted task has a higher chance of finishing afterwards, while not being increased so much that it takes back control right away.
In order to resume a preempted task it must be checked if the task can still finish within its deadline. Also if it is needed that the satellite is within range of a station, the task

must be able to finish before the station is out of reach.

3.3 The Deterministic Model

A model has been created with the goal of testing a simple strategy for collecting and transferring data between satellites within a convoy and the various stations. The satellites are modelled individually within a single convoy and a simple one-way communication system has been implemented that allows the satellites to communicate internally within the convoy.

The model consists of 4 different templates:

- Scheduler
- Processor
- Satellite
- CheckRunnable

The Scheduler, Processor, and Satellite templates are instantiated once for every satellite. The CheckRunnable template is a singleton that is used whenever a satellite has to check which of its tasks is currently available to be executed. This is done by checking if the satellite is close to a station if this is required, and whether or not the task can finish before this is no longer the case.

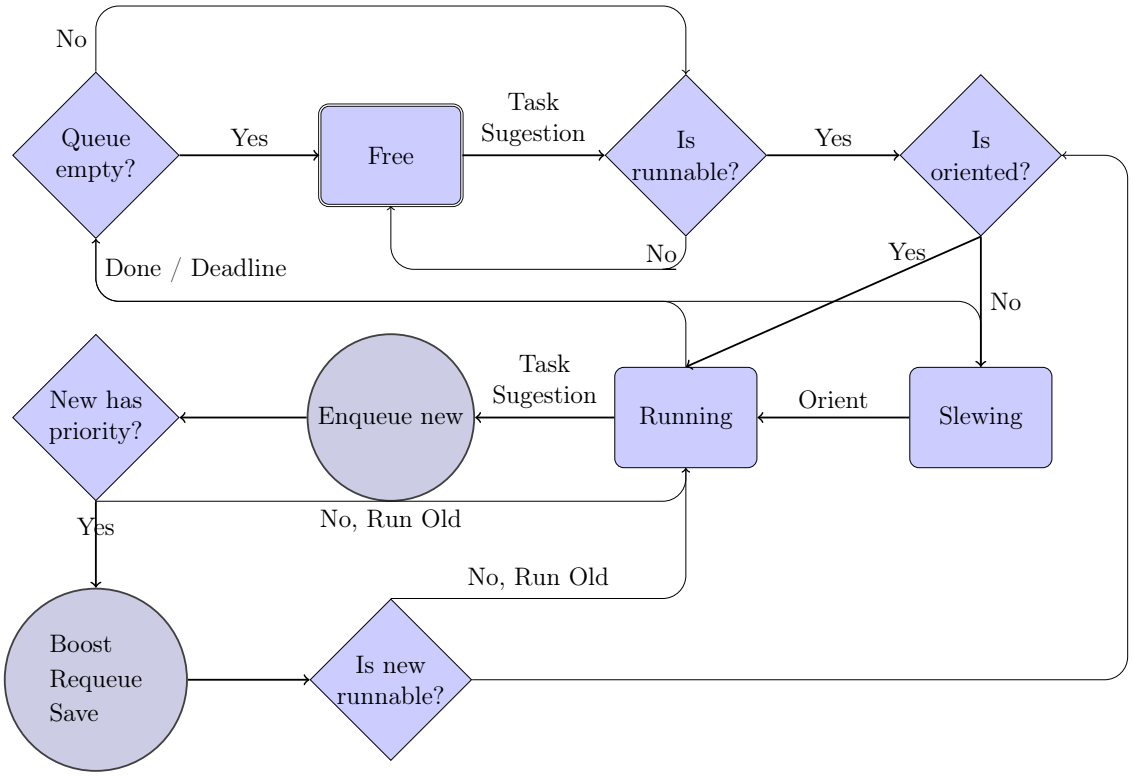

Figure 3.2: Simplified representation of the deterministic model with Free as the initial state

The Scheduler template maintains a queue of tasks and makes decisions on which task should be executed, either by modifying the queue or by preempting the current task.

Additionally, it is responsible for initialising the communication with another satellite whenever an internal communication task is to be executed.
The Processor template is waiting for synchronisations that the Scheduler and Satellite template is initiating as these synchronisations dictate when a task should be started, preempted, aborted due to internal communication failures, or when to stop slewing. It is responsible for monitoring the execution times of the tasks such that it may report when a task is completed or aborted due to a deadline violation. The Processor template will additionally modify the priority of tasks whenever they are preempted.
The Satellite template's primary objective is to suggest tasks that the scheduler should enqueue. When doing so it may modify the priority of tasks based on the satellite's current storage level. Also, the template will slew the satellite whenever the orientation does not match that required to execute the current task.

A simplified representation of the model can be seen in Figure 3.2. The figure leaves out some intermediate states and checks which are not necessary to include in order to illustrate the general flow and logic of the model. The actual model templates can be found in Appendix A.

3.3.1 Data Structures

Two structures, `TaskDescription` and `SatDescription`, have been defined in order to describe the tasks and satellites. Listing 3.1 shows the `TaskDescription` struct which contains the information used for describing a task.

```
1     /** Task Description */
2     typedef struct {
3     bool depend_station[STATIONS];
4     int execution_time;
5     int deadline;
6     int prio;
7     int data_rate;
8     int orientation;
9     } TaskDescription;
```

Listing 3.1: `TaskDescription` struct

- `depend_station[STATIONS]` is used to restrict the task such that it may only be executed when inside range of the specified stations. The field is an array where each index corresponds to a station, and a true value means that the task may only be executed when within reach of that station. If more than one station is selected, the task may be executed when any of these stations are within reach. `STATIONS` is the total number of stations.
- `execution_time` is the time it takes to complete the task
- `deadline` is the maximum duration a task can stay active before it is aborted
- `prio` is the initial priority of the task. When the Scheduler template is ordering its queue of tasks, it is doing so based on what it is able to start and what task has the highest priority. The priority of a queued task can be modified in various ways.
- `data_rate` is a value that specifies the change in memory per time unit it is executed. Tasks that collects data has a positive rate while tasks that sends data has a negative rate. A value of 0 indicates that it has no effect on the memory.

- `orientation` symbolises that the satellite must point its antenna in a certain direction. For example, in order to send data to a station the satellite must point its antenna towards Earth. A value of -1 signifies that the task is not restricted by the orientation of the satellite.

The model is made under the assumption that a satellite that is receiving data from another satellite, will never abort the task due to an exceeded deadline. The model will not handle that case and it should therefore be avoided as the transferring satellite will not be notified. The `receive` task should therefore be defined with a deadline that provides enough time for the worst case slewing time, and the execution time of the task. The transferee will be notified in case that the receiver will be preempted.

The defined tasks in the model uses these values for the `orientation` variable:

- 0 — forward
- 1 — Earth
- 2 — backwards

Forward means that the satellite is facing in the same direction as it is moving in the orbit. Earth means that the satellite is pointing towards Earth. Backwards means that the satellite is facing in the opposite direction of what it is moving in the orbit.

Listing 3.2 shows the `SatDescription` struct which contains the information used to describe a satellite in the deterministic model.

- `offset` is a duration of which the satellite must wait before it enters orbit. The satellites in a convoy are travelling in the same orbit, but they differ in how far they are in it. The offset means that is it not always the case that all satellites is within range of the same station.
- `orientation` is the initial value for which direction the satellite is facing with its antenna. This value changes as the satellite performs different tasks requiring different orientation.
- `memory` is a variable that may be incremented or decremented whenever a task with a *data_rate* is completed. The initial value of this field determines how much memory the satellite starts with. The memory can not go below 0 or surpass a maximum capacity defined within the deterministic model.
- `available_tasks[N]` is an array that specifies what tasks the satellite is able to execute. N is the total number of tasks.

```
1  /** Satellite Description */
2  typedef struct {
3  int offset;              // when does the satellite "start"
4  int orientation;         // what is the current orientation of the
       satellite
5  int memory;              // current amount stored
6  bool available_tasks[N]; // what tasks can the satellite perform
7  int suggested_task;      // what task should be performed next
8  } SatDescription;
```

Listing 3.2: `SatDescription` struct

- `suggested_task` is updated whenever the Satellite template is suggesting a new task for the Scheduler template to queue

3.3.2 Satellite

The Satellite template starts by waiting until its satellite's offset is met, which then triggers a synchronisation that starts the Scheduler template as well. Afterwards, the Satellite template will try to suggest a new task for the Scheduler template by calling the `suggest_task` function. The function is displayed in Listing 3.3. The first loop modifies the priorities of the tasks according to the current amount of stored memory, such that tasks that gather data are prioritised when the storage is low, and tasks that transfers memory are prioritised when much data is already stored. The maximum priority for a task is 100.
In the second loop, the task with the highest current priority is selected and `suggested_task` field is set. If two or more tasks are tied in priority, the first observed of the tied tasks is selected. New tasks are suggested at an interval that is determined by a global constant.

The Satellite template is responsible for slewing the satellite whenever it is required to do so. It takes an amount of time to complete one slew iteration, and the new orientation is determined with the function `turn_satellite` which can be found in Listing 3.4. The function is very simple as it just increments or decrements the orientation based on the one required to perform the current task and the satellite's current orientation. As explained in Section 3.3.1, there are currently three possible orientations, specifically 0, 1, and 2.

3.3.3 Scheduler

The Scheduler template receives suggestions, from the Satellite template, regarding which task is best to execute. Best meaning, the task with the highest priority. The task is immediately added to a queue if it is not already present in it. The addition of the task will trigger a reordering of the queue, which will result in the task with the highest priority being placed in the front.
Besides having the highest priority, the task must also be available at the current point in time. This is ensured by a synchronisation which prompts the CheckRunnable template to report which tasks that are ready to be executed The time it takes to slew the satellite is not accounted for when the CheckRunnable template checks the tasks. The windows are opened and closed according to when the satellite is within reach of the stations, as described earlier in Section 2.2.3. When a task has been selected for execution the Processor template will start executing the task and ask the Satellite template to start slewing if this is required.
Even when the Scheduler template is occupied with its current task, it may still receive suggestions from the Satellite template. When a suggestion arrives, its priority is compared against the current task's priority, and a decision based on this result determines whether the current task should be preempted or not. When a task has been preempted, it will be added to a local queue in the Processor template, which it will manage such that the tasks may be resumed later.

Whenever an internal communication task is scheduled, such as the `transfer` task,

```
1  void suggest_task() {
2       int i, j, k, count = 0, high = -1;
3       selected = -1;
4       for (i = 0; i < N - 1; i++) {
5            if ((sats[sat_id].available_tasks[i]) {
6                 if (sats[sat_id].memory <= LOW_MEMORY ) {
7                      if (jobs[i].data_rate > 0) {
8                           if (priorities[sat_id][i]*2 <= 100) {
9                                priorities[sat_id][i] = priorities[sat_id][i]
                                      * 2;
10                          }
11                          else {
12                               priorities[sat_id][i] = 100;
13                          }}
14                     else if (jobs[i].data_rate < 0) {
15                          priorities[sat_id][i] = 0;
16                     }}
17                else if (sats[sat_id].memory >= HIGH_MEMORY) {
18                     if (jobs[i].data_rate < 0) {
19                          if (priorities[sat_id][i]*2 <= 100) {
20                               priorities[sat_id][i] = priorities[sat_id][i]
                                      * 2;
21                          }
22                          else {
23                               priorities[sat_id][i] = 100;
24                          }}
25                     else if (jobs[i].data_rate > 0) {
26                          priorities[sat_id][i] = 0;
27                     }}}}
28      for (i = 0; i < N - 1; i++) {
29           if (sats[sat_id].available_tasks[i] && priorities[sat_id][i] >
                 high) {
30             if(i != RECEIVE_INDEX ) {  // it is invalid to suggest a
                   receive task (It's the transferee that takes initiative)
31               selected = i;
32               high = priorities[sat_id][i];
33             }}}
34      sats[sat_id].suggested_task = selected;
35 }
```

Listing 3.3: The function `suggest_task()`

the behaviour of the Scheduler template is changed as new conditions are introduced. When a satellite wants to communicate with another satellite, it will need to make sure that it is aligned with the one it wants to communicate with. When aligned, multiple synchronisations will be activated such that the other satellite may decide whether or not to accept the communication task. It may need to preempt the task it is currently executing in order to accept the communication, it may also decline the task which will cause the initiating satellite to abort the `transfer` task. Listing 3.5 shows a boolean function checking whether or not the satellites are aligned correctly. If the satellites are aligned, the transfer may begin. Otherwise the function returns false and the satellite continues slewing.

```
1 /** Change satellite orientation */
2 void turn_satellite() {
3      if (sats[sat_id].orientation < jobs[active[sat_id]].orientation) sats[
           sat_id].orientation ++;
4      else if (sats[sat_id].orientation > jobs[active[sat_id]].orientation)
           sats[sat_id].orientation --;
5 }
```

Listing 3.4: The function `turn_satellite()`

```
1      /** Are the satellites alligned */
2      bool rdyToTrans(){
3           if (sats[com].orientation == receive.orientation && sats[sat_id].
                orientation == transfer.orientation) {
4             return true; }
5           else return false;
6      }
```

Listing 3.5: The function `rdyToTrans()`

3.3.4 Processor

The Processor template receives tasks to execute from the Scheduler template and is responsible for monitoring the tasks' time usage such that they can be marked as completed if they reach their execution time, or aborted if their deadline is exceeded. The Processor template will also instruct the Satellite template to slew if the current orientation is divergent from what is required of the new task.
The Processor template will additionally reset the priorities of some tasks whenever a task is finished. The task that was finished will be set to its initial priority together with those that had their priority set to zero. A priority of zero may be assigned to a task when examined by the Satellite template because of the current amount of stored data.

When a task has been preempted, its current execution time is saved such that it will not have to start over if it is resumed later. The execution time is represented by a clock, but it is not possible to store clock values in other variables so the Processor template is trying to approximate the execution time. This is done by utilising a variable that is incremented every time a new task suggestion is made in the Satellite template. When the task is preempted its progress is set to the current number of task suggestion iterations, multiplied by the task suggestion frequency. The iterator is reset by the Scheduler template whenever a new task is started.
This approach is not flawless as it is only an approximation. Table 3.1 demonstrates how this approach sometimes miscalculates the actual execution time. The blue rows shows when a new task becomes the active task. Task 1 has an execution time of 7, task 2 has one of 5, and the constant which decides the suggestion frequency is 3.
Step one: The active task, the one being executed, is 1 in the beginning and it is executed for 3 time units before task 2 is suggested which then preempts the execution of task 1. Task 1's progress is calculated to be $1 * 3$ as one iteration has passed. This is correct as task 1 has indeed been executed for 3 time units.
Step two: Task 2 is then executed until it is finished, which happens after an additional 5 time units.

Step three: Task 1 can be resumed again, as seen when the time reaches 8.
Step four: Task 1 is preempted again shortly after as task 2 is suggested again after only 1 time unit.
This is a problem as task 1 has only been executed for 1 time unit, but the model believes that it has been executed for 3 before it was preempted. This means that the total execution time should be 4, but it is stored as 6. The consequence of this is that when task 1 is resumed again at time 14, the Processor template is only executing it for 1 time unit before it is completed whereas it actually needs to be executed for 2 more time units.

Time	Suggestion iterator	Execution timer task 1	Time used task 1	Execution timer task 2	Active task
0	0	0	0	0	1
1	0	1	0	0	1
2	0	2	0	0	1
3	1	3	0	0	1
3	0	0	3	0	2
4	0	0	3	1	2
5	0	0	3	2	2
6	1	0	3	3	2
7	1	0	3	4	2
8	1	0	3	5	2
8	0	0	3	0	1
9	1	1	3	0	1
9	0	0	6	0	2
10	0	0	6	1	2
11	0	0	6	2	2
12	1	0	6	3	2
13	1	0	6	4	2
14	1	0	6	5	2
14	0	0	6	0	1
15	1	1	6	0	1
16	1	0	0	0	-1

Table 3.1: Example of time approximation inaccuracy

This inaccuracy can be avoided by either setting the suggestion frequency constant to 1, or by ensuring that all tasks is dividable by the constant. Another option is to declare a separate clock for every task, for every satellite. These clocks should then be used as stopwatches and only be incremented when the associated task is being executed. The problem of this approach is the modularity as the current approach is more general and does not need to be updated or rewritten every time a new task or satellite is added to the model.

When a task is dependant on a station, the Processor template will have to chose between the available stations at the time the task is started. The chosen station will be the one that is cheapest amongst those that are en reach and the tasks is dependant on. Listing 3.6

```
1     /** Increments cost of bandwidth */
2     void calc_cost() {
3          int i = 0; price = 9999;
4          for (i = 0; i < STATIONS; i++){
5               if(jobs[task].depend_station[i] && runnable[task] &&
                    win_active[i]) {
6                 if (price > station_price_rate[i] * jobs[task].
                      execution_time){
7                   price = station_price_rate[i] * jobs[task].
                        execution_time; //choose cheapest
8                 }
9               }
10         }
11         if (price == 9999) {
12              price = 0; }
13    }
```

Listing 3.6: The function `calc_cost()`

shows the function that chooses the cheapest station and calculates the cost for executing the current task. The calculated value represents an actual price it would cost the owner of the satellite to transfer the data.

