university of groningen

Faculty of Mathematics and Natural Sciences

Computing Science: Bachelor Thesis

Sleepy for Linux

Power Management Framework for Workstations

Authors:
Thomas Hoeksema (S2349639)
Michel Medema (S2396009)

Supervisors:
drs. Faris Nizamic
dr. Apostolos Ampatzoglou
prof. dr. Alexander Lazovik

Additional supervisor:
BSc. Brian Setz

BSc project coordinator:
prof. dr. Alexandru C. Telea

July 10, 2015

Abstract

Office buildings attribute to 40% of global power consumption and are therefore an interesting target for power reduction research and efforts. In these buildings, networked workstations make up for a large part of the consumption of the entire building. One way to reduce power consumption by these workstations is to apply power management, by automatically suspending these workstations when they enter prolonged periods of inactivity.

The sleep timeout determines the amount of inactivity time after which suspension should occur. However, it is a difficult problem to determine an appropriate sleep timeout for users. This study addresses the power consumption of workstations by describing a software framework that may solve the sleep timeout issue and can be used to reduce power within the Bernoulliborg building of the University of Groningen.

Sleepy for Linux is a remote power management framework for Bernoulliborg workstations, which provides remote, external access to power management settings of the users of these workstations. Furthermore, Sleepy for Linux tracks activity data of these workstations and their users, which is available to smart external power management services, to attempt to reduce power consumption within the building.

In this research, we discuss that the software framework as presented has potential to save a significant fraction of power, based on testing data of real users within the Bernoulliborg, and describe ways to define the value of the sleep timeout. Two methods are described, namely to define a static timeout for all users, or a personalised timeout which is determined through examining the activity data of users. It is concluded that the framework is more effective when a personalised timeout is used.

Furthermore, the level of security and usability of Sleepy for Linux are also assessed in this paper, compared to the Lazy Sleep framework, a similar framework that is available for the Windows platform. The software is considered highly usable by test users and we have verified that the architecture of Sleepy for Linux has major security benefits over the architecture of Lazy Sleep.

Contents

1 Introduction

Office buildings and university sites contribute to about 40% of the world's total power consumption. [8] These sites accommodate a large amount of electronic equipment such as workstations, which, among others, contribute significantly to the total energy consumption of those buildings. [8] While these workstations serve as indispensable assets to these buildings, being used by the majority of the employees, many of these workstations are left powered even at times where they are not actually being used by any employee.

The possible reasons behind this seemingly indifferent or ignorant behaviour regarding power usage by employees vary greatly. For example, users often forget to temporarily suspend their workstation before going out for a meeting or lunch, which means a workstation is powered, yet idle, for a certain amount of time every day, while the respective owner of the workstation is not even in their office. This is of course a significant waste of energy that could have been easily prevented by the user, considering that the amount of power used by a suspended workstation is only a fraction of the power usage of an idle workstation that is fully powered. [8]

This significant waste of energy can be mitigated in two ways. First of all, users could suspend their own workstations once they decide that they will not be using it for a while, which is also referred to as manual power management. Secondly, it is possible to use automatic power management to solve this issue. This method makes use of a so-called sleep timeout, which is the amount of time after which the workstation will suspend itself if no activity has been detected by the user. By applying this method, when the users leave their idle workstations running or even leave their offices, the workstations will still suspend after some time, which saves power.

At the University of Groningen specifically, it is not very difficult to find powered workstations in the building that are not being used, even outside of office hours or during the weekend. This clearly indicates that manual power management is a very idealistic point of view that is definitely not attained in practice. We do not intend to fully research the reasons why users may forget to or purposely refrain from turning off the power to their workstation or to temporarily suspend their workstation, nor will we attempt to provide a methodology to encourage staff members to remember to power off their workstation before leaving their office. Rather, we will attempt to deploy a method of automatic power management instead. The problem with this method is that the sleep timeout is an arbitrary value that must be determined somehow.

Institutions usually deploy a static timeout for all workstations in their networks. The benefits are that this solution is easy to deploy and requires little installation effort, considering most modern systems already have support for automatic power management. However, choosing a certain value to be applied on all users has interesting side effects. For example, if the timeout is set as 15 minutes, then users who are never inactive for more than 15 minutes will never be able to save power this way. For these users, a shorter timeout would be preferable, but a shorter timeout may be too inconvenient for other users that do save a significant amount of power for a sleep timeout of 15 minutes, since their machines will suspend more frequently due to inactivity. Instead of choosing a static timeout, we will take a different approach and introduce a framework called Sleepy for Linux.

Sleepy for Linux is an extensible power management framework that attempts to reduce power consumption of workstations by providing functionality so that each user can have their own, independent sleep timeout. In order to accomplish this task, Sleepy for Linux tracks the state of the workstation continuously, which enables the framework to know, at all times, whether the user is actively working on their workstation or is idle. This activity data is gathered and transmitted anonymously and securely to an external server that organises and analyses all of the data of all the users that are using the framework. Using this data, it is possible to generate a personalised and appropriate sleep timeout for every user. The workstation of the user is then notified of this sleep timeout and will use it for automatic power management. Sleepy for Linux addresses the aforementioned problems, as it will automatically suspend the machine, especially in the case where a staff member forgets to do so, and provides a personalised timeout that will attempt to balance high power savings and low user inconvenience.

Apart from controlling the settings of the power management of the host workstations remotely, Sleepy for Linux will also provide functionality to remotely control the status of a workstation. The workstation can be remotely booted or woken up, shut down, and can be put into suspend mode or be hibernated. Sleepy for Linux is useful for both machines that run in single-user mode and multi-user mode. Respectively, these terms indicate a workstation that is owned and used only by a single staff member, and a workstation where a staff member or student can log in and access their account independent from the specific workstation that is being worked on at that moment.

By analysing the data of specific users, we assume that it is possible to find patterns that can predict how the activity load of the user will look like in the future, particularly to predict when long periods of inactivity will occur for the workstation, in which it is appropriate for the software to suspend the machine. An example of a pattern that may be expected is that the activity is related to the time of day: we assume that most workstations will be inactive when they are running at night time outside of office hours. This collection of data and patterns is subsequently used to adapt the sleep timeout for that user, so that it corresponds more realistically to the behaviour of that specific user. Furthermore, this data can also be utilised to determine when a certain workstation should or could be remotely ordered to shutdown or suspend.

Needless to say, since this framework runs on the workstations of the University of Groningen, and gathers sensitive data about user behaviour (though transmitted anonymously), it should be as secure as possible. If the software is not robust enough, then this may lead to critical issues for a large amount of workstations. If the data is not sent securely enough, or communication requires open ports, lacking security checks or lack of authentication of incoming messages, then it is possible for sensitive data to leak to third parties, or third parties could abuse the remote command system to manipulate the status of workstations.

The idea behind Sleepy for Linux originates from the Lazy Sleep software, [1] which performs a very similar task to Sleepy for Linux, apart from the fact that this software is only available for the Windows platform, where it has been successfully implemented according to the described functionality requirements. Its aim is also to reduce power usage by learning from the user's behaviour and using this data to adjust the sleep timeout in a

smart way, for example with machine learning algorithms. Lazy Sleep is to be considered a framework, rather than full-fledged power management software, because by its own, it is not able to analyse or react on activity data automatically. The software assumes that there is an external module that is capable of applying machine learning principles on the gathered activity data and is able to invoke methods on the Lazy Sleep server in order to change the sleep timeout and the status of the workstations. This assumption will be reused in the Sleepy for Linux software as well. In the section *Future Work*, the machine learning module will be mentioned again and described briefly.

Lazy Sleep was developed as part of a Green Mind project which involved reducing the environmental footprint of the Bernoulliborg on the Zernike campus of the University of Groningen on terms of water and power usage, and refuse disposal. [3] The Green Mind competition is held by the University of Groningen, where contestants have to invent methods or systems that increase the level of sustainability of buildings of the University of Groningen or of the work-flow of staff members and students.

By examining the Lazy Sleep software and the accompanying documentation, we have identified several issues that hinder the implementation and effectiveness of the Sleepy software. Five of these issues, which are important to understand before the research questions can be formulated, are the following:

- The Bernoulliborg building is home to the Computing Science and Artificial Intelligence research groups and studies, among others, and there are many multi-user workstations available for student use, as well as many single-user systems available for staff members and graduate students. However, the majority of these workstations are capable of running Linux. Since the Lazy Sleep software is only available for Windows systems, it can only reach part of the intended user population. Therefore, it is desirable to create a version of this software that runs on Linux workstations, which will be Sleepy for Linux.

- The documentation of the Green Mind project that Lazy Sleep is related to, as well as the documentation of Lazy Sleep itself, have assumed that there is a potential to reduce power usage by the results of surveys that have been taken from employees on the fifth floor of the Bernoulliborg building. [3] However, these survey results have not been verified in any way by means of the activity data that has been gathered by Lazy Sleepy. Therefore, it is yet to be seen if the software actually has the potential of decreasing power usage by the workstations. We will attempt to introduce a method to determine approximately the amount of power that could be saved, which is dependent on the intervals of inactivity of the workstations, which can be used to give a conclusion on this assumption.

- The server of Lazy Sleep must be able to distinguish workstations that have been registered and have client software running. This is accomplished by keeping track of the MAC address of the workstations; meaning that instead of distinguishing separate users, the software actually distinguishes workstations, which may have multiple users, producing activity data with different patterns. It could be more feasible and effective to record the activity data of each user separately, no matter which (both single-user and multi-user) workstation they are using.

- The usability and intrusion level of the Lazy Sleep software has never been verified through surveys or some other method of public testing. When the system is implemented, it does not only need to simply decrease power usage, but it must also be verified that the software does not interfere too harshly with the work-flow of staff members in terms of memory usage, and CPU and network performance.

- While the communication layer of Lazy Sleep is somewhat secure, by terms of the data going between these components is encrypted, the architecture of Lazy Sleep introduces some security issues if it were to be implemented in the network of the University of Groningen. For example, Lazy Sleep requires a set of open ports on both the client and server side of the application. Furthermore, no authentication is performed for incoming messages. This means that potentially, any attacker that knows the method of encryption can send remote commands to any of the client software, and these will be executed as if they were sent by the server. It is also relatively easy to set up a Man in the Middle (MiM) attack between the server and the clients, since the ports and addresses on which the Lazy Sleep components run are known in advance, to intercept the communication between these components.

Other implementation and documentation specific deficits of the software and related research have been identified as well, these will be discussed in more detail in the section *Related Work* and throughout the rest of this paper.

Summarising, we find the following **research questions** for the Sleepy for Linux project, in order of significance:

- Does the activity data of users indicate that Sleepy for Linux will be successful in decreasing power usage, and how is the amount of energy that is potentially saved calculated, based on this activity data?

- Are there significant benefits, in terms of decreasing power consumption, to determining a sleep timeout for each individual user of the workstations, rather than setting a global static timeout for all workstations, no matter which user account is currently logged in?

- Does the architecture of the Sleepy for Linux software attain the security and privacy standards as described, and is there an improvement with respect to the level of security of the Lazy Sleep software?

- What is the level of user acceptance of the software, is the software suitable to be implemented on both student and staff workstations?

To answer these research questions, the Sleepy for Linux software should be fully implemented according to the requirements provided by the stakeholders, since activity data will need to be captured and surveys will have to be conducted on test users in order to answer the first three research questions. These requirements for the application and the data will be elicited in the *Concept* section. Before we discuss these requirements, we will first discuss the similarities and differences between literature work that is related to (remote) power management (analysis) and the Sleepy for Linux project itself.

