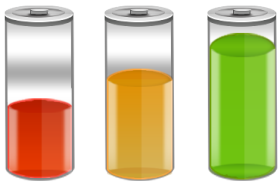
university of groningen

faculty of mathematics and natural sciences

CONTEXT AWARE POWER MANAGEMENT BASED ON USER BEHAVIOUR

BRIAN SETZ



a thesis on the topic of Computing Science

Faculty of Mathematics and Natural Sciences
University of Groningen

August 2015

Brian Setz: *Context aware power management based on user behaviour*, a thesis on the topic of Computing Science, © August 2015

SUPERVISORS:
Alexander Lazovik
Marco Wiering

LOCATION:
Groningen

ABSTRACT

Personal computers are responsible for a significant portion of the energy consumption of office buildings. There is little incentive for occupants of these types of buildings to save energy, as they are not responsible for paying the electricity bills. Enabling power management options such as sleep mode is a low effort method to reduce the energy consumption of computers. However, choosing the right timeout before a computer goes to sleep can be challenging. A sleep timeout which is too small leads to discomfort, whereas a sleep timeout which is too large results in poor energy saving efficiency. Furthermore, each user has his or her own preference when it comes to the sleep timeout. Letting the user choose their own sleep timeout is not an option, since research shows most users disable the sleep timeout completely.

In this thesis we attempt to find a solution which can determine the sleep timeout based on user behavior. Multiple models have been designed with the goal of maximizing the energy savings while minimizing discomfort. Each model has a slightly different approach to learning the optimal sleep timeout. A software solution was implemented to collect the data required to use and analyze the models. As part of this thesis we shall take a look at the architecture of this software solution and the supporting infrastructure of services which allows high performance while remaining scalable.

The models have been tested on the computers of employees of the University of Groningen during several weeks. We will analyze the results of the experiments and determine which model performs best according to some predefined criteria. We shall also take a look at the energy savings and economical savings which could be achieved if this solution were to be deployed on every computer in the Bernoulliborg.

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who have made it possible for me to complete this research. A special thanks to my supervisors, *Alexander Lazovik* and *Marco Wiering*, for their support and advice they have given during the course of this research project.

I would also like to acknowledge the important role of two PhD students of the University of Groningen, *Faris Nizamic* and *Tuan Anh Nguyen*, who have given me the opportunity to work together with them on the Sustainable Buildings project.

Furthermore, I would like to give special thanks to the *employees of the University of Groningen* who anonymously volunteered to be part of the experiments. Without them I would not have the dataset required to perform the work done in this thesis.

Finally I would like to thank my fellow student, *Ruurtjan Pul*, who has reviewed this master thesis and provided much appreciated feedback which helped tremendously.

CONTENTS

Part I

MASTER THESIS

1 INTRODUCTION

Sustainability is an important aspect in modern day society. Reducing the carbon footprint is an important aspect of working towards a more sustainable society. A report from the Intergovernmental Panel on Climate Change [32] indicates that the increase in CO_2 levels in the atmosphere is caused by human intervention with a probability of over 90 percent. Their report indicates that countries shall have to reduce CO_2 emission levels by 60 to 90 percent by 2050, otherwise temperatures will rise globally with more than 2 degrees Celsius by the end of 2050. Figure 1 depicts the CO_2 levels over the years and the alarming rate at which they increase.

> **Sus-tain-able, adjective** — *Able to be used without being completely used up or destroyed.*

Reducing the amount of energy that we consume on a daily basis is a necessity in order to achieve the aforementioned reduction in greenhouse gases and work towards a more sustainable society as a whole. There are many ways in which a reduction in energy consumption can be achieved, on many different scales.

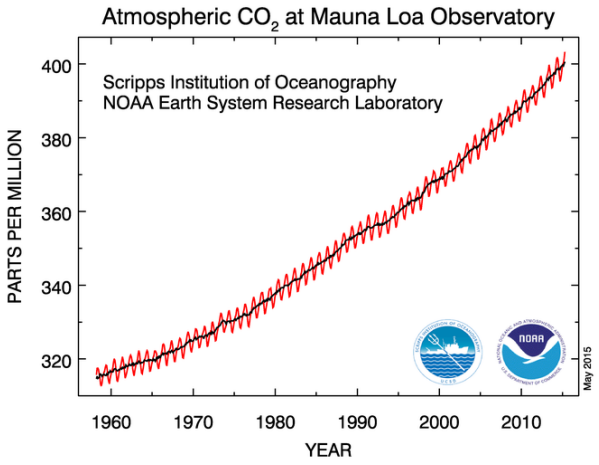


Figure 1: CO_2 level trends from March 1958 till May 2015. Taken from [32].

The IT sector is a large consumer of electricity around the world according to a report by the climate group on behalf of the Global eSustainability Initiative. They state that between 2 percent and 10 percent of the world wide energy consumption is attributed to the IT sector [1]. Personal computers in particular account for a large amount of the energy consumption within modern day buildings such as offices, universities and libraries. In most cases there is very little incentive to save energy for the occupants of these types of buildings, since they are not paying the bills of these buildings. Therefore it would be

preferable to save energy without any effort from the end user and without interfering with the end user's work flow.

In 2013 a project started with the goal of trying to determine what kind of savings would be achievable when it comes to computer usage within office buildings. This research was performed by Ruurtjan Pul and Brian Setz as part of their research internship [28]. The project was also part of the Green Mind award winning project of 2012: Sustainable Buildings [25]. The Green Mind award was an initiative of the Sustainability Steering Group of the University of Groningen. The Sustainable Buildings project is an initiative of Faris Nizamic and Tuan Anh Nguyen, both are PhD students at the University of Groningen. The Sustainable Buildings project strives to make buildings of the University of Groningen more durable.

A software application named Sleepy was developed as a result of the research done in 2013. The initial goal of Sleepy was to monitor the users of computers in order to research how much energy could potentially be saved and how much was wasted due to the computer not being in use while it was still running. The main functionalities of Sleepy were to monitor two different aspects of computers: their state and their activity level. In addition, Sleepy was also able to change the sleep timeout of computers remotely.

The state of a computer can be *on*, *off*, or *sleeping*. The activity of a computer is quantified as *active*, *idle* or *inactive*. When a computer is turned on the state of that computer has the value *on*. As soon as a computer enters sleep mode the state changes to *sleeping*. Finally, when a computer is turned off completely the state equals *off*. A computer is considered *active* when Sleepy detects activity on a computer, for example keyboard or mouse activity. When no activity is detected the activity value is *idle*. If a computer is turned off completely there can be no activity, thus the computer is considered *inactive*. Table 1 summarizes the possible values for state and activity.

State	*Activity*
On	Active
Sleeping	Idle
Off	Inactive

Table 1: Values for the state and activity of personal computers

Changes in the values of state and activity of each computer are stored in the database. Table 2 gives an abstract view of how this data is being represented inside the database. Both changes in state and activity are monitored by a sensor. For Sleepy these sensors are soft sensors, which means they are implemented as software. Each different type of sensor has a unique identifier which is called a *Sensor ID*. These Sensor ID's make it easy to determine the type of the sensor, for example whether it concerns a state or activity sensor. Each individual sensor of a certain type has a unique identifier, the *Instance ID*. Every entry logged by the sensors have a time-stamp on which the data was logged and a value representing the value of the sensor.

A more detailed description of the entire Sleepy solution, including the changes made during this research, can be found in chapter 4.

Sensor ID	*Instance ID*	*Time-stamp*	*Value*
Activity Sensor	Activity Sensor 1	12-05-14 9:48:56	Active
State Sensor	State Sensor 4	13-05-14 10:17:23	Sleeping
Activity Sensor	Activity Sensor 7	13-05-14 15:33:49	Idle

Table 2: Abstract view of how state and activity data is stored

Using the sleep mode found on computers is a way to reduce the overall energy consumption of computers. At the same time the end user does not have to put any effort into it in order to achieve these savings. The sleep mode will cause computers to enter sleep mode after a set amount of time in which the computer was not in use. A computer can be considered active when there is keyboard or mouse activity. But it also possible for applications to block the computer from entering sleep mode. For example, media players might block the computer from entering sleep mode while watching a movie.

Enforcing a policy with regards to sleep mode on computers is possible, as described by Aggar et. al in "Sustainable Computing Conserving Energy with Group Policy" [2]. In their article they describe setting up a group policy (fig. 2) which enforces power settings on Windows based computers. This kind of policy can enforce all computers to have the same sleep timeout at all time and disallow changes to the sleep timeout by the end user.

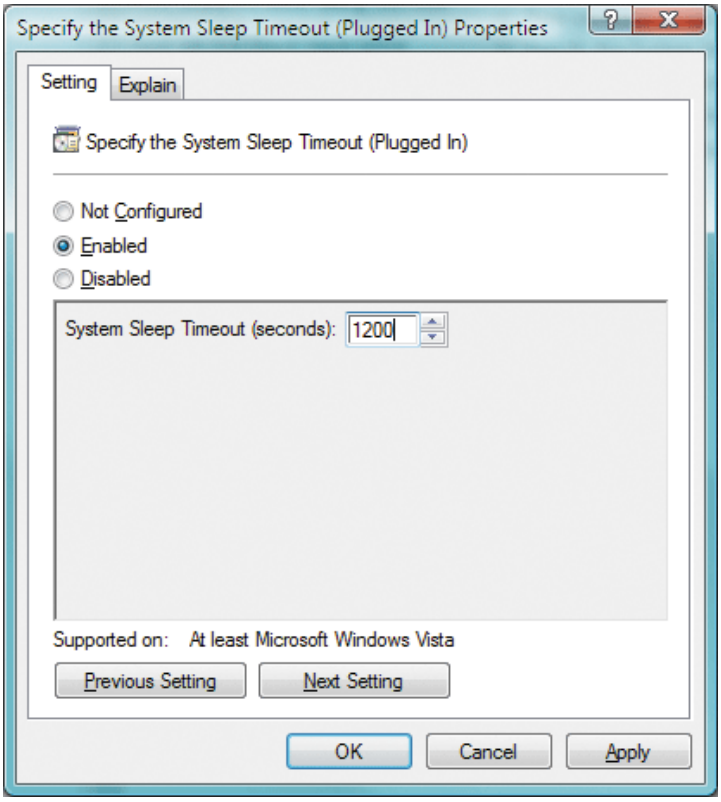


Figure 2: Enforcing a sleep timeout group policy

However, enforcing such a sleep policy comes with a major drawback: users cannot change their own sleep timeout since it is globally enforced. This could result in user discomfort as for some users the sleep timeout will be too low. But for other users the predefined sleep timeout could be too high. Low sleep timeouts may cause the computer to enter sleep mode too soon. For example, while the user is reading an article. This disrupts the work flow and causes irritation. High sleep timeouts prevent these situations from occurring but results in less energy being saved, reducing the effectiveness of enforcing said sleep policy.

A possible alternative could be to let the end user manage their own sleep timeout. The difficulty with allowing all users to change their sleep timeout is that they could set their own sleep timeout to zero, which in turn disables sleep mode on their computer completely. Or they could set a sleep timeout which is too high to be effective at saving energy. It is also possible that they simply forget to re-enable the sleep mode after disabling it for some time. As previously mentioned the occupants of office buildings usually do not have any incentive to save energy and therefore they cannot be expected to always use the optimal sleep timeout settings for a given scenario.

It appears there has to be a better way to enforce an effective policy where computers go to sleep, without introducing the user discomfort that comes with such a policy. The state and activity data collected and stored by Sleepy could potentially be used in some form as part of a learning process to optimize the way in which a computer is put to sleep. And at the same time Sleepy will allow us to measure the performance and results of such a process.

The goal of the research performed in this thesis is to decrease the energy consumption of personal computers by using the sleep mode of these personal computers in an intelligent manner. This process to determine a personalized sleep timeout is supported by the data which Sleepy collects from computers within the University of Groningen. The collected data shall be used in some form as part of a learning process which will try to find the optimal sleep timeout for individual users. At the same time it should find a sleep timeout which shall not interfere with the work flow of the person using the computer. Ultimately it will be the question whether this approach indeed results in energy savings without a negative experience for users.

We expect that when using a more optimal method to set the sleep timeout we can determine a better personal sleep timeout without breaking the user's work flow. Two research questions have been formulated based on the aforementioned problems and goals. The first research question of this thesis is as follows:

> *How can an optimal sleep timeout setting be found for individual users from data available in the dataset of state, activity and feedback information of personal computers?*

When such a method has been researched it will be interesting to see if the energy consumption of personal computers is indeed reduced without interfering with the end user. Interfering with the end user means that the sleep timeout was wrong and the computer went to sleep too early. This leads to the second research question of this thesis:

> *Does the solution used for sleep timeout optimization result in a reduction of the energy consumption of personal computers, while at the same time keeping the user comfort at high levels?*

The chapters of this master thesis are structured as follows: chapter 2 will explore work related to this master thesis. In chapter 3 the problem statement is given and formalized. Chapter 4 describes the architecture of Sleepy in more detail. The results and evaluation of the experiments can be found in chapter 5. The future works are discussed in chapter 6. Finally, the thesis is concluded in chapter 7.

2

RELATED WORK

Saving energy by putting personal computers to sleep appears to be an obvious solution for decreasing energy consumption in offices, universities, libraries and other locations that have a high concentration of computers in one place. A paper published in the journal Energy shows that up to 64 percent of all computers in offices are still powered on during after-hours [33]. Only 6 percent of all computers made use of the available power management options, such as sleep mode. Of the monitors attached to the computers around 75 percent were powered on during after hours. However, 66 percent of all monitors did use some form of power management. Based on this research it can be concluded that there are significant savings still to be made by putting computers to sleep.

A report from the Intergovernmental Panel on Climate Change [18] suggests that computers are left powered on and unattended about 28 percent of the time. The authors also show that 49 percent of computer users never or rarely turn off their computer. According to the authors the main reasons why users do not turn off their computers over night are as follows:

- It's IT policy or procedure to leave it on. (47 percent)
- My computer goes into hibernation, or sleep mode. (31 percent)
- It takes too long. (20 percent)
- It's a habit. (11 percent)
- I don't think it's important. (10 percent)
- It's a hassle. (8 percent)
- I forget. (8 percent)

Furthermore, findings in [9] also confirm that computers are often left powered on. According to this research it concerns Windows machines in particular. Their findings show that Windows laptops are often turned off, but the more power hungry desktop computers are not. Up to 50 percent of these desktop computers are left running during off peak hours. Of these computers over 75 percent could be turned off in order to achieve energy savings. The authors of the paper argue that users leave their computers turned on for the following reasons:

- There is no economic incentive as the user does not pay the bill.
- Starting computers can take a long time.
- Not wanting to lose the state of the computer upon shutdown.

- Software updates and other maintenance tasks are done at night.
- Some users want to access their data while at home.

According to [26] the energy consumption of desktop computers are around 80 watts when idling. When also including a 17-inch display monitor the energy consumption increases to around 115 watts.

These numbers show that there is a lot of potential for savings to be made by applying power management methods in a smart manner. Especially if this could be done in a non intrusive way since many users already assume their computer has sleep mode enabled. Even though in most situations sleep mode has been disabled, as is indeed the case at the University of Groningen.

2.1 SLEEP PROXY

An issue that arises when putting computers to sleep is that they are no longer available over the network. This is an issue because users want to be able to access their files remotely, and IT administrators want to be able to manage computers at any given time. In [10] a solution to this problem is proposed by introducing Network Connection Proxies, also known as Sleep Proxies. Network Connection Proxies allow computers to have a network presence while they are asleep. This is achieved by letting the Network Connection Proxy handle network traffic destined for a computer that is asleep, and wake the computer if necessary. As a result users can still remotely access their machines and IT administrators are still able to manage the computers. Thus there is no need anymore to keep computers powered on for these tasks.

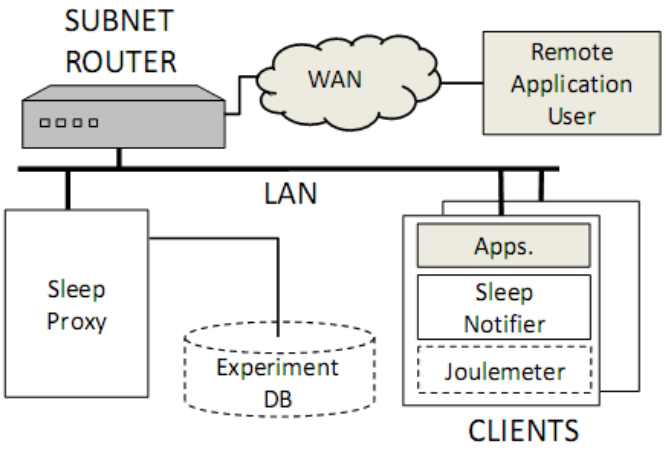


Figure 3: Sleep Proxy architecture

An actual implementation (fig. 3) of this concept is described in [29] by the Microsoft Research department. This implementation provides some solutions to issues that may arise when implementing a Network Connection Proxy. These issues include ARP probe timing, DHCP lease expiration and general failure of the sleepy proxy itself. The downside of using sleep proxies is that they are very sensitive to

changes in the network environment due to their low level implementation. More research and information on the topic of proxy servers can be found in [27] [19] [8].

The method used to detect when a computer enters sleep mode is the same method as used by Sleepy. Both solutions use Windows Power Management Events to detect changes in the power state of the computer. However, sleep proxies do not adapt to the users. As a result the behavior of the system is the same for all users, no matter their personal preferences. This is an aspect which the solution presented in this thesis attempts to improve on.

2.2 ACTIVITY DETECTION & MONITORING

To be able to determine when a computer could be put to sleep one needs to know whether it is in use. In [24] a method is described to determine the user activity based on multiple parameters. The parameters which are used are as follows: keyboard and mouse events, CPU usage and network activity. Additionally, gaze detection is used in order to determine whether a user is distracted. The gaze detection is achieved by use of a web cam which monitors eye gaze. The research also takes into account the level of user comfort, if a computer goes to sleep too soon it will cause discomfort to the user.

There are many other techniques to determine whether the user is indeed sitting behind his computer. For example, in [15] a power management system is described which uses Bluetooth enabled mobile devices to determine the location of the user. The authors were able to get within 8 percent of optimal power consumption. However, the range of Bluetooth is limited to around 10 meters, reducing the effectiveness in large, open offices.

In [35] the authors describe a WLAN based location determination system. This system makes use of the available information regarding the signal strength of received frames, and uses this to infer the location of the device from which these frames were received. WLAN determination systems usually work in two stages, offline training and online location mapping. Offline training is needed to map signal strength to locations in order to build a radio map. Once this radio map has been completed the system can map devices to locations on the radio map with some probability.

Another different approach to activity detection is described in [14]. In this paper the authors make use of an ultra-sonic location system which provides an accuracy of up to 3 cm in most cases. The system allows for high accuracy tracking of building occupants and room usage. However, this system was already built into the building in which the researchers performed their experiments. Outfitting an existing building with a similar system would come with high costs.

The software developed as part of the research in this thesis uses both keyboard and mouse events along side other events from the operating system to determine the level of activity. CPU usage and network activity would be good additions to assist with activity de-

tection. Gaze detection using a web cam is however not an option, since the software should work without requiring extra hardware. It could however become an optional addition.

2.3 USAGE PROFILING & TIMEOUT OPTIMIZATION

Usage profiling of computers can be used in addition to real time parameters to predict future user activity. In [7] a daily behavior-based usage pattern algorithm is outlined. By applying hierarchical clustering the similar usage patterns are grouped together, resulting in a set of usage profiles for a specific device. The research done in [6] explains a generic method of usage profiling using naive Bayes classifiers.

Dynamic sleep timeout optimization is needed in order to minimize the user impact of computers entering sleep mode. In [12] the sleep timeouts of printers are dynamically changed based on the time between print requests. The approach chosen in this research is to apply Hidden Markov Models to create a statistical framework for timeout optimization. More details regarding how to model power management using Markov Models can be found in [11] and in [3].

2.4 CONTEXT AWARE POWER MANAGEMENT

There are some software based solutions which share part of the same functionalities of Sleepy. One such software solution is PoliSave as described in "PoliSave: Efficient Power Management of Campus PCs" [9]. PoliSave works by allowing the user to enter a schedule for their computer: the user can specify at which time and day the computer should perform some power management action such as sleep, turn on or turn off. This allows for a great deal of personalization of the power management on user-basis. The authors indicate that usually 56 percent of the computer they monitored were turned on all the time. After applying their techniques this percentage was reduced to 6 percent, saving on average around 6 hours of on-time per work day. The average daily uptime of computers was reduced by more than 6 hours per working day, from 15.9 hours to 9.7 hours. As a result savings of over 219 kilowatts per year were achieved in their specific environment.

The downside of the approach taken by PoliSave is that the users need to manually manage their schedule for the computers and need to take into account things such as holidays if they want to be efficient. Compare this to Sleepy which automatically learns the user's schedule and does not have to deal with exceptions and it is clear that the PoliSave approach requires more commitment from the user.

A similar yet different approach is described in "E-Net-Manager: A Power Management System for Networked PCs based on Soft Sensors" [5]. The similarity between E-Net-Manager and PoliSave is the fact that they both use a schedule based approach to power management in which the end user manually needs to enter the schedule.

The difference is that E-Net-Manager employs other software-based sensors which determine whether a user is present or not. These soft-sensors include BlueTooth sensors which take advantage of how the BlueTooth handshake mechanism works. They also take advantage of other wireless techniques such as using the radio connectivity of smart-phones to determine the presence of specific users. The computer is turned on and off based on this presence. The manual calender based schedule is there for back-up in this case.

Another related software solution is described in "Gicomp and GreenOffice - Monitoring and Management Platforms for IT and Home Appliances" [16]. Gicomp realizes a centralized power management platform. The software is able to manage the power management policy of large numbers of personal computers simultaneously. The software, however, does not allow personalization of power management policies.

There are similarities but also some significant differences between these software solutions and the software solution proposed in this thesis. Sleepy does not take over control of the operating system, instead it lets the operating system decide whether the computer is in use or if it is not. The advantage of this approach is that computers are only put into sleep mode when they are not actually used, for example: when watching a movie the operating system will block the system from entering sleep mode even though there is no keyboard or mouse activity.

3 PROBLEM STATEMENT

Computers are responsible for a large portion of the total energy consumption within office buildings. The sleep mode found on personal computers is a convenient way to save energy without much effort. However, enforcing the same sleep policy on all computers for all users leads to a decrease in user comfort. Individual users have their own preferences when it comes to the sleep timeout The sleep timeout is the time before a computer enters sleep mode after being idle. It would be beneficial to have a way to determine the optimal sleep timeout setting for each user.

Because Sleepy is monitoring user behaviour on computers and storing this information there is an interesting dataset available. This dataset contains information about the state and activity of computers. It also includes information about negative feedback which can be either direct or indirect. Direct negative feedback is reported by the user using a dialog option in Sleepy. Indirect feedback comes from users waking up the computer shortly after it went to sleep. The data collected by Sleepy has the potential to be used to determine an optimal sleep timeout per user.

The problem of finding the personalized sleep timeout can be generalized as an optimization problem for sensors which have some kind of timeout that is influenced by user behaviour. An optimal timeout is a timeout that balances the benefits and the costs that come with a certain timeout setting. In this case the timeout is the sleep timeout of personal computers and one of the challenges is to find the balance between energy savings and user comfort.

Three different models have been defined in order to find the balance between the benefits and costs constraints. However, before we can take a closer look at the models we have to understand the data set on which these models are based.

3.1 DATA SET

The data set used in this research has been obtained by using Sleepy to monitor computers for an extended period of time. The computers in question are the computers used by employees at the University of Groningen who have volunteered to have Sleepy monitor their behaviour and collect the data. At the time of writing the data of approximately twenty computers is actively being collected. The time period during which this data has been collected is around two to three months, depending on the individual computer. The minimum timespan which is needed for learning the user behaviour is one month because we need to estimate the activity probability based on 1 month of historical data. For each computer there are four different types of data collected:

1. State Data
2. Activity Data
3. Feedback Data
4. Timeout Data

3.1.1 *State Data*

State data gives insight into the state of the computer. The state refers to the power management state of the computer and can assume three values: *On*, *Sleeping* and *Off*.