3.4 Experimenting with the Deterministic Model

It is now possible to perform different experiments, which allow us to observe the effect of the various features and properties of the deterministic model.

The goal for these experiments is to determine which of the features and values have the most and least significant impact on the computation time of generating and outputting a schedule, and how much it deviate from the initial configuration. This is important as it is desirable to make a non-deterministic model, but to do so a reduction in the state space is needed. We hypothesise that a non-deterministic model will encounter a large state space if all of the features from the analysis are implemented. If we are able to determine that some features can be avoided, we may be able to reduce said state space. The hypothesis is based on previous experience with the UPPAAL tool suite [2].
We will be observing the following variables:

- **Program time** the time taken to complete the query and write the resulting trace to a file
- **data values**:
  - **Data sent to Earth** describes how much data that have been sent to the stations on Earth
  - **Data gathered** describes how much data that has been collected via the `sat_gat` task, which is described in Table 3.2
  - **Data Transferred** describes how much data that has been transferred internally between the satellites
- **runs** is incremented each time a task completes before its deadline
- **delays** is incremented each time a task exceed its deadline
- **Idle** accumulated time spent idling

- **Wait** accumulated time spent waiting for another satellite to become ready for a transfer/receive task
- **Work** accumulated time spent actively executing a task
- **Slew** accumulated time spent slewing into position for a task
- **Cost of bandwidth** accumulated cost which increases every time communication with a station occurs

An additional set of experiments will be made in order to verify the correctness of the model, mainly the designed scheduling method, to see if it behaves as expected. These experiments will change the values of some of the constants in order to test that the chosen values actually affect how the model behaves. Some of the values that will be changed is the deadlines for the tasks, the thresholds for the data storage, and the priorities

All runs have a schedule length of 1440 minutes (one day), the length is a compromise between letting the model run for long enough time such that the experiments can be compared and differences may arise, and not taking to much time to compute. The problems associated with computation time is discussed later in Section 3.4.15.

The tables presenting the results of running the different experiments will be presented by comparing the values from each experiment to a base case. An initial configuration will serve as a base case representing a relatively correct scenario. The comparison will be presented as the exact values from the base case and the experiment, along with the percentage difference between the two. As the base case is assumed to be an accurate representation of the real application we hope to see the experiments yielding close to the same results, with a decreased program time.
When we are referring to one of the satellites, such as `Sat_0`, we are referring to the three instantiated templates Satellite, Scheduler, and Processor that combines to a represent a physical satellite.

3.4.1 Initial Configuration

First we made a run with the initial configuration setting a base case for the results and computation time. This initial configuration can be seen in Table 3.2 and Table 3.3 for the results.

Table 3.2 shows the global settings e.g. the number of tasks, satellites, and maximum storage capacity. In addition it shows the specification of different tasks, see Section 3.3.1, along with the specification for three types of satellites, `sat_gat` (Gatherer), `sat_trans` (Transfer), and `sat_sen` (Sender).
It can for instance be seen that the task `send_data` takes 20 units of time to complete and sends data at a rate of -2 per time unit, and the task `gather_new_data` takes 15 units of time with a rate of two. This means that 30 data can be gathered in one run, and 40 data can be sent to a station.

The three satellites may only execute the tasks listed in the Tasks column, i.e. `sat_gat` can gather and transfer data, but not send or receive data.

Most values seen will remain constant throughout the experiments, changing only a few for every run and changing them back before running the next experiment. A look at

				Memory		
Global Settings	Tasks	Sats	Stations	Max	Low	High
	4	3	2	100	30	75
TaskDiscription	Windows	Exe_Time	Deadline	Priority	Data_Rate	Orientation
send_data	(0,1)	20	26	9	-2	1
gather_new_data	(1,0)	15	21	6	2	0
transfer	(0,0)	5	17	5	-3	2
receive	(0,0)	5	11	4	3	0
SatDiscription	Offset	Orientation	Memory	Tasks	Suggested	
sat_gat	3	0	80	(0,1,1,0)	-1	
sat_trans	3	0	20	(0,1,1,1)	-1	
sat_sen	9	2	0	(1,0,0,1)	-1	

Table 3.2: Initial configuration for the experiments

Table 3.2 shows that the satellites can perform different tasks and have different initial states e.g. `sat_gat` has an offset of 3, a starting storage of 80 and can perform task one (`gather_new_data` — receive externally) and two (`transfer` — send internally), whereas `sat_sen` has an offset of 9, an initial storage of 0, and can perform task zero (`send_data` — send externally) and three (`receive` — receive internally).

Program time	Cost of bandwidth	Data sent to Earth	Data gathered	Data transferred
154	7435	640	750	990
		sat0	sat1	sat2
runs: send_data		0	0	16
runs: gather_new_data		10	15	0
runs: transfer		20	46	0
runs: receive		0	20	46
delays: send_data		0	0	0
dealys: gather_new_data		0	1	0
delays: transfer		16	0	0
delays: receive		0	0	0
Idle		382	223	784
Wait		576	51	0
Work		318	634	557
Slew		164	532	99

Table 3.3: Results of running the initial configuration

Figure 3.3 illustrates a snippet of the schedule generated running with the initial configuration. There are three rows for each satellite, the top one indicating which task is executed when, the middle one when it is slewing, and the bottom one showing when a satellite is waiting for another to communicate with. The different colours indicate different tasks, red is `send_data`, green in `gather_new_data`, blue is `transfer`, and purple is `receive`.

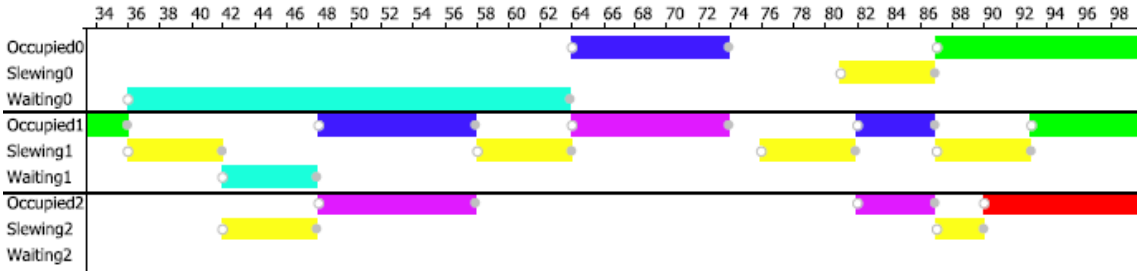

Figure 3.3: Generated schedule over the base case for three satellites

3.4.2 Experiment 1: Without Slew

Experiment: Remove the satellites' need to slew. This is done by setting the required orientation of all tasks to zero along with all initial orientations of all satellites.
Motivation: This experiment is performed to determine the effect of including slew in the deterministic model, since slew is a large component in the model. If the effect of no slew turns out to be minor while giving a significant reduction in computation time, it may not be necessary to include in a non-deterministic model.
Hypothesis: For this experiment we hypothesise

1. The `Slew-` and `Wait` clocks will become zero. Since there is no need to slew, there will be no need to wait for another satellite when communicating internally.
2. The `Work-` and `Idle` clocks will be higher. This is trivial as there is an expected decrease in slew- and wait time, hence this time must be used elsewhere.
3. There will be an increase in runs. As more time will be spent working, more tasks should be completed.
4. There will be fewer delays. As time is no longer spent slewing or waiting, there are more time to make up for being preempted.
5. An increase to all data values, sent, gathered, and transferred. As more time is spent working the data values should all be affected.
6. A decrease in program time. Since the states required for slew are no longer necessary it should be faster to run the experiment.

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	287	10505	920	1050	1275
Diff (%)	+86.0	+41.3	+43.8	+40.0	+28.8

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	23	**+43.8**
Runs: gather_new_data	10	10	**0.0**	15	25	**+66.7**	0	0	**0.0**
Runs: transfer	20	19	**-5.0**	46	66	**+43.5**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	19	**-5.0**	46	66	**+43.5**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	0	**-100.0**	0	0	**0.0**
Dealys: transfer	16	9	**-43.8**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	566	**+48.2**	223	278	**+24.7**	784	613	**-21.8**
Wait	576	307	**-46.7**	51	0	**-100.0**	0	0	**0.0**
Work	318	567	**+78.3**	634	1162	**+83.3**	557	827	**+48.5**
Slew	164	0	**-100.0**	532	0	**-100.0**	99	0	**-100.0**

Table 3.4: Results of running experiment 1

Results: The results can be found in Table 3.4.

1. Partially true. The `Slew` clocks were indeed 0 for all satellites. However, the `Wait` clock for `Sat_0`, while cut in half, was not zero. The transferring satellite is waiting for the receiving satellite whenever it itself is transferring to another satellite. It is only `Sat_0` which is waiting as it is the only satellite that may transfer to another satellite which itself is able to transfer.
2. True. Even though the percentages for the `Idle` clocks are sometimes lower than the base clocks, the sum of the `Idle` and `Work` clocks are higher than the sum of the same clocks in base.
3. True. There is a general increase in tasks completed.
4. True
5. True. There is a 20 to 44 percentage increase.
6. False. The huge increase in runs have caused the trace to include many more states which therefore significantly increases the program time.

Conclusion: Performing this experiment showed that not including slew, lead to significant variations to a majority of the observed values. We believe much of these variations are due to an oversimplification in how slew was removed.
Instead we propose a new experiment where the task execution time is increased to accommodate for the simplification. This new experiment will be used to conclude whether or not slewing should be included.

3.4.3 Experiment 2: No Slew with Increased Execution Time

Experiment: Remove the need for slewing, and increase the execution time of tasks We will increase the execution time of all tasks by the time it takes to slew one step, and modify the data value calculation so it does not include the additional time, and do the

same when calculating the bandwidth cost. Adding the time for one slewing step was chosen as the base case performed a total of 220 tasks, got delayed in performing 40 tasks, and had a total of 246 slews, thereby averaging approximately one slew per task started.
Reason: As described in Experiment 1: Without Slew, a significant variation in the data values was expected, when compared to the base case. Essentially removing slew is to remove time needed in order to complete a task. Because of this we perform this additional experiment to get more accurate results for removing slew.
Hypothesis: For this experiment we hypothesise

1. The non-clock values are overall closer to the base case, when compared to Experiment 1: Without Slew

Results: The results can be found in Table 3.6.

1. Partially true. For all of the non-clock values shown in Table 3.6, with the exception of the number of delays, the values were indeed closer. This is however not surprising as; the more time needed to complete a tasks, the more chances of a task being preempted i.e. higher risk of becoming delayed.

Table 3.5 compares how close experiment 1 and 2 were to the base case. The cells displays the total difference from the base case.
For example, experiment 2's runs cell is the sum of the difference in the amount of completed tasks. The task `gather_new_data` for `Sat_0` was completed 10 times in the base case and 12 times in this experiment, which gives a difference of 2. The amount of delays for transfer on `Sat_0` was 16 in the base and 12 in this experiment, which gives a difference of 4.
This means that the closer the value is to 0 the closer it is to the base case.

- Program time — calculated from the program time column
- Cost of bandwidth — calculated from the Cost of Bandwidth column
- Data gathered, transferred, and sent — calculated from the Data sent to Earth, Data gathered, and Data transferred columns
- Runs — calculated from the Runs rows
- Delays — calculated from the Delays rows

Experiment 2 is closer to 0 in all of the columns except for the amount of delays which means that it deviates less from the base case and is therefore considered to be closer to it. The clocks were not compared as it was necessary to change how they counted the time in order to set up the experiment, which makes the comparison uninteresting.

	Program time	Cost of bandwidth	Data gathered, transferred, and sent	Runs	Delays
Experiment 2	108	195	75	8	18
Experiment 1	133	3070	765	59	8

Table 3.5: Comparison between experiment 2 and 1

Conclusion: Comparing the results of this experiment to Experiment 1: Without Slew, the observed values are in general much closer to the base case. Unfortunately the program time was still more than 70% over the base case. These results indicate that for this model

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	262	7240	640	780	1035
Diff (%)	+70.2	-2.6	0.0	+4.0	+4.5

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	12	**+20.0**	15	14	**-6.7**	0	0	**0.0**
Runs: transfer	20	23	**+15.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	22	**+10.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	9	**N/A**	1	5	**+400.0**	0	0	**0.0**
Dealys: transfer	16	12	**-25.0**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	672	**+75.9**	223	604	**+170.9**	784	693	**-11.6**
Wait	576	341	**-40.8**	51	0	**-100.0**	0	0	**0.0**
Work	318	427	**+34.3**	634	836	**+31.9**	557	747	**+34.1**
Slew	164	0	**-100.0**	532	0	**-100.0**	99	0	**-100.0**

Table 3.6: Results of running experiment 2

slew must be kept. However, we believe the increased program time is a result of the experiment being run in a model designed to accommodate for slew. Had another model, designed not to slew, been made, we still believe there would be a noticeable decrease in program time.

3.4.4 Experiment 3: No Gather on `Sat_1`

Experiment: Remove the central satellite's ability to perform `gather_new_data`. This in done by changing the `sat_trans` task array.
Motivation: To determine the effect of having one less satellite that is able to gather data. This change make it so `Sat_1` can only transfer and receive data internally. This will help us to understand the importance of the satellites' set-up.
Hypothesis: For this experiment we hypothesise

1. Increase to the `transfer` and `receive` tasks between `Sat_0` and `Sat_1`. As `Sat_1` will now need to be fed data, which can only come from `Sat_0`.
2. Higher `idle` clock time on `Sat_1` and `Sat_2`. As `Sat_1` has to spend more time communicating with `Sat_0` in order to be fed data, it will not communicate as often with `Sat_2`.
3. Increase to the `slew` clock for `Sat_0` and `Sat_1`. `Sat_0` will slew more because it will be able to execute more tasks as `Sat_1` is more eager to receive its data. `Sat_1` will slew more because it will have to alter between communicating with `Sat_0` and `Sat_2`.
4. Decrease to the `wait` clock for `Sat_0`. The satellite will wait less because of the same reasons that was stated above.
5. Less data gathered and sent to Earth. As only one satellite can gather data and it takes more time to get the data to the one sending it to Earth.

6. Increase to the amount of data transmitted internally. As the data need to go from `Sat_0` to `Sat_1`, and finally to `Sat_2` before being sent to Earth

Results: The results can be found in Table 3.7.

1. True. The amount of completed `transfer` and `receive` tasks have more than doubled.
2. Partially true. The increase to `Sat_2`'s `idle` clock is very low. The data transfer/receive rate was high enough for `Sat_2` to continue sending data, and was therefore almost unaffected. It did however get affected as it did not communicate as much with `Sat_1`. And `Sat_1` had a significant increase in `idle` clock time.
3. Partially true. `Sat_0` did slew more, but `Sat_1` slew less. The less slewing time on `Sat_1` is a side effect of it idling much more compared to the base case. Additionally the `slew` clock is reduced as doing the same tasks multiple times in succession, will not cause it to slew back and forth. Additionally it will never have to slew in order to gather data as it cannot execute that task in this experiment.
4. True. The `wait` clock of `Sat_0` was almost halved.
5. Partially true. Less data was gathered, but the amount send to Earth was unaffected. This is most likely a result of `Sat_0` idling and waiting much more in the initial run, leaving it time to perform additional tasks.
6. True

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	135	6850	640	660	1350
Diff (%)	-12.1	-7.9	0.0	-12.0	+36.4

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	22	**+120.0**	15	0	**-100.0**	0	0	**0.0**
Runs: transfer	20	45	**+125.0**	46	45	**-2.2**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	45	**+125.0**	46	45	**-2.2**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	17	**N/A**	1	0	**-100.0**	0	0	**0.0**
Dealys: transfer	16	14	**-12.5**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	91	**-76.2**	223	627	**+181.2**	784	793	**+1.1**
Wait	576	350	**-39.2**	51	51	**0.0**	0	0	**0.0**
Work	318	616	**+93.7**	634	479	**-24.4**	557	549	**-1.4**
Slew	164	384	**+134.1**	532	284	**-46.6**	99	99	**0.0**

Table 3.7: Results of running experiment 3

Conclusion: Limiting the satellites to be able to execute less tasks seems to have a relatively small effect on the number of tasks completed. In fact the same amount of data was sent to Earth, though with less remaining storage in the system.

3.4.5 Experiment 4: Double Deadline

Experiment: Double the deadline of all tasks. This is done by modifying the `deadline` field in all `TaskDescription`s, see Table 3.2.
Motivation: An increased deadline should result in fewer delays. We therefore run this experiment to determine the correctness of the model, and that an increased deadline will indeed do this.
Hypothesis: For this experiment we hypothesise

1. Fewer delays, most noticeable on `Sat_0`. Since there are more time to complete tasks, even after them being preempted.
2. More tasks will be completed. As the preempted tasks are more likely to be finished, we expect this will result in an increased amount of completed tasks.
3. Increase to the `work` clock for all satellites. After being preempted it is more likely a task can be completed. Instead of waiting for a new suggestion the satellite can finish a task which was preempted earlier.
4. Decrease in the `idle` clock for all satellites. As the satellites will be more likely to continue preempted tasks rather than wait for a new suggestion.
5. Increase to all data values. As there will be more runs there will be moved more data.

Results: The results can be found in Table 3.8.