2 Related Work

An abundance of research is available that attempts to reduce the energy usage in office buildings and other types of buildings that sustain a large amount of daily users and visitors. All of this research accounts on the fact that these types of public buildings are responsible for a considerable portion of power usage by a city, and so they contribute significantly to the total financial and environmental costs of power usage by society. Furthermore, research shows that within these buildings, the majority of power usage is instigated by the appliances present in those buildings, in particular by workstations, lighting, printers and other office appliances. [9]

Therefore, research on how to influence these programmable devices to reduce their net power usage may be very critical tools for improving power efficiency and reducing the environmental impact, also known as carbon footprint, of these buildings. [9] Sleepy for Linux is such a tool, since it attempts to turn off appliances when they are no longer actively used. This section will discuss research that, similarly to the functionality of the Sleepy for Linux framework, also attempts to adapt power management behaviour of office appliances to user activity data and patterns.

2.1 Lazy Sleep (Windows)

Sleepy for Linux is based on the aforementioned Lazy Sleep software for the Windows platform. The most apparent functionality of Lazy Sleep has been described in the *Introduction* of this document. Apart from those requirements, Lazy Sleep also features an online overview of all the workstations that are connected to the Lazy Sleep management server. From this online overview, it is possible to remotely change the sleep timeout and idle timeout (the interval that a user has to be active in for a workstation to be considered active) of the workstations, and to send shutdown and sleep commands to any connected workstation. Furthermore, as the Lazy Sleep server is connected to the same network as the workstations it regulates, it is possible to send Wake on LAN (WOL) messages to remotely boot or wake up workstations that are not currently turned on. [1] Sleepy for Linux will follow these requirements as well, apart from the fact that WOL messages are not directly realisable due to the differences in communication protocols between Lazy Sleep and Sleepy for Linux. By thoroughly investigating the risks and limitations of the Lazy Sleep software, we have compiled the following list of issues that will have to be addressed by the Sleepy for Linux software, since they prevent the full implementation of the stakeholder requirements. Five issues that are not mentioned here have already been identified in the *Introduction* section.

In the Lazy Sleep software, the communication between the server and the client workstations happens by means of IP sockets. [1] There are several benefits to using sockets for this communication. For example, the server and clients are allowed to transmit information to each other uncoupled: the communication back and forth does not have to happen at the same time and the rate at which information is sent may differ between the directions of this channel. However, this method of communication can only be used in a single network where no port forwarding is necessary. This would mean that either each network controlled by Sleepy for Linux should have its own server instance, or port forwarding should be applied on the client workstations.

Opening ports for both incoming and outgoing information may pose serious security risks for the networks of the University of Groningen. Apart from that, the network of the University of Groningen is split up into different subsections that are not allowed to communicate directly to each other over an arbitrary port. The architecture of Lazy Sleep thus complicates, by its means of communication, the implementation of Sleepy in other parts of the university. Furthermore, we cannot expect non-technical users to understand the process of port forwarding and/or how they can apply this to their own networks. The installation and setup of the software should involve as little user interference as possible, with little to no manual configuration necessary. All in all, using sockets creates an inflexible architecture and does not promote portability and network transparency.

To manually wake up workstations using WOL messages, should the need arise to do so, the Lazy Sleep server requires the MAC address of the Ethernet adapters of the connected workstations. Subsequently, this MAC address is used as the unique identifier of these workstations. This is a questionable decision, considering there is no guarantee that MAC addresses are unique among adapters, and in occasions a workstation can have several adapters for Ethernet connections, a situation in which it is unclear which is the "right" MAC address to use. Furthermore, this limitation automatically excludes mobile devices from the Lazy Sleep server, as they have no Ethernet MAC address. From testing experience, we have found that there is no reliable way to select a MAC address from a specific adapter on Linux workstations to use as a unique identifier. We propose a different solution to distinguish between users in the *Concept* section.

The last issue with the Lazy Sleep software is that, while the communication between components is encrypted, it currently does not provide any sort of self-identification for messages that are sent between the server and the workstations, posing serious threats for the level of security and privacy that the software abides by. Therefore, any external party may impersonate the Lazy Sleep server and send unauthorised commands to the workstations and/or easily intercept or modify the encrypted activity data that is sent between the server and workstations. As discussed, the architecture of Lazy Sleep also requires several open ports, which potentially introduces security risks for every workstation that runs the software. Privacy and security are thereby exceedingly important factors for the architecture of Sleepy for Linux.

2.2 Analysis of power usage by workstations

We will now present some research which shows the impact that Sleepy for Linux could have on power reduction. Some writers argue that the power usage of office buildings has not been extensively researched, and that researchers tend to generalise the power output of all devices of an office building into a single value, rather than creating an account for each type of device in office buildings. [9] These particular writers have also shown, by repetitive measurements in office buildings in Japan, that setting the sleep timeout appropriately can lead to large power savings. They show that, for an aggressive timeout of five minutes, approximately 2% of Japan's power usage by consumers can be saved every year if office buildings implement a timeout mechanism on their workstations, showing that power management is very effective for reducing the energy use of office equipment. [9]

Further analysis of potential savings for office equipment has been performed in [8]. This study shows that lighting, workstations and the monitors of those workstations contribute to the majority of the energy usage in office buildings. The study recognises that a significant decrease in power cost and consumption can be achieved through power management of the monitors and workstations in particular. It is shown that while the power consumption is related to the energy requirements of the appliances in office buildings, choosing appliances with low power consumption is not a solution to reducing environmental footprint. Instead, the power consumption is mostly determined by the way users interact with these appliances. In the study, half of the included workstations were left on during evenings and weekends, which together account for 75% of the week. [8] The writer concludes that manual power can by itself produce impressive results if users are sufficiently educated. However, automatic power management is considered by the writer to yield the best results, as it can theoretically lead to 100% power management. [8]

Finally, we will mention Powernet, an extensive monitoring network that mines activity data from over 250 devices in a large academic building, which has made estimations of energy consumption on workstations of that building. [18] Powernet has been gathering data for over two years, which provides, according to the authors, ample level of granularity for extrapolating the data to a detailed breakdown of the power usage across the building's workstations. [18] The researchers claim that, before improvements to power usage can be identified in a specific setting, the character of power consumption of that specific setting needs to be analysed first. Many other studies have been noted to only collect data on a small scale which does not give enough insight into the actual power usage breakdown of a building. [18] This mind set is similar to the aforementioned research papers.

2.3 Sleep proxies and desktop virtualisation

Sleepy for Linux improves power management by providing an environment where each user has their own sleep timeout, regardless of the workstation that they are working on. This is just one method of smart remote power management that can lead to a reduction of power consumption. There are two other techniques that are quite well-known and frequently used as power management techniques, namely sleep proxies and desktop virtualisation, which will now be discussed.

Sleep proxies have come to existence due to the realisation that a workstation in hibernation or suspend mode suffers from degraded functionality. For example, if a machine is sleeping, it is unable to respond to requests from the network and therefore loses network presence, which is especially difficult in networks where an "always on" approach is assumed. [12] [11] [13] Another issue is that scheduled tasks such as backups must be postponed or dismissed altogether. [11]

In order to retain network presence, different machines, so-called sleep proxies are put to use that will intercept communication intended for the sleeping machines. This machine will then examine the incoming requests and will wake the sleeping machines using a WOL message, if required by the nature of the request. [12] The machines that the sleep proxies target should be allowed to sleep for a significant fraction of the time, for power reduction to be effective. [11]

The benefit of using sleep proxies is that little hardware support is required and the process can be implemented completely using software solutions, [12] this is also demonstrated by the SleepServer [13] and LiteGreen [11] applications. The costs of using sleep proxies are related due to the fact that communication patterns between clients in a network can be very complex, and can not always be solved efficiently by trivial approaches, [11] and the fact that the machines which serve as sleep proxies consume power themselves, which can derail the purpose of the sleep proxies if the amount of sleep proxies is not balanced correctly. [13] [12]

Three relevant studies were found that attempted to introduce sleep proxies intro office environments. SleepServer, which implements a very pure sleep proxy methodology as described, was found to be capable of reducing energy consumption in the range of 27% to 86% with an average savings of 60%. [13] Another study focuses on a different sleep proxy service called LiteGreen. Apart from sustaining network presence for sleeping machines, the researchers behind LiteGreen also researched the types of communication that were sent over a network. They found that broadcast and multicast messages were very common background noise in a network, and that a large amount of this communication, roughly 75%, can be safely ignored and do not have to be processed by the sleep proxies, which makes the sleep proxies even more efficient and thus save more power. [11] The last method described in [12] utilises a lightweight proxy system which has proven to half the up-time of "always-on" workstations. A warning is given about security risks related to sleep proxies: if many machines wake up simultaneously by malicious cause, this could lead to potentially dangerous power spikes. [12]

The other concept is desktop virtualisation, a process where the desktop environment of users is migrated to central servers when their workstation is suspended or turned off. When the respective workstation is reactivated, the central server transfers the migrated environment back to the host workstation. There are some performance and usability issues involved by the adoption of virtualisation. [15] [13] The obvious cost of virtualisation of workstations is the fact that the workstations, server architecture and user action sequences will have to be significantly modified. [13] However, there are some interesting benefits of using virtualisation. For example, the desktop environment of users are always on by definition, though their workstations may not be. Network presence is therefore fully sustained. Workstations can be switched off seamlessly, increasing the potential for power reduction. [15]

LiteGreen makes use of virtualisation of desktop environments in conjunction with sleep proxies. [15] Apart from manual shutdown, hibernation and suspension events, they also remotely trigger the virtualisation procedures for workstations that have been idle for an amount of time above a certain threshold. It is argued that because of virtualisation, a more aggressive threshold can be chosen to eliminate shorter inactivity periods as well, as the transition between states is made seamless by virtualisation. LiteGreen combines the two concepts to save approximately 72-74% of power consumption in a test environment. [15]

2.4 Aggressive power management for networked workstations

This research, which best resembles the requirements of this project, intends to create a similar framework to Sleepy for Linux, to be used for the workstations in the College of William and Mary located in Williamsburg, Virginia in the United States. [10] The research states that, alike the situation in the Bernoulliborg, many workstations are left powered by students and staff while they are not being used anymore, particularly in evenings and weekends. An application is proposed that determines user activity not only by user input devices such as mouse and keyboard, but also by taking into account average CPU usage over a certain period of time and whether or not there is any active ssh connection with the workstation at the moment.

The application described in this paper favours hibernation over suspending to RAM. Two main reasons are given for this decision: namely that hibernation does not require the RAM of the workstation to be powered and because the state of the session is important in case a user was logged in before hibernation or suspension to RAM. [10] However, we believe that there is a misconception by the author, as suspension to RAM also saves the state of the user session, although it does require more power to maintain the session in memory.

The sleep timeout that is used in this application is static and does not adapt to user behaviour. There are several issues with aggressive power management that may in some cases lead to more power consumption rather than reducing it. If the sleep timeout is set too aggressively, workstations will hibernate quicker and users with short intervals between their activity periods will be hindered by this approach. They will therefore spend more time on reactivating their workstation, which may, in total, cost more energy than the short time that the user would have been idle.

Although the sleep timeout will be derived differently for Sleepy for Linux, the paper mentions different methods of obtaining the time interval that the user has been idle for, which was useful in the research into the functionality of different Linux packages and libraries that we performed, prior to implementing Sleepy for Linux.