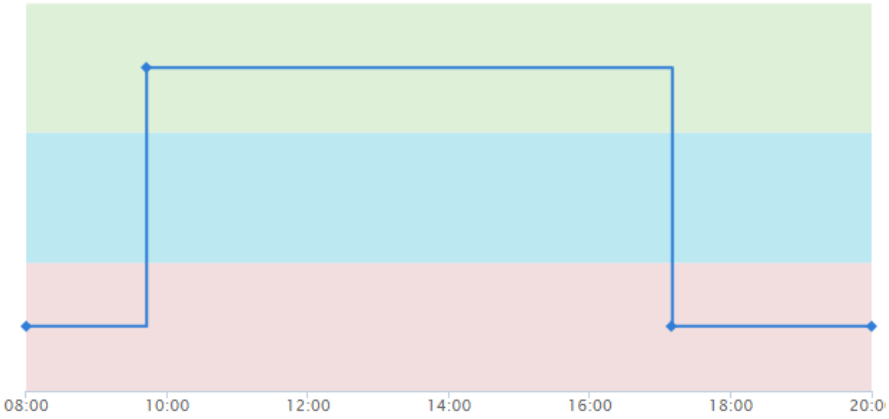


Figure 4: State data

An example of state data can be seen in figure 4. The computer changes its state from 'Off' to 'On' at around 10:00. It stays in this state for the rest of the morning and most of the afternoon, until it changes back to "Off" at around 17:00.

The way Sleepy collects this information about the state of a computer is by listening to System Power Management events [21]. Because part of Sleepy is a Windows Service it can register itself to listen to Window's *PowerBroadcastStatus* events. This allows Sleepy to detect when the computer enters sleep mode (*Suspend* event), and when it wakes up from sleep mode (*ResumeSuspend* event).

3.1.2 *Activity Data*

Activity data is collected in order to detect if a computer is actually being used while it is turned on. The activity value can be either *Active*, *Idle* or *Inactive*. Active means the computers is actively being used whereas idle means the computer is turned on but not in use. The value inactive occurs when the computer is turned off.

Figure 5 shows an example of the activity of a computer (which is the same computer as in figure 4). At around 10:00 the activity changes from "Inactive" to "Idle", then quickly after it changes to "Active". This can be explained as the user turning their computer on, waiting a minute before logging in. At 10:30 the activity changes to

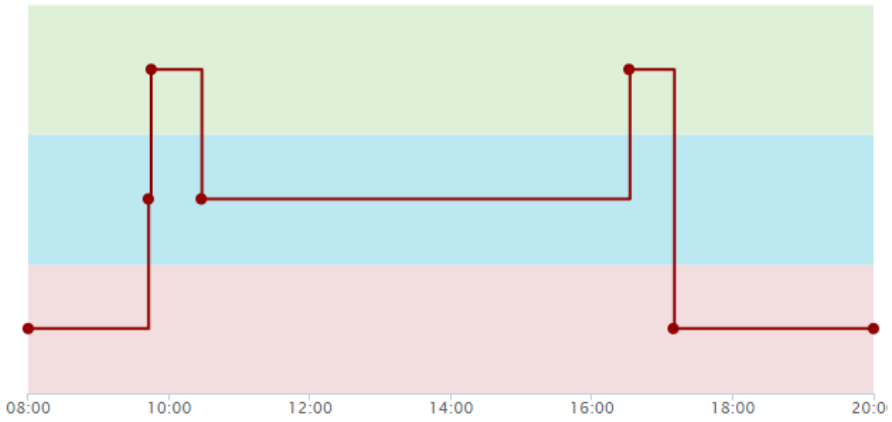


Figure 5: Activity data

"Idle" and it remains idle until 16:30 when it changes back to "Active". Half an hour later it, at around 17:00 the activity changes to "Inactive", because the computer has been turned off. There is a period of six hours, from 10:30 till 16:30, in which the computer is turned on but not in use. In this case an appropriate sleep timeout could have saved hours of energy being wasted.

Sleepy monitors the activity of the computer by reading the LASTINPUTINFO structure [20]. The structure contains the amount of ticks since there was any type of user input (e.g. mouse or keyboard input). Using the *TickCount* (the tick count per second) environment variable one can calculate the amount of seconds since there was input. When Sleepy detects there has been input it sets the active flag to true. If there hasn't been any activity for **x** seconds, then the flag is set to false. The variable **x** is called the *idle threshold*. Lower idle thresholds will cause a lot of changes between "Idle" and "Active" whereas higher idle thresholds decrease the accuracy of the data by showing "Active" for too long.

3.1.3 *Feedback Data*

Feedback is collected in order to determine whether Sleepy is working with a minimal user impact. The type of feedback that is collected is "negative feedback". As a result the user does not have to provide any feedback when everything is working as expected but only when things are not working as they should.

Time	Reason
Jan 8th 2015, 09:40:32	Enabled manually
Jan 7th 2015, 12:34:12	The computer has entered sleep mode too soon
Jan 6th 2015, 16:31:13	Enabled automatically

Figure 6: Feedback data

An instance of collected feedback can be seen in figure 6. Sleep mode was automatically enabled on the 6th of January at 16:31. The next day on the 7th of January at 12:34 the user decided to disable sleep mode because his or her computer entered sleep mode too soon. This is an example of the negative feedback that Sleepy collects. Sleep mode was automatically enabled again the following day.

Users can report their feedback using an icon presented by Sleepy in the system tray. Right clicking on this tray icon provides the user with the menu option to disable Sleepy until reboot. When this menu option is selected the user will be shown the dialog in figure 7.

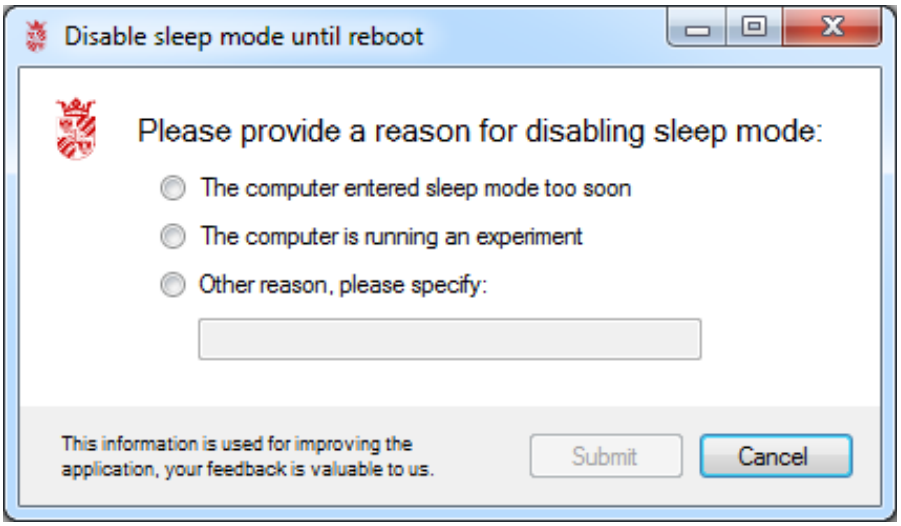


Figure 7: Negative feedback from disabling Sleepy

3.1.4 *Timeout Data*

The last piece of data that is collected is the timeout data. The timeout data encompasses the history of sleep timeout values of a computer. The sleep timeout is always represented in minutes, a sleep timeout of zero minutes means that sleep is disabled.

Time	Sleep Timeout	Reason
Jan 29th 2015, 09:13:49	Sleep disabled	Changed automatically
Jan 30th 2015, 10:24:41	30 minutes	Changed automatically

Figure 8: Timeout data

An example of sleep timeout data belonging to some computer can be seen in figure 8. As can be seen, on the 29th of January at 09:13 the sleep timeout was set to zero (disabled). The following day at 10:24 the sleep timeout was changed to 30 minutes. In both cases it was done automatically, this means Sleepy initiated the change and not the user.

3.1.5 *Extrapolated Data*

Additional data can be extrapolated based on the four previously mentioned data sets. This extrapolated data will play an important role in the models described in the following section. These additional data sets are as follows:

- Activity Probability
- Idle Time
- Negative Feedback

The activity probability is based on the activity data set. It represents the probability that a certain computer is actively in use at a given moment in time. This probability is calculated by looking at the historical activity data. To be exact, it looks at the data of the same day over the past few weeks. For example to calculate the activity probability at 14:00 on Wednesday one needs to look at the activity data of previous Wednesdays around 14:00. Based on this historical data one can determine the probability of seeing activity at a given moment in time.

Idle time can also be extracted from the activity data set. It is trivial to determine the idle time of a computer: by looking at the most recent value in the activity data set the current state of the computer can be determined. If this activity state is idle then the idle time is simply the amount of time since the activity state changed to idle. Using the idle time one can determine how long it has been since the computer has been actively used.

There already is a data set which covers part of the negative feedback which can be received. However this data set only receives negative feedback from users using the feedback dialog which is part of Sleepy. There is more negative feedback to be detected: when a computer enters sleep mode and is woken up before a certain time has passed it can also be considered negative feedback. This could mean that the sleep timeout was too short and that the personal computer entered sleep mode too soon, while the user was in fact still using the computer.

The information about the length of a sleep period can be extracted from the state data set. If the interval between sleep state and on state is too short then it shall also be considered as negative feedback. However, it is not as severe as feedback received through the feedback dialog. When a user has to use the negative feedback dialog to disable Sleepy there must have been a major impact on the user's work flow.

3.2 MODELS

Now that we understand the dataset we can take a closer look at the models which shall be used in the experiments. The models share a common goal: to minimize the amount of time in which a computer is in the idle state. Reducing this idle period will be done by putting

the computer to sleep. A reduction in idle time by putting computers to sleep leads to energy savings. However, it is important that the negative feedback remains minimal.

In order to determine the performance of each model they will be graded according to some error function. The error for choosing a certain sleep timeout can only be determined after a certain amount of time has passed. The error function for determining the performance of the model can be defined as

$$A_{idle}(t) = \{z \in A(t) \,|\, z = idle\} \tag{1}$$

$$E(t) = \sum_{z \in A_{idle}(t)} z \tag{2}$$

Where $A(t)$ is the set of time intervals for every activity change and $A_{idle}(t)$ is the set of time intervals only containing the intervals during which the activity was idle. $E(t)$ is the sum of all intervals during which the activity was reported as idle. This is the most basic form of error function that could be used to grade these models. However the error function should also include the negative feedback. Including negative feedback will result in the following error function:

$$E(t) = \theta_2 \sum_{z \in A_{idle}(t)} z + \theta_1 N(t) \tag{3}$$

Where $N(t)$ represents the negative feedback collected for timespan 't'. It can be argued that an increase in negative feedback is more costly than an increase in idle time. The exact ratio used for the evaluation of the models will be determined in chapter 5, Experiments and Evaluation.

A generic framework has been conceived based on three different types of data: activity probability, negative feedback and idle time. Three concrete models have been developed based on this generic framework. Each model becomes increasingly more complex than the previous model. The models return a personalized sleep timeout based on the given input parameters. The first model is a rule based model whereas the renaming two models are linear regression models. The weights for these linear regressions models (denoted as w_x) are obtained using linear regression. The implementation for the linear regression algorithm is provided by the Spark MLib library.

3.2.1 *Model 1 - Activity Probability*

The goal of this model is to always have a high sleep timeout whenever there is a chance of user activity at that given moment in time.

The sleep timeout becomes low when there is no probability of user activity.

$$s(t) = \begin{cases} s_{max}, & \text{if } P_a(t) > \varepsilon. \\ s_{min}, & \text{otherwise.} \end{cases} \tag{4}$$

Input:

- $P_a(t)$ – Activity probability at time t
- ε – Activity threshold
- s_{min} – Minimum sleep timeout value
- s_{max} – Maximum sleep timeout value

Output:

- $s(t)$ – Sleep timeout for time t in minutes

For the minimum sleep timeout the value will be around five minutes. The maximum sleep timeout will be around sixty minutes. The activity threshold will be set to some small value of ε. For the experiments a value of 0.05, or 5 percent was chosen.

3.2.2 *Model 2 - Activity Probability & Negative Feedback*

The goal of this model is to find the most optimal sleep timeout based on two different input parameters: activity probability and negative feedback.

$$s(t) = w_2 P_a(t) + w_1 N(t) + w_0 \tag{5}$$

Input:

- $P_a(t)$ – Activity probability at time t
- $N(t)$ – Negative feedback count at time t

Output:

- $s(t)$ – Sleep timeout for time t in minutes

A higher activity probability should lead to an increase of the sleep timeout, if the chance of a computer being in use is high it should not enter sleep mode. Furthermore, an increase in negative feedback should also lead to an increase of the sleep timeout. The difference is that an increase in negative feedback should increase the sleep timeout more than an increase in activity probability. The negative feedback at time t also includes negative feedback received before time t.

3.2.3 *Model 3 - Activity Probability, Negative Feedback & Idle Time*

The goal of this model is to find the most optimal sleep timeout based on three different input parameters: activity probability, negative feedback and idle time.

$$s(t) = w_3 P_a(t) + w_2 N(t) + w_1 I(t) + w_0 \tag{6}$$

Input:

- $P_a(t)$ – Activity probability at time t
- $N(t)$ – Negative feedback count at time t
- $I(t)$ – Idle time in minutes at time t

Output:

- $s(t)$ – Sleep timeout in minutes for time t

The first two parameters of this model work according to the same principles as described for model 2. The third parameter, idle time, also increases the sleep timeout as its own value increases. The rate of increase should be somewhere in between those of the first two parameters.

3.2.4 *Model Learning*

To be able to use the models previously described one needs to know the weights which should be used for each of the model parameters. Finding these weights can be defined as an optimization problem:

$$\min_{w \in \mathbb{R}^d} f(w) \tag{7}$$

The vector w is the weight vector which is of size d. The goal is to find the weights that minimize some objective function. This objective function $f(w)$ is defined as follows:

$$f(w) = \lambda R(w) + \frac{1}{n} \sum_{i=1}^{n} L(w; x_i, y_i) \tag{8}$$

Where $\lambda R(w)$ is the regularization parameter which controls the complexity of the model. The loss function is expressed as $L(w; x_i, y_i)$. Linear least squares with stochastic gradient descent is used to learn the weights for the second and third model. The loss function is given by the squared loss:

$$L(w; x, y) = \frac{1}{2}(w^T x - y)^2 \tag{9}$$

The training error for this model is the mean squared error which is defined as:

$$\frac{1}{n}\sum_{i=1}^{n}(s(t^{(i)}) - y^{(i)})^2 \tag{10}$$

Where $y^{(i)}$ is a data point from the training set and $s(t^{(i)})$ is the estimated value for the data point using the learned model. The training dataset for the model is based on expert knowledge of which timeout should be associated to the given input parameters. The data is split into training data and test data. Sixty percent of the data is used for training and the remaining fourty percent is used to test the trained model.

4 ARCHITECTURE AND DESIGN

In this chapter we take a closer look at the software solution which is responsible for collecting the data used by the models to determine the personalized sleep timeout. We also take a look at the supporting architecture behind the software solution.

Sleepy is a sleep management solution: it manages the sleep timeout of personal computers and monitors the computer usage. The sleep timeout determines when the computer enters sleep mode after being idle for a specific amount of time. For example, if the sleep timeout of a computer is five minutes, then the computer will enter sleep mode after being idle for five minutes. Idle usually means that there is no keyboard or mouse activity, however, some software applications are also able to prevent a computer from entering sleep mode by simulating activity. Sleepy is able to remotely changes the sleep timeout, which means as a result it is able to influence the energy consumption of computers.

4.1 BACKEND ARCHITECTURE

The Sleepy solution is part of a project named Sustainable Buildings. The goal of the Sustainable Buildings project is to make buildings more sustainable in the general sense, amongst other things it encompasses energy consumption reduction, reduction of water usage and optimisation of recycling within buildings. Software components were developed as one of the ways to achieve this reduction in energy consumption. To support these software solutions and to simplify software development by reusing as many components as possible a backend architecture was designed and implemented.

Sleepy is not the only solution collecting and storing sensor data; there are also light level sensors, motion sensors, temperature sensors, energy consumption sensors and many more. In order to accommodate any number and any type of sensor a generic way to store sensor data within the Sustainable Buildings project has been developed: the Sensor Data Storage.

4.1.1 *Sensor Data Storage*

Cassandra is the backbone of the Sensor Data Storage, it is optimized for the storage of time-based data and it is an easily scalable platform. Data is stored in column families, which to a degree are comparable to tables. The column family for the Sensor Data Storage is designed as shown in table 3.

The *sensor_id* column contains the universally unique identifier (UUID) that identifies the type of a sensor. When it comes to Sleepy there are

Columm Name	*Column Type*
sensor_id	uuid
instance_id	uuid
logged_at	timestamp
rollover_id	int
value	ascii
process_id	uuid

Table 3: Sensor Data column family in Cassandra

multiple different sensor types, an activity sensor, a state sensor, a negative feedback sensor and a sleep timeout sensor. All these types of sensors have a different UUID. However, if there are two different instances of the same sensor they have a different *instance_id* in order to uniquely identify an individual sensor. For example, when there are two computers running Sleepy they both have an activity sensor which both have the same *sensor_id*, yet the *instance_id* differs between the two sensors. The next column, *logged_at*, stores the time stamp on which the sensor data was logged, this is a UNIX time stamp. The *rollover_id* is used to easily perform a roll-over of the data stored in the column family. The actual data that a sensor needs to store is inserted in the *value* column. The *process_id* is a UUID that can be used to track streams of sensor data. For example if some sensor generates data and this data is parsed and manipulated by another sensor: in this case both these entries will have the same *process_id* in the database. This makes it easy to backtrack data steams by looking for a specific *process_id*.

4.1.2 *Sensor Data Collector & RabbitMQ*

Individual sensors do not have to directly connect to the Sensor Data Storage: there is a mechanism in place that allows for an easy method to store data in the database. The Sensor Data Collector is an application that acts as a mediator between the sensors and the database. Instead of sending the sensor data to the database the data has to be sent to the Sensor Data Collector. The way the sensor data is sent to the Sensor Data Collector is by use of a message queue on which the sensors put their sensor data. The message queue platform that is being used is RabbitMQ (fig. 9). The platform scales well and provides sufficient performance to keep up with all the different streams of sensor data.

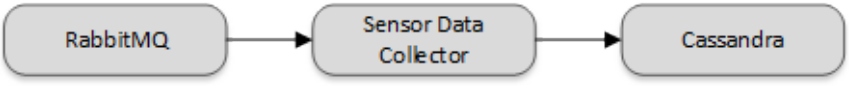


Figure 9: Sensor data flow of the Sensor Data Collector

When sensors generate data they connect to RabbitMQ as producers and publish their data to a so-called exchange. This exchange is a RabbitMQ mechanic that distributes data to specific queues based on a routing key. Each queue listens to a specific routing key and this determines which messages from the exchange get sent to the queue. The basic flow of data within RabbitMQ can be seen in figure 10.

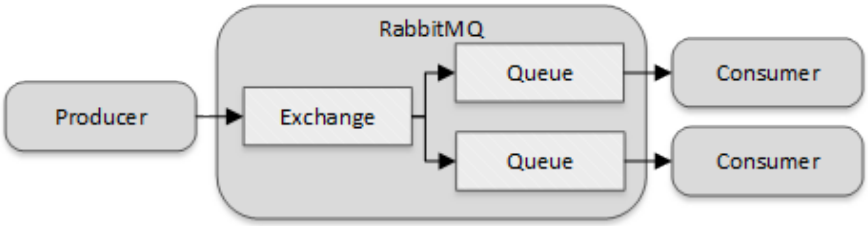


Figure 10: RabbitMQ's internal workings

Everytime a sensor publishes information to the exchange it has to specify a routing key to go along with the data. The routing key format that is used for sensor data is shown in listing 1. The *building*, *floor*, *room* and *area* parameters of the routing key are used to identify the location of the sensor. The *sensorid* and *instanceid* parameters are set to the UUID's of the corresponding sensor.

Listing 1: RabbitMQ routing key format

```
sensordata.<building>.<floor>.<room>.<area>.<sensorid>.
    <instanceid>
```

The Sensor Data Collector creates a queue in RabbitMQ that is bound to the aforementioned exchange and listens to the routing key "*sensordata*.#".This indicates that the exchange will forward all messages that start with the routing key "*sensordata*" and inserts them into the queue that belongs to the Sensor Data Collector. The advantage of this approach is that there can exist other queues which bind to the same exchange, but instead listen to a more specific routing key. For example, a queue that binds a routing key which specifies the sensor ID such that only messages from that specific type of sensor will be put in this queue, essentially filtering out all other sensors.

In order for the Sensor Data Collector to be able to understand messages from any kind of sensor there has to be a message format in place which all the sensors use when publishing their data. This message format, which can be seen in listing 2, is used for messages that are published to RabbitMQ and consumed by the Sensor Data Collector. Note that this message format uses JSON.

Listing 2: Sensor Data Collector message format

```
{
  "sensor_id":<UUID>,
  "instance_id":<UUID>,
  "timestamp":<UNIX timestamp>,
  "value":<base64 encoded byte array>,
  "process_id":<UUID>
```

```
}
```

Once more, the *sensor_id* and *instance_id* parameters contain the UUID of the corresponding sensor. The *timestamp* parameter is the UNIX time stamp of when this data was generated. The sensor data is stored in the *value* parameter, which is encoded as a byte array. The byte array itself is also encoded as base64 in order to reduce the size of the message that has to be transmitted. And finally the *process_id* parameter is present for tracking purposes. Using this message format the sensors can communicate with the Sensor Data Collector through RabbitMQ in order to get their data stored in Cassandra.

4.2 SLEEPY SOLUTION

The Sleepy solution as a whole consists of multiple components that work together. First and foremost there is the *Sleepy* component, this is the application that has to be installed on the personal computer. Next, there is the *Sleep Management Server*, it acts as a mediator between Sleepy and outside components. The last component that is part of the solution is the *Sleep Management Console*, which is responsible for the web interface that visualizes Sleepy's sensor data.

4.2.1 *Sleepy*

The Sleepy component itself consists of two separate applications. The core of Sleepy is the *Sleepy System Service*, it runs in the background as a system service. The other application is the *Sleepy User Application*, which is started every time a user logs in. Both of these applications have been developed in C#.NET using Microsoft Visual Studio 2013.

4.2.1.1 *Sleepy System Service*

Monitoring the state of the computer is one of the main tasks the Sleepy System Service performs, it does so by listening to power management events generated by Windows. These occur when a computer turns on, goes to sleep, wakes up or shuts down. These events which are generated by the operaring itself are only broadcasted to Windows Services, which is the reason why this service has to exist.

Furthermore, the service also generates the instance ID's for all the sensors that are part of Sleepy. There are four different sensors, a state sensor for monitoring the computer's state, the activity sensor which reports the activity on the computer, the negative feedback sensor which checks if the user manually disabled sleep and the timeout sensor which detects changes in the sleep timeout.

The Sleepy System Service is also responsible for the communication with the Sleep Management Server. The communication is done through TCP sockets. When the value of one of the four sensors changes it will send these changes to the Sleep Management Server. The service also sends requests for setting values and global system

information to the Sleep Management Server. The format of these messages can be seen in listening 3.

Listing 3: Communication with Sleep Management Server

```
{
  "message_type":<integer>,
  "mac_address":<string>,
  "content":<JSON>
}
```

The *message_type* specifies which of the three types of messages the service has sent: sensor updates, setting requests or global system information. The *mac_address* field indicates the computer from which this message has been received. The *content* depends on the *message_type* parameter, it is always a JSON object. If the message type is a sensor update, then the content will be the sensor data JSON object from listening 2.

The service also communicates with the Sleepy User Application instances running on the computer using Windows pipes. The reason for using pipes instead of sockets is that they are light weight and require few resources to create and destroy. The Sleepy User Application sends the time since it has last detected activity every **x** seconds and the service replies with the current sleep timeout for the computer. An example of this can be seen in listing 4.

Listing 4: Communication with Sleepy User Application

```
Sleepy User Application sends:  154   # seconds since activity
Sleepy System Service replies:   60   # sleep timeout in minutes
```

4.2.1.2 *Sleepy User Application*

Two important tasks are performed by the Sleepy User Application: the monitoring of activity and the changing of the sleep timeout. The activity sensor reports the time since the computer was last active to the Sleepy System Service. The Sleepy System Service uses this information to determine whether the computer is idle or active, as there can be multiple Sleepy User Application instances running due to multiple users being logged in at the same time. This means one user could be actively using the computer while the others are idle. It is even possible for more than one user to be actively using the computer due to remote desktop features. The service needs to take this into account. This is illustrated in figure 11.

The Sleepy User Application also provides the user with the option to disable Sleepy, this action is monitored by the negative feedback sensor and every time Sleepy is disabled this action is logged in the database. This information is used to optimize the sleep timeouts on computers monitored by Sleepy.