1. True. There was no delays to any task.
2. True. However, a total increase of only two completed tasks across all satellites.
3. Partially true. The `work` clock saw little change. While more tasks were completed time was not waisted by becoming delayed almost cancelling each other out.
4. Partially true. There was an increase in the `idle` clock time to `Sat_1` and `Sat_2`. This is most likely a result of `Sat_0` finishing more tasks.
5. False. As the same amount of tasks gathering data and sending to Earth were executed, the data values were unchanged. There was a small increase to data transferred internally, as the task was more likely to be completed.

Conclusion: As expected an increased deadline did indeed result in fewer delays. This indicates that the implementation works correctly as the defined deadline does indeed have an effect.

3.4.6 Experiment 5: No Preemption

Experiment: Make the scheduler non-preemptive. This is done by falsifying the edges leading to preemption, and forcing the Scheduler template to always take the edge back which enqueues the new task and continues the current one.
Motivation: To determine the relevance of preemption. As preemption is responsible for a significant portion of the state space it would be beneficial to investigate the effect thereof.
Hypothesis: For this experiment we hypothesise

1. Fewer delays. As tasks can no longer be preempted the only possibility for delays is that satellite keeps waiting for another satellite to transfer to.

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	154	7435	640	750	1020
Diff (%)	+0.2	0.0	0.0	0.0	+3.0

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	11	**+10.0**	15	14	**-6.7**	0	0	**0.0**
Runs: transfer	20	22	**+10.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	22	**+10.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	0	**-100.0**	0	0	**0.0**
Dealys: transfer	16	0	**-100.0**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	361	**-5.5**	223	265	**+18.8**	784	787	**+0.4**
Wait	576	551	**-4.3**	51	51	**0.0**	0	0	**0.0**
Work	318	335	**+5.3**	634	620	**-2.2**	557	554	**-0.5**
Slew	164	193	**+17.7**	532	504	**-5.3**	99	99	**0.0**

Table 3.8: Results of running experiment 4

2. Small increase in runs. Since time is no longer waisted on tasks becoming delayed, however tasks can no longer preempt the active task when their window comes up, which dampens the increase.
3. Increase to all data values. Assuming our first hypothesis is correct there will be performed more tasks, and thereby higher data values.
4. Shorter program time. As preemption, in our deterministic model, takes many states to achieve, not having it will cause the trace to become smaller and thereby lessen the time it takes to write the trace.

Results: The results can be found in Table 3.9.

1. False. However, without preemption there are no possibility for the tasks to become delayed, with the exception of waiting for another satellite to become ready when attempting to communicate internally. Therefore this task was the only one to experience delays.
2. Partially true. The increase to runs turned out to be significant.
3. False. There was a significant increase in runs between `Sat_0` and `Sat_1`. However, the same amount of data was sent to Earth and nearly the same amount gathered. This indicates an inefficient use of time, as the satellite can never exceed its maximum storage capacity but can still initiate a `gather_new_data` when at say 90% storage. Doing so will cause the satellite to reach 100% storage but completing the task will take the same amount of time as it would otherwise. This occur because a task can be queued while the same task is being executed, and when the task is done it may be run again.
4. False. The program time remained approximately the same.

Conclusion: At first glance the preemption feature seems to be of small importance as the observed data values are very close to that of the base case. However, a closer look at

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	158	9775	640	775	1080
Diff (%)	+2.8	+31.5	0.0	+3.3	+9.1

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	18	**+80.0**	15	19	**+26.7**	0	0	**0.0**
Runs: transfer	20	26	**+30.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	26	**+30.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	0	**-100.0**	0	0	**0.0**
Dealys: transfer	16	17	**+6.3**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	515	**+34.8**	223	415	**+86.1**	784	789	**+0.6**
Wait	576	393	**-31.8**	51	53	**+3.9**	0	0	**0.0**
Work	318	408	**+28.3**	634	653	**+3.0**	557	550	**-1.3**
Slew	164	124	**-24.4**	532	319	**-40.0**	99	101	**+2.0**

Table 3.9: Results of running experiment 5

the results shows that significantly more runs of the task `gather_new_data` was completed. This shows an inefficient use of the task as explained above. This shows that preemption is important for an effective use of time.

3.4.7 Experiment 6: No Preemption, No Queue

Experiment: Like in the previous experiment, we have removed preemption. In addition to this we have made it so that tasks can not be queued while another one of the same type is being executed.
Motivation: The main problem with the previous experiment Experiment 5: No Preemption, was the inefficient use of time. However, we believe this to be a problem with other functionality in the deterministic model, and not because preemption must be included.
Hypothesis: For this experiment we hypothesise

1. Data values close to the base case. However, without the inefficient use of time when gathering data.
2. All four clocks will have small variations compared to the base case.

Results: The results can be found in Table 3.10.

1. True. There was small variations to the data values, without a large amounts of time waisted. However, 26 `gather_new_data` tasks were completed meaning the Data_gathered value should have been 780. This is caused by the `HIGH` threshold being 75 rather than 70. Had this been the case, we believe there would have been no inefficient use of time.
2. Partially true. Most of the clocks were close to the base case. However, a few differed significantly. This was mainly the clocks on `Sat_0` and the `idle` clock on `Sat_2`. As

Sat_0 completed more tasks it makes sense that it would have spent more time working and slewing. As for Sat_1's increased idle clock, this is most likely a result of it receiving more data form Sat_0, and thereby reached the HIGH storage level more often and having to wait for Sat_2 to become available for transfer.

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	136	7470	600	740	1065
Diff (%)	-12.0	0.5	-6.3	-1.3	7.6

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	15	**-6.3**
Runs: gather_new_data	10	14	**40.0**	15	12	**-20.0**	0	0	**0.0**
Runs: transfer	20	27	**35.0**	46	44	**-4.3**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	27	**35.0**	46	44	**-4.3**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	0	**-100.0**	0	0	**0.0**
Dealys: transfer	16	26	**62.5**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	357	**-6.5**	223	362	**62.3**	784	826	**5.4**
Wait	576	456	**-20.8**	51	51	**0.0**	0	0	**0.0**
Work	318	381	**19.8**	634	571	**-9.9**	557	520	**-6.6**
Slew	164	248	**51.2**	532	458	**-13.9**	99	96	**-3.0**

Table 3.10: Results of running experiment 6

Conclusion: Changing the preemption experiment so it would not queue tasks while working, turned out to be a success. This has showed that preemption may not be necessary. There was an increased amount of delays but also a decreased Program time. Based on this we believe using preemption may lead to a better scheduler, but will have a longer computation time.

3.4.8 Experiment 7: Double Maximum Storage

Experiment: Double the storage capacity of all satellites. This will also affect the HIGH and LOW thresholds as they are based on the maximum storage.
Motivation: As we have chosen the storage capacity we wish to investigate if the this will affect the system behaviour.
Hypothesis: For this experiment we hypothesise

1. Small increase in gather_new_data tasks completed. Since the initial storage value starts further from the HIGH threshold, it will execute this task more early on. However, once reaching this it will behave like the base case.
2. Increase to data gathered. The data gathered is a direct result of the amount of gather_new_data tasks completed.
3. Decreased idle clock and increase in the work clock for Sat_0 and Sat_1. Since they will perform more tasks early on they should work more and idle less.

Results: The results can be found in Table 3.11.

1. True. The total number of uppVargather_new_data tasks performed was increased from 25 to 32.
2. True. 28 % more data was gathered.
3. True

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	149	8640	600	960	1050
Diff (%)	-3.0	+16.2	-6.3	+28.0	+6.1

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	15	**-6.3**
Runs: gather_new_data	10	13	**+30.0**	15	19	**+26.7**	0	0	**0.0**
Runs: transfer	20	21	**+5.0**	46	49	**+6.5**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	21	**+5.0**	46	49	**+6.5**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	1	**0.0**	0	0	**0.0**
Dealys: transfer	16	22	**+37.5**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	290	**-24.1**	223	79	**-64.6**	784	793	**+1.1**
Wait	576	622	**+8.0**	51	48	**-5.9**	0	0	**0.0**
Work	318	386	**+21.4**	634	743	**+17.2**	557	556	**-0.2**
Slew	164	144	**-12.2**	532	572	**+7.5**	99	93	**-6.1**

Table 3.11: Results of running experiment 7

Conclusion: As expected changing the maximum storage allowed the satellites to gather more data. Mostly the experiment deviated little from the base case, and no changes were unexpected. This confirms that the storage capacity is implemented as intended.

3.4.9 Experiment 8: No Window Dependencies

Experiment: Tasks can now be started at any time as none of them are window dependant. This means it will always be possible to gather and send data.
Motivation: Removing the concept of window dependencies will be a significant inaccuracy. However, it is interesting to investigate how much this will deviate from the base case as it most likely cause a noticeable reduction in program time.
Hypothesis: For this experiment we hypothesise

1. `Sat_0` will not be able to perform any tasks. It is unable to execute the `send_data` task, which means it has to transfer it data when the memory is full. But `Sat_1` will prioritise its own `transfer` task and since it can execute `gather_new_data` at any time it will always have something to do that has a higher priority than receiving data.
2. Increase in `work` clock time for `Sat_1`
3. Increase in `work` clock time for `Sat_2`
4. `Sat_1` will have little to no `idle` clock time

Results: The results can be found in Table 3.12.

1. True
2. True. An increase of just 10 %, as a result of much of the decreased idle time is used slewing.
3. True
4. True. Its `idle` clock was reduced by 97 %.

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	120	0	645	720	660
Diff (%)	-22.0	-100.0	+0.8	-4.0	-33.3

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	21	**+31.3**
Runs: gather_new_data	10	0	**-100.0**	15	24	**+60.0**	0	0	**0.0**
Runs: transfer	20	0	**-100.0**	46	44	**-4.3**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	0	**-100.0**	46	44	**-4.3**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	1	**0.0**	0	0	**0.0**
Dealys: transfer	16	10	**-37.5**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	673	**+76.2**	223	6	**-97.3**	784	562	**-28.3**
Wait	576	707	**+22.7**	51	75	**+47.1**	0	0	**0.0**
Work	318	54	**-83.0**	634	698	**+10.1**	557	734	**+31.8**
Slew	164	6	**-96.3**	532	661	**+24.2**	99	144	**+45.5**

Table 3.12: Results of running experiment 8

Conclusion: this experiment shows that the concept of windows should be included. The most noticeable impact of course being the total starvation of `Sat_0` as it can not unload its data. But the inefficient utilisation of `Sat_2`'s `send_data` should not be ignored either.

3.4.10 Experiment 9: Smaller Difference Between High and Low Thresholds

Experiment: The difference between the `HIGH` and `LOW` thresholds will be decreased. `HIGH` will be changed to 60 from 75 and `LOW` to 40 from 25.
Motivation: The thresholds is a choice taken in designing our scheduler. Because of this, changing the defined thresholds is amongst the most important values to test.
Hypothesis: For this experiment we hypothesise

1. More internal communication tasks will be completed. The satellites are considered to be at high storage level more of the time and will therefore want to transfer data more often.
2. Fewer `gather_new_data` tasks performed. For the same reasons as above.

Results: The results can be found in Table 3.13.

1. False. There was a decrease to the amount of data transferred between `Sat_0` and `Sat_1`. This is most likely because `Sat_1` performed more `gather_new_data` tasks,

making data from `Sat_0` less important, i.e. `Sat_1` prevented `Sat_0` from performing as many tasks.

2. True. However, it only went down by 1 in total.

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	150	7240	640	720	915
Diff (%)	-2.5	-2.6	0.0	-4.0	-7.6

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	7	**-30.0**	15	17	**+13.3**	0	0	**0.0**
Runs: transfer	20	15	**-25.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	15	**-25.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	1	**0.0**	0	0	**0.0**
Dealys: transfer	16	22	**+37.5**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	393	**+2.9**	223	169	**-24.2**	784	784	**0.0**
Wait	576	652	**+13.2**	51	51	**0.0**	0	0	**0.0**
Work	318	267	**-16.0**	634	657	**+3.6**	557	557	**0.0**
Slew	164	128	**-22.0**	532	563	**+5.8**	99	99	**0.0**

Table 3.13: Results of running experiment 9

Conclusion: Due to how the scheduler is designed, the thresholds seem to have little impact on the data values. However, setting them closer have caused noticeable more delays on `Sat_1`. Indicating that doing so is not beneficial.

3.4.11 Experiment 10: Larger Difference Between High and Low Storage

Experiment: The difference between the `HIGH` and `LOW` thresholds will be increased. `HIGH` will be changed to 90 from 75 and `LOW` 10 from 25.
Motivation: The thresholds is a choice taken in designing our scheduler. Because of this, changing the defined thresholds is amongst the most important values to test.
Hypothesis: For this experiment we hypothesise

1. Less data send to Earth. Gather data tasks will be suggested when the satellite is closer to maximum storage capacity, which should result in less efficient use of the `gather_new_data` task, which in turn means that less data can be send to Earth and less `send_data` tasks will be executed.
2. The task `gather_new_data` will be executed more
3. The task `send_data` will be executed less
4. The `receive` and `transfer` task will be executed less.

Results: The results can be found in Table 3.14.

1. True. However, the difference is very small and the same amount of `send_data` tasks was completed, indicating it was not utilised as efficient as could have been.

2. True. As the task is at a high priority a larger percentage of the time it will be executed more often. However, the data_gathered value it has changed little, showing an inefficient use of time.
3. False. The total amount of data gathered was not much lower, which is the reason that `Sat_2` was still able to perform as in the base case.
4. True

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	170	9580	630	790	915
Diff (%)	+10.41	+28.9	-1.6	+5.3	-7.6

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	15	**+50.0**	15	21	**+40.0**	0	0	**0.0**
Runs: transfer	20	15	**-25.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	15	**-25.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	0	**-100.0**	0	0	**0.0**
Dealys: transfer	16	16	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	638	**+67.0**	223	326	**+46.2**	784	771	**-1.7**
Wait	576	290	**-49.7**	51	54	**+5.9**	0	0	**0.0**
Work	318	330	**+3.8**	634	669	**+5.5**	557	567	**+1.8**
Slew	164	182	**+11.0**	532	391	**-26.5**	99	102	**+3.0**

Table 3.14: Results of running experiment 10

Conclusion: Once again changing the thresholds yielded little variation to the data values. However, significantly more `gather_new_data` tasks was completed, once again showing an inefficient utilisation of the task. The thresholds should therefore not be changed in this manner.

3.4.12 Experiment 11: Double Priority for Non-internal Communication

Experiment: Double the priority of task `gather_new_data` and `send_data`.
Motivation: Performing this experiment will tell if increasing the priority of the already high priority tasks will yield different results.
Hypothesis: For this experiment we hypothesise

1. Little to no effect on any of the variables. The priority of said tasks is already of higher priority in the base case.

Results: The results can be found in Table 3.15.

1. True. All values remained the same.

Conclusion: performing this experiment showed that nothing was changed when comparing to the base case. This shows, not unexpectedly, that increasing the priority of already high priority tasks does not change anything.

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	154	7435	640	750	990
Diff (%)	-0.1	0.0	0.0	0.0	0.0

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	10	**0.0**	15	15	**0.0**	0	0	**0.0**
Runs: transfer	20	20	**0.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	20	**0.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	1	**0.0**	0	0	**0.0**
Dealys: transfer	16	16	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	382	**0.0**	223	223	**0.0**	784	784	**0.0**
Wait	576	576	**0.0**	51	51	**0.0**	0	0	**0.0**
Work	318	318	**0.0**	634	634	**0.0**	557	557	**0.0**
Slew	164	164	**0.0**	532	532	**0.0**	99	99	**0.0**

Table 3.15: Results of running experiment 11

3.4.13 Experiment 12: Double Priority for Internal Communication

Experiment: Double the priority of task `transfer` and `receive`. This will cause the `transfer` task to become the highest prioritised task and should therefore be executed more often.
Motivation: This will make transferring data the most important task. Indicating that moving data to a satellite which can send it to Earth, is the primary responsibility.
Hypothesis: For this experiment we hypothesise

1. More `transfer` and `receive` tasks will be completed
2. Fewer `gather_new_data` and `send_data` tasks will be completed
3. Slightly less data will be send to Earth. The priority system will make sure that the `gather_new_data` and `send_data` tasks will not be ignored. The tasks will just be executed after the `transfer` and `receive` tasks. Additionally, the `gather_new_data` task will be more effective as it sometimes gather more data than what can be stored, but by transferring before the data is collected, more space on the gathering satellite will be free thereby allowing more of the data to be stored.

Results: The results can be found in Table 3.16.

1. True. `Sat_0` transferred an additional 2 times, it is however not as much as expected.
2. False. The total amount of `gather_new_data` and `send_data` tasks was the same as in the base case.
3. False. The amount of data sent was unaffected.

Conclusion: This experiment resulted in little change to the data values. However, there where more delays, and more work time. An increased work time may seem to be a good thing, but if nothing is accomplished, the energy may simply have been waisted. Therefore

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	157	7435	640	750	1020
Diff (%)	+1.9	0.0	0.0	0.0	+3.0

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	16	**0.0**
Runs: gather_new_data	10	11	**+10.0**	15	14	**-6.7**	0	0	**0.0**
Runs: transfer	20	22	**+10.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	22	**+10.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	4	**N/A**	1	1	**0.0**	0	0	**0.0**
Dealys: transfer	16	21	**+31.3**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	263	**-31.2**	223	188	**-15.7**	784	781	**-0.4**
Wait	576	698	**+21.2**	51	54	**+5.9**	0	0	**0.0**
Work	318	387	**+21.7**	634	672	**+6.0**	557	559	**+0.4**
Slew	164	94	**-42.7**	532	528	**-0.8**	99	102	**+3.0**

Table 3.16: Results of running experiment 12

the priority of the internal communication tasks should not be increased to exceed the other tasks.