2.5 Other activity-controlled office appliances

Sleepy for Linux is a system based on personalising and remotely adjusting the sleep timeout of networked workstations for the University of Groningen. However, workstations are not the only type of appliances that are suitable for this kind of power management. We will now briefly discuss two other types of appliances for which smart timeout-based systems have shown to greatly reduce total power consumption, to reinforce the approach behind Sleepy for Linux.

Besides workstations, lighting in office buildings is a large source of power consumption. [17] Similarly to workstations, when lighting is controlled manually, users may often be reluctant to manually switch off lighting in rooms when it is no longer necessary. [17] [20] Automatic control of lighting may therefore be desirable, as they may turn off lighting when no more user presence is detected or when sufficient daylight is present in the respective rooms, in which case there is no need for additional lighting either. [17]

Kaneko et al [20] and Fischer et al [17] both describe a lighting system that is automatically managed by a machine learning module that adapts to user input and behaviour, based on parameters that can be set according to the preferences of users. Kaneko found that by automatically controlling the dim rates of lighting on employee floors of office buildings, it was possible to save 25% of the power consumption by means of lighting, while providing appropriate illumination for every present employee. [20] Fischer suggests a highly efficient simulated illumination model where users are represented as a set of parameters which indicate their preferences regarding illumination. This is achieved through a minimal network of wireless sensors that, apart from detecting user presence automatically, will in future work also address the aforementioned insight regarding daylight infiltration. [17]

Attempts have been made to reduce energy consumption of heating, ventilating and air-conditioning (HVAC) systems as well. [14] [16] There are of particular interest in countries where temperature or humidity variations are common, such as Japan, as these appliances are by necessity present in office buildings. The researchers of [14] argue that a clever choice of HVAC technologies and automatic control of these appliances can lead to improved equipment efficiency which potentially reduces power consumption in air-conditioned buildings. Other researchers have implemented models using reinforcement learning to make air-conditioning more energy efficient, by only automatically turning them on when it is desired by users, feeding to this process so-called daily action plans that are submitted by users. [16]

2.6 Context-Aware Power Management

Power consumption in buildings has risen in the past few decades due to the increase in desktop computing equipment in offices. [23] Apart from this increase in desktop computing equipment, the advent of ubiquitous computing, a concept in which computing equipment pervades the user environment in all sorts of devices in any location, complicates power reduction and power management in general even further, as all of these devices in the environment consume power in one way or another. [25] However, this kind of pervasive computing could also prove to be beneficial to power management, since information about the context of the user is very likely to be available and accurate in these kinds of computing environments: the devices are dealing with detailed user behaviour (user location, user activity, user presence, etc.) opposed to desktop computing equipment. [23]

This is where the notion of Context-Aware Power Management (CAPM) comes into play. CAPM attempts to minimise the overall electricity consumption of a building while maintaining acceptable performance for the users of the devices in those buildings. [23] In such power management systems, researchers argue that the primary context that should be available is the activity status of a device: whether a user is currently using their device and if they are about to start using their device. [25] The purpose of the Sleepy for Linux framework is to obtain the data that supports the determination of these context types, which is why research on CAPM and its potential impact are interesting to this project.

These studies show that power management is baked into the process of context-aware systems. Research states that the requirements for context-aware techniques are its accuracy, its complexity, usability and power consumption. [22] All of these factors are also important for Sleepy for Linux's activity sensor.

Devices such as smart phones, tablets and sometimes laptops usually travel along with their respective owners from place to place within a building. Therefore, several tests have been performed to see if location-aware power management frameworks are capable of providing enough insight over users to reduce power consumption. [23] [22] These studies recognise that the level of power consumption of the devices used by these context-aware techniques may harm the acceptance of the software if the consumption is too high. [22] Furthermore, they show that location-aware applications are capable of reducing power consumption where possible. [23] [22] However, these results are usually very marginal and are highly variant between users. [25] Authors claim that using only location as a context is not enough to create an accurate power management framework, and that when more types of sensors are added, such as an acoustic sensor in combination with location sensors, accuracy will increase. [23]

One example of CAPM that is closely related to Sleepy for Linux is elaborated on in [25]. In this research, the authors combine the activity data from the workstation (similar to Sleepy for Linux) with the data of a presence sensor in terms of Bluetooth tags in users' devices. They note that this approach works well for predicting when a monitor should be turned off, but may incur too much user inconvenience when used for workstations. [25] However, as Sleepy for Linux uses an independent timeout per workstation, we expect to receive better results than this study, where a global static timeout was used.

Wearable devices, in particular smart phones and smart watches, have stimulated the development of distributed sensing applications. [28] However, if such a device is continuously measuring data from the environment, such as location or acceleration data, then this has a large impact on the battery lifetime of such a device. [24] Therefore, increasing the battery lifetime of mobile devices in sensing applications is a primary research area in CAPM. User context is available in abundance on smart wearable devices as the user often interacts with these devices. The user context changes continuously, and therefore applications can dynamically adapt (or temporarily disable) their sampling rate or algorithms in order to reduce the power consumption of the device. [28] This study has shown that this can lead to increasing the lifetime of smart phone batteries fivefold, [28] while another study shows that decreasing the sampling rate of sensing applications can lead to a fourfold increase in battery duration, while maintaining great accuracy, [24] which will significantly benefit the deployment of these devices in real environments. [24] [28]

However, mobile CAPM is not only effective for distributed sensing systems, but also for regular devices. A study has been performed that proposes a system for CAPM that will warn the system when the battery of their mobile device is running low. [26] This system uses the location of the user to determine the next charging opportunity for the device, predicts the amount of time that the user will be using their device between charging opportunities, and finally can also determine the actual lifetime of a device if the current application context is maintained. [26] The system shows minimal prediction errors and is regarded as being able to maximise user convenience while distributing power resources as efficiently as possible. [26]

Another system that heavily uses CAPM is the AlarmNet system, which is a system that attempts to improve the health care in elderly homes, by using wireless sensor technology to monitor behaviour patterns of resident, in order to apply tailored context-aware protocols for those residents. [27] As a side result, the system also implements features for enhancing resident privacy and for providing medical practitioners with important events that happen in the living quarters of these residents. [27]

As a final example, CAPM can also be used in situations where no human users are present. For example, CAPM can also be applied for determining power efficient routes in wireless ad hoc networks. [21] Rather than performing shortest-hop routing for every package sent over the network, several metrics are introduced that are based on the power consumption of communication nodes within these networks, which are subsequently used to construct a cheapest path through the network in terms of power consumption. The researchers show that they are capable of power reductions in the range of 40% to 70% without significantly increasing the delivery time of messages, and that these techniques can be used in most traditional routing protocols. [21]

2.7 Summary

From literature research, we can conclude that there are many research papers that have analysed the potential benefits of power management on office appliances, in particular lighting, workstations, monitors and in some cases air-conditioning devices. All discussed research agrees that there is abundant room for improvement in power efficiency for all those devices if automatic power management is used, rather than manual power management by users. This supports the assumptions that we have made earlier in this paper. Apart from automatic power management, we have seen how context-aware techniques can, in combination with power management techniques, effectively and efficiently decrease power usage with minimal effects on perceived performance of the devices for users in very diverse situations.

However, while there are some initiatives for creating smart software for suspending inactive workstations, none of these initiatives actually implement remote power management in the way Sleepy for Linux intends to. Rather, they attempt to reduce power usage by aggressively suspending workstations using very short sleep timeouts. We have discussed how aggressive timeouts can lead to user dissatisfaction which can severely harm the adoption of such software on a larger scale. Using the concepts introduced by Sleepy for Linux, we predict that it will be possible to create user-friendly power management without risking the level of adoption by potential users.

There have also been more rigorous attempts at remote power management which involve using sleep proxies and using virtualisation to reduce power consumption, while sustaining network presence and user-friendliness respectively. While these techniques have been designed specifically for use in enterprise networks, which the networks of the University of Groningen could technically apply to as well, we do not consider these for Sleepy for Linux, due the fact that this requires large changes in workstation and network behaviour that may not favour the adoption of the software just for the purpose of better power management. Needless to say, time and resource constraints also prohibit us from researching and/or utilising these methods.

3 Concept

The goal of the Sleepy for Linux project is to provide an answer to several research questions that involve power reduction of workstations, and which attempt to assess security and usability levels of the Sleepy for Linux framework. To reiterate, we have derived the following research questions earlier on in this paper:

- Does the activity data of users indicate that Sleepy for Linux will be successful in decreasing power usage, and how is the amount of energy that is potentially saved calculated, based on this activity data?
- Are there significant benefits, in terms of decreasing power consumption, to determining a sleep timeout for each individual user of the workstations, rather than setting a global static timeout for all workstations, no matter which user account is currently logged in?
- Does the architecture of the Sleepy for Linux software attain the security and privacy standards as described, and is there an improvement with respect to the level of security of the Lazy Sleep software?
- What is the level of user acceptance of the software, is the software suitable to be implemented on both student and staff workstations?

The problem that Sleepy for Linux addresses has been discussed in the *Introduction* and a brief description of the framework has been given. In this section, we will elaborate on the analysis of the problem, the software requirements for the framework and our methodologies for answering these research questions.

3.1 Problem analysis

For this project, the scope of the problem is limited to the power consumption of the workstations at the University of Groningen, specifically in the Bernoulliborg building on the Zernike campus. As demonstrated by the Green Mind competition, staff and students of the Bernoulliborg value sustainability highly and environmental sustainability has become a trademark of the Bernoulliborg. Many plans of action have already been developed and executed that attempt to reduce energy consumption of the appliances in the offices of staff members, to reduce the amount of time that lighting is active within the building, to streamline the process of collecting refuse for easier recycling purposes and to limit the amount of water used by introducing filters for taps and information stickers for toilet users. These methods have succeeded in reducing water and energy usage significantly. [3]

However, there is still room for improvement. The Bernoulliborg contains several hundreds of workstations available for the staff members of various research groups, research graduates, staff members of the building and students. These workstations host in principle two operating systems, namely Microsoft Windows and a GNU/Linux distribution. We have discussed that workstations are one of the primary sources of an office building's power consumption. Needless to say, the Bernoulliborg is such a building, and a large amount of power is indeed consumed by these workstations. [3] There are potentially opportunities for power reduction for these workstations if it can be shown, somehow, that these workstations are not using their energy efficiently with respect to their usage by staff and students.

To understand if it is possible to reduce power consumption in this area of electrical equipment, we need to know how and when this power is being consumed by the workstations. It is possible to find out how efficient workstations are using power, by looking at how users are using their workstation. We can distinguish between two states for now, a workstation being active and a workstation being idle. Respectively, this means that a user is currently using their workstation, and that a user is not using their workstation, but it is powered. The following graph (Figure 1) shows the state of an example workstation in the Bernoulliborg between 12:00 and 18:00 on a specific day.

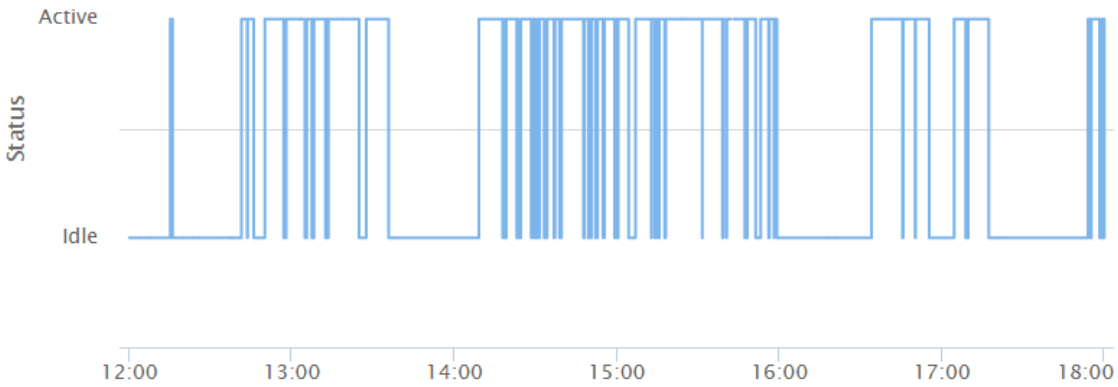

Figure 1: Activity data of an example user in the Bernoulliborg.