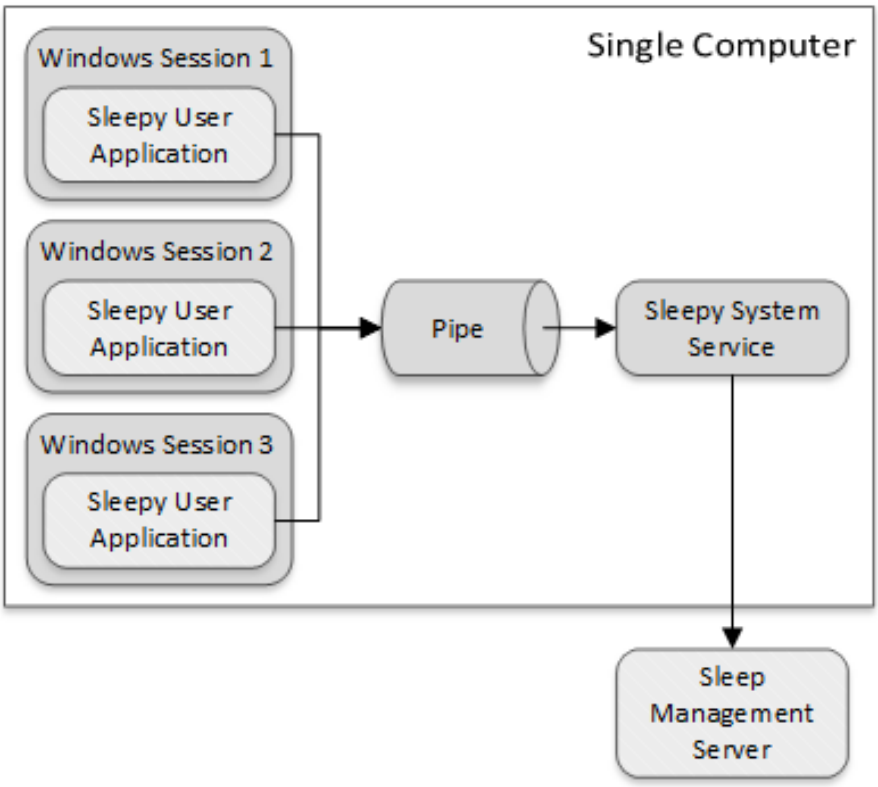


Figure 11: Multiple instances of Sleepy User Application on one computer

4.2.2 *Sleep Management Server*

The Sleep Management Server acts as a gateway for instances of the Sleepy System Service. It has been written in Java using the IntelliJ IDEA development environment. When the service starts it connects to the Sleep Management Server and sends some basic information, this allows the Sleep Management Server to know that this specific computer can be found on a certain MAC and IP address. More than one service can be connected to the Sleep Management Server, as is depicted in figure 12.

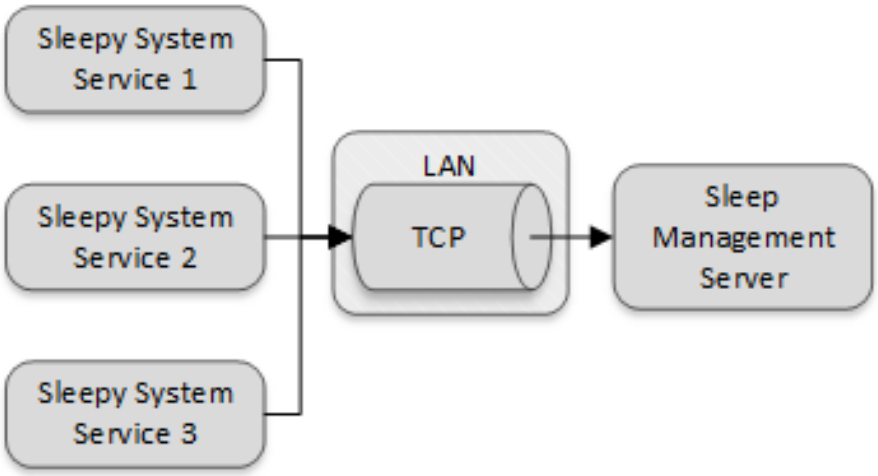


Figure 12: Multiple services connected to one Sleep Management Server

One of the tasks which is handled by the Sleep Management Server are setting requests from services. When a service starts it needs to request the current settings for the sleep timeout and the idle threshold. This is done by sending a setting request message (listing 5) to the Sleep Management Server, which in turn replies with a setting reply message (listing 6).

Listing 5: Setting request

```
{
  "setting_request":<string>
}
```

Listing 6: Setting reply

```
{
  "setting_name":<string>,
  "setting_value":<string>
}
```

The Sleep Management also makes sure that sensor updates produced by services are published to RabbitMQ. Once published to RabbitMQ it will be up to the Sensor Data Collector to read the message and enter it into the Cassandra database, where it is permanently stored.

4.2.3 *Sleep Management Console*

The Sleep Management Console has been developed to be able to visualize all the data generated by Sleepy. It mainly has three purposes: display the current state of the system, give access to historical data and send commands to computers running Sleepy. Just like the Sleep Management Server this component has also been written in Java, making use of servlet technology to provide data to the frontend.

MAC Address	Sleep Timeout	Idle Threshold	Status	Activity
2C44FD0D4CAF	Sleep is disabled	300 second(s)	Off	Inactive
10604B78DD93	Sleep is disabled	300 second(s)	On	Idle
A0D3C1156034	Sleep is disabled	300 second(s)	Off	Inactive
2C44FD327F71	Sleep is disabled	300 second(s)	Off	Inactive
7446A0C0684C	Sleep is disabled	300 second(s)	On	Idle
10604B828C93	Sleep is disabled	300 second(s)	On	Idle
B4B52FC8BCB4	Sleep is disabled	300 second(s)	On	Idle

Figure 13: Overview page of the Sleep Management Console

The currrent state of the system is displayed on the overview page. A fragment of the overview page presented by the Sleep Management Console can be seen in figure 13. The actual overview page displays more information about the version of Sleepy that is installed, details about the operating system as well as the time since a computer has been last seen.

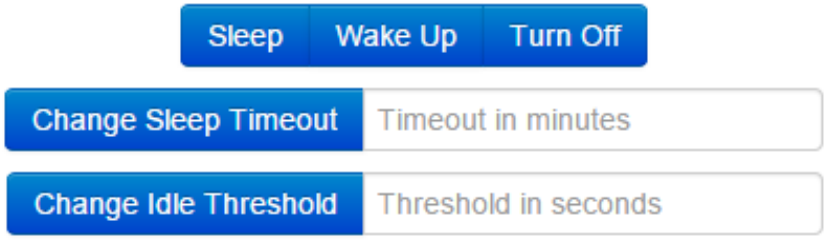


Figure 14: Controls for commanding Sleepy System Service instances

Controls can be found at the bottom of the overview page (figure 14). These controls allows administrators of the system to control the computers and the instances of the Sleepy System service which are running on those computers. It is possible to put a computer to sleep, wake it up, or turn it off completely. The sleep timeout and idle threshold settings can also be changed using these controls.

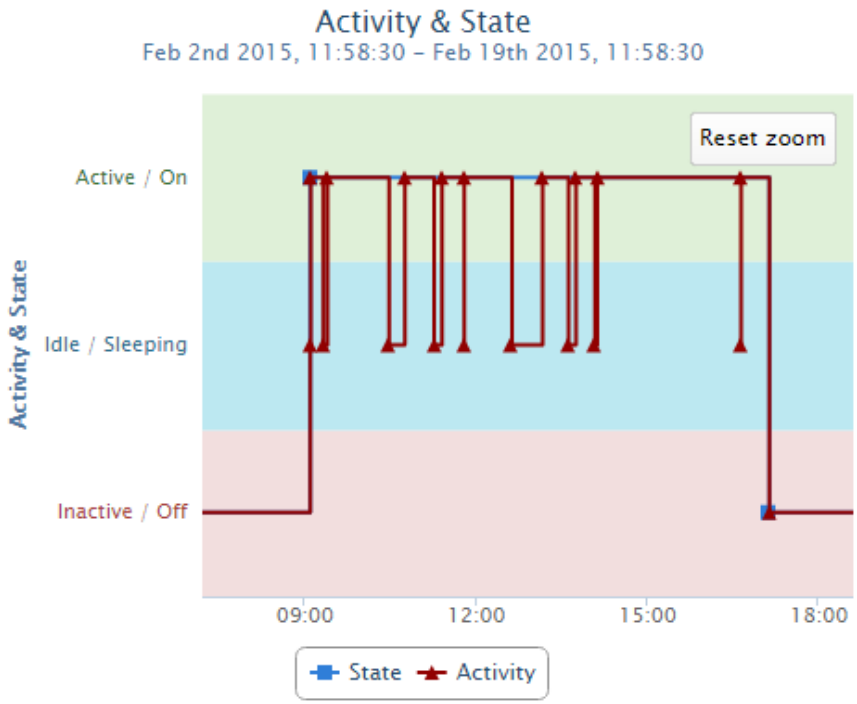


Figure 15: Historical state and activity data

Historical information can be viewed for each computer on the overview page. This allows administrators to verify the data collected by Sleepy and detect if there are any problems with either the computer or Sleepy. Figure 15 shows an example of the historical state and activity data of a computer.

Request	*Action*	*Description*
GET	workstation	Get one specific computer
GET	workstations	Gets all computers
GET	state	Get the state data for a specific computer
GET	activity	Get the activity data for a specific computer
GET	disable	Get the disable data for a specific computer
GET	timeout	Get the timeout data for a specific computer
POST	sleep	Send the sleep command to a computer
POST	wakeup	Send the wakeup command to a computer
POST	turnoff	Send the turn off command to a computer
POST	timeout	Change the sleep timeout of a computer
POST	threshold	Change the idle threshold of a computer

Table 4: Sleep Management Console REST-interface

All data displayed on the Sleep Management Console is extracted from the Sensor Data Storage, the Cassandra database. The way this data is made available to the frontend is by use of a REST-interface. The REST-interface is described in table 4. The frontend calls the REST-interface using jQuery's AJAX-functionality [17] for a more modern user experience.

4.3 GLOBAL ARCHITECTURAL VIEW

Sleepy shares its underlying architecture with other solutions that are part of the Sustainable Buildings project. This architecture has been developed alongside Sleepy and is now a critical part of both Sleepy and also the Sustainable Buildings solution. The goal of this global architecture is to make integration of different components within the Sustainable Building ecosystem more simple.

One of the ways this is achieved is by using a shared code base for all the elementary tasks that services within this ecosystem have to perform, such as communicating with message queues and databases. This shared code base has been named system-core and contains multiple modules that are available for use. These modules include:

- AMQP, for communication with RabbitMQ
- Cassandra, for accessing the Cassandra database
- Graph-DB, for accessing the OrientDB graph database
- Rest-Spray, for setting up a REST service
- Sensor-Data, for handling sensor data
- Service-Core, required by all other modules
- ZooKeeper, for communication with the ZooKeeper server

Using these modules makes the process of building and maintaining services go faster since there is a lot of code that can be reused. For example, the sensor data collector makes use of the AMQP module for retrieving messages from RabbitMQ. It also uses the Cassandra module to insert these messages into the database. And since the messages are sensor messages it uses the Sensor-Data module to handle the messages.

Other services that make use of this shared code base include a planning service [13] for managing the environment in the restaurant of the Bernoulliborg, services that control actuators and report their status to the message queue and services that provide REST interfaces for front-ends in order to expose data from the Cassandra database.

Each of these services have to register themselves in ZooKeeper in order to able to monitor which services are running and which aren't. Registering in ZooKeeper also makes it easier for services to communicate with each other without having to pre-configure the IP-addresses in every service. The only IP-address a service should know is the ZooKeeper IP-address.

The global architecture is a multi-layered architecture, as can be seen in figure 58. A larger image of the global architecture can be found in appendix A. Each of the layers in this architecture is separated and has its own responsibility.

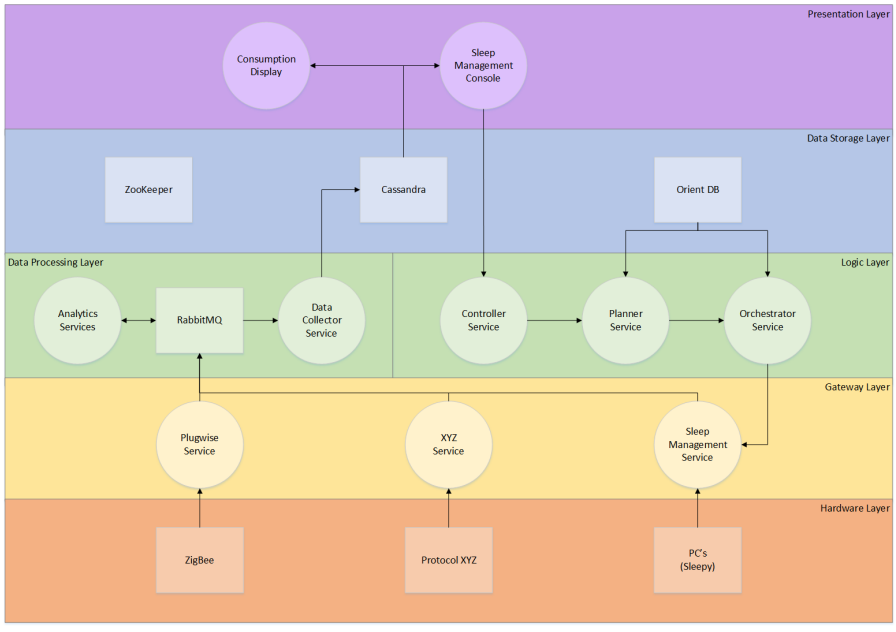


Figure 16: Global Architecture

The hardware devices, such as actuators and sensors, are all part of the *hardware layer*. Each hardware type has its own service associated with them. These services reside in the *gateway layer*.

The *gateway layer* is responsible for the communication with sensors and actuators. Services in this layer can provide REST-interfaces for controlling actuators or publishing data from sensors to the message queue.

Once data has been published to the message queue it enters the *data processing layer*. A special type of service, called the analytics service will pre-process the data if needed, for example it may combine data from multiple sensors into one. Eventually the data will be collected by the sensor data collector.

ZooKeeper, Cassandra and OrientDB are the data stores which make up the *data storage layer*. ZooKeeper stores information about services, Cassandra stores information about sensors and OrientDB stores information about the physical architecture of the system.

When the data has been stored it can be displayed by services in the *presentation layer*. Examples of these are the Sleep Management Console and an Energy Display which displays the total energy consumption within the Bernoulliborg.

Sometimes actions need to be performed based on events, the services in the *logic layer* perform these tasks. The controller service, planner service and orchestrator service work together to create a plan, a set of actions, based on incoming sensor data. The planner will execute these actions by communicating with the services in the *gateway layer*. Thus we have come full circle.

5 EXPERIMENTS AND EVALUATION

Experiments have been performed to determine the effectiveness of the three different models defined in chapter 3. The experiments lasted three weeks, and during these weeks each computer used every model. The experiments started on the 1st of June 2015 and ended on the 22nd of June 2015. In this chapter the results of the experiments are analyzed in order to determine which of the three models performs best, and how much energy can potentially be saved. Determining the best model is done using the model evaluation as described in the problem statement. Not only the performance of the models are analyzed but also the performance of Sleepy itself. We will also take a look at the economic savings that can be achieved when deploying Sleepy on a larger scale.

In total fourteen computers have been chosen to be part of the experiments. Each of the selected computers are used by staff members of the University of Groningen. Due to limitations imposed by the system administrators of the University of Groningen it was not possible to install Sleepy on more computers. It would have been interesting to compare the results from staff member computers and the results from the publicly available student computers. The reason being that the publicly available computers for students are often the computers which are left running while not being actively used.

Identifier	*Model Week 1*	*Model Week 2*	*Model Week 3*
Computer 1	1	2	3
Computer 2	1	2	3
Computer 3	1	3	2
Computer 4	1	3	2
Computer 5	2	1	3
Computer 6	2	1	3
Computer 7	2	3	1
Computer 8	2	3	1
Computer 9	3	1	2
Computer 10	3	1	2
Computer 11	3	2	1
Computer 12	3	2	1
Computer 13	4	4	4
Computer 14	4	4	4

Table 5: The fourteen computers chosen for the experiments

The computers which have been chosen are shown in table 5. Each computer is identified by Sleepy using a MAC address but for readability purposes numbers have been assigned to them instead. Each computer is tested over a three week period in which the model changes every week. The table shows which models were used during the first, second and third week. Models 1, 2 and 3 refer to the models described in chapter 3. Model number 4 refers to the control model, which is not using a sleep timeout. This simulates the case for all computers at the University of Groningen which do not have Sleepy installed.

5.1 COMPUTER USAGE PROFILING

Predicting when a computer will be in use is a very important aspect in the process of determining the right personalized sleep timeout. Each of the three models take this into account by using the activity probability. The activity probability represents the probability of seeing activity on a computer at a certain moment in time. A usage profile can be generated when calculating the activity probability at multiple points during a given time interval.

The time series data related to these computers appears to have a weekly seasonality. This is as expected as users often work on the same days every week. Therefore, historical data from the same day of the week is used to predict the future activity.

Data from up to four weeks ago is used in the prediction of the activity probability. Including more than four weeks of data would result in the data set not adapting to changes in circumstances, for example holidays or sick leave. Including less than four weeks in this timespan would result in less accurate predictions, therefore four weeks was chosen to find a balance. When predicting the activity probability for a certain moment in time a window of ten minutes is used, thus if there was activity in the past during this window then it counts towards activity for that moment in time. These parameters have been chosen after some experimentation to produce the most accurate results. Equation 11 shows how the profiles are calculated.

$$P_{activity}(t) = \frac{\sum_{i=1}^{n} f(t - weeks(n))}{n} \tag{11}$$

where:

- f(t) is 0 (no activity) or 1 (some activity) depending on whether there was activity at the given moment in time.

For example, in order to predict the activity probability on Monday the 16th of February at 12:00 one has to look at the past four Mondays (four weeks window). For each of these past Mondays the activity data between 11:55 and 12:05 (ten minutes window) has to be checked and if there has been activity between these points in time, one needs to take it into account when predicting the activity probability.

We shall now take a closer look at the resulting profiles for each of the fourteen computers. The profiles shown in the following figures are the average profile of all three weeks. The probability ranges anywhere from zero percent to one hundred percent. Where zero percent indicates there is no predicted activity, and one hundred percent indicating that there is a probability of a hundred percent of seeing activity.