3.4.14 Experiment 13: Finer Granularity on Suggestion-timer

Experiment: The task suggestion interval is set to 1 instead of 3. This value is a compromise between satellite efficiency and trace length reduction. The value decides how often the satellites will have to evaluate the tasks in order to find the best ones to execute.

Motivation: With the inaccuracy described in Section 3.3.4, regarding the suggestion timer, it should be tested what the impact is thereof. This experiment will be on a model more precise than that used in the base case, and the hope is that it will show little deviation there from.

Hypothesis: For this experiment we hypothesise

1. Much increased program time. The finer granularity will result in a much larger state space.
2. Slight decrease in the `idle` clocks. The satellites should be more efficient.
3. Small variation in the data values. The finer granularity should mean that the satellites will sometimes chose to preempt a task that the base case would have finished.

Results: The results can be found in Table 3.17.

1. True. There is a significant increase in program time as it has more than doubled.
2. Partially true. `Sat_0` experienced a slight increase in the `idle` clock whereas the other satellites idled less. This is most likely a result of `Sat_1` executing more tasks.
3. True

	Program time	Cost of Bandwidth	Data sent to Earth	Data gathered	Data transferred
Base:	154	7435	640	750	990
Result:	348	7275	600	750	930
Diff (%)	+126.2	-2.2	-6.3	0.0	-6.1

	Sat_0			Sat_1			Sat_2		
	Base	Result	Diff (%)	Base	Result	Diff(%)	Base	Result	Diff (%)
Runs: send_data	0	0	**0.0**	0	0	**0.0**	16	15	**-6.3**
Runs: gather_new_data	10	8	**-20.0**	15	17	**+13.3**	0	0	**0.0**
Runs: transfer	20	16	**-20.0**	46	46	**0.0**	0	0	**0.0**
Runs: receive	0	0	**0.0**	20	16	**-20.0**	46	46	**0.0**
Dealys: send_data	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: gather_new_data	0	0	**0.0**	1	1	**0.0**	0	0	**0.0**
Dealys: transfer	16	16	**0.0**	0	0	**0.0**	0	0	**0.0**
Dealys: receive	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**
Idle	382	399	**+4.5**	223	172	**-22.9**	784	779	**-0.6**
Wait	576	676	**+17.4**	51	51	**0.0**	0	0	**0.0**
Work	318	229	**-28.0**	634	602	**-5.0**	557	546	**-2.0**
Slew	164	120	**-26.8**	532	599	**+12.6**	99	99	**0.0**

Table 3.17: Results of running experiment 13

Conclusion: This experiment shows that the inaccuracy does indeed have an impact. While there are little deviation to the data values and cost of bandwidth, there are some noticeable differences in the clock values, mainly on `Sat_0`.
Performing this experiment took more than twice the time to complete, for little variations when considering the system as a whole. Because of this we believe that introducing this inaccuracy was just in regards to testing the model, but should be removed if used in practise.

3.4.15 Computation Time

An experiment with 10 satellites was also performed, as it is expected that three satellites will be insufficient in representing a full convoy. However, writing these traces was more time consuming and required larger amounts of storage capacity.
Getting a trace with 10 satellites took 3 hours and 38 minutes, and the trace required over 25GB of storage. This led us to estimate the increase in storage and time per satellite added. We found that for n satellites the file size for $n + 1$ satellites would be $\approx$ 66% higher.
We found the function of time required to get a trace to be: $\approx 100.0464893 * EXP(4.878053161 * 10^{-1} * n)$.
EXP returns Euler's number raised to a power [15]. The equation is found using the online tool *three points parabola calculator*[16], the points are measured by ourself with a script to keep track of time where x is the number of satellites and y is the time required for SMC to complete the query and output the trace to a file.
From the equation we can conclude that the desired 30 satellite configuration would take over seven years to complete and 681TB of storage.

The 10 satellite experiment was executed on a laptop and could naturally have been faster

by using a stronger computer. It should be noted that it is not the actual computation time of completing the query that is time consuming, for example, the base case is completed in under two seconds. The time comes from having to write the trace to a file or some other output.
We also attempted running a query with 30 satellites where only the final state was printed. This experiment was completed in just 53 minutes, and showed that the satellites would perform approximately the same amount of tasks per satellite.

If the amount of states written in a trace could be reduced, or if the states themselves could be made smaller i.e. only including values relevant for the schedule, the execution time and file size would be lessened as well. The trace is necessary as it is analysed in order to create a schedule.

3.5 Conclusion

In this section we will make a final conclusion on the experiments performed on the deterministic model. We will focus on what features should be included in the construction of a non-deterministic model, and whether or not our choices in designing a scheduler have been valid.

The scheduler behaved as expected. Consider for instance the experiment with double storage causing small variations mainly in the beginning of the schedule. Or the one with a more frequent suggestion-timer, which caused little variation to the data values and number of runs.

These experiments showed that it will be necessary to include the concept of windows as not doing so causes extensive variations to the schedule where tasks are performed at times where they would otherwise not be available.
Based on experiment two Section 3.4.3, we believe that slew does not need to be included as long as this is compensated for, by increasing the execution time of the different tasks. With the designed scheduler preemption also appeared to be a necessity which should be included in any future model. However, Section 3.4.7 showed that preemption may not be needed and the poor results in the previous experiment was a result of the model being build to include it.

The experiments involving more than three satellites, Section 3.4.15, showed that it is possible to find a schedule with this method. However, in order to get a schedule it would require modifications to SMC in order to modify its output. Such modifications would be to change which information is included in each state. In order to construct the schedule in would only be necessary to include four variables per satellite, these are as follows:

- Active — what task the satellite is currently performing
- Start time — a time stamp for when the Processor template entered the location Occupied
- End time — a time stamp indicating when the Processor template left Occupied
- Slews — how many slews were performed prior to entering Occupied

This would cause an enormous reduction in the amount of information required to print a state thereby making it possible to compute and output the trace faster and at a lower memory requirement. Running the initial configuration with three satellites results in each state describing over 1300 variables and clocks, rather than the necessary twelve. Additionally all states would not even be needed, for instance all states included because of the CheckRunnable template could be excluded.

Non-deterministic Model 4

In this chapter we will expand upon the knowledge gained during the previous chapter, Chapter 3, specifically what we learned from examining the experiments. We will continue to explore the possibilities of modelling convoys of satellites where each convoy represents a number of satellites that follows the same orbit.

4.1 Scenario

As described in Chapter 1, the scenario we wish to model is made out of multiple satellites spread across multiple convoys, where each satellite can communicate with other satellites, and stations at specific times, as according to Section 2.2.
This model is less detailed, as the focus will be on convoys of satellites, instead of examining individual satellites. In this new non-deterministic model, a global Scheduler template will suggest tasks to the convoys which they may choose to accept or reject, which makes this model non-deterministic as the choices are unpredictable. Some restrictions have been made such that some choices are forced, but this will be explained in detail later.
Given our results from the experiments in Section 3.5 and prior knowledge about CORA [2], each convoy will use a non-preemptive scheduler with Value Based Priority (VBP). This decision is made to reduce the number of choices the model will be able to make, and ultimately decrease the state space which will be essential to run queries with schedule lengths of around 1440 time units.

Our results from the benchmark in Section 3.5 highlighted the following:

- Preemption will not be part of the non-deterministic model due to it increasing the state space substantially with little improvement to the schedule
- Slew is also scraped from this model, given we can obtain close to similar results by assuming one slew is always required for each task executed
- Stations is kept because it gives a more realistic picture and naturally decreases the state space as the number of choices are limited at some intervals where the convoys are forced to reject a task due to the task requiring a window that is not available at the given time
- Cross convoy communication opportunities will be used due to it having similar effect as stations, where the state space is reduced and staying true to the environment the satellites are in

The satellites will be able to perform the three tasks: `gather_new_data`, `send_data`, and `transfer` which were described in more detail in the previous scenario, which can be seen in Section 3.1. A difference in how the tasks function in comparison the deterministic

model is that `transfer` now sends data to other convoys, and not to satellites within the convoy.

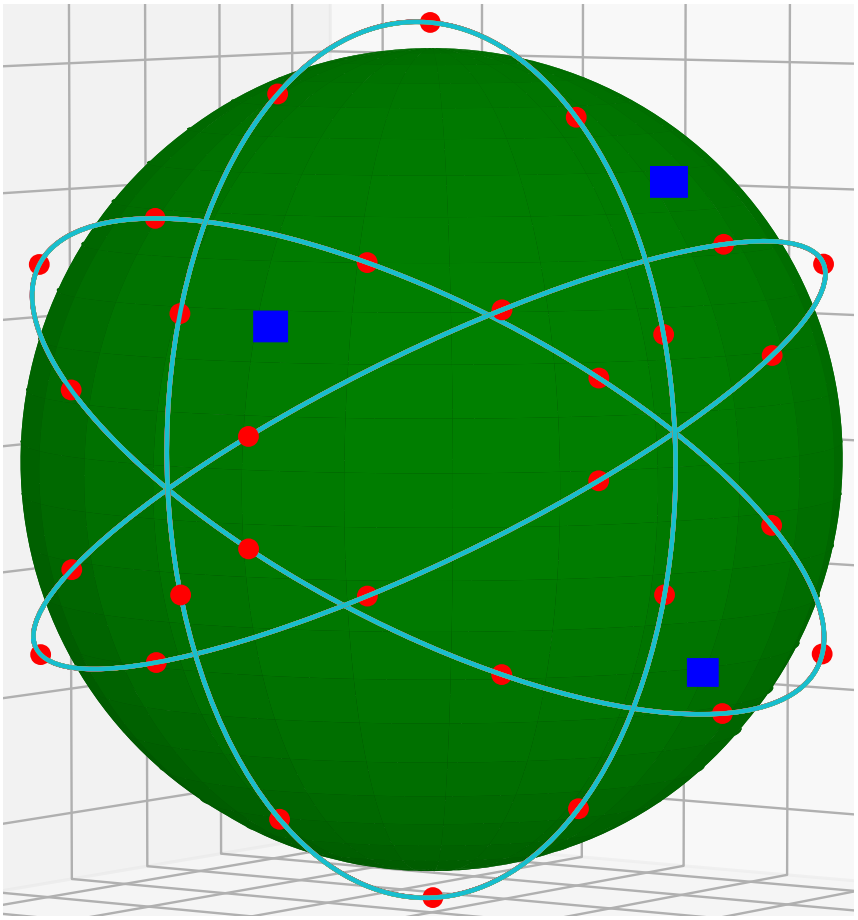

Figure 4.1: Three convoys with ten satellites each, and three stations

Figure 4.1 gives a visual representation of the scenario. The figure has blue coloured stations and three convoys, coloured teal, each convoy has ten satellites marked with a red dot. When the teal lines cross, the convoys will be able to communicate with each other, this has been referred to previously as a cross convoy communication opportunity. The figure is only a still picture of how the system could look like, and the orbits will change over time, thereby moving when the cross convoy communication opportunities are available. The same is true for when the satellites may communicate with the stations.

4.2 UPPAAL CORA

CORA is a branch of UPPAAL, which uses the concept of cost to make optimal reachability analyses. Optimal refers to finding the path with the minimum cost [17].
Using this the before mentioned unpredictable choices will be based on what is best rather than simply being random. Unlike UPPAAL which uses timed automata to model a problem, CORA uses linearly priced timed automata. CORA has the same functionally in regards to `guards`, `updates`, etc. on locations and edges as that described in Section 2.4.

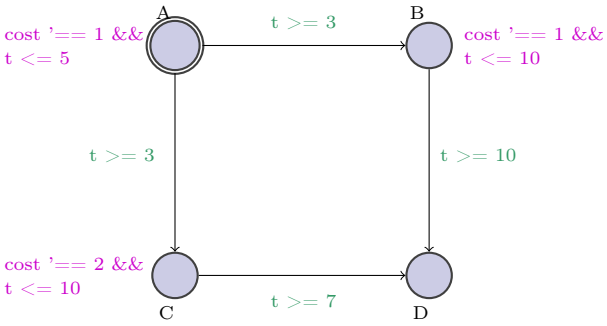

Figure 4.2: Template with `cost`

Figure 4.2 illustrates a model which will be used to demonstrate how CORA uses `cost` along with an exhaustive search strategy to find the optimal path. Whenever times passes, the `cost` is incremented by the sum of all specified cost rates from the active locations in every template.
The goal for this example is to find the path with the lowest `cost` that leads to location `D`. The model start in its initial location, `A`, where the only option is to delay for three time units, which increases the `cost` by 3. After the delay, three options become available. It can either delay again, transition to `B`, or transition to `C`. CORA will explore all of these options, as well as all options these may spawn.

It might appear the better choice is to transition to `B` as the cost for that location is cheaper than the cost of `C`. By taking that path it is forced to wait five units of time before transitioning to `D`, leading to a final cost of 10. The best path from the model in Figure 4.2 is to wait for five time units in `A`, then transition to `C` where it then waits for an additional two time units before transitioning to `D`. This means the optimal path has a cost of 9, because it costs 5 to wait in `A` and 4 to wait two units of time in `C`. This example shows how the greedy choice, in this case transitioning to `B`, is not always the best choice in the long run. It is not difficult to manually discover the optimal path for this particular model, but it becomes increasingly difficult when the model gets larger and more complex.

4.3 CORA Model

This section details the non-deterministic CORA model. It covers the templates in the model and the defined data structures along with the priority approach, flow of the model, and concerns regarding the state space. Lastly experiments is performed and a conclusion on the findings will be made.

4.3.1 Overview

The non-deterministic model consists of two templates, Scheduler and Convoy, and both can be viewed in Figure 4.3 and Figure 4.4 respectively. Their responsibilities are listed below:

The Scheduler template's responsibilities:

- Suggest and assign tasks for each Convoy template
- Progression of time
- Ensure performed actions are allowed within the parameters (e.g. a convoy may not exceed its maximum storage capacity)

The Convoy template's responsibilities:

- Randomly accept or reject task suggested by the Scheduler template
- Calculate the `cost` for its current status
- Distribute accepted tasks to its satellites

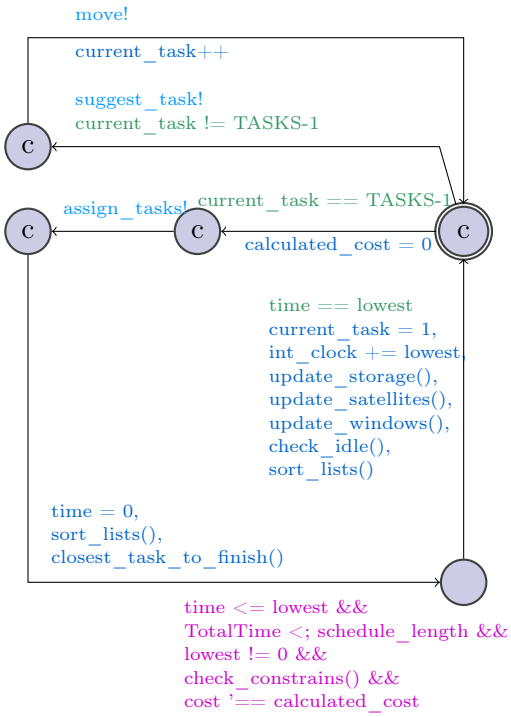

Figure 4.3: The Scheduler template

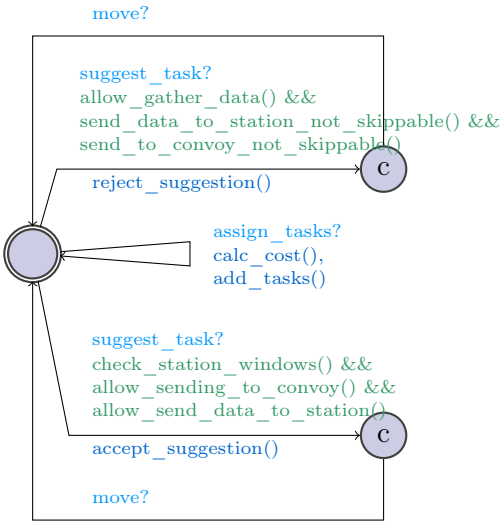

Figure 4.4: The Convoy template

4.3.2 Data Structure

The model utilises the following data structures; `Stations`, `Convoys`(not to be confused with the template Convoy), `Tasks`, `Executions`, and `Scheduler`(not to be confused with the template Scheduler).

`Stations`'s goal is to keep track of how much data that has been sent to the individual station along with the cost rate. It has two attributes: `cost` — the bandwidth cost of sending data to that particular station and `data_received` — how much data that has been sent to the individual stations. The struct is used alongside a three dimensional array which is used to determine when a convoy is within reach of a station, this will be explained in more detail later.