The graph shows that in this specific time frame for this specific example user, the user is not active for the full 100% of the time frame. There are both large gaps (longer than 30 minutes) and small gaps (shorter than 5 minutes) of idleness in this graph, in which the user's workstation is powered but is not being used. In these periods of time, the workstation is powered without a purpose, as no tasks are being performed on it by its user. Therefore, the summation of these idleness periods is what could potentially be saved in terms of power. For this specific user, the prospects of potential power reduction are quite positive, as we can see that this user is idle for almost half of the time in this specific time frame.

We have discussed that there is a sleep timeout which determines when a workstation should be considered idle enough for action to be taken upon that workstation. There are two ways to save power in these instances, which are shutting down and suspending a workstation. To verify that performing these actions will actually reduce overall power consumption, we present the following graph (Figure 2) about power consumption levels by workstations. There are other states that are not covered in this graph, such as hibernation, but since these are not very relevant to our project we will not discuss these in detail.

Four states are visible in this graph. "Active" and "Idle" have the same meaning as described before: the workstations are powered and users respectively are and are not using their workstations to perform tasks. The state "Off" refers to a workstation being powered down, so no user interaction is possible during this state. Furthermore, the "Suspended" state refers to a workstation being suspended to RAM.

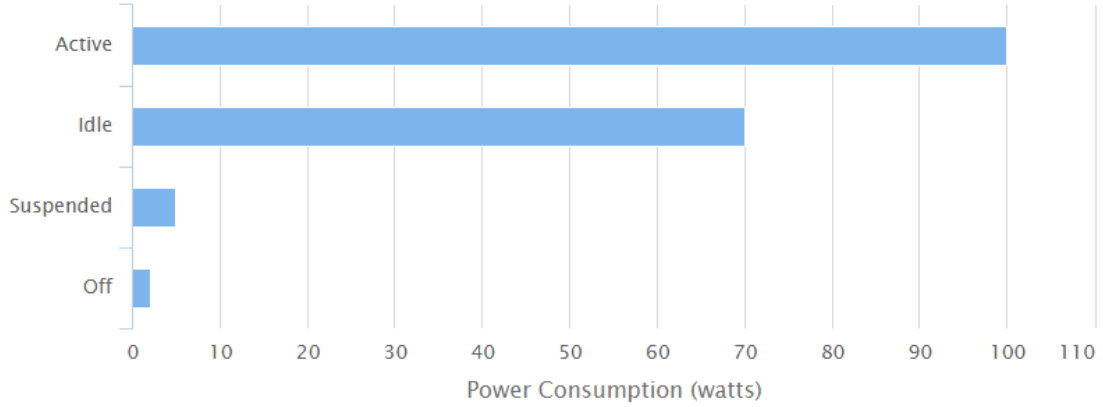

Figure 2: Power consumption levels of workstations. [8] [11] [15]

Active workstations use the most power out of these four states. While idle machines are also powered, they use less power than active workstations. The difference between active and idle workstations is that the processor is being used more frequently during active periods, due to the fact that the user is actively performing tasks on their workstation, which is not the case during idle periods. [8] When a workstation is turned off, the workstation will actually still consume a bit of power, due to the fact that the network interface needs to stay powered. When a workstation is suspended, the state of the machine is saved to RAM, which enables the workstation to power down most of its components apart from the network interface and the RAM, saving a large amount of power compared to the active and idle states. The special attribute of suspended workstations is that they respect both power consumption and user convenience: it is relatively quick to resume a workstation from suspension, while it usually takes more time (and power) to start up a workstation that is completely powered down.

From this graph we can conclude that it may be beneficial to suspend workstations when they are not being used by their users, as this may potentially save a significant portion of power. Suspending a workstation also retains user convenience, which is not true if we were to use a technique alike fully powering down a workstation or applying hibernation. While hibernation uses even less power than suspension, as much as a powered down workstation, it is not feasible to be used in this situation because of the time (and power) that it takes to wake up a workstation from hibernation, opposed to suspension. It should be noted that the values in the graph are not actual consumption levels that we have measured in the Bernoulliborg, but have been conceived from several literature references that assess power consumption of workstations. The absolute values may not represent the Bernoulliborg as accurately as possible, but these proportions of power consumption for the different states of the workstations will hold in the Bernoulliborg.

3.2 Requirements analysis

The problem that this project addresses is now clear: workstations in the Bernoulliborg are frequently idle for extended periods of time, which unnecessarily consumes a large amount of power and actually accounts for a significant portion of total building consumption of the Bernoulliborg. This can be solved by suspending the workstations when they are idle, as suspended workstations consume far less power compared to powered workstations which are idle.

For the research questions of this project, it has to be established whether Sleepy for Linux is capable of enabling power savings and if the suggested techniques that can be applied by Sleepy for Linux are effective. In order to establish this, activity data from test users in the Bernoulliborg, such as presented in the graph in the previous section, is needed. However, this data is not automatically available, as most systems do not have any standard software to measure this kind of data over a certain amount of time. However, Sleepy for Linux does measure this data, and therefore a prototype of Sleepy for Linux will have to be implemented. Furthermore, the research questions about privacy, security and usability also call for a working prototype that can be compared to the Windows version of Sleepy for Linux and that can be evaluated using user surveys and interviews.

Since a working prototype of Sleepy for Linux is necessary, this section will discuss the software requirements of the framework in detail. First of all, we will reiterate by giving a short description of the functionality of the software as perceived by users and components within the system. Afterwards, the requirements as deduced from the stakeholders and this description will be presented in categories. Furthermore, some research that was performed regarding the technicalities of the workstations of the Bernoulliborg (operating system, distribution) will be included as well.

3.2.1 Software description

Sleepy for Linux is a remote power management framework that provides external modules with remote access to various parameters of the power management of registered workstations. In particular, the framework allows external modules to set the sleep timeout of a specific (set of) workstation(s). Apart from that, it is also possible to remotely command a workstation to suspend or shut down.

The interaction of the user with the system is very simple. First of all, the user will install the framework using a simple installation package that is easy to understand for both technical and non-technical audiences, and requires little to no manual configuration. Once the software is installed and the workstation is rebooted, Sleepy for Linux will go into effect. The user will see a tray icon in the system tray menu of their Linux distribution which will display whether Sleepy for Linux is active or disabled. Upon interacting with the tray icon, the user is able to show information on the framework and it should be possible for the user to disable and re-enable the framework through this tray menu. When disabling the framework, users are obliged to specify a reason as to why they are disabling the application.

While the framework is enabled, it will listen to remote commands (shutdown/suspend) from a central server, and will continuously gather activity data from the users that are logged in on the workstations. If the user is not using their device, they are labeled as idle, while in the opposite case they are labeled as active. The distinction between idle and active is achieved through a so-called idle timeout, which will be discussed in the requirements. The server receives all the data from the client framework applications and returns the remote commands (if any are present for a specific workstation) and updates on the sleep timeout and idle timeout for those workstations. The central server stores all of the status data from clients and workstations and also exposes several endpoints for external modules to interact with, namely to access the activity data and to manipulate workstations and users remotely (e.g. change the sleep timeout, idle timeout, or to suspend or shut their workstation down). All of the data captured by Sleepy for Linux is anonymised and it should not be possible for the authorised external modules to directly link activity data to specific people in the Bernoulliborg building.

Concluding the description of the software, we will identify the stakeholders in this project. The students and staff of the Bernoulliborg are the primary end-users of the Sleepy for Linux framework and are therefore important stakeholders. Other stakeholders are the CIT staff, which are responsible for the workstations in the Bernoulliborg and the software running on them, since the target area of deployment is the workstations of the Bernoulliborg. As external modules will interact with the framework to modify power management settings, the developers of such modules should be considered, as they dictate the amount and types of endpoints that are available through the central server. Lastly, the University of Groningen as a whole would benefit from this framework through potential power reduction, which can, apart from the Bernoulliborg, eventually also be distributed to other parts of the University. We have conducted interviews with several students, a research graduate, and some staff members about the functionality of the Sleepy for Linux framework. Furthermore, an author of the Green Mind competition [3], who was involved in the elaboration of Lazy Sleep, as well as the authors of Lazy Sleep themselves [1] have helped us conceive a list of requirements for the Sleepy for Linux framework. These requirements will now be listed.

3.2.2 Functional requirements

The functional requirements describe the user functionality that the aforementioned stakeholders have requested directly or have been deduced from deliberation with the stakeholders. See *Appendix A: Table of functional requirements* for a detailed list of all the functional requirements.

A. User Interface
The user is able to interact with the application through a user interface. This user interface consists of several components. There is an application indicator, also known as a tray icon, visible in the system tray bar of the window manager of the desktop session while the application is running. This indicator displays the logo of Sleepy for Linux. When clicked, the indicator will show a drop-down box with a set of options for the user. The options open up windows that allow the user to view information about the application and to perform certain actions that will be discussed.

B. Disabling Sleepy for Linux
When the application is running on a workstation and is enabled, the user of that workstation can temporarily disable the functionality of Sleepy for Linux. Through the system tray menu, a window can be brought up in which the user is asked to specify a reason for disabling Sleepy for Linux. The user can pick from a number of preset reasons or specify a custom reason. When the user confirms for a certain chosen reason, Sleepy for Linux will disable power management, so that the workstation will no longer be suspended when the user is idle. Sleepy for Linux will keep power management disabled until the user logs out and logs back in again, until the system is rebooted, or until the user manually re-enables Sleepy for Linux through the same system tray menu.

C. Monitoring client status
The Sleepy for Linux application gathers data about the current status of the end-user's user sessions and monitors the activity of the end-user in case their user session is active. This data, of all users, is gathered at the Sleepy for Linux server. We have created an online overview that is able of showing all of this status data in one overview for all of the registered clients. This data is also available through exposed API implemented by the server that allows authorised parties to request anonymous user activity data.

D. Monitoring workstation status
Apart from gathering user session activity data, Sleepy for Linux also gathers workstation data, particularly about the status of the workstation. Similarly to how the data of end-users is presented in an online overview, we have also implemented an overview for workstations. The types of data that are stored per end-user and workstation are further detailed in a later section.

E. Client/workstation manipulation
Sleepy for Linux offers an architecture that allows for remote management of the sleep timeout and idle timeout of client applications, which are respectively the amount of idle time after a workstation of the specific end-user will suspend, and the amount of idle time after which a workstation will actually be regarded as idle. The amount of idle time is defined as the time that has passed since the last activity event by the end-user. The majority of these event are triggered through an input device such as a mouse or keyboard.

3.2.3 Technical requirements

The technical requirements describe the functionality of Sleepy for Linux regarding external modules, communication between modules and interacting with lower level hardware and/or software. See *Appendix B: Table of technical requirements* for a detailed list of all technical requirements.