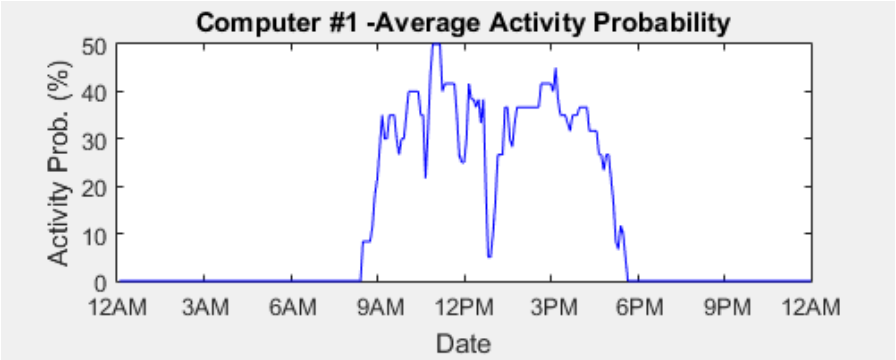


Figure 17: Computer 1 - average activity probability

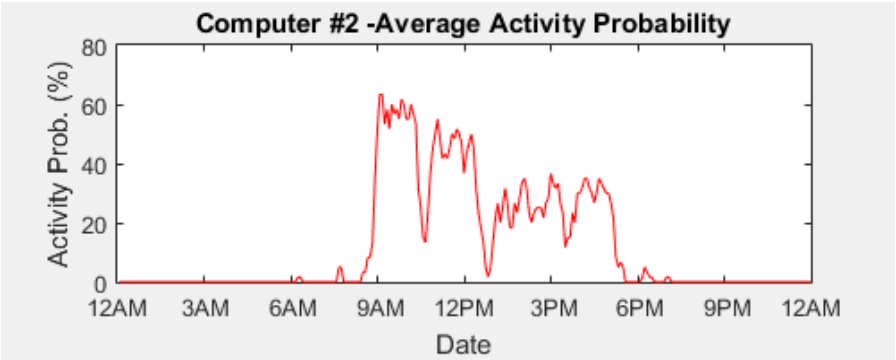


Figure 18: Computer 2 - average activity probability

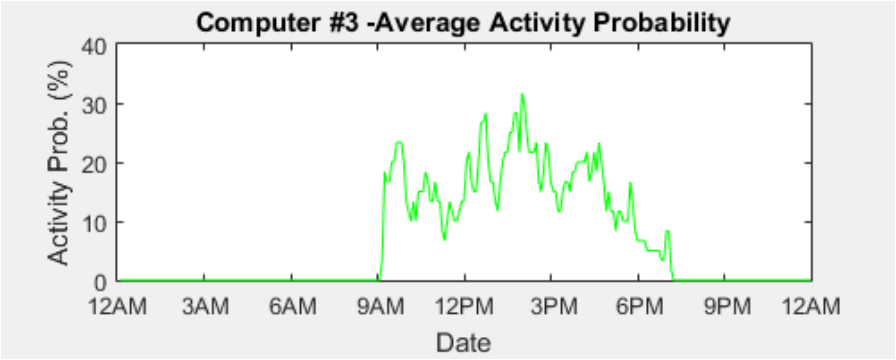


Figure 19: Computer 3 - average activity probability

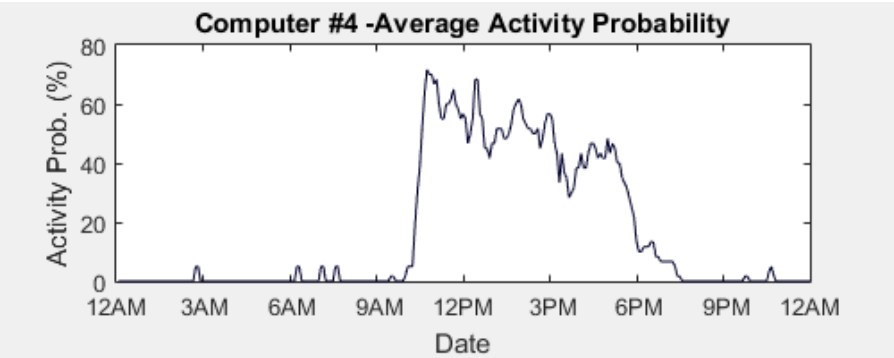


Figure 20: Computer 4 - average activity probability

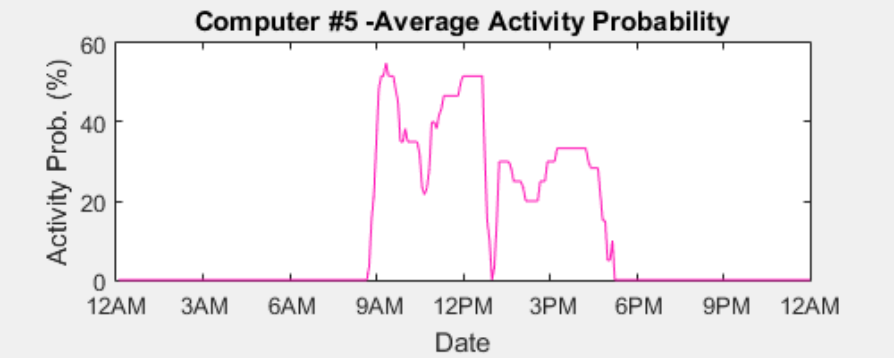


Figure 21: Computer 5 - average activity probability

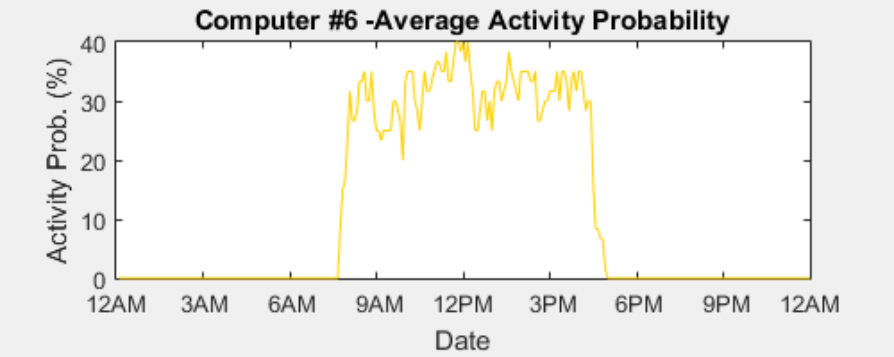


Figure 22: Computer 6 - average activity probability

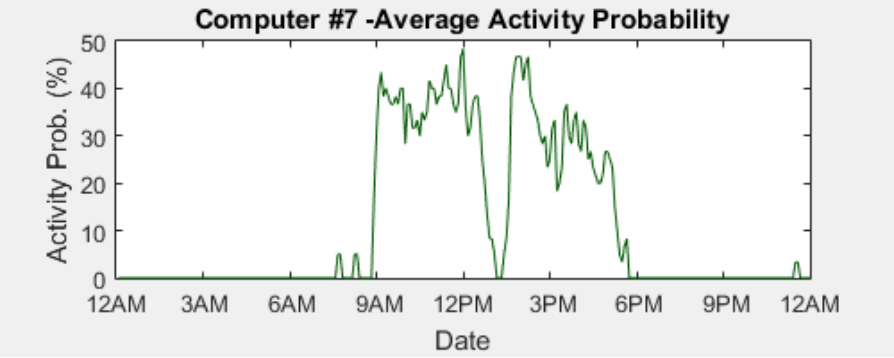


Figure 23: Computer 7 - average activity probability

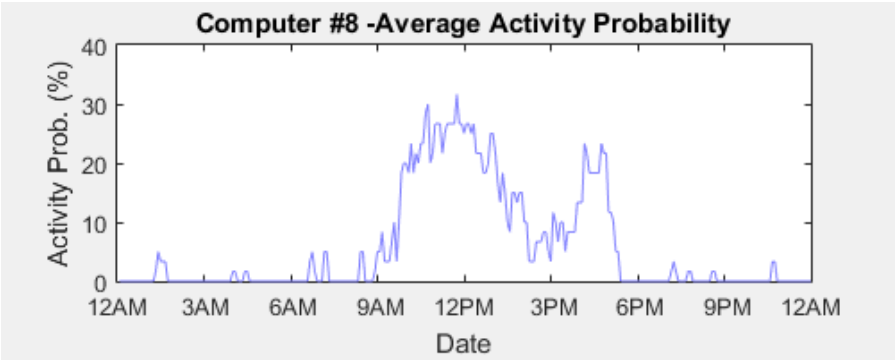


Figure 24: Computer 8 - average activity probability

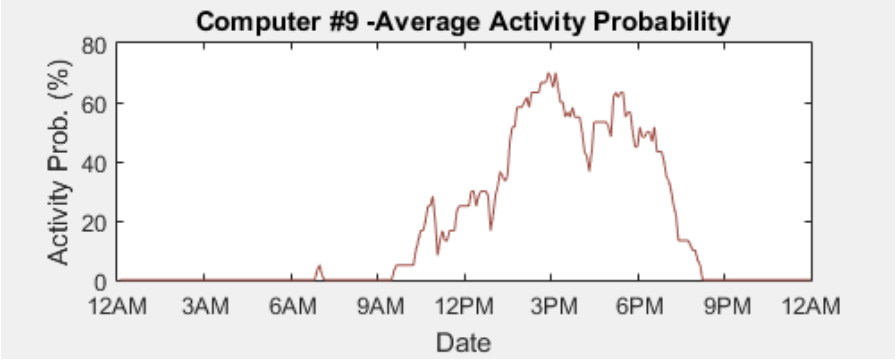


Figure 25: Computer 9 - average activity probability

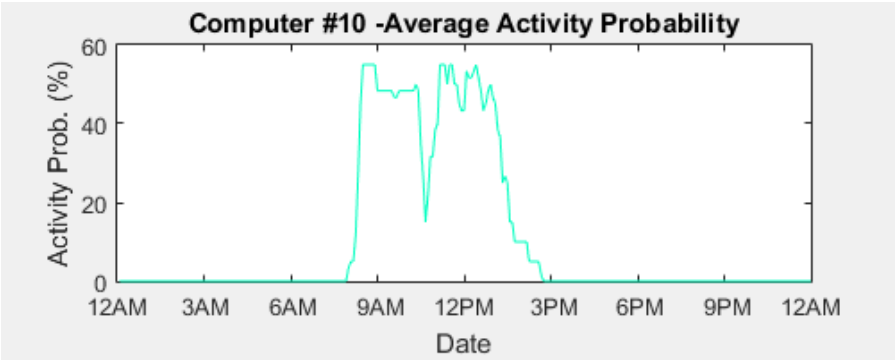


Figure 26: Computer 10 - average activity probability

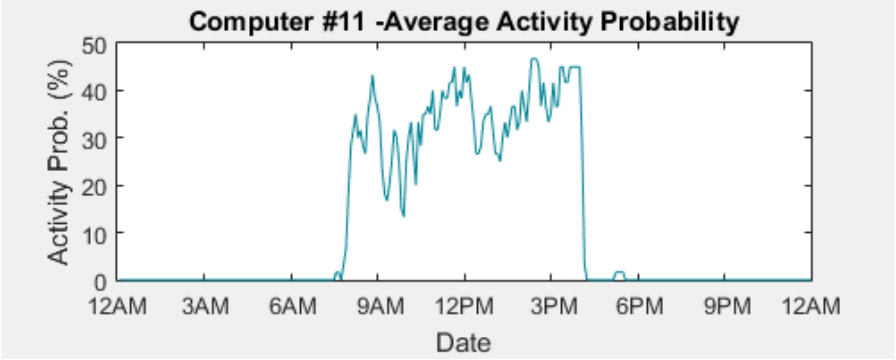


Figure 27: Computer 11 - average activity probability

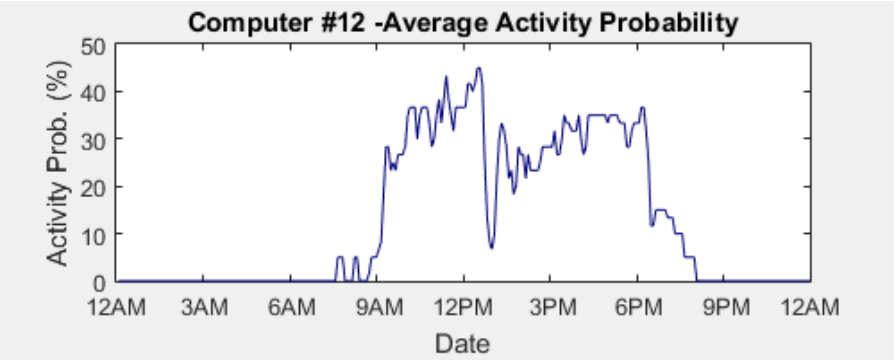


Figure 28: Computer 12 - average activity probability

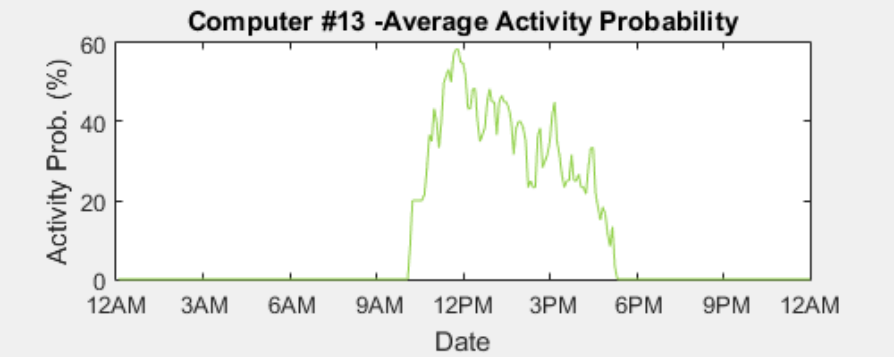


Figure 29: Computer 13 - average activity probability

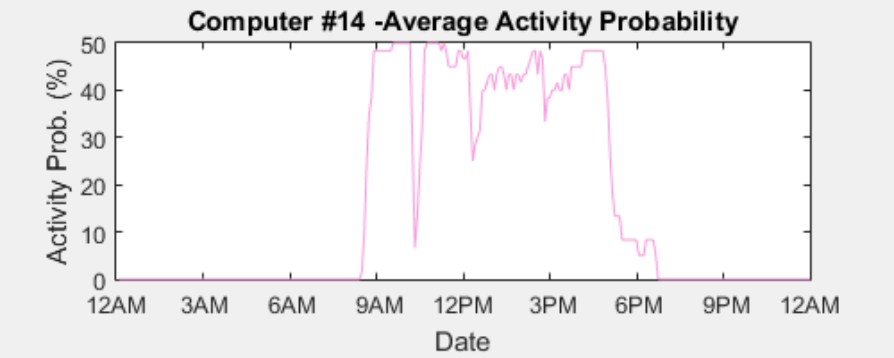


Figure 30: Computer 14 - average activity probability

These average profiles were generated by combining the daily profiles over the three week period during which the experiments were running. The learning algorithm only uses the daily profiles, and not these average profiles. The average profiles are displayed here to show patterns and provide insight. All of the complete daily profiles used in the process to learn the optimal sleep timeout can be found in appendix B.

A very common pattern which can be seen in most of the profiles is that the activity probability drops at certain times. The most common drop occurring at around 1 PM. This can be seen mostly in the profiles of computer 1 (fig. 17), computer 2 (fig. 18), computer 5 (fig. 21), computer 7 (fig. 23), computer 12 (fig. 28) and computer 14 (fig. 30). This can be explained by users leaving their computer for a lunch

break. Another common pattern is a drop in the activity probability around 10 PM or 11 PM. The pattern can be clearly seen in the profiles of computer 2 (fig. 18), computer 5 (fig. 21), computer 10 (fig. 26), computer 11 (fig. 27) and computer 14 (fig. 30).

Another pattern that can also be seen in the activity profiles is that most users use their computer almost exclusively between 8AM and 7PM. These patterns which can be seen in the average profiles verify that there are moments during the day on which energy could be saved. For example, during lunch time and other breaks the computer could be put into sleep mode. This could significantly reduce the energy consumption of the computer for up to an hour every working day.

These patterns verify that there are indeed opportunities to save energy, even if users shut down their computer at the end of every day. Assuming an average working day of eight hours and a lunch break of thirty minutes means that the computer could potentially sleep for $\frac{0.5}{8} = 6.25\%$ of the time of a full working day. Thus it is expected that at the very least the efficiency of the models should be around 6.25%.

5.2 SOFTWARE PERFORMANCE

One of the main goals of this thesis is to optimize the sleep timeout of computers while keeping user comfort at high levels. Therefore it is important to make sure that the software that has been developed to monitor computers does not disrupt the work flow of the user in the first place. The main way in which the software could potentially interfere with the end user is by slowing down the computer due to consuming large amount of resources.

In previous versions of Sleepy, which were developed before the start of this master project, there were many issues with the software itself which caused user discomfort. Some of these included crashes, memory leaks and high CPU usage. Therefore it is important that this new version of Sleepy that has been developed during this master project does not suffer from the same shortcomings as its predecessors. The following resources are interesting to look at in order to determine if the software itself interferes with the user:

- CPU cycles.
- Random-access memory.
- Disk input-output.
- Network input-output.

Three of these four resources have been monitored and analyzed in order to determine the impact that Sleepy has on the host computer. It is critical that the sleep software does not slow down the computer because this would have a large negative impact on the user comfort levels before even trying to optimize the sleep timeout. The disk

input-output has not been measured since Sleepy does not perform any actions which access the disk, except during startup.

5.2.1 *CPU Profiling*

In order to profile the CPU usage the dotTrace profiler has been used. The dotTrace profiler is a profiler for .NET applications, it allows for on-demand performance profiling as it can be easily attached to processes that are already running. This makes it a good choice for testing Sleepy, since Sleepy consists of more than one application running at the same time. Both the Sleepy System Service and the Sleepy User Application have been profiled. For both the distinction was made between the initialization phase and the normal operation, in order to better understand what happens during the initialization phase. The initialization phase consists of the first few seconds after starting the application whereas normal operation is everything after the initialization phase.

The profiling is done at the level of methods, this way we can easily discover which methods are potential bottlenecks and which methods should be optimized. When looking at methods we look at two different states: waiting and running. These refer to the state of the thread executing these methods. Waiting methods do not consume CPU cycles but show the reason why threads are in the waiting state. Running methods do consume CPU cycles and should be optimized when needed. Both the Sleepy System Service and Sleepy User Application have been profiled during a one hour period.

Core	*Percent*
CPU Core 8	34.1
CPU Core 3	30.2
CPU Core 1	20.9
CPU Core 7	7.8
CPU Core 4	3.4
CPU Core 6	1.7
CPU Core 5	1.4
CPU Core 2	0.5

Table 6: Load distribution of Sleepy System Service during initialization

First we shall analyze the CPU usage of the service part of Sleepy: the Sleepy System Service. The total CPU utilization during the initialization phase was 3.1 percent. The distribution of this load across all the CPU cores can be seen in table 6.

Table 7 shows the percentage of time during which threads were in a certain state during the initialization phase of the service. As can be seen the service spent a large portion of time waiting. When a thread is in waiting state the thread consumes no CPU cycles.

Thread State	*Percent*
Waiting	93.2
Running	6.8
Ready	0.02

Table 7: Thread state of Sleepy System Service during initialization

In table 8 we can see the methods which spent the most amount of time in the waiting state while the service was initializing. The first method is used when connecting to the Sleep Management Server, it waits for the connecting to be completely set up. The second method is used to put threads to sleep. Other methods consumed significantly less time in the waiting state.

Method	*Percent*
System.Net.Sockets.DoConnect	25.6
System.Threading.Thread.Sleep	22.1

Table 8: Top methods of Sleepy System Service in the waiting state during initialization

Next we shall take a look at the top methods during the running state. These are the methods which are responsible for the CPU usage during the initialization phase. The six methods consuming the most CPU time are displayed in table 9.

Method	*Percent*
Sleepy.Program.cctor	18.5
Sleepy.Service.SleepyCore.cctor	10.9
SleepyLibrary.JSON.JSonMessage.ToString	9.5
Newtonsoft.Json.JsonSerializer.GetMatchingConverter	6.2
Sleepy.Service.SleepyCore.Start	4.7
SleepyLibrary.OS.OsController.GetMacAddress	4.6

Table 9: Top methods of Sleepy System Service in the running state during initialization

The first two methods, as well as the fifth method, are responsible for initializing the service. They start the threads, make the TCP connections, read the configuration and so on. Therefore it is not surprising to see them at the top of the list. What is more surprising is to see that serializing JSON messages is a relatively expensive operation, as both the third and fourth method account for about fifteen percent

of the total CPU load during initialization. Perhaps even more surprising, retrieving the MAC address of the active network interface accounts for almost five percent of the total CPU load.

Now that we know the statistics for the initialization phase we shall take a closer look at the CPU load during normal operation. We start again with the average CPU utilization, which is on average 0.04 percent during normal operation. The utilization per core can be seen in table 10.

Core	*Percent*
CPU Core 2	22.4
CPU Core 7	17.8
CPU Core 3	14.0
CPU Core 1	13.3
CPU Core 5	12.5
CPU Core 4	9.5
CPU Core 6	6.5
CPU Core 8	3.9

Table 10: Load distribution of Sleepy System Service during normal operation

As can be seen in table 11 the service is idle almost all the time. This is expected as the service is mainly responsible for passing messages from the Sleepy User Application to the Sleep Management Server. When it is not passing these messages it is listening for new messages.

Thread State	*Percent*
Waiting	99.9
Running	0.03
Ready	< 0.01

Table 11: Thread state of Sleepy System Service during normal operation

The methods that spent the most amount of time in the waiting state during normal operation are very similar to the ones during the initialization phase. The only addition is the last method that can be seen in table 12. This method is responsible for receiving data from TCP sockets, it blocks the thread when waiting for data.

Method	*Percent*
System.Threading.Thread.Sleep	21.1
System.Net.Sockets.DoConnect	18.6
System.Net.Sockets.Receive	18.6

Table 12: Top methods of Sleepy System Service in the waiting state during normal operation

During normal operation both JSON handling (method four and six) and TCP connections (method two, three and five) account for a large portion of the CPU utilization as can be seen in table 13. This is as expected as the Sleepy System Service connects to the Sleep Management Server via TCP sockets and sends JSON messages across. A bigger surprise is the first method, which is responsible for reading the current sleep timeout from a configuration file. As this value might change the service needs to refresh it often, therefore it needs to read the file and parse the setting. This appears to be a relatively expensive operation.

Method	*Percent*
Sleepy.Service.SettingsManager.GetAppSetting	18.9
SleepyLibrary.Network.TcpServer.HandleConnection	17.7
SleepyLibrary.Network.TcpServer.ConnectionCallback	17.3
Newtonsoft.Json.Serialization.JsonSerializer.CreateObject	4.1
SleepyLibrary.Network.Client.TcpClient.Connect	4.0
Newtonsoft.Json.Serialization.JsonSerializer.PopulateObject	3.2

Table 13: Top methods of Sleepy System Service in the running state during normal operation

The next step is to profile the Sleepy User Application during the initialization phase. This has been done in the same way as the Sleepy System Service. First we shall take a look at the CPU utilization during the initialization phase. During the initialization phase the CPU utilization was 2.3 percent on average. The load per CPU core can be seen in table 14.

Core	*Percent*
CPU Core 3	31.6
CPU Core 6	22.6
CPU Core 5	20.3
CPU Core 4	12.5
CPU Core 1	7.4
CPU Core 7	2.8
CPU Core 2	2.5
CPU Core 8	0.3

Table 14: Load distribution of Sleepy User Application during initialization

During initialization the thread states are very similar to the Sleepy System Service: there is some CPU activity but the application is mostly idle. The exact numbers can be found in table 15.

Thread State	*Percent*
Waiting	91.2
Running	8.7
Ready	0.1

Table 15: Thread state of Sleepy User Application during initialization

When looking at the top methods (tab. 16) which are in the waiting state it also look very similar to previous results. Connecting to TCP sockets is once again responsible for the most amount of time spent in the waiting state. A new method which also spends a reasonable amount of time waiting is the Windows Form message loop. This message loop is there because the System User Application has a graphical user interface which listens to certain Windows events. Other methods spent significantly less time in the waiting state.

Method	*Percent*
System.Net.Sockets.DoConnect	42.0
System.Windows.Forms.ApplicationManager.MessageLoop	3.2

Table 16: Top methods of Sleepy User Application in the waiting state during initialization

In table 17 the methods which consume the most amount of CPU cycles are shown. The first two methods are responsible for initializing the application, starting the threads, building the user interface and so on. Thus is it expected that they consume more CPU cycles during the initialization phase. The third and fifth methods setup the socket connection between the Sleepy User Application and the Sleepy System Service. Once again finding the MAC address of the active network interface takes a relatively large amount of time.

Method	*Percent*
Sleepy.Program.Main	20.7
Sleepy.Program.cctor	16.3
Sleepy.Network.Client.SleepyServiceTcpClient.Write	7.3
Newtonsoft.Json.JsonSerializer.GetMatchingConverter	5.5
SleepyLibrary.Network.Client.TcpClient.Connect	4.2
SleepyLibrary.OS.OsController.GetMacAddress	3.8

Table 17: Top methods of Sleepy User Application in the running state during initialization

Finally we can take a look at the performance of the Sleepy User Application during normal operation. When under normal operation

the CPU utilization is around 0.09 percent. The CPU utilization per CPU core can be found in table 18. The Sleepy User Application only consists of three threads, therefore it is not as easy to distribute the load compared to the Sleepy System Service.

Core	*Percent*
CPU Core 1	28.2
CPU Core 3	16.3
CPU Core 5	14.0
CPU Core 7	11.9
CPU Core 2	11.8
CPU Core 4	6.7
CPU Core 6	6.4
CPU Core 8	4.8

Table 18: Load distribution of Sleepy User Application during normal operation

During normal operation the Sleepy User Application spends most of the time waiting. The reason for this is because the Sleepy User Application periodically sends activity information back to the Sleepy Management Server. Once this data has been sent the application goes back to sleep until it is time for another update. The percentage of time spent in each state can be seen in table 19.

Thread State	*Percent*
Waiting	99.7
Running	0.2
Ready	0.06

Table 19: Thread state of Sleepy User Application during normal operation

When looking at the methods which spent the most time in the waiting state there are only two main entries as can be seen in table 20. The other methods spent less than 1 percent of the time in the waiting state. Neither of these methods are unexpected as the Sleepy User Application only periodically sends updates (method one) and listens to Windows events (method two).

Method	*Percent*
System.Threading.Thread.Sleep	34.7
System.Windows.Forms.ApplicationManager.MessageLoop	30.3

Table 20: Top methods of Sleepy User Application in the waiting state during normal operation

Finally we take a look at the top methods in the running state during normal operation. The first method in table 21 practically consumes all of the total CPU cycles used by the Sleepy User Application. This method is used to change the sleep timeout. In order the change the sleep timeout a new process has to be created which spawns a command prompt, this command prompt executes the command to change the sleep timeout of the current power management profile. Spawning a process is a very expensive operation, as can be seen in the table.

Method	*Percent*
SleepyLibrary.OS.OsController.ExecuteCommand	93.5
SleepyLibrary.Network.Client.TcpClient.Read	1.0
Sleepy.Application.SleepyUserApplication.Run	0.7

Table 21: Top methods of Sleepy User Application in the running state during normal operation

5.2.2 *Memory Profiling*

It is important that both the Sleepy System Service and the Sleepy User Application do not consume large amounts of random access memory. If they would it could cause the host computer to slow down and possibly run out of memory. This is especially true if there are memory leaks in either of the two application. Therefore the memory usage has been profiled in great detail using a tool named dotMemory. Profiling the memory usage has been done over the course of one hour. A snapshot of the memory usage has been taken every twenty minutes. We only take a look at the memory actually used by Sleepy, and not the overhead that comes with using the .NET framework.

First we shall take a look at the memory usage of the Sleepy System Service. The results of each snapshot can be seen in table 22. It is clear that the memory usage remains almost constant. The small differences come from String object allocations which are created when sending and receiving JSON messages. These objects are collected by the garbage collector after some time.

Snapshot	*Memory Usage*
Snapshot 1	731.44 KB
Snapshot 2	737.44 KB
Snapshot 3	734.35 KB

Table 22: Memory usage of Sleepy System Service

When taking a closer look at which objects are responsible for consuming the memory it becomes apparent that the String objects do indeed account for most of the memory usage. Table 23 provides a summary of the top five object types responsible for consuming the most amount of memory.

Object Type	*Memory Usage*
String	208.68 KB
Hashtable[]	62.25 KB
Int32[]	47.04 KB
Object[]	33.77 KB
String[]	22.38 KB

Table 23: Memory usage of Sleepy System Service per object type

Since String object consume by far the most amount of memory let us take a closer look at these String objects to determine where they originate from. The top three results of inspecting the String objects is shown in table 24. Rather surprisingly, handling JSON objects is not responsible for most of the String allocations. Both the configuration of the logging system and the configuration of the Sleepy System Service itself account for almost all of the String object allocations.

String Allocation Reason	*Memory Usage*
Logging Configuration	112.85 KB
Application Configuration	46.78 KB
JSON Handling	29.86 KB

Table 24: Top String memory usage of Sleepy System Service

Next we will perform the same analysis for the Sleepy User Application. We start once again with taking a look at the three memory snapshots, which can be seen in table 25. The memory consumption of the Sleepy User Application remains very much constant during the profiling period. However, compared to the Sleepy System Service the amount of memory being consumed is rather high. Especially considering the fact that the Sleepy User Application is a very basic application in terms of complexity when compared to the Sleepy System Service.

Snapshot	*Memory Usage*
Snapshot 1	2.16 MB
Snapshot 2	2.16 MB
Snapshot 3	2.16 MB

Table 25: Memory usage of Sleepy User Application

Table 26 shows the object types which consume the most amount of memory. It appears that byte arrays are responsible for the large

difference in memory consumption. After closer inspection it appears that it is in fact a single byte array of 1.58 MB which contains the Sleepy User Application logo stored in the icon format.

Object Type	*Memory Usage*
Byte[]	1.58 MB
String	196.11 KB
Hashtable[]	55.22 KB
Int32[]	32.16 KB
Object[]	28.30 KB

Table 26: Memory usage of Sleepy User Application per object type

When ignoring this byte array the memory consumption is very similar to the memory consumption of the Sleepy System Service. The memory consumption when excluding the byte array is around 580 KB. Which is indeed lower than the Sleepy System Service, as would have been expected.

5.2.3 *Network I/O Profiling*

Sleepy relies on a network connection since it depends on the Sleep Management Server for configuration. Since Sleepy is aimed at office buildings it is important that the bandwidth consumption remains low. This is especially true in offices with large numbers of computers. Once again we make the distinction between the initialization phase and normal operation. The reason for this is that during the initialization phase a lot of additional network traffic occurs. The packet analyzer Wireshark was used to capture packets between the Sleepy Management Server and the Sleepy System Service.

During the initialization three messages are sent between the client and the server. These messages add up to 1.77 Kilobyte. This included setting up the TCP connection and all the TCP ACK, SYN and FIN packets. Thus every time a computer with Sleepy installed on it starts it will require 1.77 KB of bandwidth. If 500 computer were started at the same time, this would require less than 1 Megabyte, or 8 Megabit, of bandwidth. Something a modern network can easily handle as 100 Megabit connections are standard and 1 Gigabit is not uncommon.