`Convoys` defines the limits for the convoy, i.e. the number of satellites, storage capacity, and the maximum number of parallel executions of the same task. It has six attributes: `satellite` — which defines how many satellites the convoy has available, `send_to_station` — how many stations the convoy comes in contact with throughout its schedule length and how many satellites that can be assigned to each station, `send_to_convoy` — identical

to `send_to_station` except it describes the relation with other convoys, `gather_data` — number of satellites that can gather data, `storage` — how much data the convoy can store in total, `tasks_depending_on_windows` — describes which tasks that are depending on which windows.
`Tasks` details the essential information for each task. It has three attributes: `task_type` — integer id, `execution_time` — how long it takes to complete the task, and `data_rate` describes the amount of data that is being sent when this task is active.
`Executions` contains the same attributes as `tasks`, but has one additional attribute: `to` — integer id that refers to which station or convoy that a satellites is sending data. The struct is used to reduce the state space as it allows us to declare some of the `Tasks` structs to be constant which means they will not be considered when a new state is made.
`Scheduler` keeps track of all satellites across all convoys. It consists of a two dimensional array where the first index number refers to the convoys and the second index number corresponds to a satellite, the array is filled with initialised `Executions` structs.

4.3.3 Suggestion and Assignment of Tasks

The Scheduler template assigns a value to `current_task`, which indicates what type of task is currently being suggested to the Convoy templates. The suggestion is sent when the channel `suggest_task` is triggered, this is done for every task, whenever a task has finished. The synchronisation is initialised from the upper loop in the Scheduler template and can be seen in Figure 4.3. When the Convoy templates receives the synchronisation they can either transition up or down, shown in Figure 4.4. This will then cause the templates to either reject or accept the task.

Value Based Priority

When all tasks have been through the decision process, it is then up to the individual Convoy templates to determine how many satellites should be assign to each task. This is done using VBP.
VBP is based on the state of the Convoy template i.e. the current amount of data stored, which windows that are active, and the state of other convoys. The goal for Equation (4.1) is to return the number of satellites that should be assigned to `gather_new_data`. The equation's variables are detailed below:

- `obs_gather_cost[id]`, `obs_sender_station_cost[id]`, and `obs_sender_convoy_cost[id]` have a value between zero and ten depending on the status of the Convoy. For example, if the convoy has no data in storage and no satellites assigned to the `gather_new_data` task, the priority will be set to ten after the calculation. This is considered bad in almost any scenario except for the beginning where it is unable to avoid this initial value.
- `proc_in_use[id]` tells the model how many satellites are currently working.
- `convoys[id].satellites` correlates to the number of satellites the Convoy have.

In Equation (4.2) we see the actual values for the previous equation when the for the model time is zero. This results in the equations returning the value nine, meaning that

all satellites will be assigned to the `gather_new_data` task.

$$\frac{obs\_gather\_cost[id] * (convoys[id].satellites - proc\_in\_use[id])}{obs\_gather\_cost[id] + obs\_sender\_station\_cost[id] + obs\_sender\_convoy\_cost[id]} \tag{4.1}$$

$$\frac{10 * (9 - 0)}{10 + 0 + 0} \tag{4.2}$$

Similar calculations are made for when the Convoy template assigns satellites to the tasks `send_data`, and `transfer`, but those calculations use other factors to assign the correct number of satellites. For example, if a Convoy have a high storage and it is currently in range of a station, the `cost` of not assigning satellites to the `send_data` task should be high, but if the station that is available right now is five times as expensive as one that is soon to be available it should reduce the number of satellites assigned and wait for the other station to become available.
This is also done for the `transfer` task, but since there is no bandwidth cost associated with transferring data between convoys, the distribution calculation is altered. Instead, the Convoy observes how much storage is available on the other Convoy and compare the percentage to its own. The non-deterministic model tries to encourage the transfer of data to other convoys when one of the convoys have a lot of available storage. It is beneficial to transfer data to another convoy as it may be the case that the other convoy has access to a cheaper station.

Illustrative representation of VBP in the non-deterministic model

In Figure 4.5 an illustration of the VBP measures can be seen, on the x axis we have time and the y axis is the value of the different lines. The blue line represents the accumulated total storage, red is how much data that has been sent to all stations, same for yellow with respect to convoys, and last the green represents the constant maximum storage.

The blue line, total storage, will rise whenever data has been collected and decrease whenever data is sent to a station or another convoy. There is a spike where it is close to reaching maximum capacity which can be caused by a number of different reasons. In this scenario it seems it is not within reach of a station, or the cost of the station is too high compared to all the stations it can potentially communicate with later.
As time passes the yellow line goes up indicating it is transferring its data to another convoy until it finally reaches a station in which it find suitable to send its data to. Figure 4.6 follows the same time line as Figure 4.5, but shows how the different parameters goes up and down according to total storage capacity and available stations/convoys.

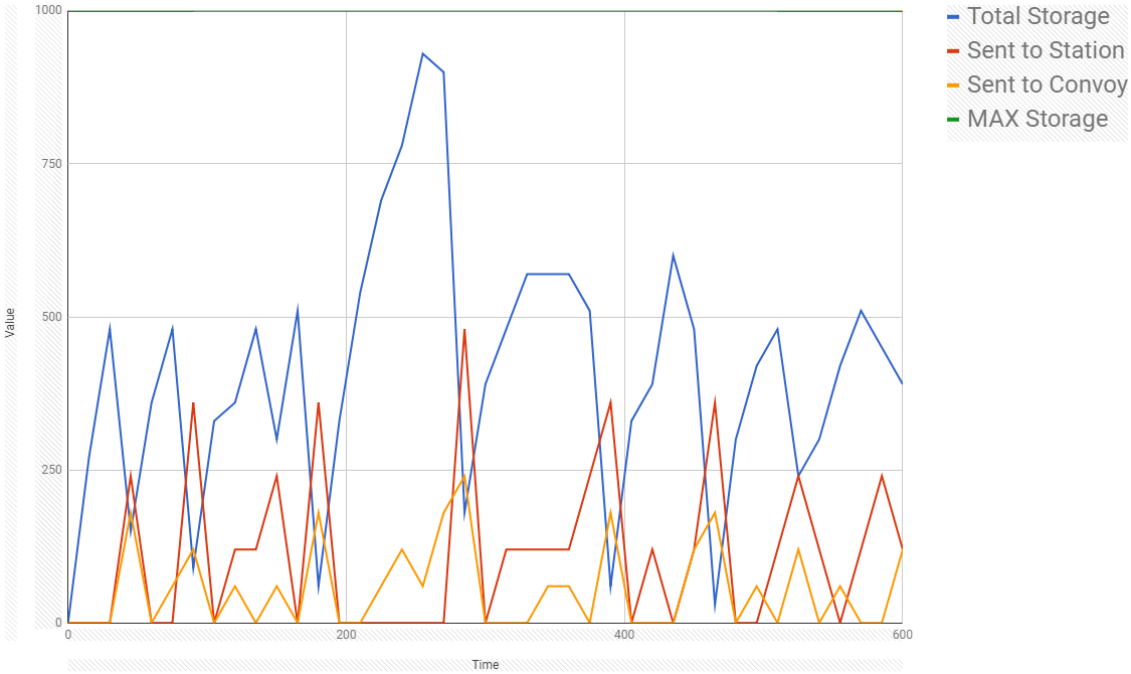

Figure 4.5: Monitor total storage, amount of data sent to stations, and convoys over 600 time units

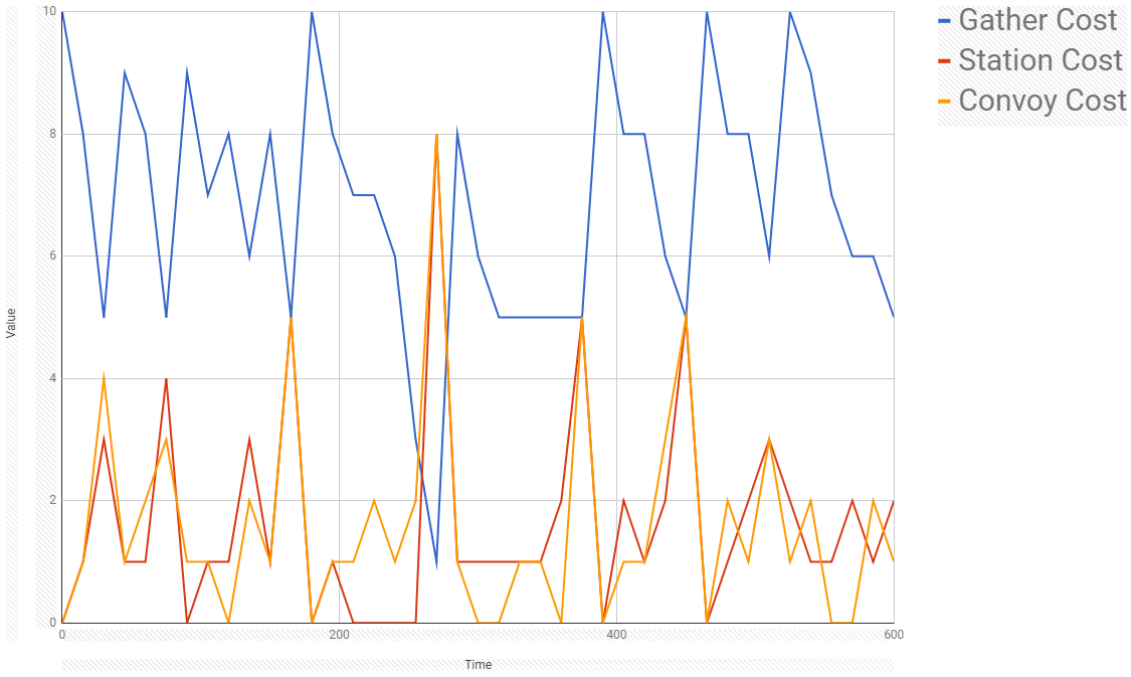

Figure 4.6: Monitor cost of different parameters over 600 time units

Stations

The stations are defined and used identically to how the deterministic model handled them. This means a convoy can have multiple satellites communicating with a single station. The stations have no limit on how many satellites that may send data to it simultaneously.

Windows

The windows behaves similar to those used in the deterministic model, there is however a difference in when they are available for the satellites. The satellites are distributed evenly around the orbit, meaning the convoy will always have a number of satellites in range of a station, if the current orbit comes within range. The windows are used to represent Earth's rotation and how the rotation affects what land mass the satellites flies above. As Earth rotates, the orbit is slightly skewed as it does not adjust to the rotation.
We handle this by assigning a constant number of satellites that may communicate with the stations, as long as the orbit comes in reach of the them.

Cost

The `cost` is calculated in the same manner as how we assign tasks to each Convoy template. The formula for this can be seen in Listing 4.1. The `cost` is based of a range between 0 and 10 for each Convoy . Each Convoy may have multiple stations available at a given time and all of them are examined since the bandwidth cost may vary between the stations. When an available station is inspected, a number is computed that uses the following factors:

- Percentage storage — uses current storage and maximum storage level
- Percentage cost — uses current station cost and the most expensive station cost out of all those available to the convoy
- Percentage occupied — uses current number of satellites currently assigned to sending data to the station, out of the maximum number of satellites able to send to the station

This number represent the cost for communicating with the stations i.e. if the convoy has full storage and all of its satellites assigned to `send_data` it results in a cost of 0. However, when we have half storage and two station's with different bandwidth cost and no satellites assigned to `send_data`, both stations should produce a cost. The station with a low bandwidth cost should produce a high cost, so future decision made by CORA should be to minimise this cost by assigning satellites to start sending data to the station, on the other hand if only the station with high bandwidth cost is available it should produce a low cost, so CORA is only incentivise to use this when the cost starts to raise, which only happens when storage levels start to increase as a result of not sending to any station or convoys while still gathering data. After each station have been calculated we add up the values and divide it to fits into a range between 0–10, as seen on line 28–34. The final accumulated cost is then assigned to the variable `calculated_cost`, seen on line 35.

```
1      internal_cost = find_highest_cost();
2      for (s = 0; s < STATIONS ; s++){
3           sender_cost[id][s] = 0;
4           if (convoys[id].send_to_station[s] != 0){
5                if (
6                    ((100-storages[id]*100/convoys[id].storage)
7                 * ((internal_cost*100)/stations[s].cost))
8                 + (((internal_cost*100)/stations[s].cost)
9                 * (100-send_to_station[id][s]*100/convoys[id].
                      send_to_station[s]))
10                < ((storages[id]*100/convoys[id].storage)
11                * ((internal_cost*100)/stations[s].cost))
12                + (((internal_cost*100)/stations[s].cost)
13                * (send_to_station[id][s]*100/convoys[id].send_to_station[s
                      ])))
14                {
15                     sender_cost[id][s] = (
16                        (100-storages[id]*100/convoys[id].storage)
17                     * ((internal_cost*100)/stations[s].cost))
18                     + (((internal_cost*100)/stations[s].cost)
19                     * (100-send_to_station[id][s]*100/convoys[id].
                          send_to_station[s]));
20                }
21                else{
22                     sender_cost[id][s] = (
23                        (storages[id]*100/convoys[id].storage)
24                     * ((internal_cost*100)/stations[s].cost))
25                     + (((internal_cost*100)/stations[s].cost)
26                     * (send_to_station[id][s]*100/convoys[id].
                          send_to_station[s]));
27                }
28                total_cost += (internal_cost*100)/stations[s].cost;
29                n++;
30           }
31           else{
32                sender_cost[id][s] = (
33     (storages[id]*100/convoys[id].storage)
34              * ((internal_cost*100)/stations[s].cost))
35              + (((internal_cost*100)/stations[s].cost)
36              * (send_to_station[id][s]*100/convoys[id].send_to_station[s]));
37           }
38           total_cost += (internal_cost*100)/stations[s].cost;
39           n++;
40      }
41      obs_sender_station_cost[id] += sender_cost[id][s]/(100/n);
42 }
43 obs_sender_station_cost[id] /= total_cost/10;
44 obs_sender_station_cost[id] /= n;
45 calculated_cost += obs_sender_station_cost[id];
```

Listing 4.1: Code for calculating `cost`

4.4 State Space Concerns

With three defined tasks and three convoys in the non-deterministic model, each time the convoys are suggested a task it creates a maximum of $(2^3)^3$ choices. Table 4.1 illustrates the magnitude given 1–5 suggestion(s).

Table 4.1 indicates that the number of actions the model can take quickly grows out of

proportion. This led us to restrict the non-deterministic behaviour in some parts of the model. One instance where the non-determinism have been restricted is when determining how many satellites that should be assigned to a given task.

Suggestions	Traces to explore:
1	512
2	262.144
3	134.217.728
4	68.719.476.736
5	35.184.372.088.832

Table 4.1: Calculating number of traces/options to explore

Based on previous experience [2] with how many traces CORA quickly generates, we would like to limit the number of choices that the model is presented with. To do so, four measures have been implemented which are optional and can be tweaked to generate a best trace within the constraints.

1. The `gather_new_data` task must be accepted if the storage is below 50 % for the individual convoy
2. The `send_data` task cannot be accepted if storage is below 50 % of the convoy's total storage, and must be taken if the storage is above 75 %
3. The `transfer` task is identical to `send_data` in which it only has non-deterministic behaviour when storage is between 50 and 75 %
4. The `gather_new_data` task is always accepted regardless of the convoys storage, this overrules the first reduction to even further reduce state space

These constraints are then wrapped in methods that evaluates to true or false, and placed on the edges that accept or reject tasks in order to limit CORA's options. The effect of these measures will be experimented with in order to measure how much they impact the model.

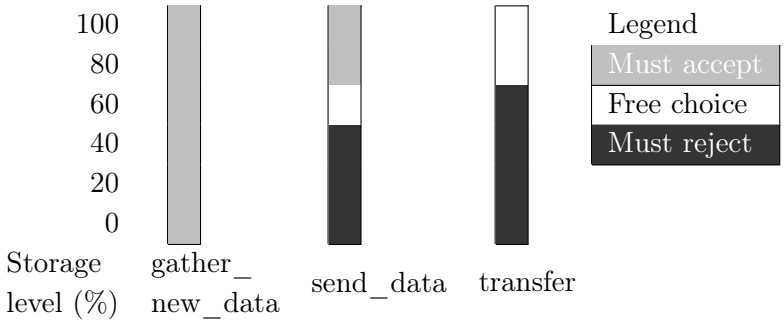

Figure 4.7: Thresholds for the individual tasks

In Figure 4.7i a visual illustration of these storage percentage restrictions can be seen. The three columns corresponds to the three tasks, where the column height indicates at what storage level the different task can be taken, i.e. `send_data` cannot be taken before the convoy's storage is above 50 %, and must be taken when above 75 %. CORA has free choice in-between the the 50 and 75 % thresholds.

4.5 Experimenting with the Non-deterministic Model

In this section we will perform several experiments on the non-deterministic model. We will be exploring which parts of the model have the most significant impact in regards to the state space. As in Section 3.4 our hypothesis and motivation for each experiment will be presented prior to discussing the results.
The reasoning for performing these experiments are to test which values have the most and least significant impact on the model. To test the implemented scheduling method, as well as the feasibility of using a non-deterministic approach to generating a trace.

The results of each experiment will be presented next to the results of running the initial configuration and the percentage difference between the two. The section will end with a comparison of the maximum trace length for each experiment and lastly a conclusion capturing the findings of the experiments.