A. Measuring user activity
To determine the current status of a user, Sleepy for Linux must be able to determine whether a user is actively using the workstation or not. To do this, the number of seconds since user activity was last detected is tracked. User activity that should be tracked consists of mouse movement, keyboard input events, and also actions such as playing a video should be considered as user activity. When no activity has been detected for a certain amount of time, the user is considered idle, determined by the idle timeout.

B. Accessing power management
Sleepy for Linux does not suspend the workstation itself. The task of suspending the workstation is left to the power management of the operating system. Sleepy for Linux merely updates the sleep timeout of the power management to reflect the sleep timeout that is supplied by the server. Suspension of the workstation is done through power management (instead of forcing this through a bus message after a certain amount of time) to prevent disruption of the normal operation of the operating system. The power management libraries offer many different settings regarding power management, and circumventing the power manager would render these settings obsolete. Finally, Sleepy for Linux also toggles a property of power management to disable or re-enable all power management based on user feedback through the disable and enable menus.

C. Executing remote commands
The workstation can receive remote commands from the Sleepy for Linux and these commands should be executed by the workstation. Two commands are supported at this point, namely shutdown and suspend. A workstation should execute these commands immediately as they are received. Naturally, the two commands respectively shut down the workstation and suspend the workstation to RAM. Note that in this case, Sleepy for Linux does circumvent power management and immediately suspends the workstation by itself.

D. Catching user and workstation events
In the functional requirements, we have described that Sleepy for Linux should be able to keep track of the status of workstations and end-users. To be able to determine these states, it is necessary to catch certain events that are fired by components of the operating system both in user session mode and system (root) mode. The status of the workstation and end-users then changes upon the occurrence of these events, so these events should be propagated to the central server which also keeps track of activity data of end-users. Examples of these events are booting a workstation, shutting down, suspending or hibernating a workstation and resuming from suspension or hibernation, or logging into or out of a certain user account.

E. Uniquely identifying clients
Every Sleepy for Linux client application should be uniquely identifiable. This means that every different user account, be it on the same workstation or on different workstations, has a unique identifier that allows the server to separate the activity data of one user from the other. It is necessary to track data separately for each end-user, since sleep timeouts and idle timeouts may be different for each end-user.

F. Installation and compatibility A future goal of Sleepy for Linux is that it should run on the workstations of the University of Groningen, specifically on the Linux version that is running on the workstations in the Bernoulliborg. We have identified that the Linux versions running on these machines are the Xubuntu 14.04 LTS version and the Ubuntu 10.04 version or higher, which use either the Unity, GNOME or XFCE window managers depending on the workstation, so Sleepy for Linux should be functional on any combination of these window managers and distributions and their versions. Furthermore, to contribute to usability and ease of adoption, the software should be installed by a single install script without user intervention.

3.2.4 Non-functional requirements

The non-functional requirements define criteria that all functionality of the Sleepy for Linux application and server should adhere to. See *Appendix C: Table of non-functional requirements* for a detailed list of all non-functional requirements.

A. Privacy and security
Every user of Sleepy for Linux can be uniquely identified. This is necessary for determining a different sleep timeout for every user. However, it should not be possible to trace the gathered activity data back to a specific person. It is very important that the data is anonymous, otherwise the activity data can be used to track a specific user. This also implies that the activity data should be sent to the server over a secure connection. When the activity data is sent over an insecure connection, third (unauthorised) parties can potentially intercept the data of a certain user, which is a violation of the security and privacy of that user.

B. Scalability
Once every so often, the Sleepy for Linux clients send data to the server about user activity. The clients also poll the server for remote commands that may have to be executed. The server must process these requests and generate a response for the client in all of these cases. When the number of clients increases, the number of requests made to the server increases linearly to the amount of connected clients. The server architecture must be able to scale well to many incoming requests, up to several hundreds of requests at the same time, to be able to process all this data, since the targeted area of deployment contains many workstations, of which a large fraction can be powered at the same time.

C. Portability
In the technical requirements we have described the system specifications of the Bernoulliborg systems. Based on this information, it has been decided that Sleepy for Linux will support the latest (and some earlier) versions of Ubuntu and Debian, with the Unity, GNOME2, GNOME3, XFCE, LXDE, and potentially the KDE, MATE and Cinnamon window managers. Preferably, the software should be written and compiled in such a way that only one binary package will be necessary which runs equally robust on each of the supported operating systems and window managers.

D. Performance and usability
The user should not notice the fact that Sleepy for Linux is running on their workstation, by means of a decrease in the performance of the workstation. Therefore, the resources used by Sleepy for Linux should be kept to a minimum. Furthermore, The amount of resources used also affects the power consumption of the workstation. Users may be very cautious in adopting the software into their working environment. The usability of the software is of very high importance for that reason. If the user experiences too much continued inconvenience from using the software, the chances are that the user will be tempted to deinstall the software, which is specifically true for such an application as Sleepy for Linux that does not offer immediate convenience benefits to the user.

E. Interoperability
Lazy Sleep (Windows) is a part of a larger system conceived by the Green Mind proposal. [3] The data obtained by Lazy Sleep is being fed to this system by means of a queue. Sleepy for Linux should be compatible with this queue architecture, since integration is highly desirable. Furthermore, access to the system is provided by means of an external API. This API provides the ability to modify the settings of the users, such as the sleep timeout, and the possibility to send commands to the workstations.

3.3 Data requirements

The software requirements for the implementation of the Sleepy for Linux have been established. One of the important functions of Sleepy for Linux is that it centrally gathers activity data from the connected users, and that this data can be used to modify the power management of these workstations remotely. There are two research questions that involve this activity data that is gathered by Sleepy for Linux, namely the following:

- Does the activity data of users indicate that Sleepy for Linux will be successful in decreasing power usage, and how is the amount of energy that is potentially saved calculated, based on this activity data?
- Are there significant benefits, in terms of decreasing power consumption, to determining a sleep timeout for each individual user of the workstations, rather than setting a global static timeout for all workstations, no matter which user account is currently logged in?

For the implementation of Sleepy for Linux, it is useful to understand what kind of data is needed to perform the analysis for these two research questions. Furthermore, to answer these research questions, a thorough analysis on activity data is required, which will only be available through the use of Sleepy for Linux. Therefore, this section will discuss the types and quantities of data that the Sleepy for Linux framework should gather from users, how this data will be represented, how the data will be obtained and how the data will be analysed with respect to answering the research questions.

3.3.1 Identification of types

Sleepy for Linux distinguishes between workstations and users, so that users can travel between workstations and retain their personalised power management settings. Both of these types of entities within the framework have an activity status at any given point in time. Some of these different values for the activity status have already been mentioned in this paper. Workstations represent the actual physical devices that are connected to the framework, and can take any of the following list for the activity status:

- *On* - When a workstation has this status, it means that it is currently powered. This does not automatically mean that a user is currently logged in on the workstation, because when the workstation is still displaying the log-in screen, the workstation is also considered powered.
- *Off* - A workstation is considered off when it is powered down, in this state the workstation is using the least amount of power, as only the network interface is powered.

- *Sleeping* - When a workstation is manually or automatically suspended to RAM (or hibernated by the user), it is considered to be in this state. We will often refer to a suspended workstation or user as "sleeping" due to the terminology used in the Green Mind proposal and the Lazy Sleep documentation, [1] [3] as well as to correspond better to the etymology of the framework.

For users of these workstations, we assume that any user can only be logged into one workstation at a time, and that the activity status can take any of the following values at any point in time:

- *Active* - A user is active when they are logged in at a workstation connected to the Sleepy for Linux framework, and are currently performing tasks on that specific workstation. This is measured through events from input devices such as mice and keyboards.
- *Idle* - A user is idle when they are logged in at a certain workstation, but have not generated any activity for a certain amount of time, which is determined by the idle timeout.
- *Off* - The user is not currently logged in at any workstation that has Sleepy for Linux installed. Note that this is different from the "Off" state for workstations.
- *Sleeping* - The workstation at which the user is currently logged in is in the "Sleeping" state, which means that the workstation is suspended (or hibernated), so the user is suspended as well. When a workstation is sleeping, all of users that are logged in at that workstation should be considered sleeping as well.

To construct the activity status data as described in the problem analysis, these states of workstations and users need to be known. Therefore, the Sleepy for Linux framework should be able to construct two-dimensional continuous signals which display the activity status variations for specific workstations and users, over a certain amount of time. These signals can then be used to deduce the amount of idle time of a certain user, to deduce the length of the idle periods, and to make claims about the power reduction that a certain sleep timeout could provide for a certain user, as well as decide whether a personalised sleep timeout is preferable over a global timeout.

Apart from activity data, the framework should also capture negative feedback of users, which happens when they disable the framework and provide a reason for doing so. While these negative feedback messages are not directly useful for answering our research questions, they will be necessary in the case that an external module, which implements machine learning to modify the sleep timeouts, will be produced. This module will most likely attempt to minimise user inconvenience in relation to power reduction, and negative user feedback is a valid indicator for estimating user inconvenience. [4]

3.3.2 Acquisition and quantification

The targeted area of deployment are the workstations of the Bernoulliborg at the Zernike campus of the University of Groningen. Therefore, it would be ideal to test the Sleepy for Linux framework on these workstations and use the test data from these users and workstations to formulate an answer to the research questions at hand.

However, during the early weeks of this project, it had become clear that this was not possible due to the fact that the CIT staff would not like to run experimental power management software on these workstations. What complicated the adoption of Sleepy for Linux during the time span of this project even further is the fact that the software of the networked workstations is centrally managed. Therefore, the framework was to be installed on all networked workstations at the same time, or none at all.

There are special workstations in the Bernoulliborg for employees of the University that are partially managed by the users themselves. This means that these users have limited access to commands for installing software packages. However, even this limited amount of access is not enough to install and run all of the required resources by Sleepy for Linux. Therefore it was decided to test the software on personal workstations of users at their homes and completely unmanaged workstations within the Bernoulliborg. We were also given access to several workstations in the Bernoulliborg that were both unmanaged and not appointed to an employee to test the Sleepy for Linux framework on.

To gather the data, a testing phase will take place where 6 to 10 testers will run the Sleepy for Linux prototype on a specific workstation that they either use within the Bernoulliborg or in a home situation. The framework will be collecting activity data of these users during the testing phase. For this testing phase, the sleep timeout will be set to infinite, so that we can measure the actual user behaviour without interference of automatic power management. Then, we can use the data to analyse if a sleep timeout would reduce power consumption and whether a personalised timeout is the best approach for the situation in the Bernoulliborg. The testers consist of staff members, research graduates and several third year students of the Computing Science department of the University of Groningen. We estimate that we will need at least two weeks of activity data from these testers in order to perform a viable analysis that answers our research questions. This is considered to be a reasonable amount of time for the testing phase, considering the relatively short amount of time that is available to do our research.

It was necessary to gather a significant amount of data to answer the stated research questions through a testing phase. The various types of data that are collected are necessary to answer these research questions. Furthermore, the usability of the framework must also be verified as part of our research, which can only be performed if the framework has been tested by the targeted group of users. The possibility was considered to use data from the original Lazy Sleep (Windows) framework in the data analysis for this project. However, it was not possible to extract the data due to time constraints and technicalities regarding the production environment of Lazy Sleep. This data was also recorded with an arbitrary finite timeout rather than an infinite timeout. This means that there is a limit to the length of the idle periods of this data, and is therefore too biased to be used for the elaboration of our research questions, since it is affected by automatic power management.

Besides gathering the activity data, the testing phase will also be used to get insight into the user acceptance of Sleepy for Linux, through a questionnaire, which is handed out to the testers after the testing phase is finished. We will elaborate more on the user acceptance testing later on in this paper, and will also describe that 6 to 10 users, which is usually considered an amount that is too low to get reasonable results, is actually enough in this situation.