After initialization when normal operation starts the client and the server communicate approximately once every five minutes. The total size of all packets involved in these periodic updates is 0.805 KB, less than half of the packet size during the initialization phase. Therefore it is safe to say that even with a large number of computers with Sleepy installed the load of the network is minimal.

5.3 RESULTS

Each model has been tested and evaluated for exactly six days, starting on Monday and ending on Saturday. The seventh day of the week, Sunday, was used for transitioning between models. The exact times between which the models were evaluated are as follows: from Monday 00:00:00 until Saturday 23:59:59. This means the length of each timespan is 518,399 seconds, or approximately 144 hours. To see which model was used during which week for a given computer, please take a look at table 5.

All data was extracted from the database after the experiments were finished. The extracted data was stored in CSV files for easy parsing. The MATLAB software was used to parse these CSV files and produce the graphs which are visible on the following pages.

5.3.1 *Model Performance*

To determine which model performs better a formula was defined in chapter 3. This formula has two parameters as inputs, the idle time and the negative feedback score. Once these parameters are known the model can be evaluated. The goal is to find a model which minimizes the idle time and at the same time keeps the overall negative feedback as low as possible.

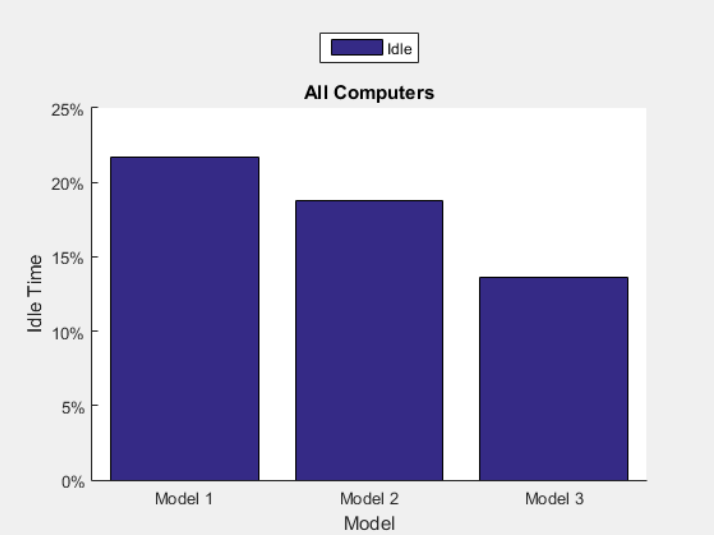


Figure 31: Total idle time per model

Let us start with taking a look at the idle times that are associated with each model. For each model we shall determine the total amount of time spent in the idle state per model. Figure 31 shows this percentage spent in the idle state of the total amount of time for each model. As was predicted, the least aggressive model, model 1, has the highest idle time percentage. Whereas the most aggressive model, model 3, spent the least amount of time in the idle state. This can be explained by the fact that model 1 is very conservative with the sleep timeout, and therefore often leaves the computer idling. A detailed

overview of the state of each computer for each model can be found in appendix C.

When looking at the idle time in hours instead of as the percentage of total we can see there is a significant different between the three models. Table 27 shows the idle time per model in hours, and also the average idle time per computer. It can be seen that the difference in idle time between the least and most aggressive model is almost one hour per computer. Model 3 is more aggressive the longer a computer is idle, therefore it is not surprising that this model has the least amount of idle time.

Model	*Hours Idle*	*Hours Idle per Computer*	*Std. dev.*
Model 1	31.19	2.60	1.73
Model 2	27.05	2.25	1.76
Model 3	19.57	1.63	1.56

Table 27: Total idle time in hours per model

Next, let us take a look at how each model influences the time spent sleeping. The graph displayed in figure 32 shows how much percent of the time the computers were asleep when using a certain model. A significant difference can be observed between model 1 and the other two models.

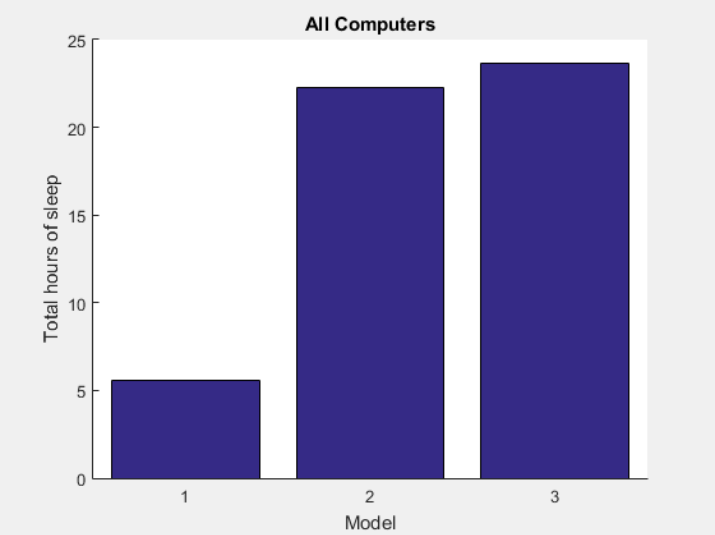


Figure 32: Total time sleeping per model

The difference in hours of sleep is due to the fact that the first model puts the computer to sleep when there is no probability of seeing activity. Which means that there are fewer moments when it puts the computer to sleep, when compared to the other two models. The second and third models are more aggressive when it comes to choosing when to put the computer to sleep and therefore result in more hours of sleep.

The exact number of hours of sleep for each model are shown in table 28. When looking at the hours of sleep per computer it becomes apparent that using model 1 results in about 30 minutes of sleep per computer per week. Whereas using model 3 leads to almost 120 minutes of sleep per computer per week. Model 2 is only slightly behind model 3 with around 110 minutes of sleep per computer per week. Using models 2 or 3 results in over four times as many hours of sleep compared to model 1.

Model	*Hours Sleeping*	*Hours Sleeping per Computer*	*Std. dev.*
Model 1	5.54	0.46	0.42
Model 2	22.26	1.85	2.29
Model 3	23.68	1.97	3.17

Table 28: Total sleep in hours per model

Finally, let us take a look at the negative feedback associated with each model. An interesting fact is that the only kind of negative feedback was caused by a computer going to sleep too early, while the user was still using the computer. There were no reports of negative feedback through the dialog which disables the software completely. Even though users were made aware of this possibility while they were using the software.

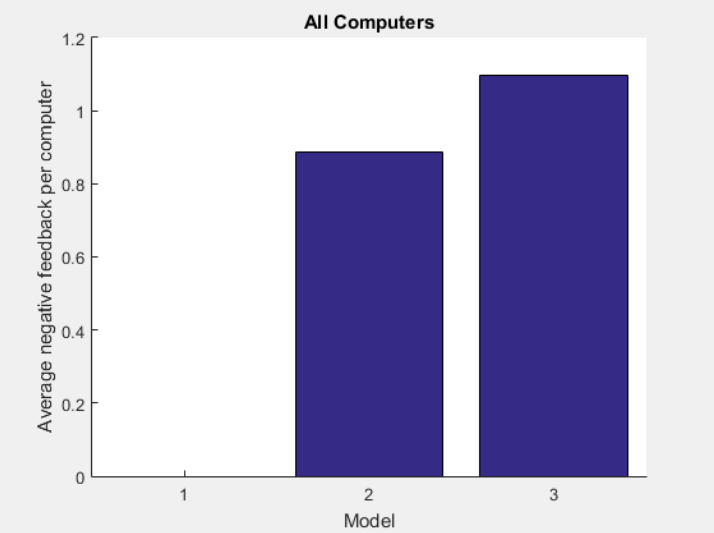


Figure 33: Average negative feedback score per computer

The number of negative feedback occurrences per computer can be seen in figure 33. Surprisingly there is only one model which does not have any negative feedback associated to it, which is model 1. This can be explained by the fact that this model is the least aggressive model. There were 11 occurrences of negative feedback when using model 2. And 14 occurrences of negative feedback while model 3 was in use. This equals to a negative feedback score of 0.917 per computer for model 2 and 1.167 per computer for model 3.

Another interesting aspect to look at is the amount of negative feedback generated per hour of sleep. This ratio of negative feedback per hours of sleep gives a good indication of how expensive (in terms of discomfort or negative feedback) each hour of sleep is. For model 1 this number is $\frac{0}{5.54} = 0$. Whereas this number is $\frac{11}{22.26} = 0.49$ for model 2. Finally, for model 3 the negative feedback per hour of sleep is $\frac{14}{23.68} = 0.59$. These results are not unexpected as model 1 generated no negative feedback and therefore is the least expensive in terms of discomfort per hours of sleep. However, model 2 and model 3 are very close together cost-wise.

Now that we know the results in terms of idle time, hours of sleep and negative feedback we can apply the cost function defined in chapter 3 to determine the ranking of each model. The equation is as follows:

$$E(t) = \theta_2 \sum_{z \in A_{idle}(t)} z + \theta_1 N(t) \tag{12}$$

A lower score means that the model is better according to our cost function. We can enter the values for the idle time and the negative feedback. For example when entering these values for model 2 the equation would become:

$$E(t) = \theta_2 27.05 + \theta_1 11 \tag{13}$$

Now we have to choose our θ_1 and θ_2. Let us set $\theta_2 = 1$ such that:

$$E(t) = 27.05 + \theta_1 11 \tag{14}$$

We want our θ_1 to increase the value of negative feedback in some cases, since negative feedback should always be penalized due to the focus on preventing discomfort. This can be done by using the ratio of negative feedback per hours of sleep. A higher ratio should increase the negative feedback score, since this means the model was less effective at keeping the negative feedback at a minimum.

We choose $\theta_1 = (1 + R(t))$ where $R(t)$ is the negative feedback per hour of sleep ratio for the given timespan. Thus for model 2 it would be $\theta_1 = (1 + 0.49) = 1.49$. Thus the equation becomes:

$$E(t) = 27.05 + 1.49x11 = 43.44 \tag{15}$$

For model 1:

$$E(t) = 31.19 + 1x0 = 31.19 \tag{16}$$

For model 3:

$$E(t) = 19.57 + 1.59 * 14 = 41.83 \tag{17}$$

Therefore the ranking is as shown in table 29. With model 1 being the best model in terms of minimizing user discomfort and model 3 and model 2 being second and third in the rankings.

Ranking	*Model*	*Score*	*Idle Time*	*Sleep Time*	*Sleep/Feedback*	*Feedback*
1	Model 1	31.19	31.19	5.54	0.0	0.0
2	Model 3	41.83	19.57	23.68	0.59	14
3	Model 2	43.44	27.05	22.26	0.49	11

Table 29: Parameters and ranking per model

5.3.2 *Energy & Economic Savings*

In the previous section we ranked the models based on a cost function which penalized negative feedback. A different way to look at each model is by taking energy savings and economic value into account. For the building managers of offices and other large buildings the economic perspective and the sustainability aspects might also be very interesting. Especially since energy costs are rising and governments pushing organizations towards a more sustainable environment. For example in the Netherlands, where the government provides a number of subsidies for energy related initiatives [30].

As mentioned in chapter 2 the energy consumption of a computer is around 115 watts according to Nordman et al [26]. According to Roberson et al [31] the power consumption of desktop computers is around 105 watts. In [4] it is said that the energy consumption of computers ranges anywhere from 77 watts to 322 watts. All sources however agree that the power consumption when sleeping is around 5 watts. These numbers are assuming there is one desktop computer with one monitor. In the following calculations we shall use 150 watts as the average power consumption, to take into account the multi-monitor desktop setup. Furthermore, we shall use the current energy tariffs in the Netherlands, which is 22 eurocents per kilowatt-hour [23] at the time of writing.

kWh when On	0.15
kWh when Sleeping	0.005
kWh Tariff 2015	€0.22
Computers in Bernoulliborg (BB)	500

Table 30: Constants used in Energy & Economic calculations

Table 30 shows an overview of the constants which shall be used in the calculations for the energy and economic savings. The number of computers in the Bernoulliborg is estimated to be around 500. This estimation is based on the fact that there is room for 350 employees in the Bernoulliborg [34], where we assume at least one computer per employee. We also have to include the number of computers for

students, this number is estimated to be around 150. This is based on the fact that there are six large computer rooms on the second floor with around 25 computers each.

Model	*Total Sleep*	*Sleep per Computer*	*Sleep per PC per Month*
Model 1	5.55 hours	0.46 hours	2.31 hours
Model 2	22.26 hours	1.85 hours	9.27 hours
Model 3	23.68 hours	1.97 hours	9.87 hours

Table 31: Sleep per month

The amount of sleep per computer per month is shown in table 31. The difference between model 1 and the other models is significant: model 1 results in only 2.31 hours of sleep whereas both other models result in 9 to 10 hours of sleep. The monthly hours of sleep were calculated by multiplying the sleep per computer times 5. This is because the hours of sleep per computer is taken over a 6 day period. It is important to know the number of hours a computer is asleep in order to determine the monthly kWh savings per model.

Model	*Sleep Consumption*	*On Consumption*	*Monthly Savings*
Model 1	0.012 kW	0.35 kW	0.33 kW
Model 2	0.046 kW	1.39 kW	1.35 kW
Model 3	0.049 kW	1.48 kW	1.43 kW

Table 32: Savings in kWh per month per computer

To determine the monthly energy savings we have to determine two things: the number of kilowatts consumed per computer while it was sleeping and the number of kilowatts (kW) the computer would have consumed if it would have been turned on instead of asleep. These numbers can be seen in table 32. For example, a single computer using model 1 would have consumed 0.012 kW per month while it was sleeping. If the computer was turned on instead of in sleep mode it would have consumed 0.35 kW per month. Therefore 0.33 kW are saved per computer per month using model 1.

Model	*Savings per Computer (kWh)*	*Savings per Computer (euro)*
Model 1	4.02 kW	€0.89
Model 2	16.13 kW	€3.55
Model 3	17.17 kW	€3.77

Table 33: Annual energy & economic savings per computer

Over the course of a year the savings per computer add up to a significant number of kilowatts as can be seen in table 33. Even using model 1 saves around 4 kW per computer per year. This equals to around 90 eurocents worth of energy savings per computer per year.

Using model 2 or 3 results in 16 to 17 kilowatts of energy savings, or 3.55 euro to 3.77 euro of economic savings.

Finally, let us take a look at the annual savings that could be achieved in an actual office building. In this example we shall use the Bernoulliborg to demonstrate the energy and economic savings. We assume there are 500 computers in the Bernoulliborg.

Model	*Annual Savings (kWh)*	*Annual Savings (euro)*
Model 1	2010.68 kW	€442.35
Model 2	8067.44 kW	€1774.84
Model 3	8583.49 kW	€1888.37

Table 34: Annual energy & economic savings in the Bernoulliborg

The results for the Bernoulliborg can be seen in table 34. If all computers in the Bernoulliborg used model 1 then the savings would be 2010.68 kW, which amounts to 442.35 euro. The biggest savings can be achieved using model 3: the savings are 8583.49 kW or 1888.37 euro. It is interesting to note that this is enough energy saved to power an average sized Dutch household for almost two and a half years [22].

6 FUTURE WORK

Due to the time-bound nature of a master project there are still areas in which further work can be performed. The following areas have been identified:

Cross Platform Support: To be able to cover all kinds of computers there should be support of other operating systems. Currently the only supported operating system is Windows, from Windows XP and onward. The reason Windows is the first operating system to be supported is because in the Bernoulliborg the majority of computers are Windows based. The next step would be to support Linux based computers, of which there are also quite a few in the Bernoulliborg.

Two attempts were made to implement a Linux version, one by the author of this thesis and one by Bachelor students as part of their own thesis. Neither solution is complete but they have proven that it is possible to support Linux based operating systems. Once Linux support has been realized the next candidate would be Apple's OSX. There have been no attempts yet to support OSX.

User Survey: In order to learn even more about user comfort levels a survey should be held among the participants of the experiments. At the time of writing this survey is being performed, however due to the summer holidays the number of responses is too low to be included as part of this thesis. In the survey the following questions have been asked:

- I turn off the computer after leaving the office
- Enabling sleep mode has made me more aware of how I use my computer
- Enabling sleep mode has made me put the computer to sleep more often
- Enabling sleep mode has made me turn the computer off more often
- On some occasions the computer entered sleep mode while I was actively using the computer
- Having to wake up my computer from sleep mode disrupts my work flow

Multiple Data Sources: Currently the solution only uses one source of information to determine the probability of seeing activity at a given moment in time, namely the historical activity data. It would be interesting to see how extra data sources would influence the system.

For example, rules could be added based on common working hours which increase the base probability of seeing activity. Between 9:00 AM and 5:00 PM on working days it would be more likely to see activity on computers in most offices, this could be added as a rule. An even better data source would be agendas or calendars: this would especially be useful for classrooms and to take holidays into account.

Reinforcement Learning: Instead of using linear regressions models one can also look at other methods of modeling a personalized sleep timeout. A good candidate would be to explore the possibilities that reinforcement learning offers. Possible actions of a basic reinforcement learning model could be increasing or decreasing the sleep timeout. Where short term rewards are the expected energy savings directly associated with using a specific sleep timeout. The long term rewards could balance the user comfort and the energy savings, as the effect on the user comfort is only noticeable after a certain amount of time has passed.

7 CONCLUSION

In this thesis we have looked at how context aware power management based on user behavior can be implemented and used in practice using a software solution. We have also looked at several different models and the savings which were achieved by each of these models by performing experiments over the course of a month. Let us look at the research questions posed in chapter 1. The first research question posed is as follows:

> *How can an optimal sleep timeout setting be found for individual users from data available in the dataset of state, activity and feedback information of personal computers?*

We can answer this research question based on our findings from chapter 3 and chapter 4. The Sleepy software solution allows us to collect all the required datasets of state, activity and feedback information. It also allows us to manipulate the sleep timeout remotely.

We defined three models to be tested. The three different models each have a different approach to determine the personalized sleep timeout for a computer. Model 1 was designed to maximize user comfort by making sure that the computer is not put to sleep as long as there is some predicted activity. Whereas model 3 tries to be more aggressive by also taking a look at how long the computer has been idle. The longer the computer has been idle, the more aggressively the sleep timeout is set. Model 2 is positioned somewhere between model 1 and model 3.

Each model was trained using the linear least squares method with stochastic gradient descent. The input parameters for these models are provided by the previously mentioned datasets. The Sleepy software is used to collect the required datasets in real time. By using an error function to determine the ranking of each model we were able to determine which model is more optimal according to this error function.

The second research question concerns the results of using the models in practice, it is formulated as follows:

> *Does the solution used for sleep timeout optimization result in a reduction of the energy consumption of personal computers, while at the same time keeping the user comfort at high levels?*

This question can be answered by looking at chapter 5, in which the results of the experiments are presented and discussed. The result from the experiments show that the energy consumption of computers can indeed be reduced when using power management based on user behavior. The amount of energy which is saved depends on the model which is used. Models which try to set the sleep timeout

more aggressively save more energy, but this is at the expense of user comfort. However, during the experiments even the most aggressive model did not generate a large amount of negative feedback. Therefore it can be concluded that the solution for sleep timeout optimization indeed reduces the energy consumption of personal computers and does not have a significant impact on the user comfort levels.

The level of savings which can be achieved can be quite significant. Especially in office buildings with large amounts of computers. If the solution would have been deployed on all computer in the Bernoulliborg building of the University of Groningen the savings would range between 442.35 euro and 1888.37 euro, depending on the chosen model. And unlike many other sustainable projects and initiatives this solution would not require any modifications to be made to the building. It only requires a light weight software application to be installed on every computer. Therefore the deployment costs and operational costs would be very low.

It is probable that even higher savings could be seen, especially if the solution would have been deployed to student workplaces instead of the computers of employees. The reason for this is because students are more likely to leave the computers turned on. Whereas the employees that participated turned their computer off after leaving the office almost every time. It would have also been interesting to see how the solution handles these public student computers, as they are used by many different students and each of them has their own behavior. Furthermore, it also seems plausible that the solution would have learned the classroom schedules of these student workplaces, putting computers to sleep when there are no lectures being given in the room.

We can conclude that it is indeed possible to find a personalized sleep timeout setting using the state, activity and feedback datasets. And we can also conclude that the personalized sleep timeout does reduce the energy consumption of computers. The level of user comfort depends on the model which is used. But even when using aggressive models the comfort levels do not drop drastically while the energy savings do increase significantly. We hope the work done in this thesis paves the way for future research on this topic.