4.5.1 Initial Configuration

Before performing the experiments we will be presenting the initial configuration and the results of finding the best trace after 200 units of time. The reason for only running the experiments for 200 units of time is that CORA, when finding the best trace, uses exhaustive space exploration which may consume too much memory to finish. It is possible to get a trace for 300 time units with the base case, but since some of the experiments may consume more memory, we want to test with a shorter trace.
The results of the initial configuration run will be used as a base case, to which the results of each experiment will be compared against.

The base case consist of two convoys with ten satellites in each. Additionally, each convoy will have two stations each where their tasks `send_data` can be executed when the convoy's orbit is in range. Both convoys will be able to perform the same type of tasks, have equal storage capacity, and equal initial storage level. All this can be seen in Table 4.2 along with values associated with the stations, and tasks.
The `Tasks` column in Table 4.2 lists four tasks, even though the Global Settings row dictate that there are only three tasks available. The reason for this conflict is because only three tasks can be initiated by the convoy itself, the forth one, `Receive_From_Convoy`, is only assigned when another convoy is transferring data to it via the `transfer` task.

Base Case

From the base configuration we produce values for the following variables, seen in Table 4.3. This is the values that will be used for our base case, to see changes when running experiments.

- Cost of Bandwidth (CoB) – Accumulated cost of all the data sent to each station
- CoB:ratio – Indicate how much each single data unit costed to send.
- Station data – Show how much data each station has received, where the numbering refer to individual station.
- Total data – accumulated data of all the stations.
- Runs:<TASK> – How many times each convoy has executed that task
- Transfer – How much data the convoy has transferred.

					Convoy Thresholds		Station Thresholds	
Global Settings	Tasks	Convoys	Stations	Storages	Low	High	Low	High
	3	2	4	(500, 500)	70	100	30	40

Convoys	satellites	send_data	transfer	gather_new_data	storage	tasks_depending_on_windows
Convoy #1	10	(2, 1, 0, 0)	(0, 3)	10	1000	((-1, -1, -1, -1), (1, 1, 0, 0), (-1, -1, -1, -1), (-1, -1, -1, -1))
Convoy #2	10	(0, 0, 1, 2)	(3, 0)	10	1000	((-1, -1, -1, -1), (0, 0, 1, 1), (-1, -1, -1, -1), (-1, -1, -1, -1))

Stations	cost	data_recieved
Station #1	2	0
Station #2	5	0
Station #3	3	0
Station #4	2	0

Tasks	task_type	execution_time	data_rate
gather_new_data	0	15	2
send_data	1	15	4
transfer	2	15	4
Receive_From_Convoy	3	15	4

Table 4.2: Initial configuration for the experiments

- Receive – How much data the convoy has received.
- Storage – How much data is left in each convoy.

General fields			Convoy fields		
				convoy_1	convoy_2
CoB		6420	Runs:send_data	20	24
CoB:ratio		2.61	Runs:gather_new_data	50	38
Cora: Cost		2600	Runs:transfer	6	2
Station data	1	840	Runs:receive	2	6
	2	360	Transfer	360	120
	3	420	Receive	120	360
	4	840	Storage	410	530
Total data		2460			

Table 4.3: Results of running the initial configuration

4.5.2 Experiment 1: Finer Granularity of Scoring

Experiment: Scores given when assigning tasks have been changed from 0–10 to 0–100.
Motivation: This experiments is performed to observe the effect of scoring with higher ranges to determine the impact compared to the base case. Testing this is important as the 0–10 range cuts off the most significant decimal, which may be more important than initially thought.
Hypothesis: For this experiment we hypothesise

1. Close to similar results across all fields

Results: The top section of Table 4.4 shows close to similar results compared to the base case. Whereas the bottom part shows that the path taken is remarkably different.
The reasons why it did not take the same path is a result of some of the priorities not having the same score e.g. the score of two tasks in the base case may both have been 5, but a wider range revealed these to be 51 and 58.
Additionally we notice that the CORA cost has decreased while the CoB increased, despite maintaining the same total amount of data gathered from the stations. This is because

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	6540	2.659	2390	840	420	360	840	2460
Diff (%)	**+1.9**	**+1.9**	**-8.1**	**0**	**+16.7**	**-14.3**	**0**	**0**

	`convoy_1`			`convoy_2`		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	21	**+5.0**	24	23	**-4.2**
Runs:gather_new_data	50	40	**-20.0**	38	48	**+26.3**
Runs:transfer	6	2	**-66.7**	2	4	**+100.0**
Runs:receive	2	4	**+100.0**	6	2	**-66.7**
Transfer	360	120	**-66.7**	120	240	**+100.0**
Receive	120	240	**+100.0**	360	120	**-66.7**
Storage	410	410	**0**	530	530	**0**

Table 4.4: Results of running experiment 1

CORA cost is not strictly tied to the CoB, but also consider the storage level of each convoy, and how many satellites each convoy have assigned to sending data to a station. This indicates that the process of how the CORA cost is calculated should be reworked to focus more on CoB. It should be changed such the CORA cost rises whenever the CoB goes up.

4.5.3 Experiment 2: Double Satellites in All Convoys

Experiment: Increase the number of satellites in each convoy from 10 to 20. Additionally the number of satellites that can be assigned to each task, and the storage capacity have been doubled.
Motivation: To test what impact increasing the number of satellites will have to the maximum schedule length.
Hypothesis: For this experiment we hypothesise

1. CoB will approximately double
2. Stations data received will doubled
3. More `send_data` and `gather_new_data` task executions overall
4. Expect similar maximum trace length. This may seem counter intuitive but since we hypothesise adding satellites does not introduce new choices, it will have little effect on the state space.

Results: Doubling the amount of satellites shows that our first three hypothesis are true, and the last is partially true as it was still able to run 200 time units. Also worth noting is the decrease in CORA cost, indicating that with double the satellites it is better at balancing its storage level throughout the run.

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	13140	2.639	1985	1680	780	840	1680	4980
Diff (%)	+**104.7**	+**1.1**	**-23.7**	+**100.0**	+**116.7**	+**100.0**	+**100.0**	+**102.44**

	`convoy_1`			`convoy_2`		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	41	+**105.0**	24	42	+**75.0**
Runs:gather_new_data	50	85	+**70.0**	38	80	+**110.5**
Runs:transfer	6	3	**-50.0**	2	1	**-50.0**
Runs:receive	2	1	**-50.0**	6	3	**-50.0**
Transfer	360	180	**-50.0**	120	60	**-50.0**
Receive	120	60	**-50.0**	360	180	**-50.0**
Storage	410	670	+**63.4**	530	760	+**43.4**

Table 4.5: Results of running experiment 2

4.5.4 Experiment 3: Adding a Convoy

Experiment: Increase the number of convoys from 2 to 3. The new convoy will have 10 satellites as the other convoys, and have access to the stations #2 and #3
Motivation: To investigate the impact an additional convoy will have to the maximum schedule length.
Hypothesis: For this experiment we hypothesise

1. 50 % higher CORA: cost
2. 50 % higher CoB
3. 50 % more data send to stations
4. More internal communication
5. Reduced maximum trace length. Unlike adding satellites, adding convoys causes the model to make more choices and thus increase the state space.

Attempt 1

Results: CORA runs out of memory and can therefore not produce any results

Attempt 2

Adjustments: Changing the lower threshold for convoys from 50 to 55. This reduce the number of choices taken, and may allow the query to finish.

Results: All of the stated hypothesis are true, except for the 50 % higher CoB, which increased by 67,3 %. Total internal communication for the base case was 8 which increased to 9, however, we expected this to be higher because convoy #3 only had access to stations with a higher CoB, making it less ideal to send to.
From the results we see that `convoy_3` ended up getting the most `Receive_From_Convoy` tasks. This is unexpected as `convoy_3` has access to the more expensive stations. This is likely a result of the other convoys assigning their satellites first, and `Receive_From_Convoy`

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	10740	2.934	3980	840	720	1260	840	3660
Diff (%)	**+67.3**	**+12.4**	**+53.1**	**0**	**+100.0**	**+200.0**	**0**	**+48.78**

	convoy_1			convoy_2			convoy_3		
	Base	Result	Diff (%)	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	20	**0**	24	23	**-4.1**	N/A	20	**N/A**
Runs:gather_new_data	50	45	**-10.0**	38	49	**+28.9**	N/A	41	**N/A**
Runs:transfer	6	2	**-66.6**	2	4	**+100.0**	N/A	3	**N/A**
Runs:receive	2	2	**0**	6	2	**-66.6**	N/A	5	**N/A**
Transfer	360	120	**-66.6**	120	240	**+100.0**	N/A	180	**N/A**
Receive	120	120	**0**	360	120	**-66.6**	N/A	300	**N/A**
Storage	410	500	**+21.9**	530	440	**-16.9**	N/A	530	**N/A**

Table 4.6: Results of running experiment 3

can not be rejected. Alternatively, it can be the result of an inaccuracy in the scoring function.
It is difficult to compare these results to the base case, as an adjustments was needed before being able to run the query, this could potentially lock out better paths.

4.5.5 Experiment 4: Different Task Execution Time

Experiment: Change the execution time of all task which where 15, so `gather_new_data` takes 14 time units, `send_data` takes 15, and `transfer` and `receive` takes 16.
Motivation: This is a more realistic set up as it is doubtful all tasks will take the same- or even a dividable amount of time. This will impact the maximum trace length significantly, because the Scheduler template is able to suggest tasks more rapid.
Hypothesis: For this experiment we hypothesise

1. Reduced trace length
2. Slightly higher amount of tasks being performed

Attempt 1

Results: CORA runs out of memory and can therefore not produce any results

Attempt 2

Adjustments: Changing convoy lower threshold from 50 to 70.

Results: Given the first attempt failed, reduced schedule length is confirmed. The second hypothesis was partially true as both `send_data` and `gather_new_data` has increased but by much more than expected. Whenever a task finishes new tasks are suggested to the convoys. In the base case, tasks were suggested at time 15, 30, 45 but this experiment could potentially change these timestamps to 14, 15, 16, 28 and so on. But due to it not producing a result with similar settings to the base case it is difficult to compare the

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	8488	2.666	1917	1072	532	524	1056	3184
Diff (%)	+**32.2**	+**2.1**	**-26.3**	+**27.6**	+**47.8**	+**24.8**	+**25.7**	+**29.43**

	convoy_1			convoy_2		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	27	+**35.0**	24	27	+**12.5**
Runs:gather_new_data	50	70	+**40.0**	38	66	+**73.7**
Runs:transfer	6	2	**-66.7**	2	2	**0**
Runs:receive	2	2	**0**	6	2	**-66.7**
Transfer	360	128	**-64.4**	120	128	+**6.7**
Receive	120	128	+**6.7**	360	128	**-64.4**
Storage	410	766	+**86.8**	530	678	+**27.9**

Table 4.7: Results of running experiment 4

validity other than changing the execution times have a substantial effect on the maximum trace length.

4.5.6 Experiment 5: Removing Constraints

Experiment: Each convoy will be able to accept or reject any task including `gather_new_data` i.e. the convoys are never forced to allocate satellites to perform a specific tasks.
Motivation: This experiment will make all choices truly non-deterministic rather than some of the choices being based on values specified by us. This is going to test the effect of the forced behaviour in the model.
Hypothesis: For this experiment we hypothesise

1. Reduced maximum trace length
2. Lower CORA cost

Results: It was not possible to produce a trace of time 200 for this experiment. Because of this no results have been included. Regardless, this shows that it is needed to enforce some behaviour, otherwise CORA will consume all the available memory.

4.5.7 Experiment 6: Increase Windows' Span

Experiment: Increase all station windows by 20 %, meaning that the satellites have 20 % more time to send data.
Motivation: As the duration satellites are over stations are defined using a formula with parameter values defined by us, it should be tested whether or not these values are appropriate for the initial configuration.
Hypothesis: For this experiment we hypothesise

1. More data sent to the stations with a bandwidth cost of 2 or less, and less data sent to the other stations

2. Minimal effect on trace length
3. Lower CORA cost
4. Increase in `gather_new_data` and `send_data` tasks. As there are better possibilities for sending data, there will be more storage available to gather additional data.

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	7680	2.56	2360	1080	420	420	1080	3000
Diff (%)	+**19.6**	**-1.9**	**-9.2**	+**28.6**	+**16.7**	**0**	+**28.6**	+**21.95**

	`convoy_1`			`convoy_2`		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	25	+**25.0**	24	28	+**16.7**
Runs:gather_new_data	50	56	+**12.0**	38	50	+**31.6**
Runs:transfer	6	3	**-50.0**	2	1	**-50.0**
Runs:receive	2	1	**-50.0**	6	3	**-50.0**
Transfer	360	180	**-50.0**	120	60	**-50.0**
Receive	120	60	**-50.0**	360	180	**-50.0**
Storage	410	410	**0**	530	530	**0**

Table 4.8: Results of running experiment 6

Results: The two stations with a bandwidth cost of 2 received significantly more data, with a lower CORA cost and better CoB:ratio. One of the more expensive stations also experienced an increased in data received. This is because the two convoys are able to send more and thus have the available storage to gather more data as seen in the bottom part of the table.

4.5.8 Experiment 7: Decrease Windows' Span

Experiment: Decrease all station windows by 20 %.
Motivation: The above experiment showed it was impactful to increase the window span. We wish to know whether or not this will be the case if these are decreased.
Hypothesis: For this experiment we hypothesise

1. Less data sent to all stations
2. Higher CORA cost
3. Higher CoB:ratio
4. Less `gather_new_data` and `send_data` executed

Results: None of the hypothesis held true. The only change is `convoy_2` has performed fewer `send_data` tasks. The reason why there are no variations to the amount of data sent to the stations, is that the base case has stopped before three of the `send_data` tasks were completed. The amount of task runs are incremented when the they are assigned whereas the data are not updated until the tasks are completed.
The reason for there being no difference otherwise is likely due the windows still being large enough for the satellites to finish the tasks they are executing.

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	6420	2.61	2600	840	360	420	840	2460
Diff (%)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

	convoy_1			convoy_2		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	20	**0**	24	21	**-12.5**
Runs:gather_new_data	50	50	**0**	38	38	**0**
Runs:transfer	6	6	**0**	2	2	**0**
Runs:receive	2	2	**0**	6	6	**0**
Transfer	360	360	**0**	120	120	**0**
Receive	120	120	**0**	360	360	**0**
Storage	410	410	**0**	530	530	**0**

Table 4.9: Results of running experiment 7

4.5.9 Experiment 8: Increase Threshold for Convoys

Experiment: Widen the interval for which CORA may choose to send to another convoy. This interval are changes from 50–100 to 25–100.
Motivation: As thresholds are values defined by us, it should be tested whether or not the chosen values are valid or should be changed.
Hypothesis: For this experiment we hypothesise

1. Lower CORA cost. As the model is able to make more choices it should be able to find a better path.
2. Higher CoB:ratio. As the data is expected to be transferred more between convoys it may cause small variations to the amount of data received by each station e.g. station #1 might receive 30 less data and station #4 receive 30 more data.
3. Increase in the number of `transfer` tasks

Results: All hypothesis where true except for CoB:ratio being higher. This may be a result of the cheaper stations already being used to their maximum capability.

4.5.10 Experiment 9: Decrease Threshold for Convoy

Experiment: Change the threshold for convoy decisions from 50–100 to 75–100 i.e. narrowing the range of when the convoy may chose to transfer to another convoy.
Motivation: As thresholds are values defined by us, it should be tested whether or not the chosen values are valid or should be changed.
Hypothesis: For this experiment we hypothesise

1. Higher CORA cost
2. Fewer `transfer` tasks

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	6420	2.61	2540	840	360	420	840	2460
Diff (%)	**0**	**0**	**-2.3**	**0**	**0**	**0**	**0**	**0**

	convoy_1			convoy_2		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	20	**0**	24	24	**0**
Runs:gather_new_data	50	42	**-16.0**	38	46	**+21.1**
Runs:transfer	6	5	**-16.7**	2	4	**+100.0**
Runs:receive	2	4	**+100.0**	6	5	**-16.7**
Transfer	360	240	**-33.3**	120	240	**+100.0**
Receive	120	240	**+100.0**	360	240	**-33.3**
Storage	410	410	**0**	530	530	**0**

Table 4.10: Results of running experiment 8

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	6420	2.61	2695	840	360	420	840	2460
Diff (%)	**0**	**0**	**+3.7**	**0**	**0**	**0**	**0**	**0**

	convoy_1			convoy_2		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	20	**0**	24	24	**0**
Runs:gather_new_data	50	44	**-12.0**	38	46	**+21.1**
Runs:transfer	6	0	**-100.0**	2	0	**-100.0**
Runs:receive	2	0	**-100.0**	6	0	**-100.0**
Transfer	360	0	**-100.0**	120	0	**-100.0**
Receive	120	0	**-100.0**	360	0	**-100.0**
Storage	410	470	**+14.6**	530	500	**-5.7**

Table 4.11: Results of running experiment 9

Results: Both statements where true. Showing that increasing and narrowing the span where the model may chose to transfer has the opposite effect.

4.5.11 Experiment 10: Increase Threshold for Stations

Experiment: Increase the upper and lower limit for when CORA can decide to send data to a station, changed from 30–40 to 20–60.
Motivation: As thresholds are values defined by us, it should be tested whether or not the chosen values are valid or should be changed.