3.4 User acceptance testing

So far, the analysis and methodologies for answering all research questions, with the exception of the last one, have been derived. This last research question relates to the level of user acceptance of the framework, namely:

- What is the level of user acceptance of the software, is the software suitable to be implemented on both student and staff workstations?

Several techniques were used to stimulate the adoption of the software by the targeted end-users and to assess the level of user acceptance of the framework. The methodology of determining how usable the Sleepy for Linux framework is, from an end-user perspective, will now be described in detail. We will start by explaining which intrinsic values are important in order to convince and stimulate users into adopting the software, using the Technology Acceptance Model. Subsequently, our assessment method for usability will be described. Lastly, we will comment on the relevance of the results based on our limited sample size of users.

(We will use the terms *usability* and *user acceptance* interchangeably, as they arguably measure the same underlying software aspects.)

3.4.1 Improving user acceptance

The Sleepy for Linux software does not directly create any benefits for the targeted users. The goal of the framework is to enable remote access to the power management of workstations and to measure activity behaviour of users. While the framework does intend to reduce power consumption, which can be perceived by users as a positive feature, this feature on its own may not convince users that the framework is useful. After all, the reason why this framework may save power is due to the fact that users are currently utilising power resources wastefully while their workstations are being powered. Therefore, tools should be considered to improve upon the user acceptance of the framework as much as possible.

One of such tools is the Technology Acceptance Model (TAM), which is a model in the information sciences that attempts to explain how user acceptance and adoption of a certain type of technology can be achieved in practice. [29] Figure 3 shows the general layout of the TAM and all of its components as conceived by Davis. [29] [30]

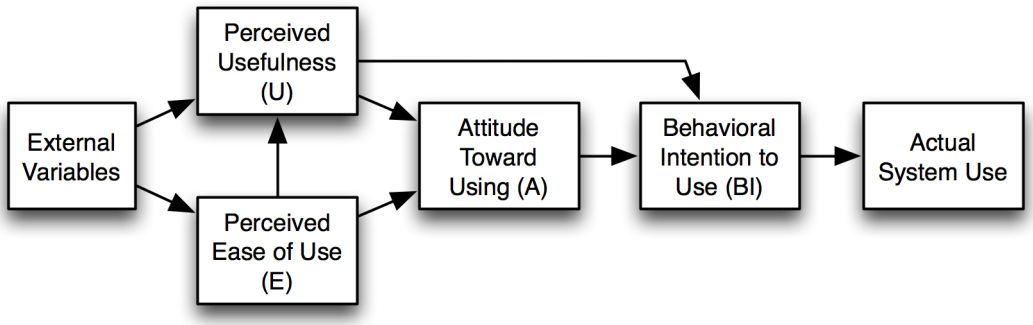

Figure 3: The Technology Acceptance Model as described by Davis. [29]

The TAM lists several key factors that will affect how users perceive the Sleepy for Linux framework. We will describe how the TAM can be used to increase the effectiveness of our software framework in terms of being appealing to future users. First off, the TAM contains several components, most importantly:

- *External Variables* - These are in principle the design features of the software package. These features can be experienced as both positive and negative by the end-user and may influence their level of Perceived Usefulness and Perceived Ease of Use.
- *Perceived Ease of Use* - This is described by the authors of the TAM as "the degree to which a person believes that using a particular system would be free from effort". [30] It signifies that users will be more likely to adopt a certain software system if they are under the impression that it will not cause any major inconvenience during the installation procedure and the period that the software is running.
- *Perceived Usefulness* - This is originally defined as "the degree to which a person believes that using a particular system would enhance his or her job performance". [30] Users will be more likely to adopt software into their working environment if it can be shown that the software is in some way capable of streamlining their work flow or simplifying a task that they often perform.

The TAM describes that maximising Perceived Ease of Use and Perceived Usefulness through the use of External Variables, such as designated design features regarding the software, will result in maximal user adoption and actual use of the software. For this reason, the interface of Sleepy for Linux should be designed to be as simple, yet informative, as possible. Besides this, the software should not interfere at all with the work flow of the user unless the user specifically requests certain information of the software framework or attempts to disable the framework temporarily.

Furthermore, we have also developed a portal website for the software, of which the home page is depicted in the second half of *Appendix F: Screenshots*. The portal explains what the purpose of the software is and how it can benefit our targeted test user audience. Furthermore, information about this project is presented. There is also a page that hosts a download with the package that is needed to install Sleepy for Linux, which informs the user with detailed steps about the installation and deinstallation procedure. The platform website is designed to propagate a professional and open impression about the framework to potential users. By conveying our information to users through this portal website, we hope that users will perceive that the software is easy to use and that it is useful in terms of reducing power consumption. According to the TAM, this will help maximise the attitude of end-users towards using the software, which will eventually lead to actual system use by these targeted users.

3.4.2 System Usability Scale

The System Usability Scale (SUS) is a "quick and dirty" usability study that allows one to quickly and easily assess the usability of software or of a service. [35] It assesses the usability through a questionnaire, containing 10 questions. A scale is provided, ranging from strongly disagree to strongly agree, from which one of the options should be chosen. Each of the questions has a certain score attached to it, depending on the chosen option.

With these scores, an overall score can be calculated for the questionnaire, which can be matched against a scale, that indicates how the perceived usability of the software or service is. The scale ranges from 0 to 100 and a higher score means a better perceived usability by the user. [35] Using the SUS for the evaluation of the usability of Sleepy for Linux seems appropriate, because Sleepy for Linux, as perceived by the user, is quite limited in terms of user interface and ways for the user to influence the framework. Using a highly advanced usability study would therefore be unnecessary and in fact counter-effective. Also, the SUS survey is short, it contains only ten questions, and is thus easy for users to complete. Finally, the SUS can be used effectively for small sample sizes. This is important, since the amount of users that are testing Sleepy for Linux is limited.

When a tester has filled in the questionnaire, the usability score of that questionnaire can be determined by calculating the scores for all the individual questions. For the questions with uneven numbers (1, 3, 5, 7, 9) the score is the scale position minus one. For the questions with even numbers (2, 4, 6, 8, 10) the score is calculated by subtracting the scale position from five. All of these individual scores are then summed up and the resulting number is multiplied by 2.5. [36] Subsequently, the final score is obtained, which can be used to determine the usability of Sleepy for Linux. To obtain the total score, taking into account all the questionnaires, all the scores are summed up and are then divided by the number of questionnaires.

3.4.3 User sample size evaluation

As has been described, the sample size of the testing audience for Sleepy for Linux is limited to the range of 6 to 10 testers. The reader may wonder if this amount of testers is actually enough to find a reasonably relevant value for the user acceptance and if this amount of testers is enough to identify most of the usability issues with the software. We will argue now that this is actually enough for Sleepy for Linux.

There is a study that formulates an algorithm for determining the percentage of usability problems can be found, given a certain amount of test users. [37] Figure 4 shows the formula that was conceived, which is in fact $F(i) = N \cdot (1 - (1 - L)^i)$, where $F(i)$ is the amount of usability problems found for i distinct test users, where N is the total number of usability problems and L is a proportion of usability problems that a single user can discover while testing a software application. The study argues that this value is 31%. This would mean that with only 6 users, the minimum amount of testers for Sleepy for Linux, we are capable of finding 89% of the usability problems that exist for the application.

Of course, there are other studies that doubt the formula that has been presented. For example, [33] states that while the formula has been validated several times by comparing the value of L to studies on usability problems, the formula itself makes unwarranted claims about the differences and similarities between distinct test users. It is stated that the formula only applies when the software that it is tested on is relatively simple, and that for web-based tests with reasonably big software projects, the formula fails spectacularly to calculate the right amount of usability problems found. [33] Their conclusion is that a simple, arbitrary L value can only be reliable in rare occasions. A more useful model would be to predict the probability that at least one usability problem remains after a certain amount of users have tested the software. [33]

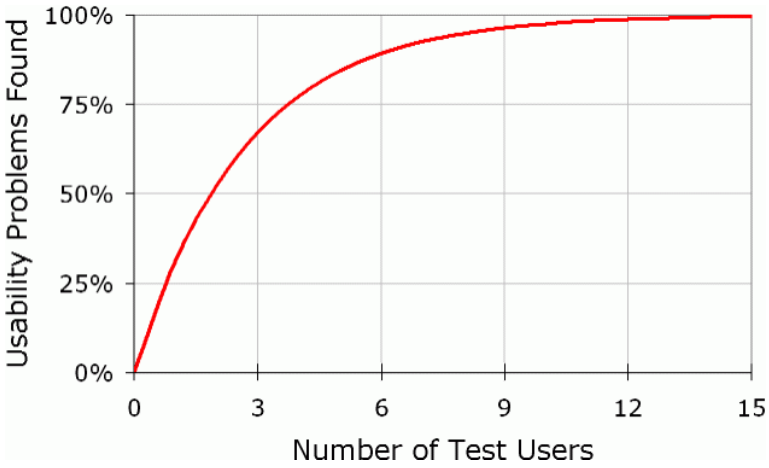

Figure 4: Graph of the number of usability problems found versus the number of test users. [37]

Another study that criticises the formula is presented in [32], which claims that the formula can be applied in most occasions for web-based testing, but that the chosen value of $L = 31\%$ is too high. Rather, this study finds a value for L that is much lower, in the order of $L = 10\%$, meaning that considerably more testers are needed to find a large fraction of the usability problems. Furthermore, other researchers argue that it is not feasible to rely on these differences between users and construct formulas that attempt to index these differences. For example, study shows that for randomly chosen sets of users, the amount of usability problems found may vary greatly, one set of users finding only 55% and others finding 95%. [34] However, the writer does not deny the fact that a large amount of usability problems can be found with a limited amount of testers.

Although some of this criticism is well-warranted when large scale application testing is considered, it is not reasonable in the situation of Sleepy for Linux. As the authors of the original study state when they conceived the described formula, the most important concept of the graph is that "zero users gives zero insights". [37] What this means is that for simple applications, it is always best to perform testing regardless of the size of your testing audience, as this will definitely lead to 0 or more usability problems being found. Furthermore, we argue that since the application is very small, with a limited user interface and ways for the users to interact with the software, the amount of usability problems will be very low and users are more likely to be able to traverse all execution paths of the software within the user interface during the testing phase. Therefore, individual users are more likely to find more usability problems and the value for L in the formula will be decent enough to find a large amount of problems with the limited amount of testers. Besides, we will not only conduct simple surveys in order to assess the usability of Sleepy for Linux, but we will also conduct user interviews in which we propose open questions to the test audience and request them to list the experienced usability problems and suggestions that they have for Sleepy for Linux. By performing these two usability studies in parallel, we think that we can make reasonable assumptions about the level of user acceptance of the framework.

3.5 Summary

In this section the research problems have been further elaborated upon and for each of the research questions, a research methodology has been described. These methodologies revealed that implementation of a prototype of the Sleepy for Linux framework is necessary in order to formulate an answer for these research questions.

The requirements of the framework have been described as elicited from the stakeholders of the project, which detail the perceived functionality by users, the layout of the user interface, the software qualities of the framework and the technical feats that have to be accomplished for the implementation of the prototype. These requirements are necessary to verify that the prototype adheres to the functionality that the stakeholders expect from the framework. The prototype is also used to measure the level of security and usability of the framework, both of which are qualities that are subject for a research question. Furthermore, requirements were given to the types and quantities of data that Sleepy for Linux should collect from its users, which is necessary to answer the research questions about power reduction for workstations and automatic power management.