Part II

APPENDIX

A

GLOBAL ARCHITECTURE

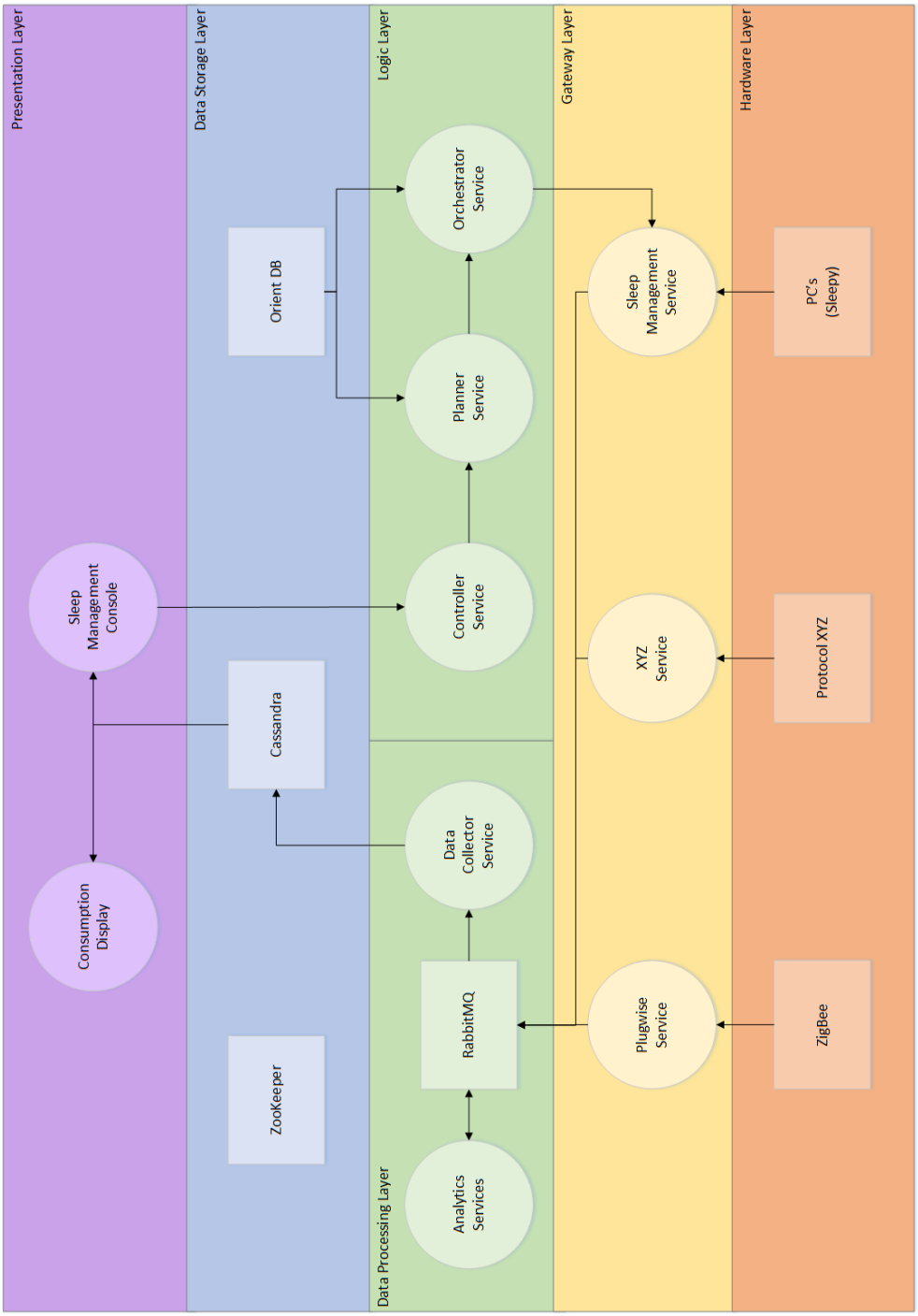


Figure 34: Global Architecture

B

ACTIVITY PROFILES

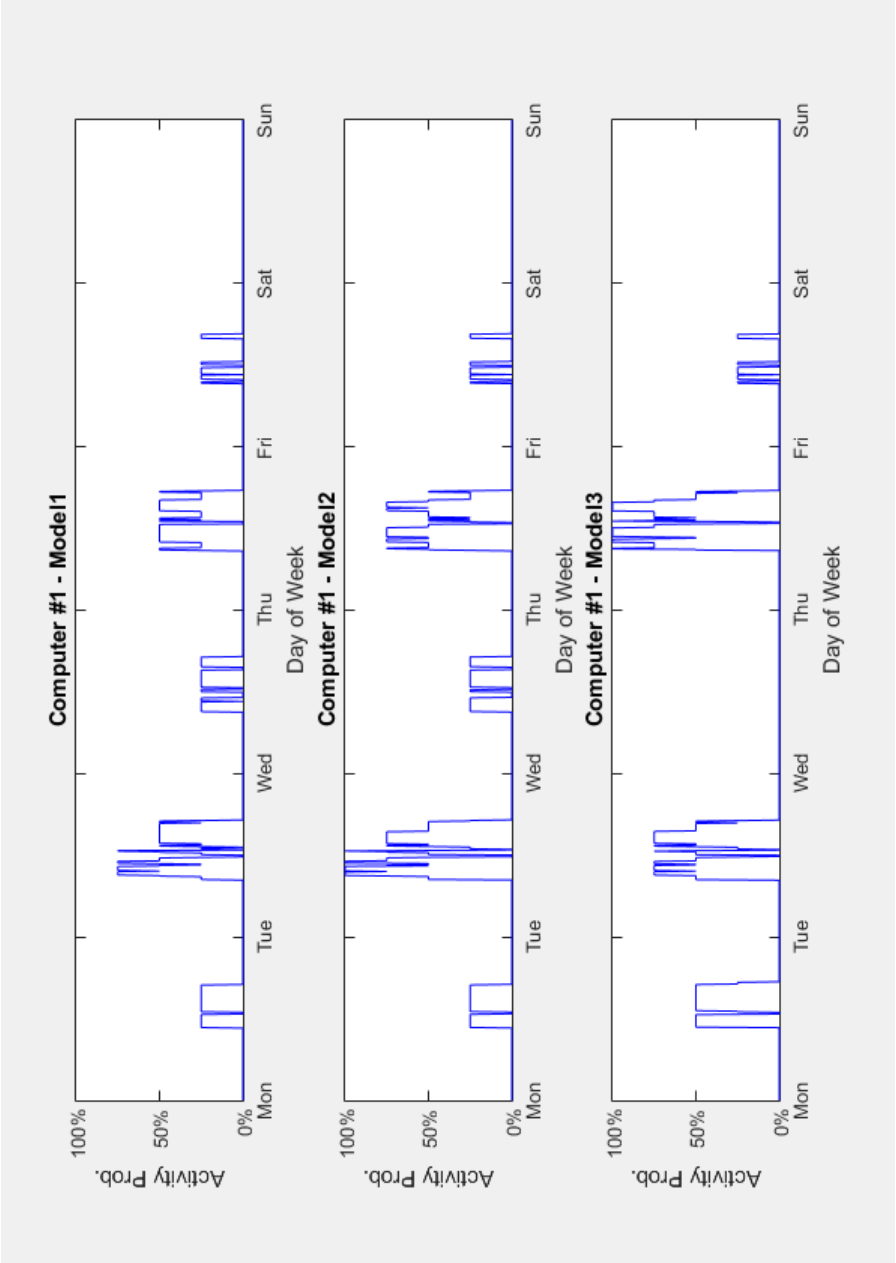


Figure 35: Computer 1 - Activity Probability per Model

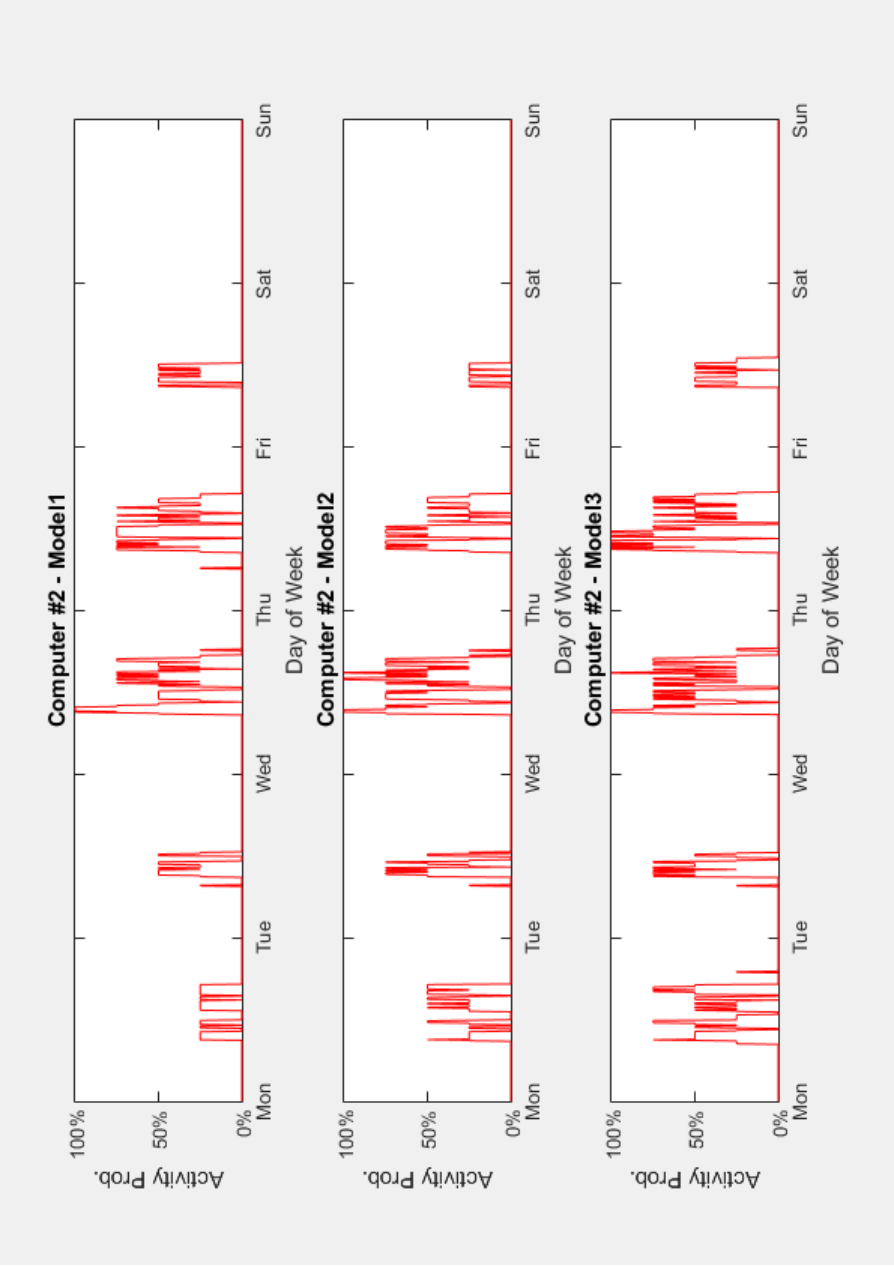


Figure 36: Computer 2 - Activity Probability per Model

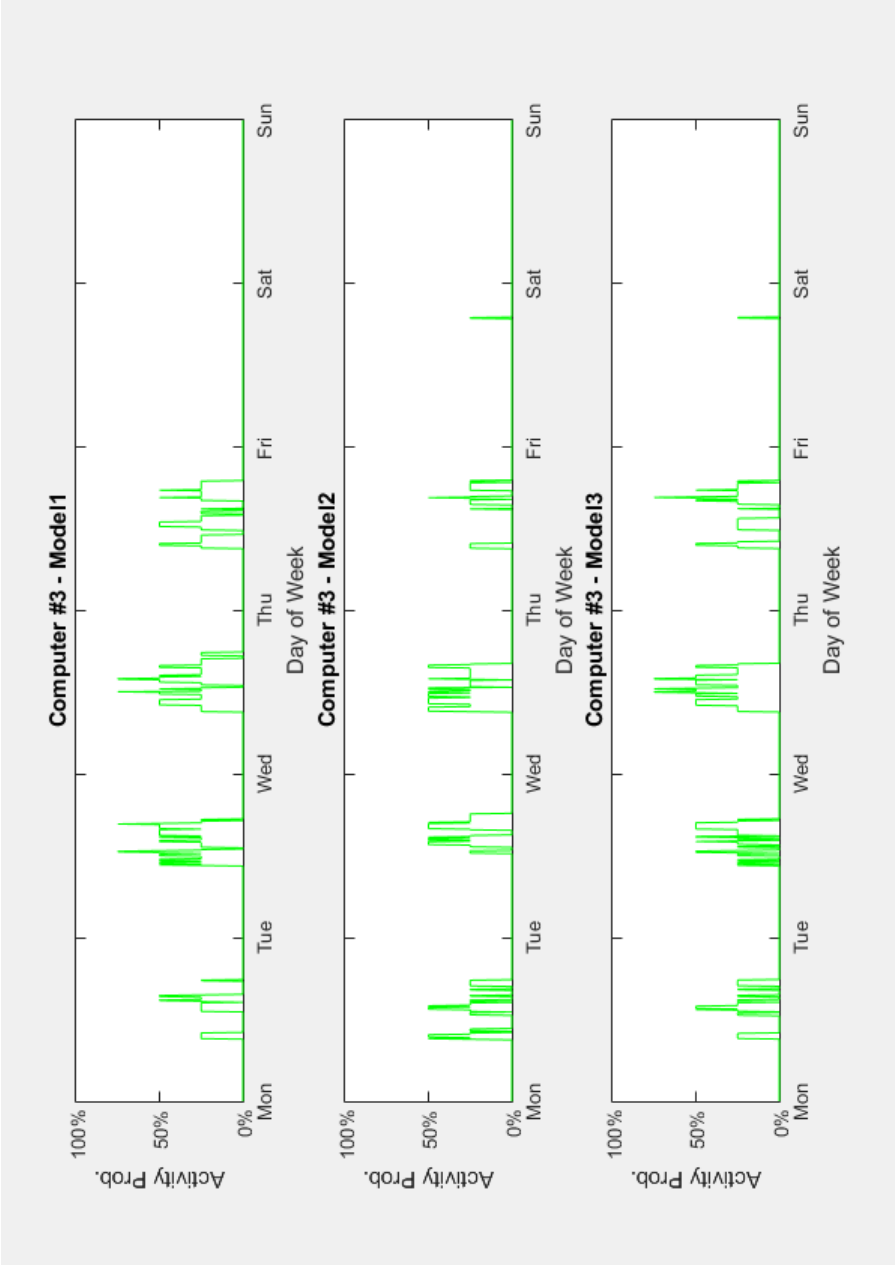


Figure 37: Computer 3 - Activity Probability per Model

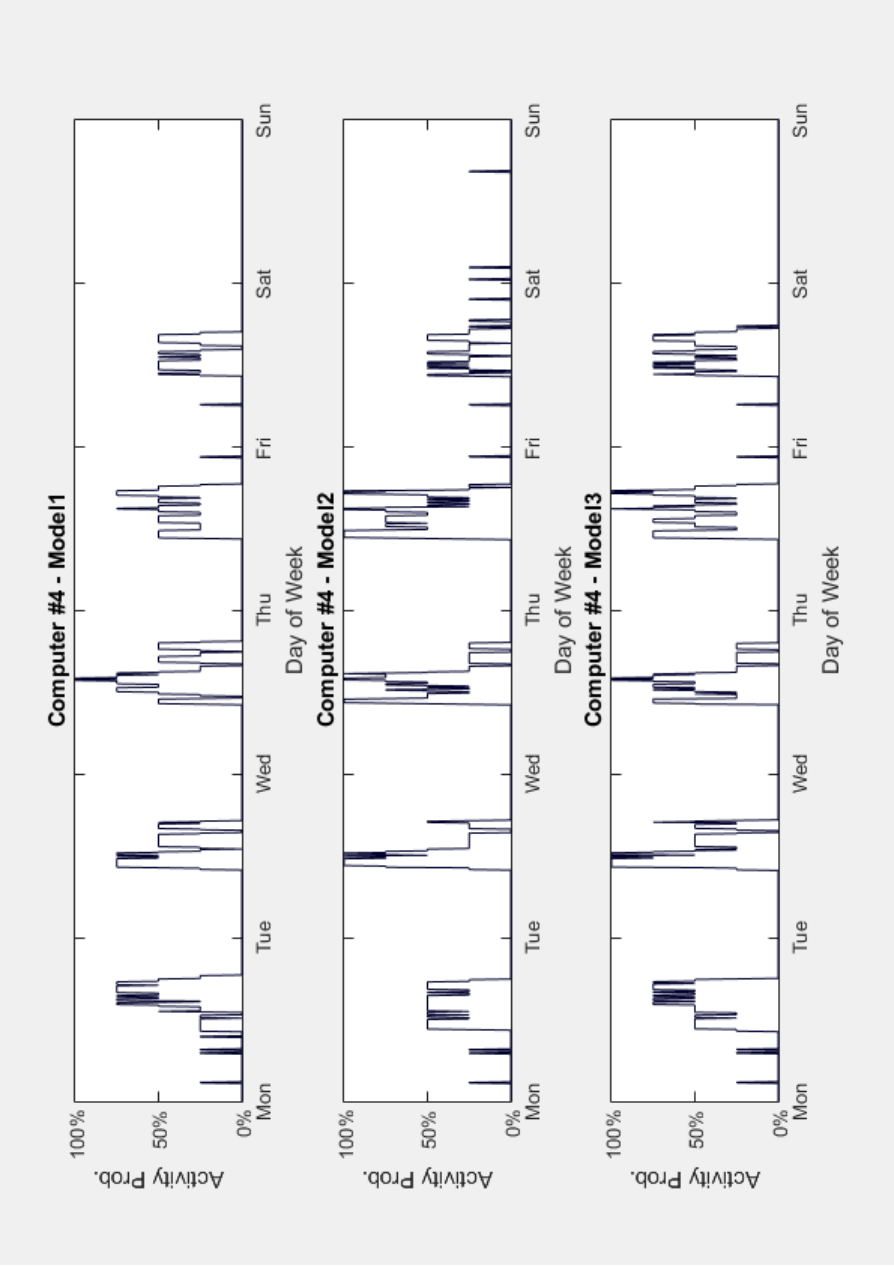


Figure 38: Computer 4 - Activity Probability per Model

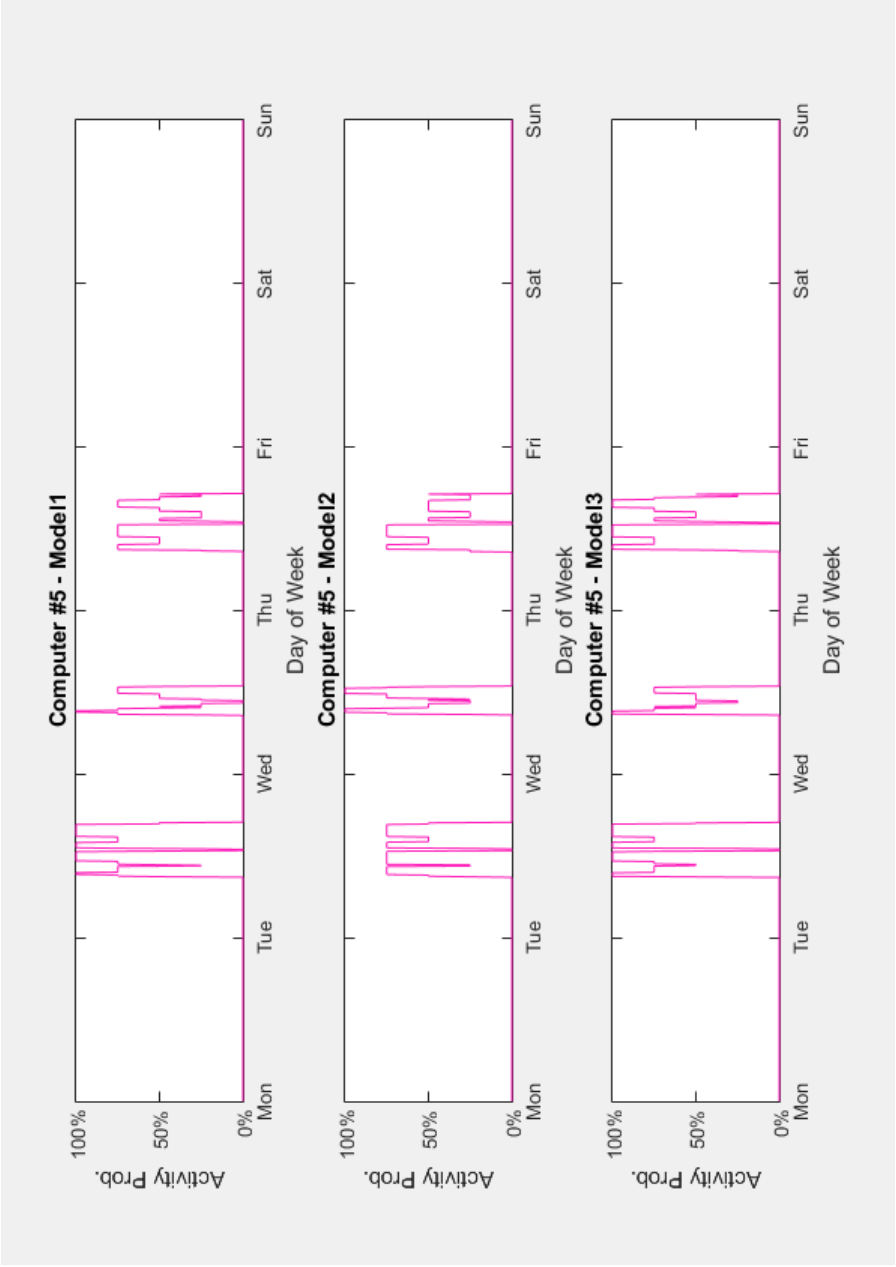


Figure 39: Computer 5 - Activity Probability per Model

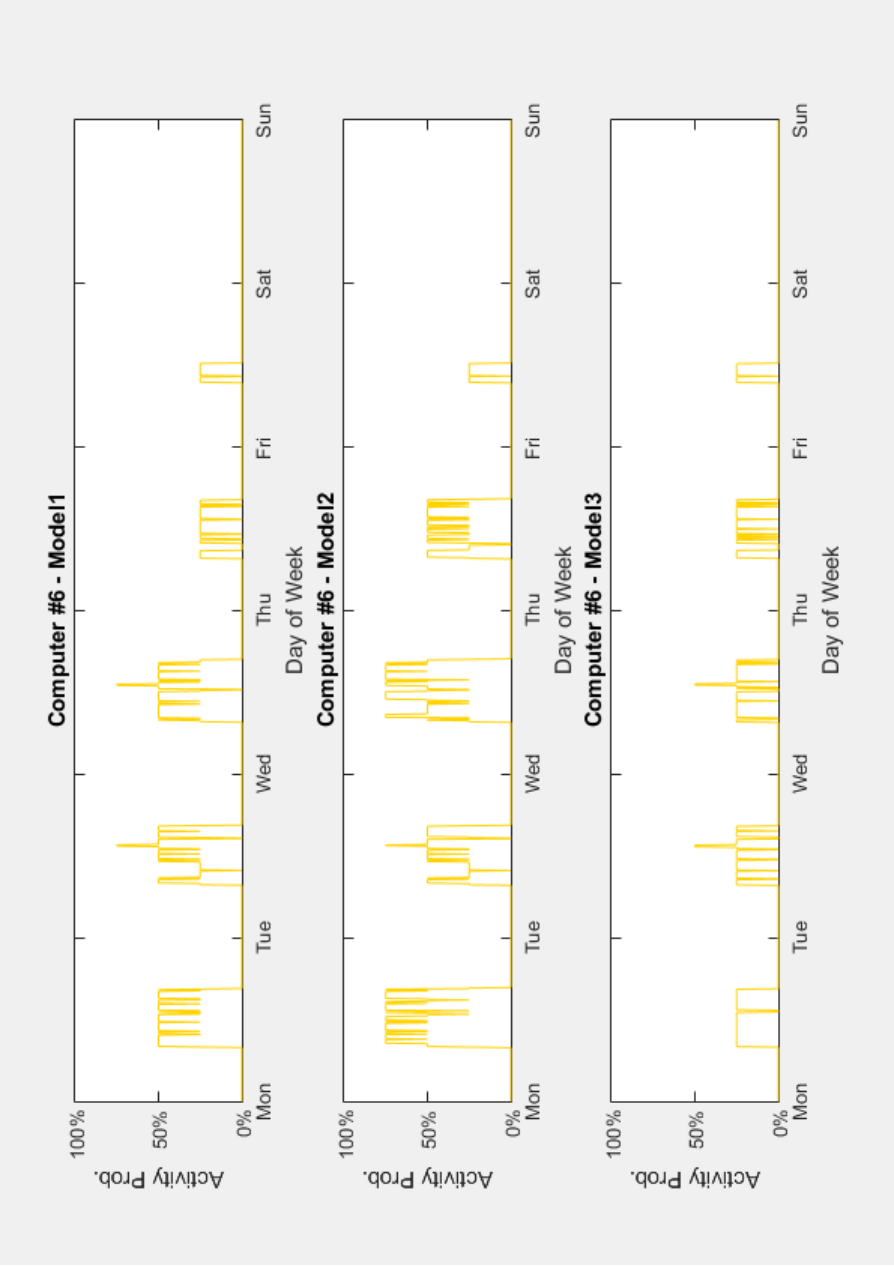


Figure 40: Computer 6 - Activity Probability per Model

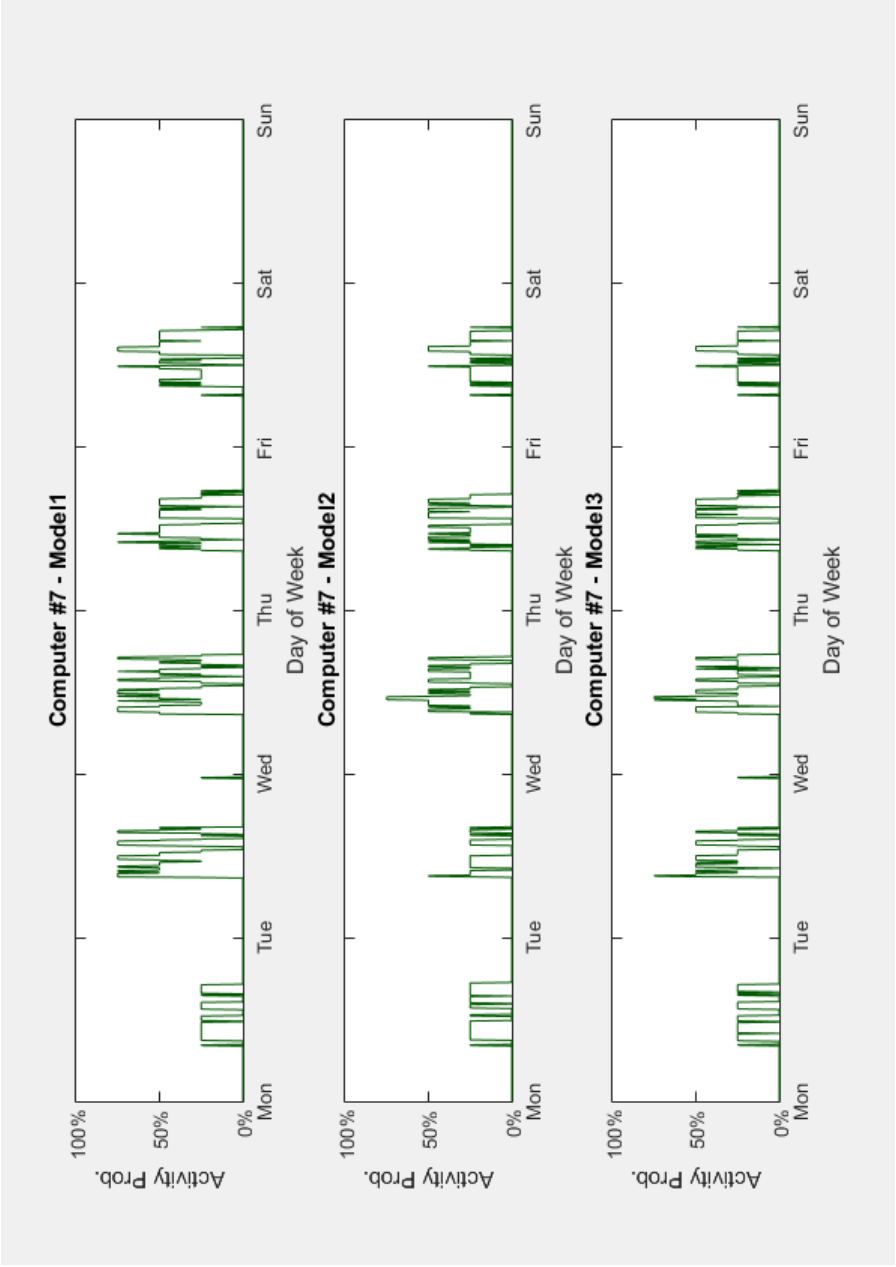


Figure 41: Computer 7 - Activity Probability per Model

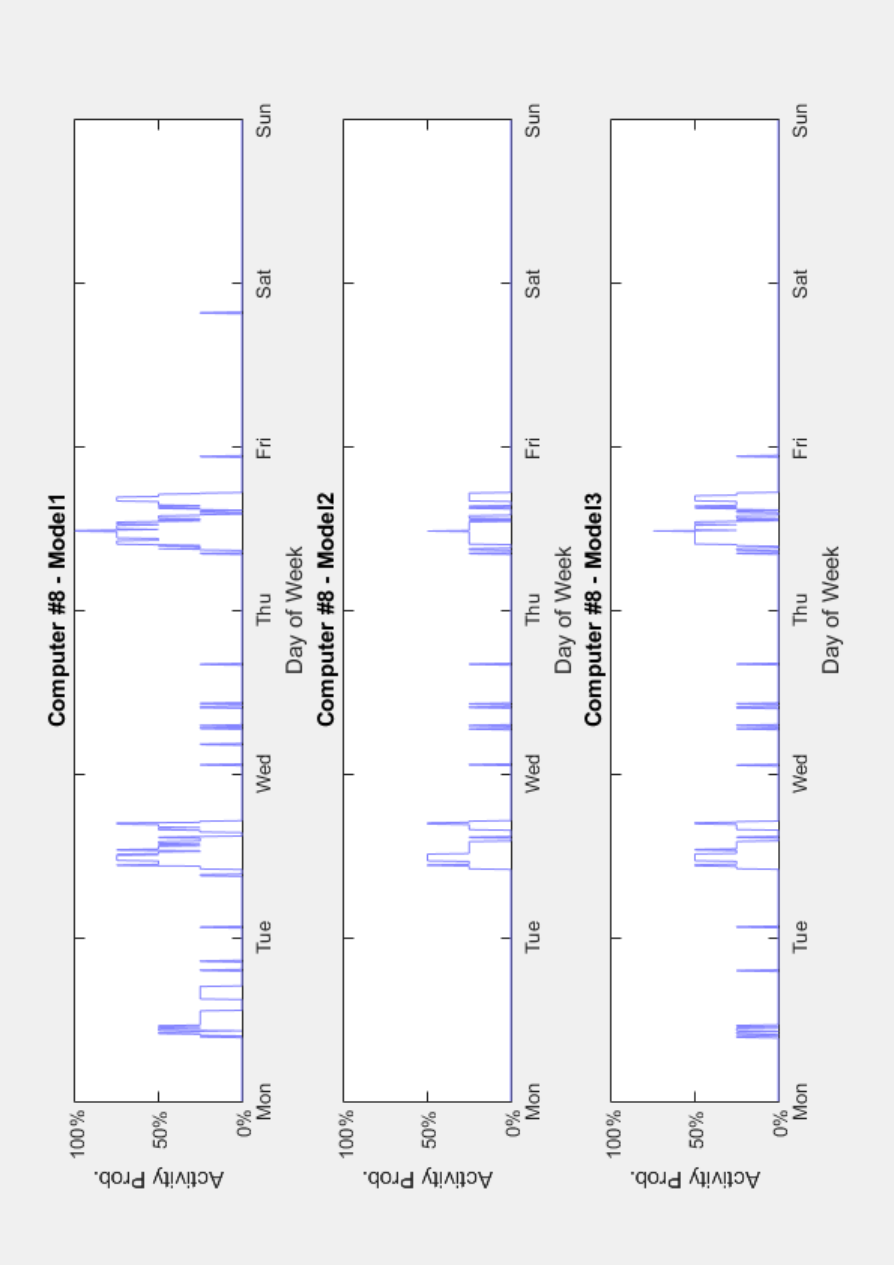


Figure 42: Computer 8 - Activity Probability per Model

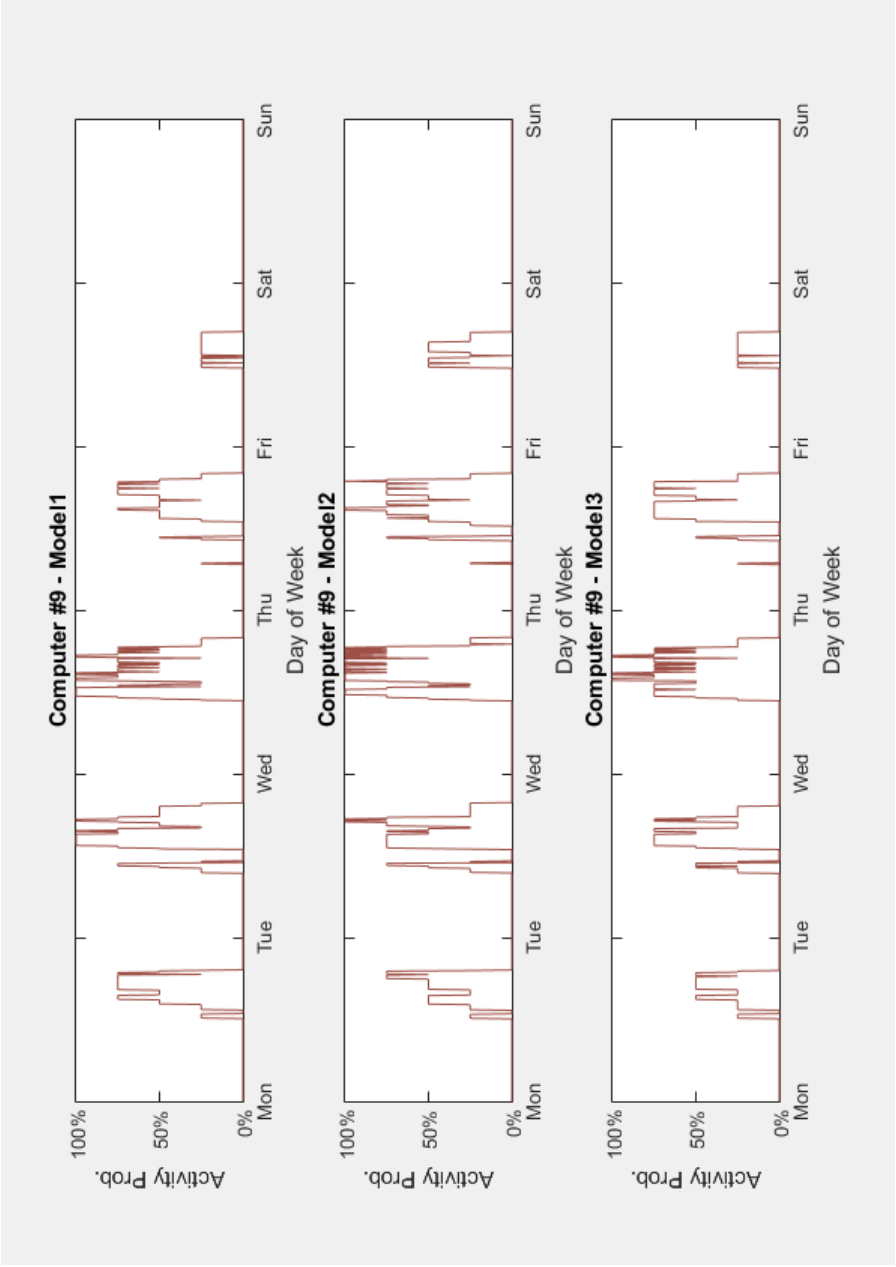


Figure 43: Computer 9 - Activity Probability per Model

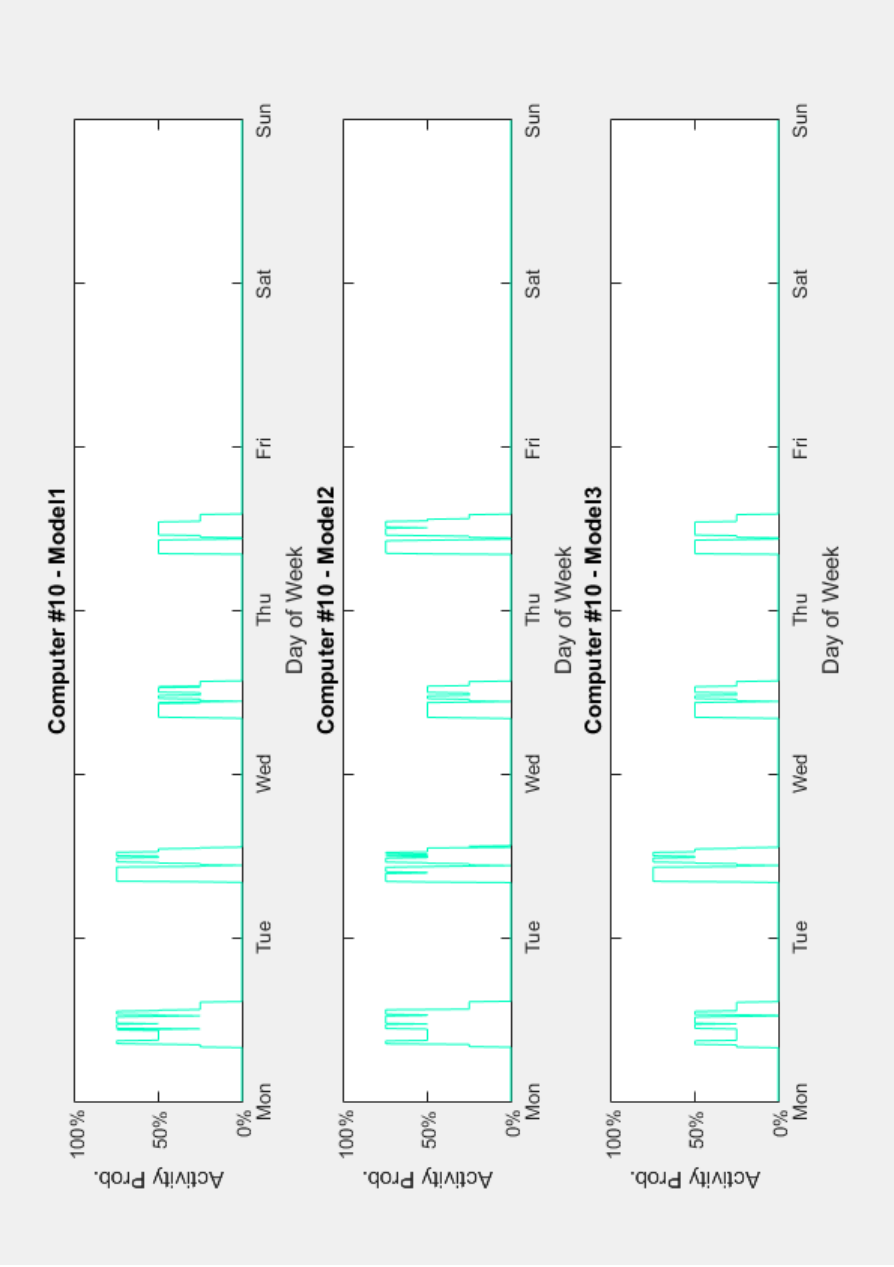


Figure 44: Computer 10 - Activity Probability per Model

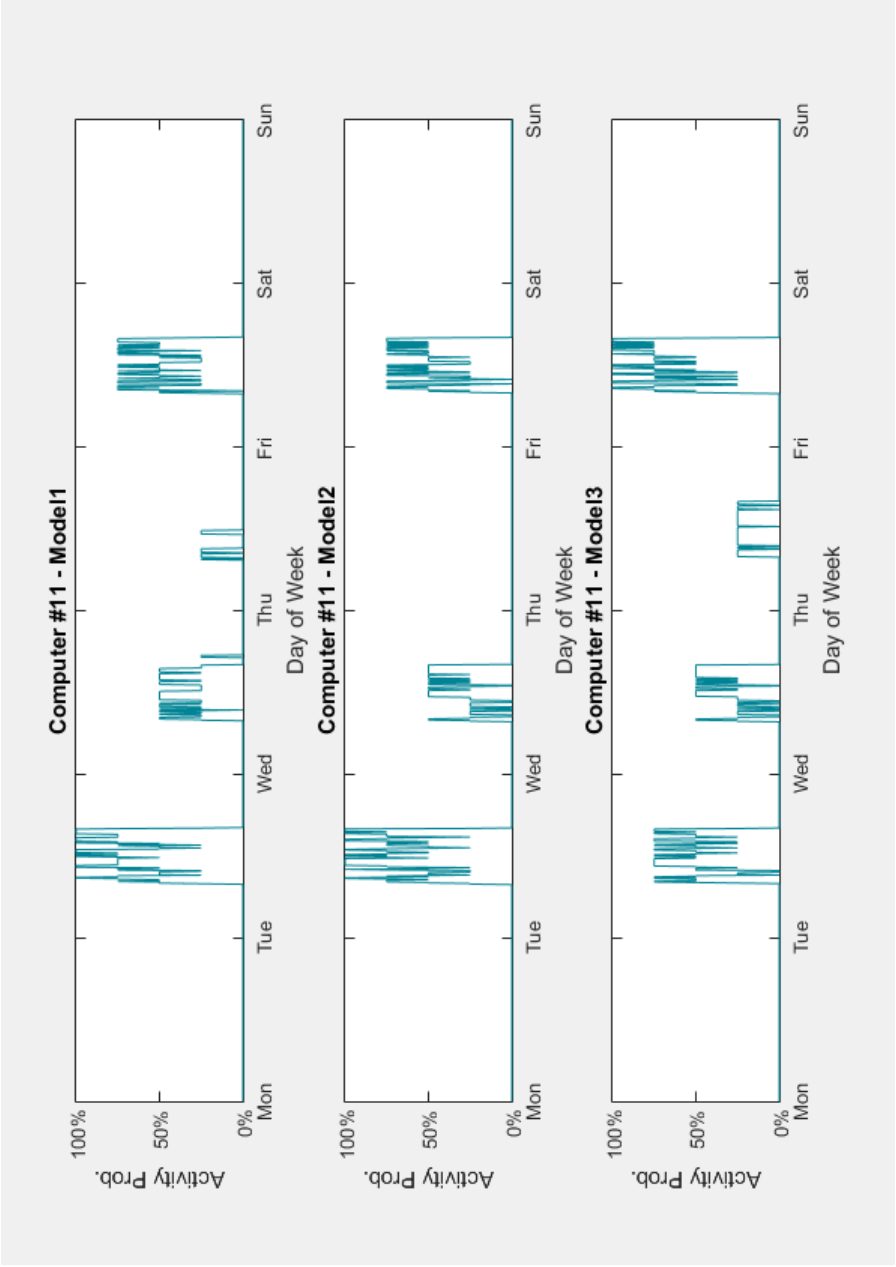


Figure 45: Computer 11 - Activity Probability per Model

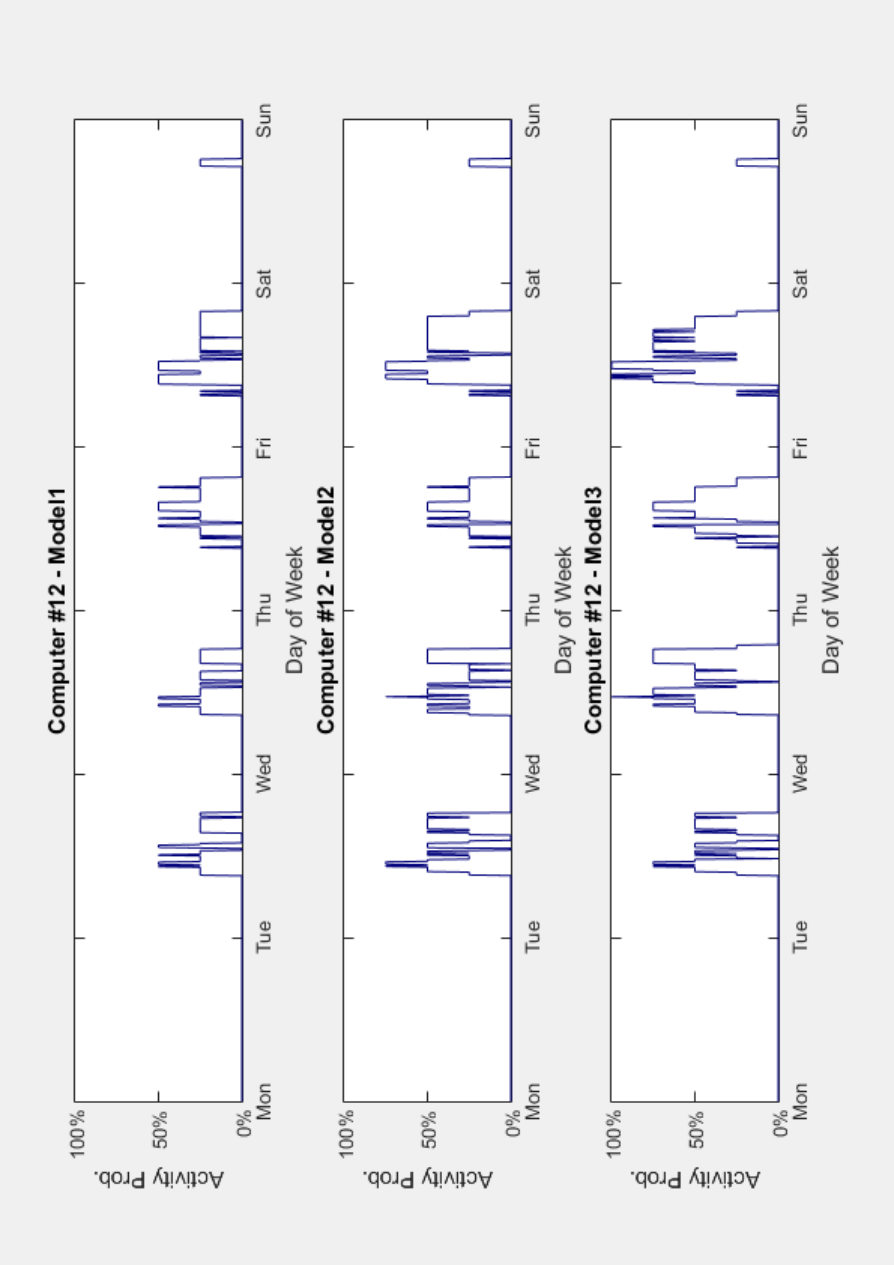


Figure 46: Computer 12 - Activity Probability per Model

C

ACTIVITY DATA

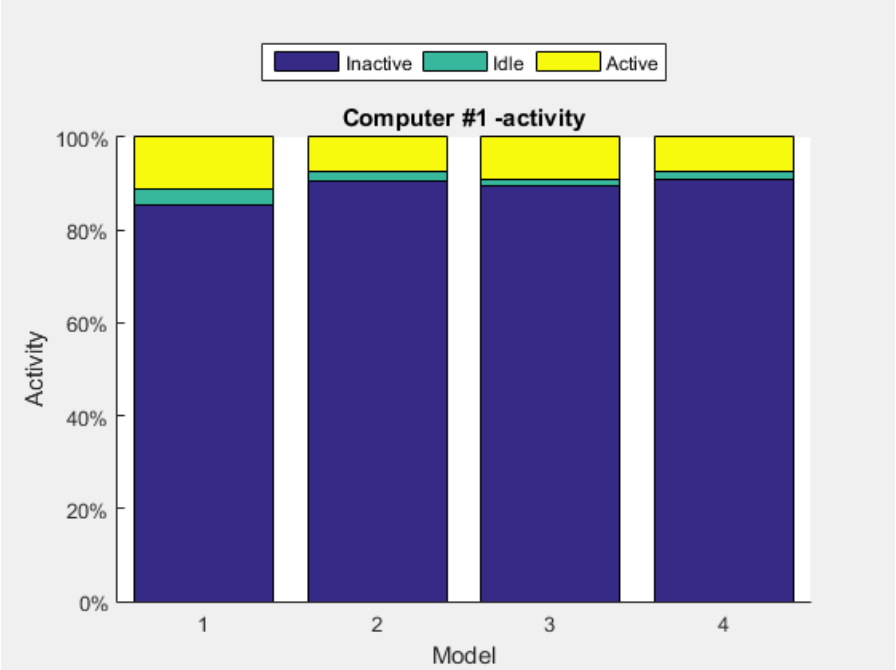


Figure 47: Computer 1 - Activity Data per Model

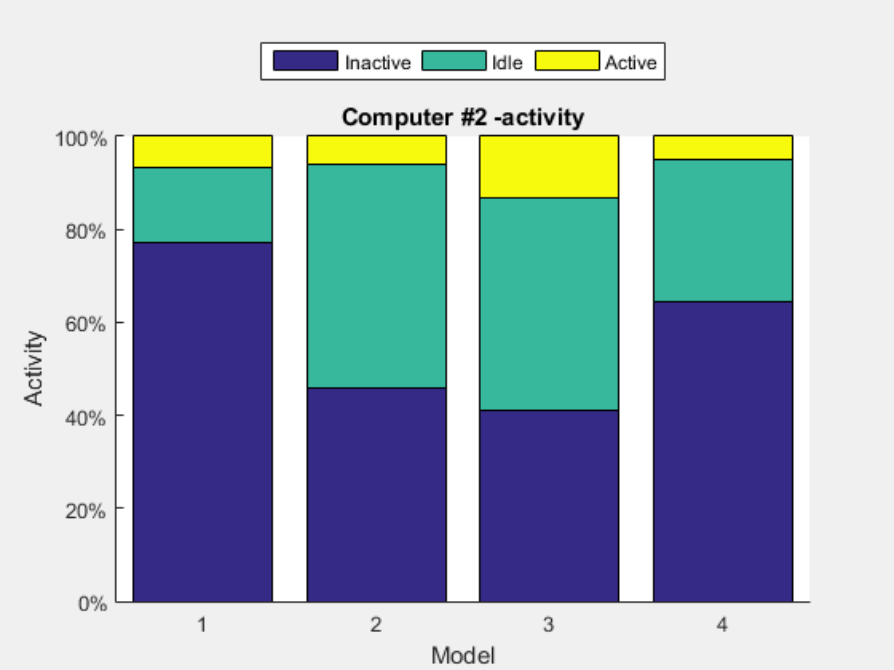


Figure 48: Computer 2 - Activity Data per Model

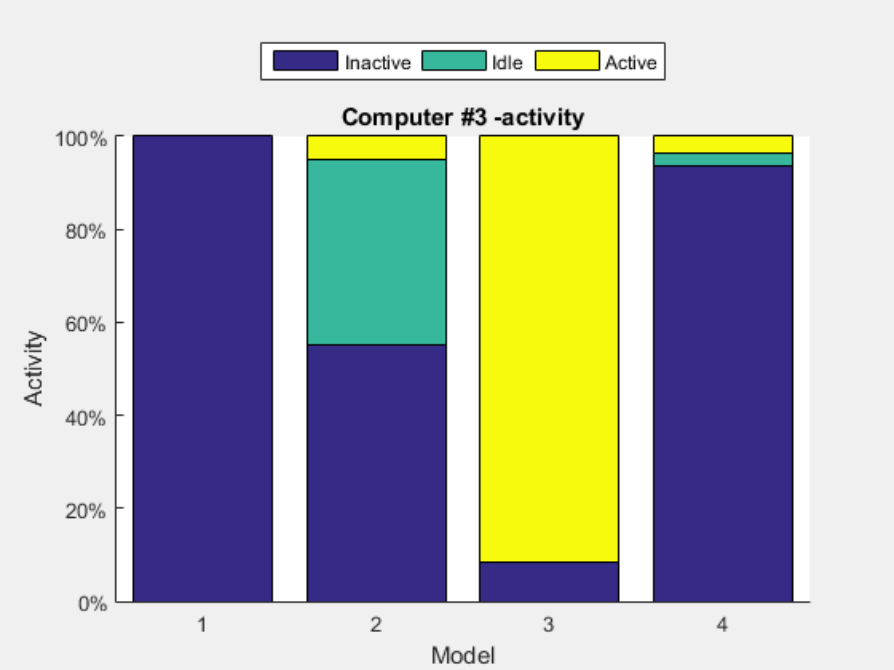


Figure 49: Computer 3 - Activity Data per Model

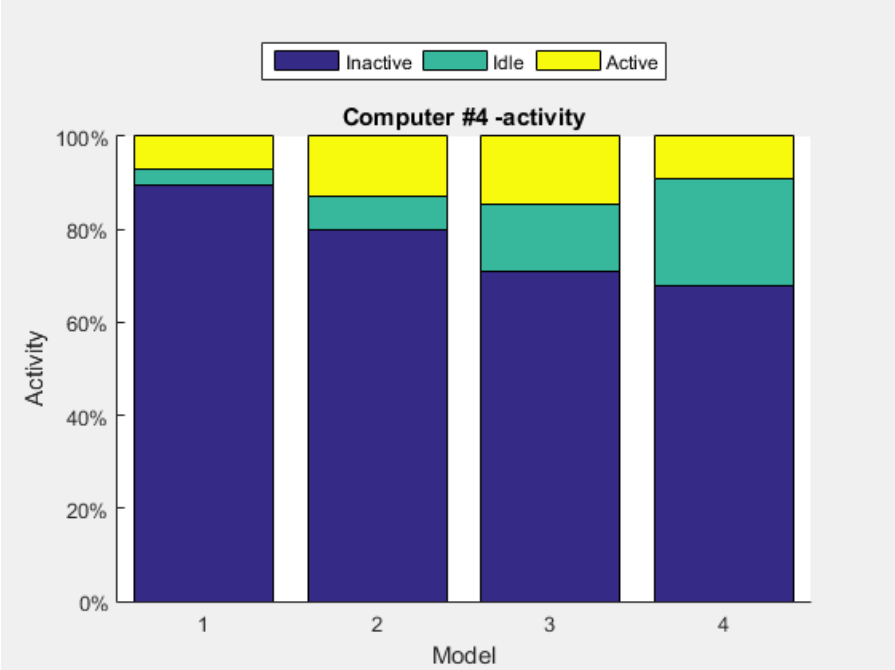


Figure 50: Computer 4 - Activity Data per Model

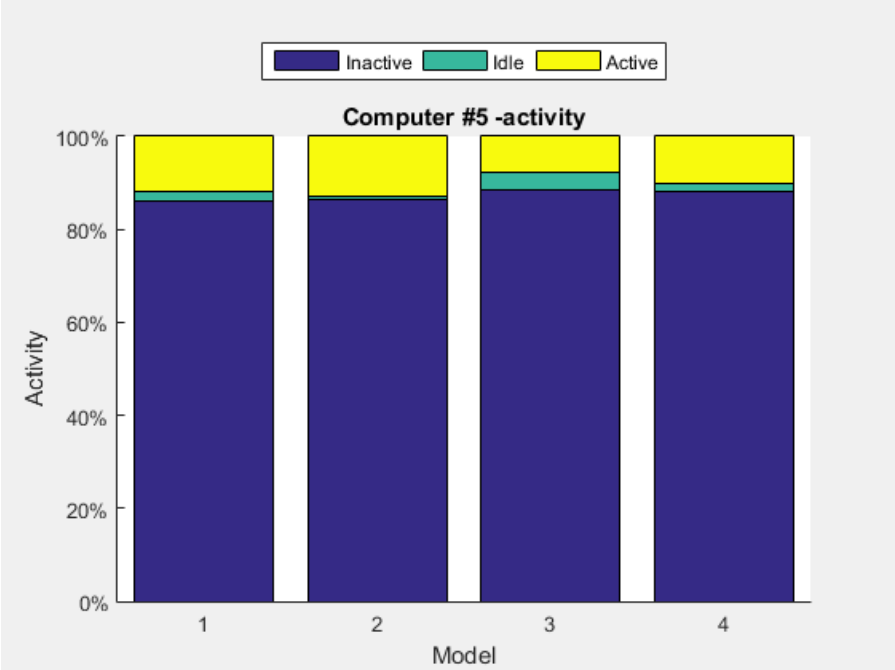


Figure 51: Computer 5 - Activity Data per Model

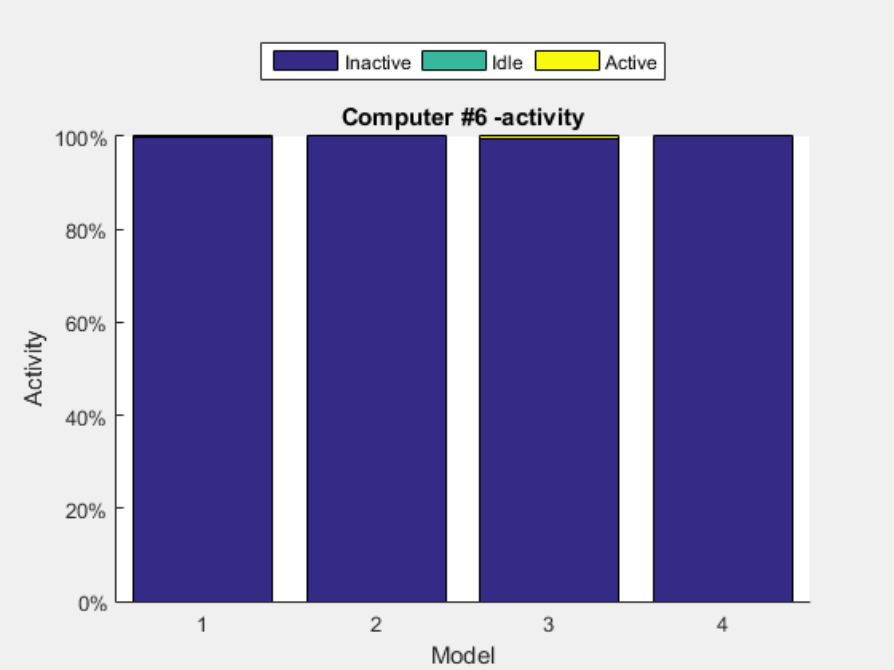


Figure 52: Computer 6 - Activity Data per Model

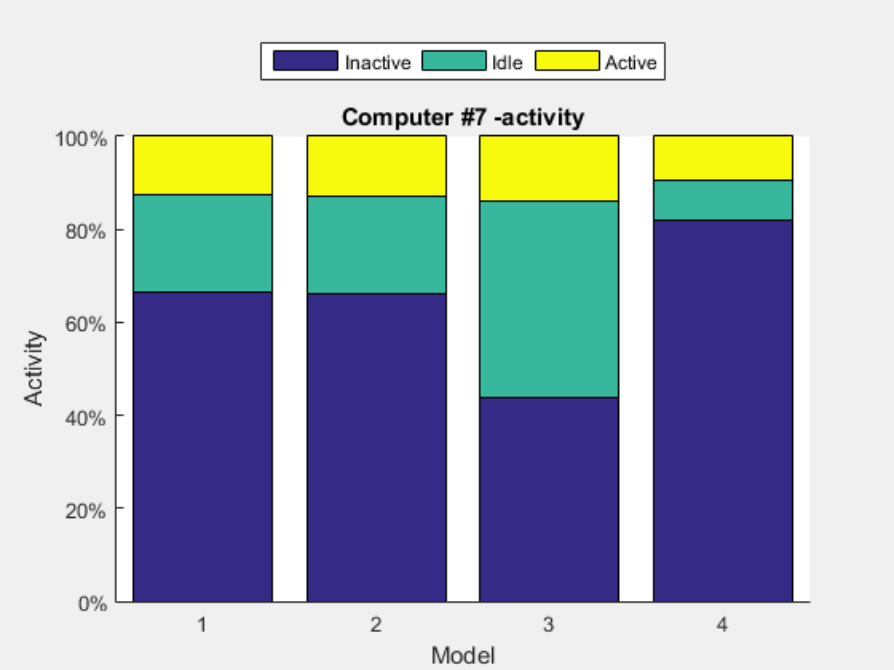


Figure 53: Computer 7 - Activity Data per Model

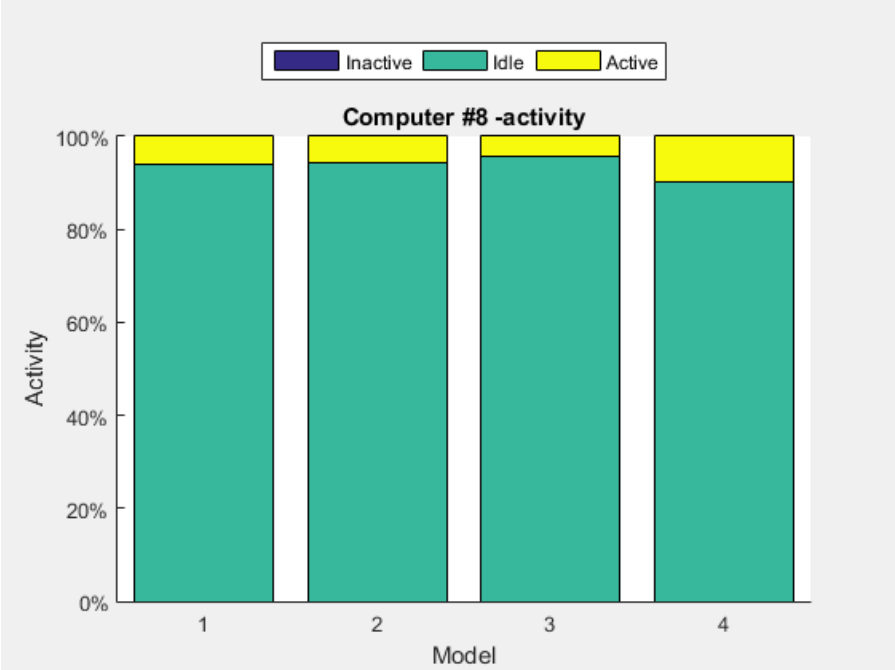


Figure 54: Computer 8 - Activity Data per Model

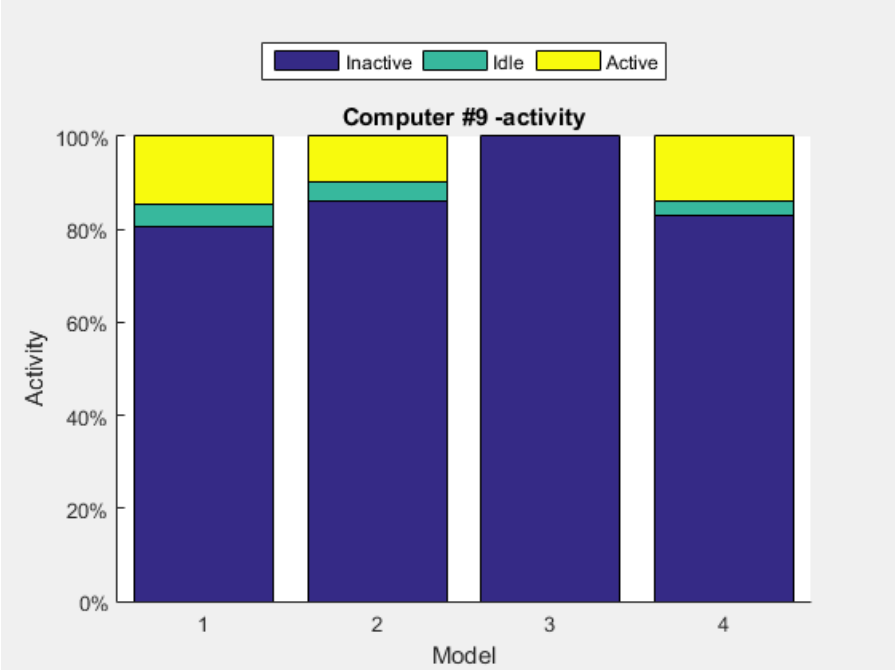


Figure 55: Computer 9 - Activity Data per Model

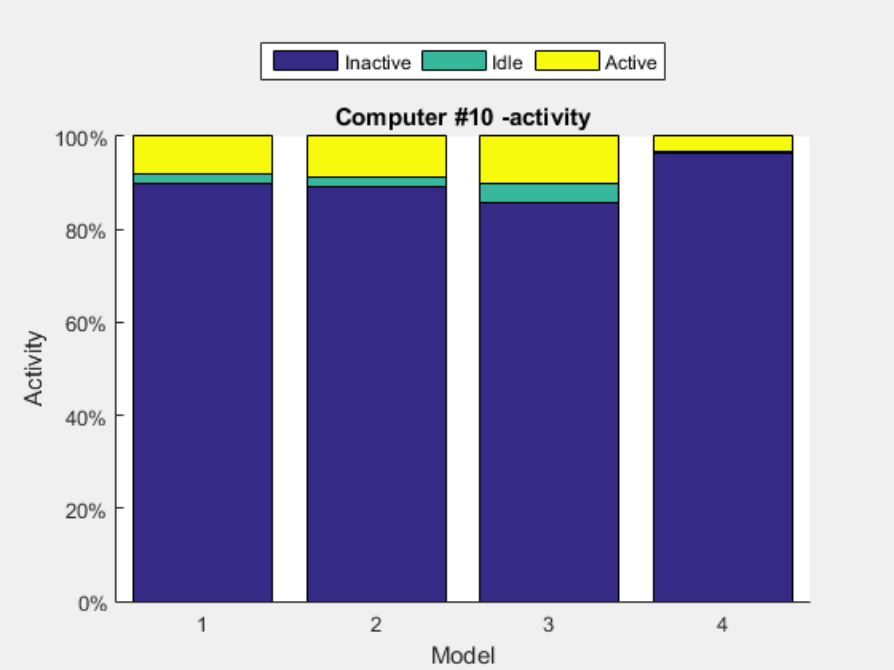


Figure 56: Computer 10 - Activity Data per Model

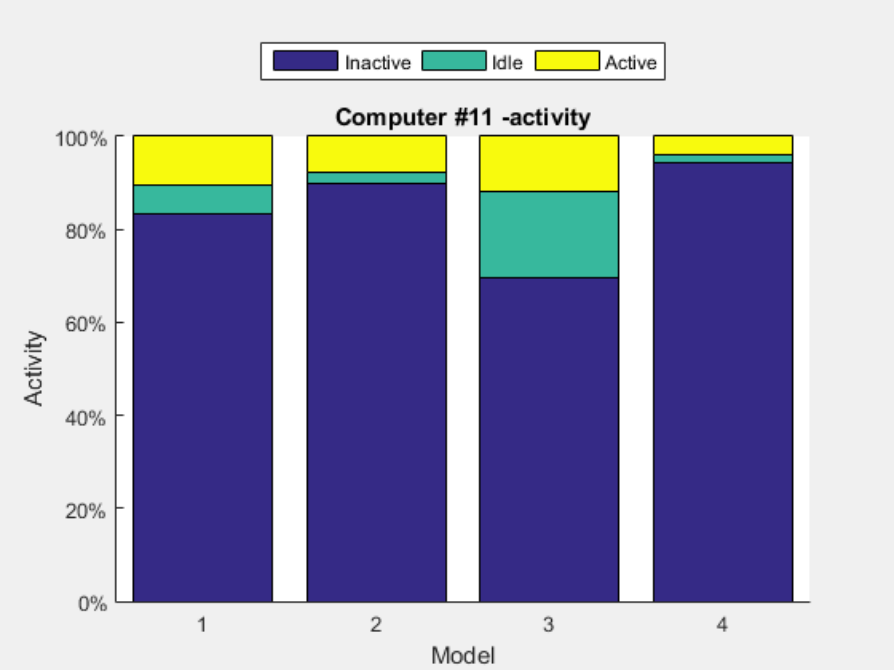


Figure 57: Computer 11 - Activity Data per Model

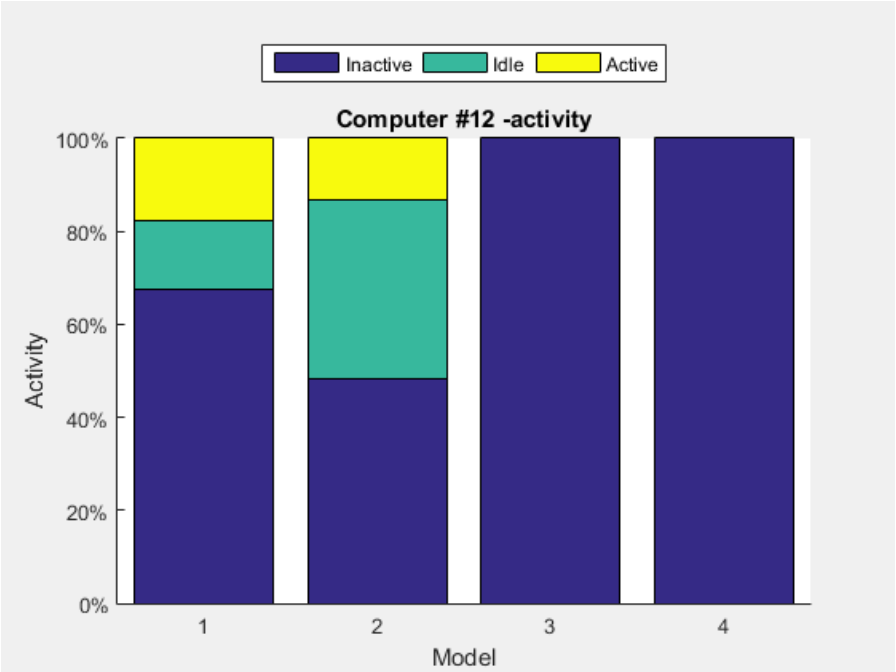


Figure 58: Computer 12 - Activity Data per Model

BIBLIOGRAPHY

[1] Smart2020: Enabling the low carbon economy in the information age. *The Climate Group - Global eSustainability Initiative*, 2008. URL http://www.smart2020.org/_assets/files/02_smart2020Report.pdf.

[2] M. Aggar, P. Stemen, and M. Walsh. Sustainable computing, conserving energy with group policy. *Microsoft TechNet magazine*, May 2008. URL https://technet.microsoft.com/en-us/magazine/cc462804.aspx.

[3] Luca Benini, Alessandro Bogliolo, Giuseppe A. Paleologo, and Giovanni De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, Volume 18, Issue 6*, June 1999. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=724463&isnumber=15604.

[4] Michael Bluejay. How much electricity do computers use?, March 2012. URL http://michaelbluejay.com/electricity/computers.html.