Hypothesis: For this experiment we hypothesise

1. Lower CORA cost
2. Higher CoB:ratio
3. More total data sent

				Station data				
	CoB	CoB:ratio	CORA cost	1	2	3	4	Total
Base	6420	2.61	2600	840	360	420	840	2460
Result	6420	2.61	2600	840	360	420	840	2460
Diff (%)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

	convoy_1			convoy_2		
	Base	Result	Diff (%)	Base	Result	Diff (%)
Runs:send_data	20	20	**0**	24	21	**-12.5**
Runs:gather_new_data	50	50	**0**	38	38	**0**
Runs:transfer	6	6	**0**	2	2	**0**
Runs:receive	2	2	**0**	6	6	**0**
Transfer	360	360	**0**	120	120	**0**
Receive	120	120	**0**	360	360	**0**
Storage	410	410	**0**	530	530	**0**

Table 4.12: Results of running experiment 10

Results: All hypothesis where false, this shows that the non-deterministic model will never consider sending data with a storage level below 30 %. Additionally it seems it will try to send when above 40 % storage.

4.5.12 Schedule Length Experiment

Throughout most of the experiments it was possible to finish the query finding a trace of length 200 without any additional modifications. However, for a few of the experiments it was not possible to finish the query. Because of this and to answer the many hypothesis revolving around the maximum trace length, a last experiment will be performed. In this experiment we will find the maximum trace length for each of the experiments described in this section.
This will be done by testing different intervals, as finding the precise value would take many attempts. The intervals there will be tested are; 50, 75, 100, 150, 200, 300, 400, 600, 800, 1200, and 1600.
In Table 4.13 a list of all the experiments can be seen along with their maximum trace length. Most noticeable is *Decrease threshold for convoys* as it was able to find a trace of 1600 time units. The reason the independent experiment can do so is that it can remain below the convoy threshold in most of the run, reducing the number of times it has to make a decision. In the other experiments this was not possible but by decreasing the threshold for when to transfer to other convoys the amount of choices where decreased significantly.

We can conclude that with the non-deterministic model the most demanding factors are multiple tasks with different execution times, and allowing CORA to find the optimal

path without guiding it. Meaning, sometimes enforcing what tasks to execute. These experiments are mentioned as they where only able to produce a trace of 75 time units.

Experiments	Schedule Length
Initial Configuration	300
Finer Granularity of Scoring	400
Double Satellites in All Convoys	300
Adding a Convoy	150
Different Task Execution Time	75
Removing Constraints	75
Increase Windows' spans	400
Decrease Windows' spans	300
Increase Threshold for Convoys	200
Decrease Threshold for Convoys	1600
Increase Threshold for Stations	200

Table 4.13: Maximum schedule length for each experiment.

4.6 Conclusion for Experiments

Through the experiments some valuable information was gathered. Most of the results from the experiments holds true to the hypothesises, but the once that stand out where changing granularity of scoring, decrease windows' span, increase threshold for stations, and decrease threshold for stations.

Changing granularity of scoring causes the best trace to make some different decisions early on in the trace, which resulted in the final one to be significantly different. This did not end up as a problem as the CoB, CoB:ratio, CORA cost and data sent to the stations values were similar to that of the base case, which would imply that the base granularity is sufficient for finding the optimal path. We learnt, from the same experiment, that the calculation for the CORA cost could be improved such that the CoB:ratio will have a larger impact.

Decrease windows' span did not behave as expected, because it stayed true to the base case and as we assumed it would decrease in data sent to the stations. This may be because that only decreasing the amount by 20 % was not enough to make an impact to when we used to send data to the stations.

Similarly, the increased threshold for stations experiment did not behave as expected either. The reason for this might be that when the storage for a given convoy is low, it assigns many satellites to the `gather_new_data` task. This results in the storage level potentially increasing such that the 40 % threshold is met.

The experiments show us the following results; the base case configuration may not be the best configuration in regards to showcasing some of the implemented features that were altered and tested, such as the thresholds and window spans.
Given that not much improvement was shown in some of the experiments where CORA was less restricted in choosing what tasks should be executed. This indicates that non-

determinism may not be the most important factor to consider when making a schedule. The underlying logic for determining optimal decisions based on the knowledge about the satellites or convoys state may provide better results. This is tied with our finding from the trace length observations, which indicate that if we where to give CORA more freedom, it would not be able to produce a schedule longer than an two hours at best.

Discussion 5

In this chapter we will discuss some of the alternatives to the choices taken throughout the project. This will focus on the construction of the different models, and why we have done what we did. Each topic will highlight a problem that we encountered after testing or reviewing a particular feature, and we will discuss alternative routes that could have been taken, with the knowledge obtained since. There may not always be a solution to the specific topic and if a solution is found there are in most cases no proof it would be an improvement. But it is important to consider potential limitations.

5.1 Task Suggestion

When a new task is suggested in the deterministic model it is suggested because it has the highest priority. Ties are resolved by choosing the task with the lowest index in the array that contains all of the tasks. The suggestion does not consider whether the task is available or not. This means that some of the suggested tasks can not be started right away. This causes the satellite to idle if it does not have any available tasks in its queue.
This issue could be resolved by changing how suggestions are made. Instead of sending a single task as a suggestion to the Scheduler template, an array of tasks that are arranged by their current priority could be sent. The Scheduler template would then check which are available and enqueue the available one with the highest priority.
This feature will probably make the scheduler more efficient in executing more tasks, but it was not made into an experiment in Section 3.4 as we were not testing the efficiency of the scheduler.

5.2 VBP in the Deterministic Model

The non-deterministic model implemented a VBP for determining each task's importance. This allow for a more dynamic task priority with a finer granularity in comparison to how the deterministic model handles priority. This approach would have been interesting to implement in the deterministic model. We believe doing this would result in a better schedule i.e. more data would be sent to Earth.

5.3 Non-Deterministic Base Case

The results of the non-deterministic experiments showed in some cases that the base case may not have been configured optimally in regards to showcasing all of the features of the model, this was clear from the experiments regarding the lower threshold for station,

which had little to no effect on the results. The problem stems from a lack of us not observing some potentially impactful variables, that we did not know could have helped gain the necessary insight into the behaviour of the model. The variables we would had like to observe should provide information about the maximum potential data sent to each station based on their windows, and convoy efficiency in regards to how well the convoys utilised the satellites they may communicate with. With these variables, we will be able to observe if a station is already used to its maximum potential. This would potentially alter the base configuration and thus the base case. With a better base case, it might have been possible to have gained more insight into the model and possible undiscovered problems.

5.4 VBP

The priorities used to calculate the importance of each task have some minor flaws, which will be explained in this section along with some possible addition to how the implemented VBP could become more accurate. When calculating the VBP for determining the number of `send_data` tasks that should be assigned, the percentage- storage, cost of station, and satellites already sending to the station, are used. It is mainly the cost of station that should be changed so it is aware of what the cheapest and most expensive station overall is, instead of only considering the once currently available.
If a convoy only has one station available to it, it will assume that the station is the best to send to. This is somewhat true, but if it was able to see what convoys it could communicate with and what the cost of their available stations are, it might be a better choice sending its data to other convoys rather than sending to an expensive station.

The same applies for the VBP calculation when concerning `send_to_convoy`. This should include the knowledge about what station the receiving convoy has available to it. As it does not make sense to transfer data to a convoy not associated with any stations, or to send to a convoy associated with more expensive stations only.

5.5 Windows

The non-deterministic model uses the same concept of windows as the deterministic model, but because it looks at all the satellites at a much greater scale some modifications was made to the windows. The convoy consist of a number of satellites that are evenly distributed in the orbit. Each convoy have a variable to define how many satellites are over a station, so when one satellite goes out of range of a station, another enters goes into range.
Given that the station's range is defined by a radius, its area is circular and it is therefore not correct to assume that some defined amount of satellites can always be in range of the station, even thought the orbit is above it. The reason for this is that when the orbit of the convoy first enters the station's circular area, only one satellites will be able to communicate with it briefly. As time passes the windows increase until one of the satellites are always above the station's area. This continues to increase until the satellites are right above the station in which the window starts to shrink again. This is a result of the orbit not matching the rotation of the Earth.

5.6 Schedule

It was discovered that the non–deterministic model was unable to produce a schedule. Each convoy have a list of satellites representing if they are currently executing a task. Additionally storage is not tracked on each satellite but rather as a total amount on the convoy. When we find the lowest amount of time that should be passed in order for a task to finish, we first sort the list based on execution time. When new task are assigned they choose the first available satellite.
This mean that even if we simply added an id to the satellites to keep track of them when we sort the list, it will still result in the first couple of satellites will be used more then the once in the end of the list. For us to fix this each satellites will need to have its own storage and not use a total storage over all the satellites in the convoy. By doing so would require some more modification to some of the logic, the convoy will no longer need to calculate how many satellites should execute n task, but the satellites itself will use the VBP to determine what task it should execute. Additionally the checking when a satellites it within a window should be similar to that of the deterministic model.

5.7 Satellite Offset Strategies

The satellites in the deterministic model is distributed throughout the orbit with an offset to their current position, this is handled by delaying until the offset is reached before a satellite may enter the location `Start`. And when it is checked whether or not a satellite is within a window, its offset are subtracted from the total time in order to answer this.
The problem with the deterministic solution is that the satellites with an offset are forced to idle until that offset is met. This means the smaller an offset, the longer a schedule. It is also problematic if those with a small offset are unable to gather data themselves, and thus dependent on others to transfer data to them, as they will have to wait for them to become active.

Instead all satellites could start working at time zero until the specified schedule time, being considered to be at $time + sat.offset$ time. This means that the model would always produce a schedule over the specified amount of time. However, for the very first schedule this may cause some inaccuracies as, say the first satellite launched would not be able to work while the others are being launched.

5.8 Predicting Storage Level

In the deterministic model it might have been beneficial if the Satellite template was able to predict the future storage capacity of the satellite. Consider a satellite at a storage level of 50% currently gathering data, which will cause it to reach 80% when finished. This is considered to be a `HIGH` storage level and gathering more data would not yield the maximum amount of data. It could have been implemented such that the satellite used this information to suggest a transfer task rather than another gather task as the gather task will quickly be preempted anyway.
We believe doing this would cause a minor reduction to the state space as well as a schedule which utilise time more efficiently.

5.9 Dynamic Execution Time For Tasks

All of the tasks have a fixed execution time that must be met before the task is considered complete. This could be changed such that time required in order to complete a task was based on the state of the satellite. Instead of assigning a constant execution time, this could be recalculated every time a task is suggested or about to be started. For example, the execution time for sending data to a station could be based on the current amount of stored data, the calculation would then be: $execution\_time = |stored_data/data_rate|$ or if not able to empty its storage prior to the window ending simply run until the window does so. The *stored_data* is the amount stored data on the satellite and *data_rate* is the amount of data that is send per time unit for the task. The deadline should be adjusted as well in order to fit the new execution time. Additionally, if the task is restricted by a window, the deadline should be set to when the window closes. If this approach was taken, the use of the processors will more efficient.

The time is already approximated by the use of a variable that is incremented every time a new task suggestion is made in the Satellite template, as described in Section 3.3.4. But that solution is flawed as it is sometimes imprecise or very expensive in regards to program time if set to do so every time unit, as showed in Section 3.4.14.

5.10 Predicting Windows

Both models suffer from not having the ability to predict upcoming windows. The deterministic model could determine a better path if it had knowledge about upcoming windows. In this way it would be able to idle for a while instead of starting another task just for it to be preempted moments later.
For the non–deterministic model, combined with CORA's exhaustive search strategy, it already produces the best path. The potential benefits of adding this prediction to the non–deterministic model is to reduce the amount of choices CORA has to traverse. This would free up state space and could result in fewer constraints and an overall better path because of it.

5.11 Include Schedule in Model

As described in Section 3.4.15, the most time– and storage consuming part of generating a schedule was writing the trace. A way to potentially avoid this, instead of modifying SMC, would be to store the necessary information in arrays in the model. By doing this the schedule would be included in each state, increasing the size of the states but making only the very last state relevant to print, which is already possible.
Such arrays should include at least a satellite identifier and the values described in Section 3.5, these being; task id, start time, end time, and number of slews.

Conclusion 6

In this chapter we will conclude on this report and the project as a whole. Through this we will answer the problem statement written in Chapter 1:
How can different forms of model checking be applied to generating a schedule for multiple satellites in a convoy, and how does such methods compare against each other?
The model checking tools used for this project are UPPAAL- CORA and SMC.

We constructed a deterministic model in SMC with the goal of testing how influential some of the features we presented in Chapter 2 were, and in order to test the feasibility of generating a schedule using a detailed model. The features were tested as it would be beneficial to know which of them could be excluded from the non-deterministic model as we hypothesised that the non-deterministic model would suffer from a large state space, despite of it being less detailed.
The deterministic model implements a preemptive scheduler with dynamic prioritising. This model was able to generate a schedule with three satellites for an entire day and print the finished schedule within few minutes (2 minutes and 34 seconds). The required time was drastically increased to 3 hours and 38 minutes when ten satellites was scheduled. A schedule for thirty satellites was calculated in 53 minutes, but the trace was omitted as we predicted that it would take seven years to print it.
We believe this method is feasible for generating schedules for a convoy of thirty satellites if it is possible to alter SMC such that only some variables are printed. We do not believe this method is feasible for generating schedules for a convoy of thirty satellites with the current version of SMC.

We constructed a non-deterministic model in order to utilise CORA's cost optimisation. The model was less detailed as the focus was not on the individual satellites, but rather on convoys. As the model was constructed in CORA, all choices taken were the best possible in regards to how the cost rate is calculated.
This model implements a VBP scheduler, meaning the priorities changes throughout the run based on the state of the model. We were able to generate a trace of 5 hours with the initial configuration of the model. The length of the schedule could be extended by restricting the options available for some given states. Noticeably the experiment revolving around decreasing the threshold for convoys, Section 4.5.10, yielded similar results to that of the base case, even though the CORA cost had little impact. Additionally it was possible to produce a significantly longer trace as the configuration was more restricted. This experiment signifies that the scheduling logic, on its own, is capable of making close to optimal choice's.

This leads us to believe that implementing scheduling logic similar to that in the non-

deterministic model, into a deterministic model is the more favourable option.

Future Work 7

This chapter describes some of the work or unexplored areas within our problem domain that could help further explore this area.

7.1 Change of Vision

During the end of our semester GOMspace held an open house where they talk about their latest project and the company's history, in which we attended. Here some useful information was provided which changes the perspective of our current scenario. During their presentation they talked about the usage and problems they encountered when developing their GOMX-4A and GOMX-4B satellites. Each of the satellites have multiple modular slots, that are sold to other companies which create their module and can then be integrated into one of the satellites to gather information.

GOMspace also mentioned that some of the companies would like to take high quality photos of earth, such images could have a file size up to 4Gb. Sending such a file down to Earth takes about 30 days.
This deviate from our scenario about tasks having relatively small execution times and raises the question if there is a better way to represent tasks. The tasks could be changed such that they no longer have an execution time. Instead tasks should be viewed as being active and inactive, and a task would only become inactive by being preempted, finishing, or if its requirements are no longer upheld. The ramifications of this change is uncertain and will need to be tested further to prove if this new way of handling task actually is beneficial.

7.2 Disregard Delayed Tasks

As described in Section 3.5, it would be desirable to modify what information is written. Additionally it would be helpful to disregard tasks from the final schedule, which became delayed. This could be done using an extra script, scanning through the schedule deleting tasks which was not completed.

7.3 UPPAAL Stratego

CORA was introduced in Section 4.2 as a tool that finds the path with the lowest cost, thereby finding the optimal path. However, as stated in Section 4.6, the time it takes to find such a path can be problematic. This makes it necessary to introduce methods for reducing

the state space and thereby the time it takes to find such a path. This unfortunately may compromise the best schedule to be the best with the given restrictions.

This problem of balancing computation time and cost optimality is what brings attention to UPPAAL Stratego. Stratego is used to analyse a Stochastic Priced Timed Game (SPTG). It comes with an extended query language that introduces strategies that the user can formalise. The strategies are first class objects that may be compared, optimised and used when performing model checking. Strategies restrict which options are available at a given state in order to secure that some goal may be accomplished. Such a goal could be to always reach some location within before some time limit.

Stratego offers different kinds of strategies that can be used in either statistical model checking or symbolic model checking. The symbolic constructed strategies can be used to make reachability queries such as the one shown in Equation (7.1)

$$strategy\ InTime = control : A <> MyTemplate.LocationA\ \&\&\ total\_time\ <=\ 100 \tag{7.1}$$

$InTime$ ensures that the location `LocationA` can be reached before the clock $total\_time$ reaches 100 time units. The strategy can then be used in conjunction with other queries as seen in Equation (7.2)

$$E <>\ MyTemplate.LocationB\ \&\&\ MyValue\ <=\ 15\ under\ InTime \tag{7.2}$$

That query asks if there exists a path that leads to `LocationB` where $MyValue$ is equal to or less than 15, while still adhering to the restrictions of the strategy $InTime$.