Finally, the user acceptance testing of the framework has been described in detail. A testing phase of two weeks will take place in which a limited amount of testers will test the application, so that they may provide input on user acceptance levels through surveys and user interviews after the testing phase has been finalised. We have explained that using the Technology Acceptance Model, we can increase the perceived usefulness and perceived ease of use of the application. A platform website has been created for Sleepy for Linux that fulfils this purpose.

The user acceptance of the framework will be tested through the System Usability Scale, a "quick and dirty" usability study, and detailed user interviews with open questions. While the amount of testers is small, we have argued that the amount of testers will generate enough and relevant data to be used for user acceptance studies, because the framework itself has a limited user interface and control of the framework by the user is limited to a couple of functions. Therefore, it is likely that all users can fully test the application in the testing phase.

The methodology to be used for the research questions has been analysed and proposed in this section. In the next section, these methodologies will be realised and more information will be given on the implementation for Sleepy for Linux and the data analysis will be described.

4 Realisation

In the *Concept* section, the base structure of the framework has been determined and the methodologies for answering the research questions have been formulated. Furthermore, directions were given for how the testing phase of the framework is to be executed. In this section, the implementation of the Sleepy for Linux framework will be thoroughly discussed, including the architecture of the various components and the underlying Linux mechanisms and libraries that make these components function in the way they do. Finally, the execution of the testing phase will be discussed, as well as how the data that resulted from this testing phase should be analysed.

4.1 Implementation

Starting off, the implementation of Sleepy for Linux will be explained. Before going into detail for the individual components that inhabit the Sleepy for Linux framework, the overall architecture of the system will be discussed. Since this is a remote power management framework, the system comprises a central server to which a large amount of clients in the form of workstations can be connected. These components communicate to each other using particular protocols and message policies that have been designed specifically for this framework. The communication layer will therefore be explained in detail. After the overall architecture and communication layer are established, the architectural layout of both the server and client side of the framework will be laid out to the reader. Finally, we comment on the existence of external modules and why they are important to the Sleepy for Linux framework.

4.1.1 System Architecture

Figure 5 shows the overall architecture of the system. The system consists of an arbitrary number of workstations, a central server and (potentially) external modules. On the workstations, which are shown on the left in the figure, a service is running at multi-user (system) level in root mode (also known as superuser mode), and there exists only one running instance of such a service per workstation. The service is in principle a loop that will communicate with the server every iteration through the workstation API, where it can send updates to and retrieve pending commands to execute, if there are any.

Apart from the service component, there is also a client component running for each user session that is active. In other words, every user that is logged in on a workstation has their own client component running in the user mode of that user, and is only running for a user while that specific user is logged in. These clients communicate with the server, through the client API, where they can send status updates to and obtain (among other data) the sleep timeout from the server for their respective user.

The figure shows several types of workstations. A single-user workstation (A) is shown at the top of the diagram. Only a single user exists on these workstations and typically it is not possible for other users to use this workstation or for the account to migrate to another workstation. A multi-user workstation (B) is also shown, where zero or more users can be logged on at the same time, but the accounts still cannot travel between workstations. When multiple users are logged on at a certain time, a separate client is running on the workstation for every user.

The last type of workstation is the networked workstation (C and D). This type of workstations is particularly important for the targeted area of deployment. Workstations in the Bernoulliborg are connected to a network so that users can log in from any workstation in the building. Therefore, their power management settings and Sleepy for Linux instances should not be linked to specific workstations. A user can switch between workstations within a network, but will still be seen as the same user by the Sleepy for Linux system. This means that the user will receive the same power management settings, no matter which workstation is used within the network.

The distinction into two sets of components, clients and services, is due to some limitations of the X11 screen saver extension library, that is used by Sleepy for Linux to determine the user activity status on a workstation, since the library is capable of obtaining the exact time that has passed since the user has last shown any activity on the workstation. For this library, activity is measured through events from input devices such as mice or keyboards, but a user may also be considered active by this library when, for example, a video is currently playing in any application running on the workstation. We have explained that the idle timeout is defined to be the amount of inactivity time after a user is considered to be actually idle. The time since the last activity that is returned by the X11 screen saver extension, combined with the defined idle timeout, is then used to determine whether a user is actively using their workstation or is to be considered idle.

While this may seem as an ideal solution for this application, there is a peculiarity that has to be dealt with. The library can only obtain the idle time when it is called from an active user session in the user mode of that session. When two users, say user 1 and user 2, are logged in, this means that in the user mode of user 2 and in root mode, it is not possible to determine if user 1 is idle or active. This can only be determined in the user mode of user 1. Subsequently, it is not possible to detect the "user" activity or inactivity for workstations where no user is currently logged in, and any software that uses this library cannot be running outside of these user sessions. The clients are for this reason forced to run separately for each user in the user mode of that user. However, to shut a workstation down or to suspend it, UNIX commands have to be called from the root mode of the workstation. Since we cannot call these commands from user mode, which is what the separate client applications are running in, it is necessary to distinguish a second type of component that runs in root mode, which is the service.

Not only have we made a distinction between software components for this reason, but we have actually separated the entities for which data is recorded as well. For the Lazy Sleep (Windows) framework, activity data is gathered per workstation rather than for each user. This means that when multiple users use the same workstation, they will all contribute to the same activity data for that workstation. However, since we are running a separate Sleepy for Linux client application instance for each user, it makes much more sense to gather data for each individual user account that interacts with the workstations rather than only collecting data per workstation. This also naturally enabled the concept that we have described earlier in this section, namely that users should be able to travel between workstations and retain their connection to their specific data and power management settings.

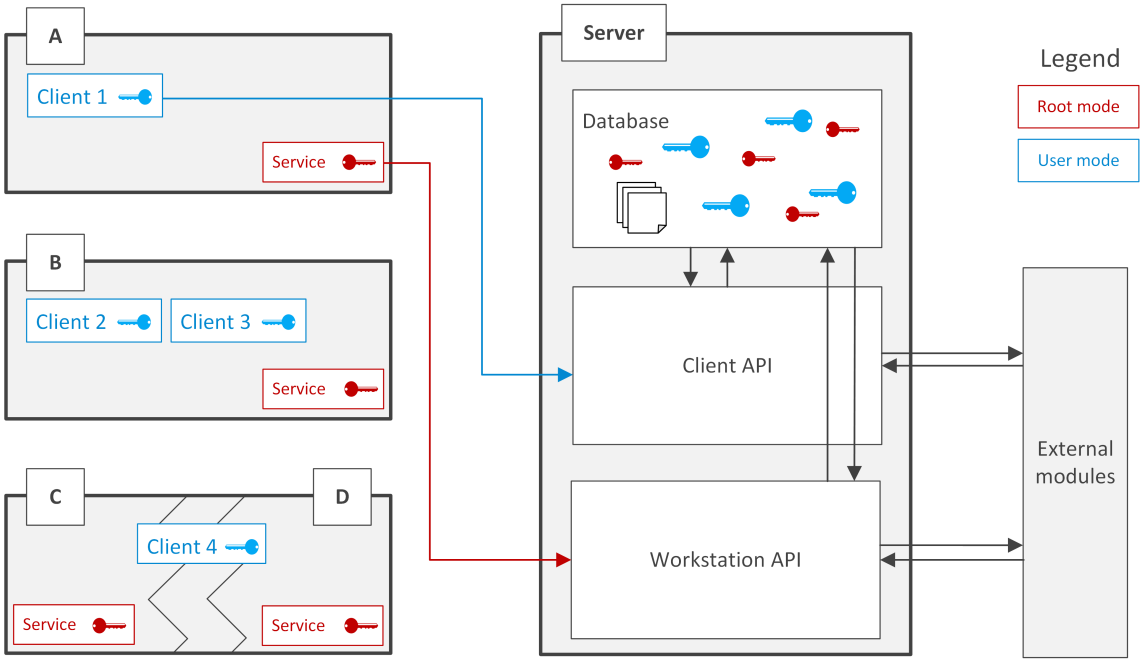

Figure 5: System Architecture of the Sleepy for Linux framework.

Now that the workstations have been described, we will talk about the server side portion of the framework. Both of the APIs that are available on the server interact with a database. The activity data received from the clients and services, about respectively user activity/events and workstation events, is stored in this database, as well as pending commands that are to be executed by the services running on the workstations. To keep track of which activity data belongs to which client, each client has a unique identifier, which is also stored in the database. The workstations must also be uniquely identified, to keep track of which commands are meant for which workstation. To uniquely identify each workstation, the MAC address of one of the available network interfaces of the workstation is used, since a MAC address is somewhat guaranteed to be universally unique and is needed to remotely wake up the workstation when it is powered off using WOL magic packets.

The last component of the system is the set of external modules, which are shown on the right of the figure. External modules interact with the system through both of the APIs that are exposed by the server. These modules can use the workstation API to send commands to workstations, and the client API can be used to set a sleep timeout or idle timeout for a specific user. Obtaining activity data or the last known status of a certain user or workstation could also be obtained through these APIs, but is currently not yet implemented. The next section will discuss the communication between workstations, the server, and the external modules in more detail.

4.1.2 Communication Layer

The workstations, server and external modules need to communicate with each other to exchange data and issue requests to be performed on workstations and the data. For this communication between the different components in the system, the HTTP protocol is used. Implementing a communication layer using the HTTP protocol is desirable of the use of other communication principles such as IP sockets, as is implemented by Lazy Sleep (Windows), because of several advantages that will be discussed. However, using the HTTP protocol means that Sleepy for Linux will not be directly compatible with the existing server architecture of Lazy Sleep.

To be able to use the HTTP protocol, a web service is running on the server, which is constantly waiting for incoming requests. All the requests are initialised by the software components running on the workstations itself, and data that must be sent from the server to the workstations can only be passed to the workstation in form of a reply to these requests. As a result of this coupled method of communication, the server cannot directly send commands to a workstation or update the sleep timeout of a user directly. However, this also means that authentication of messages is guaranteed, as the client authorises the reply message of the server, which contains the commands and adjustments to the timeout, using the certificate of the server. Furthermore, when a workstation is located in a different network than the server, there is no need for a user to forward any ports when the HTTP method of communication is used over IP sockets. It also does not expose additional ports on the network in most cases, since HTTP traffic is usually already allowed on institutional networks.

Another advantage of the use of the HTTP protocol is the fact that is widely used, and many additional protocols exist that work in conjunction with it. Because Sleepy for Linux should ensure the privacy of the users, the communication must happen in a secure way. This is relatively easy to achieve by using the Transport Layer Security (TLS) protocol, which is a protocol that works on top of the HTTP protocol. Using this protocol also makes it possible to validate the identity of the server, which means that it is much harder to execute an impersonation attack that could trick workstations into sending data to or receiving replies from a malicious and unauthorised third party instead of the legitimate server.

An issue that is not addressed by the use of TLS is the fact that one client can in principle send data on behalf of another client. The certificate can only be used by the clients to validate the server's identity. It is not possible for the server to determine the client's identity in a reliable way. However, every client owns a unique identifier, which is randomly generated by the server and is only known internally to the client and server. It is therefore difficult to figure out which identifier is used by another client. Admitted, it will always remain possible to obtain the identifier of another client by accessing the files of the associated user account and search for the file in which Sleepy for Linux stores this unique identifier. In the network of the University of Groningen, it is fortunately not possible to view the files of another user in the network.