[5] Simone Brienza, Fabio Bindi, and Giuseppe Anastasi. E-netmanager: A power management system for networked pcs based on soft sensors. *International Conference on Smart Computing (SMARTCOMP)*, November 2014. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7043846.

[6] C. Chang, M. M. Drugan, P. Verhaegen, A. Nowe, and J. R. Duflou. Finding days-of-week representation for intelligent machine usage profiling. *International Conference on Future Information Technology*, 2013. URL https://ai.vub.ac.be/sites/default/files/Finding_Days-of-week_Representation_for_Intelligent_Machine_Usage_Profiling_final.pdf.

[7] Y. Chen, Y. Ko, and W. Peng. An intelligent system for mining usage patterns from appliance data in smart home environment. *Technologies and Applications of Artificial Intelligence*, November 2012. URL http://dx.doi.org/10.1109/TAAI.2012.54.

[8] S. Cheshire. Method and apparatus for implementing a sleep proxy for services on a network. *United States Patent N. 7,330,986*, February 2008. URL http://www.google.com/patents/US20060253720.

[9] L Chiaraviglio and M. Mellia. Polisave: Efficient power management of campus pcs. *Telecommunications and Computer Networks (SoftCOM), 2010 International Conference on Software*, September 2010. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5623620.

[10] K. J. Christensen and F. Gulledge. Enabling power management for network-attached computers. *International Journal of Network Management Volume 8, Issue 2*, April 1998. URL http://www.csee.usf.edu/~christen/energy/ijnm98.pdf.

[11] Victor Ciriza, Laurent Donini, Jean-Baptiste Durand, and Stephane Girard. Optimal timeouts for power management under renewal or hidden markov processes for requests. 2009. URL https://hal.archives-ouvertes.fr/hal-00412509v2.

[12] Jean-Baptiste Durand, Stephane Girard, Victor Ciriza, and Laurent Donini. Optimization of power consumption and user impact based on point process modeling of the request sequence. *Journal of the Royal Statistical Society Series C*, pages 151–165, 2013. URL https://hal.archives-ouvertes.fr/hal-00412509v4.

[13] I. Georgievski and A Lazovik. Utility-based htn planning. *21st European Conference on Artificial Intelligence*, 2014.

[14] R. K. Harle and A. Hopper. The potential for location-aware power management. *Proceedings of the 10th international conference on Ubiquitous computing*, September 2008. URL http://www.cl.cam.ac.uk/research/dtg/www/publications/public/rkh23/ubicomp202-harle.pdf.

[15] Colin Harris and Vinny Cahill. Power management for stationary machines in a pervasive computing environment. *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005. URL https://www.scss.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-48.pdf.

[16] Mateusz Jarus and Ariel Oleksiak. Gicomp and greenoffice - monitoring and management platforms for it and home appliances. *Energy Efficiency in Large Scale Distributed Systems*, August 2010. URL http://dl.acm.org/citation.cfm?id=2695310.

[17] The jQuery Foundation. Perform an asynchronous http request., January 2015. URL http://api.jquery.com/jquery.ajax/.

[18] S. Karayi. Pc energy report 2007. 2007. URL http://www.1e.com/energycampaign/downloads/1E%20Energy%20Report%20US.pdf.

[19] R. Khan, R. Bolla, M. Repetto, R. Bruschi, and M. Giribaldi. Smart proxying for reducing network energy consumption. *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, July 2012. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6267032.