The statistical strategies allows for optimisation towards some goal such as minimising a value. An example of this can be seen in Equation (7.3).

$$strategy\ MinValue\ =\ minE\ (MyValue)\ [<= 200]\ :\ MyTemplate.LocationA\ under\ InTime \tag{7.3}$$

$MinValue$ will be generated by repeatedly combing different strategies where the options they present for any given state are price-optimised. By repeatedly combing different strategies, the system learns how to more effectively reach a goal. The end result is a deterministic strategy that is near-optimal in reaching the goal. It is near-optimal as a simulation based method is used for learning the strategies. The goal of the strategy presented in Equation (7.3) is to minimise the $MyValue$ variable while still conforming to the strategy $InTime$.

The reasoning for using this tool to find schedules for convoys of satellites, is its ability to intelligently restrict what options are available for any given state, thereby theoretically reducing the computation time. The downside is that, unlike CORA, we are never guaranteed finding the optimal schedule. However, if the near-optimal schedules it may present are *good enough*, this may be an acceptable trade-off.

Glossary

CoB	Cost of Bandwidth. 55–65
CORA	UPPAAL Cost Optimal Reachability Analysis. iii, v, 12, 45–47, 52, 54–66, 70, 71, 73
DP	Dynamic Priority. 10, 12, 16
FIFO	First In, First Out. 10
FP	Fixed Priority. 10
LEO	Low Earth Orbit. 5–7
NASA	National Aeronautics and Space Administration. 1
RAAN	Right ascension of the ascending node. 9
RR	Round Robin. 10
SMC	UPPAAL Statistical Model Checking. iii, v, 12, 15, 42, 43, 70, 71
SPTG	Stochastic Priced Timed Game. 74
SRF	Shortest Remaining Time first. 10, 11
TLE	Two-line element set. 8, 9
TROPICS	Time-Resolved Observations of Precipitation structure and storm Intensity with Constellation of Smallsats. 1, 3, 4
VBP	Value Based Priority. iii, 45, 49, 50, 67–69, 71

List of Figures

List of Tables

Listings

Bibliography

[1] William J. Blackwell and Scott Braun. *Time-Resolved Observations of Precipitation structure and storm Intensity with a Constellation of Smallsats*. 2016. URL: https://tropics.ll.mit.edu/CMS/tropics/Mission-Overview (visited on Feb. 27, 2018).

[2] Anders Lykke Matthiassen, Jacob Nielsen, and Oliver Brun Køblsted. *Verification and Cost Optimal Nanosatellite Battery-Aware Schedule Production*. Jan. 2018. URL: http://projekter.aau.dk/projekter/files/267936373/Verification_and_Cost_Optimal_Nanosatellite_Battery_Aware_Schedule_Production___deis903e17.pdf (visited on Feb. 27, 2018).

[3] Lars Alminde. *Private Conversation*. Conversation via mail and one meeting at GomSpace. 17 2017.

[4] Elizabeth Mabrouk. *What are SmallSats and CubeSats?* 2017. URL: https://www.nasa.gov/content/what-are-smallsats-and-cubesats.

[5] Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krčál, Gilles Nies, and Marvin Stenger. "Battery-Aware Scheduling in Low Orbit: The GomX–3 Case". In: *FM 2016: Formal Methods*. Ed. by John Fitzgerald, Constance Heitmeyer, Stefania Gnesi, and Anna Philippou. Cham: Springer International Publishing, 2016, pp. 559–576. ISBN: 978-3-319-48989-6.

[6] Søren Nørgreen Gustafsson. *Open house pressentation*. GOMSpace held an open house convention where they pressented their GOMX-4 satellites and answered questions. Apr. 2018.

[7] Ian Poole. *Satellite Orbit Types & Definitions*. URL: http://www.radio-electronics.com/info/satellite/satellite-orbits/satellites-orbit-definitions.php.

[8] GISGeography. *Polar Orbit vs Sun Synchronous Orbit*. Feb. 2018. URL: https://gisgeography.com/polar-orbit-sun-synchronous-orbit.

[9] Tom Henderson. "Circular Motion and Satellite Motion - Lesson 4 - Planetary and Satellite Motion". In: (). URL: http://www.physicsclassroom.com/class/circles/Lesson-4/Mathematics-of-Satellite-Motion (visited on Feb. 21, 2018).

[10] *Satellite orbiting Earth*. URL: http://www.pngmart.com/image/49166.

[11] Jevon James. *How To Read GPS Coordinates*. URL: http://www.ubergizmo.com/how-to/read-gps-coordinates/ (visited on Feb. 27, 2018).

[12] Dr. T.S. Kelso. *Frequently Asked Questions: Two-Line Element Set Format*. Mar. 2014. URL: https://www.celestrak.com/columns/v04n03/ (visited on Feb. 22, 2018).

[13] Chris Veness. *Calculate distance, bearing and more between Latitude/Longitude points*. 2017. URL: https://www.movable-type.co.uk/scripts/latlong.html (visited on Feb. 27, 2018).

[14] Ashutosh Juneja. *Preemptive vs. Non-Preemptive Process Scheduling*. URL: https://study.com/academy/lesson/preemptive-vs-non-preemptive-process-scheduling.html (visited on Feb. 27, 2018).

[15] Google. *EXP - Docs editors Help - Google Support*. URL: https://support.google.com/docs/answer/3093411?hl=en.

[16] Free mathematics tutorials. *Three Points Parabola Calculator*. URL: http://www.analyzemath.com/parabola/three_points_para_calc.html.

[17] Gerd Behrmann. "UPPAAL CORA, UPPAAL for Planning and Scheduling". In: (). URL: http://people.cs.aau.dk/~adavid/cora/download.html.

Deterministic Model A

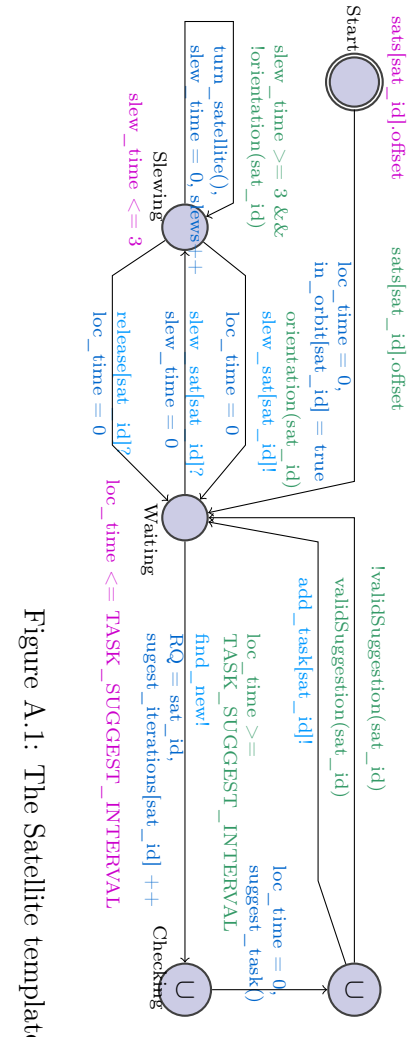

Figure A.1: The Satellite template

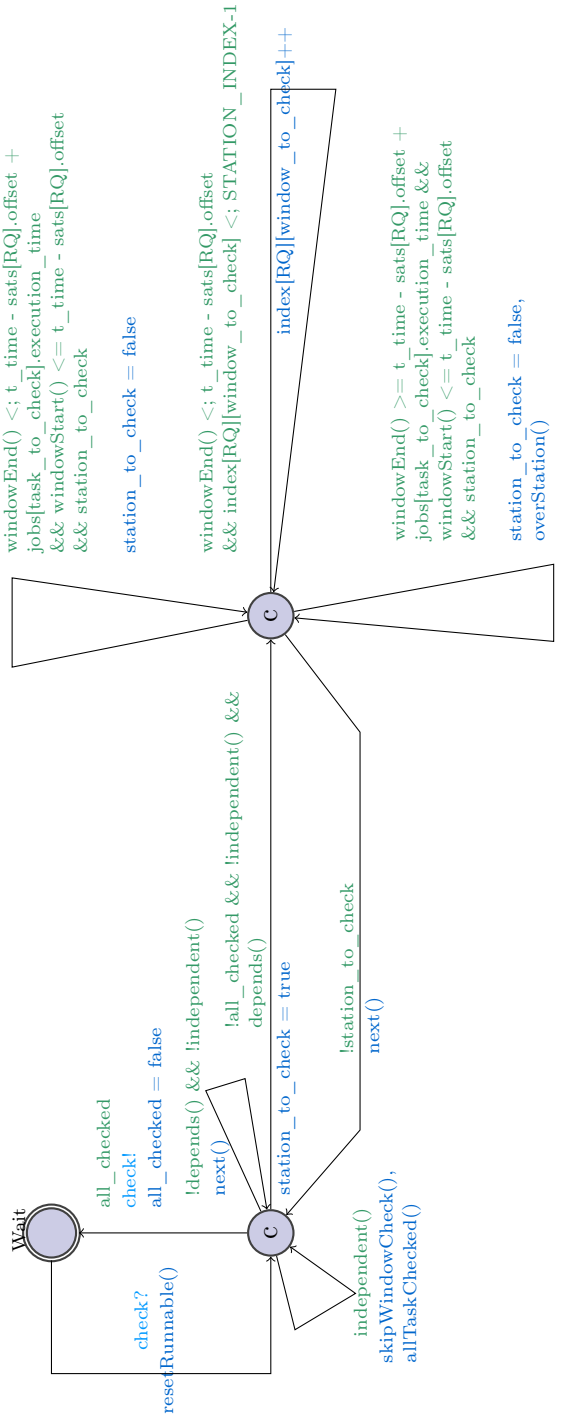

Figure A.2: The CheckRunnable template

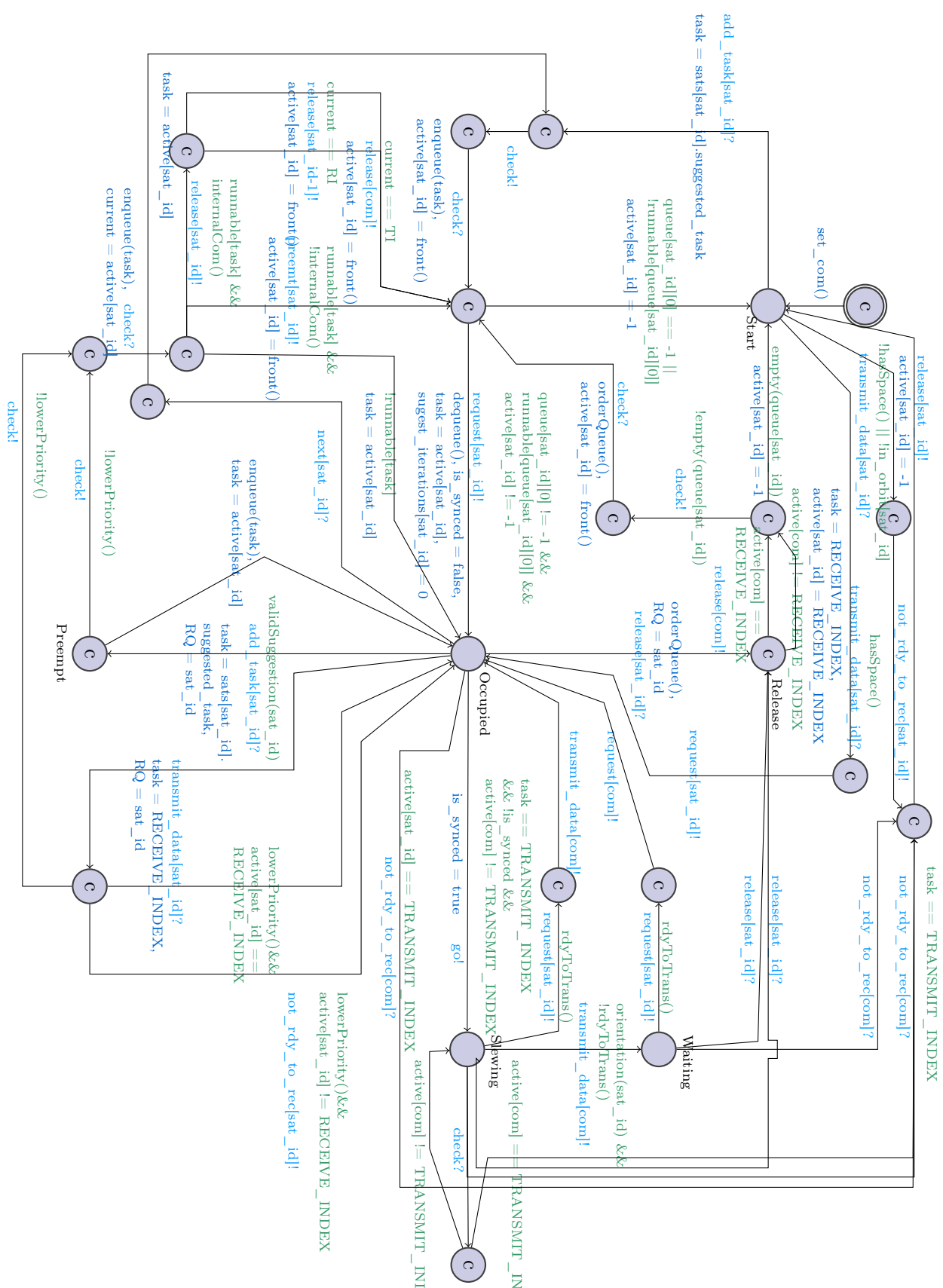

Figure A.3: The Scheduler template

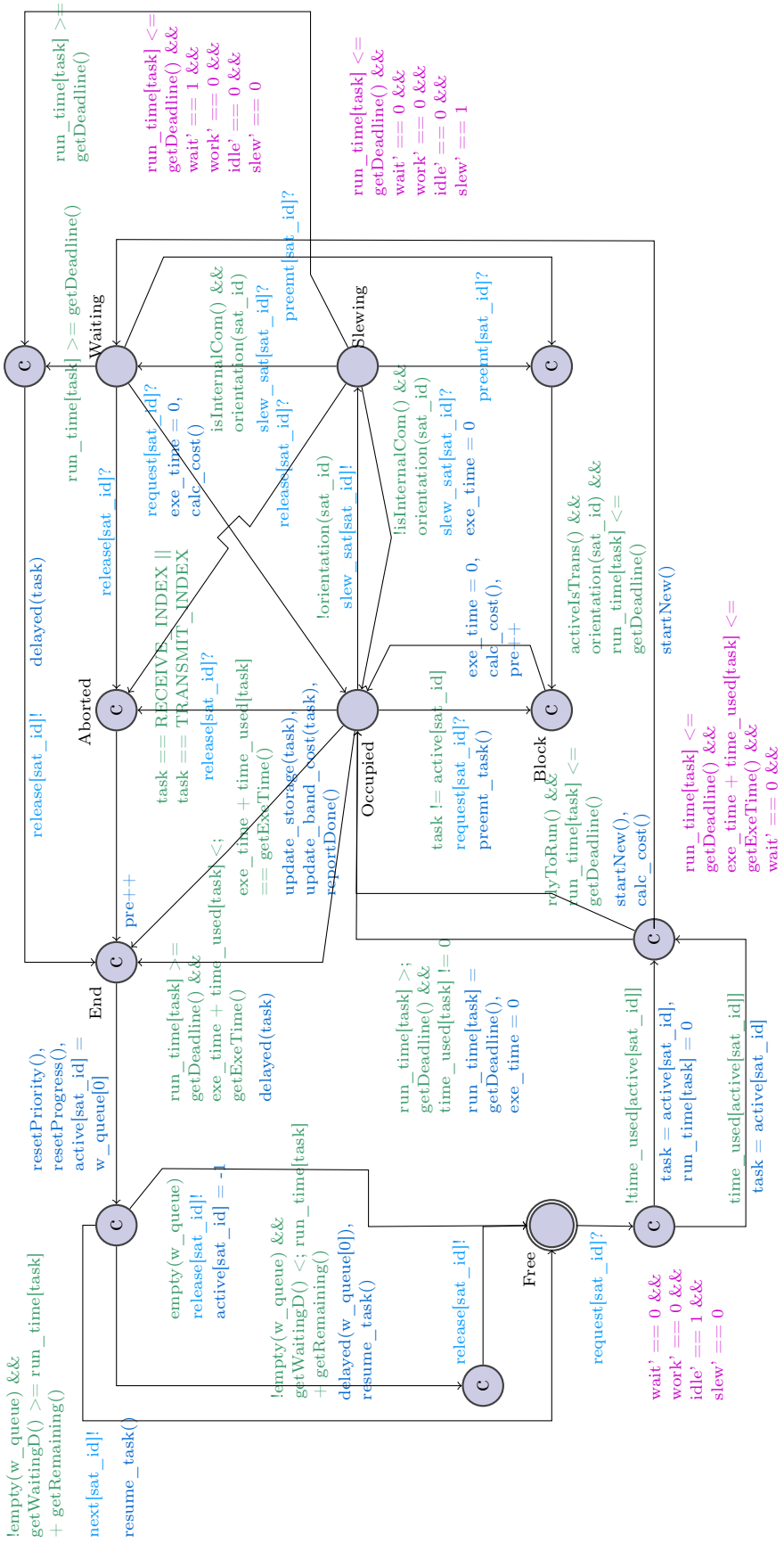

Figure A.4: The Processor template