[20] Microsoft. Lastinputinfo structure, January 2015. URL https://msdn.microsoft.com/en-us/library/windows/desktop/ms646272(v=vs.85).aspx.

[21] Microsoft. System power management events, January 2015. URL https://msdn.microsoft.com/en-us/library/windows/desktop/aa373223(v=vs.85).aspx.

[22] Mileu Centraal. Gemiddeld energieverbruik, June 2015. URL http://www.milieucentraal.nl/energie-besparen/snel-besparen/grip-op-je-energierekening/gemiddeld-energieverbruik/.

[23] Mileu Centraal. Energieprijzen, July 2015. URL http://www.milieucentraal.nl/energie-besparen/snel-besparen/grip-op-je-energierekening/energieprijzen/.

[24] S. Munir, J. A. Stankovic, C. M. Liang, and S. Lin. Reducing energy waste for computers by human-in-the-loop control. *IEEE Transactions on Emerging Topics in Computing*, 2013. URL http://www.cs.virginia.edu/~stankovic/psfiles/energywaste_ieee-1.pdf.

[25] F. Nizamic, T. A. Nguyen, A. Lazovik, and M. Aiello. Greenmind - an architecture and realization for energy smart buildings. In *International Conference on ICT for Sustainability*, pages 20–29, 2014. URL http://dx.doi.org/10.2991/ict4s-14.2014.3.

[26] Bruce Nordman and Ken Christensen. Greener pcs for the enterprise. *IEEE IT Professional*, July 2009. URL http://www.computer.org/csdl/mags/it/2009/04/mit2009040028-abs.html.

[27] Bruce Nordman and Ken Christensen. Improving the energy efficiency of ethernet-connected devices: A proposal for proxying. *Ethernet Alliance*, October 2007. URL http://www.ethernetalliance.org/wp-content/uploads/2011/10/Proposal_for_Proxying_edit.pdf.

[28] R. Pul and B. Setz. Research internship report. Technical report, University of Groningen, June 2013.

[29] J. Reich, M. Goraczko, A. Kansal, and J. Padhye. Sleepless in seattle no longer. *USENIX Annual Technical Conference*, 2007. URL http://research.microsoft.com/jump/131390.

[30] Rijksdienst voor Ondernemend Nederland. Regelingen: Topsector energie (tse), July 2015. URL http://www.rvo.nl/subsidies-regelingen/regelingen-topsector-energie-tse.

[31] Judy A. Roberson, Gregory K. Homan, Akshay Mahajan, Bruce Nordman, Carrie A. Webber, Richard E. Brown, Marla McWhinney, and Jonathan G. Koomey. Energy use and power levels in new monitors and personal computers. *Environmental Energy Technologies Division*, July 2002. URL http://enduse.lbl.gov/Info/LBNL-48581.pdf.

[32] Solomon S., D. Qin, M. Manning, Z. Chen, M. Marquis, K.B. Averyt, M. Tignor, and H.L. Miller. Climate change 2007: The physical science basis. *Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA*, October 2007. URL http://www.ipcc.ch/pdf/assessment-report/ar4/wg1/ar4_wg1_full_report.pdf.

[33] C. A. Webber, J. A. Roberson, M. C. McWhinney, R. E. Brown, M. J. Pinckard, and J. F. Busch. After-hours power status of office equipment in the usa. *Energy Volume 31, Issue 14*, November 2006. URL http://dx.doi.org/10.1016/j.energy.2005.11.007.

[34] Wikipedia. Bernoulliborg, June 2015. URL https://nl.wikipedia.org/wiki/Bernoulliborg.

[35] Moustafa Youssef and Ashok Agrawala. The horus wlan location determination system. *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, 2005. URL http://dl.acm.org/citation.cfm?id=1067193.