An overview of the messages that are passed between the workstations and the server is given in Figure 6. The figure shows how the communication between the workstations and the server is organised, and displays the interactions between the workstations and the server.

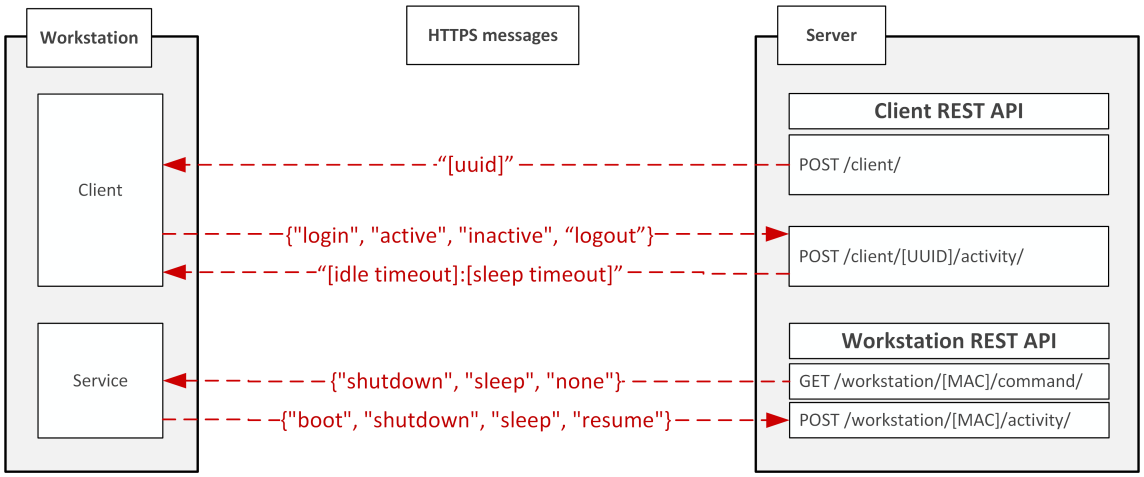

Figure 6: Communication Layer of the system. Sets, depicted between curly brackets, indicate that any of these strings can be used for this message. Variables are depicted between square brackets.

The server exposes several endpoints through a Representational State Transfer (REST) API, which are divided into two main categories, the client and workstation API, as depicted in the system architecture. The first consists of the endpoints that are used by the clients and the other of the endpoints used by the services that are both running on the workstations connected to the framework. Only the endpoints relevant for the communication between the workstations and the server are displayed in the figure. A full overview of the endpoints will be given in the description of the server architecture.

When the client is executed for a specific user for the very first time, it must obtain a unique identifier from the server, also referred to as the registration procedure. This identifier is stored both on the server and at the client in a location where only the current user may access the file containing the identifier. When the client is running, it periodically sends events about the activity of the user to the server, and as a response it receives the potentially updated idle timeout and the sleep timeout from the server. If the timeouts do not change for an extended period of time, then the clients will receive the same values for the timeouts multiple times, though this will naturally not influence the application in a harmful way. The service regularly checks if there are any pending commands on the server for the workstation on which the service is running. The service also sends system events to the server, for example when the workstation is booted or shut down, but does not receive any response. More information on the events and data that are sent by users and workstations will be described in the client architecture section.

4.1.3 Server Architecture

This section describes the architecture of the server. An overview of the server, along with some of the components with which the server interacts are shown in Figure 7. The server implements a REST API for both the clients and the workstations which has been depicted in the figure. The workstations use their endpoints to send system events relevant to power management to the server, and obtain pending commands from the server. Clients use their endpoints to register themselves at the server and in exchange receive a unique identifier, and to post activity status data and user events to the server. It is also possible for external modules to interact with the workstations through some of these endpoints, particularly the ones that are addressed with arrows in the figure. External modules can send instructions to or modify the power management settings of a specific user or workstation with the use of these endpoints.

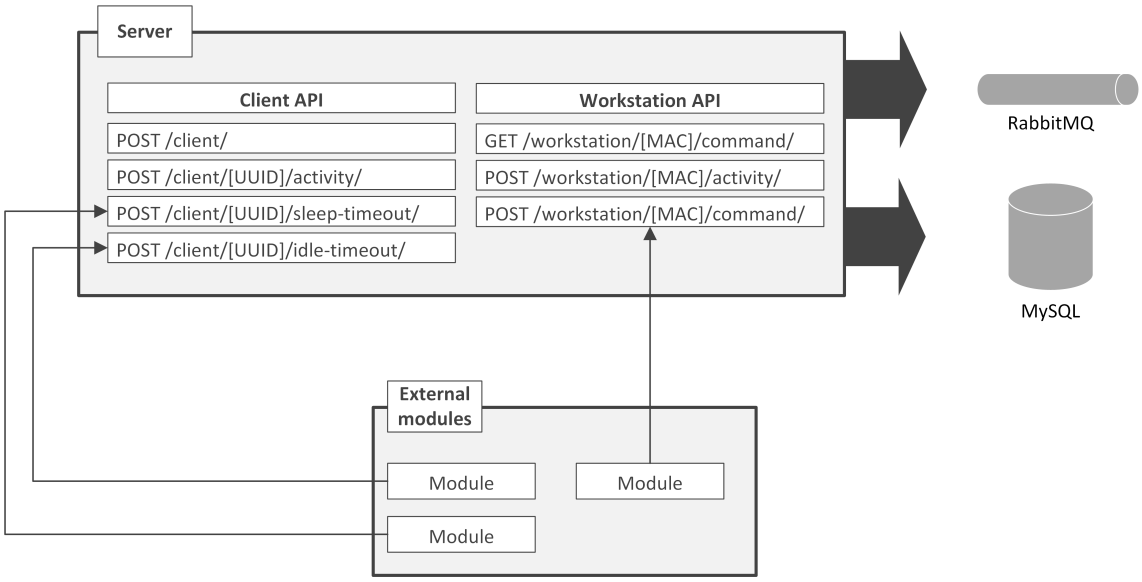

Figure 7: Architecture of the server used by Sleepy for Linux.

The server contains a database, which is used to store activity data and events of clients and workstations. When a client requests a unique identifier from the server, the server also stores this identifier in the database. This identifier can then be linked to the received activity data from that client. The identifier is also used to look up the sleep timeout and idle timeout that belong to a certain user. The unique identifiers of the workstations are also stored in the database, which are used to link the received system events to the correct workstation and to keep track of which instructions are addressed to which workstation.

Apart from its own database to store sensor data, the server architecture of the Sleepy for Linux framework also interacts with a set of message queues implemented using RabbitMQ. These queues act as a bridge between the server of Sleepy for Linux and the architecture of the Green Mind proposal that has been made for the Bernoulliborg. [3] The reason for wanting this type of compatibility is the fact that this is how Lazy Sleep (Windows) communicates its gathered data with the rest of the Green Mind system.

To ensure compatibility with the Green Mind system in the Bernoulliborg, this output structure of data into message queues has also been made available in Sleepy for Linux. The Green Mind system uses, among other things, the activity data received from workstations and users for various purposes. Since Sleepy for Linux already gathers this data, the external system does not implement this itself, but relies on the data obtained by Sleepy for Linux. We do not discuss the implementation details of these message queues here, but instead refer to the Lazy Sleep documentation. [1]

The members of the two sets of APIs that are available to clients, services and external modules will now be discussed. The first set contains the endpoints for the clients and the second set contains the endpoints for the workstations. The following list describes all the endpoints that are available through the REST API for the clients:

- **POST /client/** This endpoint is used to register a new client. The MAC address of the workstation and the user name must be sent along with the request. If no exceptions occur, the response contains the Universally Unique Identifier (UUID) assigned to the client. The server will store the UUID in a database table, along with the date and time of the registration and the sleep and idle timeout of the client. The sleep and idle timeout are initially set to a predefined default value.
- **POST /client/[UUID]/activity/** Clients can post their activity data at this endpoint. The URL must contain the UUID of the client and the current status of the client must be sent along with the request. The values that this status can take are defined in Figure 6. In the case of a "login" message, the MAC address must also be provided in the call, so that the server knows at which workstation the user is currently logging in. This is important in case an external module wants to shut down or suspend a workstation at which a specific user is currently logged in. As a result of this API call, the client will obtain the idle timeout and sleep timeout. The activity data received by the server is stored in the database as well as sent to the respective RabbitMQ message queue.
- **POST /client/[UUID]/sleep-timeout/** This endpoint can be used to change the sleep timeout of a client. The client, whose sleep timeout should be changed, is specified in the URL of the endpoint by their UUID. The new sleep timeout is sent along with the data field of the request and the server updates the sleep timeout of the corresponding client in the database.
- **POST /client/[UUID]/idle-timeout/** This endpoint can be used to change the idle timeout of a client. Similarly to the previous API call, the client is specified by the UUID, and the new idle timeout is sent along with the request, which subsequently replaces the current idle timeout of that specific client that is stored in the database.

The available endpoints for the services (workstations) are described in the following list:

- **GET /workstation/[MAC]/command/** When a command is pending for the specified workstation, this command is returned. Before the pending command is returned, it is removed from the database at the server, so that it is not possible for a workstation to receive the same command multiple times. When there is no command currently pending for the workstation that is specified, the server responds with a standard *none* reply. The rest of the possible replies are given in Figure 6.

- **POST /workstation/[MAC]/command/** This endpoint can be used for executing remote commands on a workstation, identified by its MAC address. The command that should be executed on the specified workstation is sent along with the request, which can be either "shutdown" or "sleep", the latter referring to the suspension of the workstation. The server will store the command for the specified workstation in the database, until another command is received for that workstation or until the command is retrieved by the workstation. When another command is sent, the previously stored command is overwritten.
- **POST /workstation/[MAC]/activity/** The system events of the workstation, generated by the service and several event scripts that will be described in the client architecture, can be sent to this endpoint. The endpoint does not return any meaningful data to the client, but simply acknowledges that the request has been processed. The events received by the server are also stored in the database.

Not all of the available endpoints are being used by the clients, services and other components that are running on the workstations. The endpoints that are not being used by these components are made available to allow the integration of external modules, as has been described in the previous architecture sections. For example, an external component could be taking care of determining a sleep timeout for the clients based on the obtained activity data for those clients, using machine learning algorithms. This component could then use the corresponding endpoint to update the sleep timeout of the respective clients. It is also possible for an external component to send sleep or shutdown commands to the workstations through the API, if they are authorized to do so.

4.1.4 Client Architecture

The last (arguably most important) part of the implementation that will be discussed is the architecture of the Sleepy for Linux client application and event scripts, which reside at a specific workstation where Sleepy for Linux is installed. First, the components of the client are explained along with the responsibilities of each component. Subsequently, an overview is given of the events that Sleepy for Linux detects on a workstation. Finally, some of the tasks performed by the components of the Sleepy for Linux client are discussed in greater detail.

A. Components
Figure 8 shows the components of Sleepy for Linux that are running on workstations. Sleepy for Linux makes use of a software package called Upstart that is available by default on Ubuntu systems and is available through the package repository of the other distributions Sleepy for Linux supports. Upstart is an event-based, script-invoking service which handles starting and stopping of tasks and services before, at, or after certain observed system events, such as boot or shutdown of the machine, that can be chosen by the developer. This allows the developer to create daemon-like applications relatively easy by adding a single Upstart script to a certain directory at the machine on which their applications run. Virtually any sort of task can be performed at each of these events.

Furthermore, Upstart jobs are split up into two categories: user session jobs and system jobs. User session jobs run for each user, in the environment of those users, using the system events that are only applicable for that user. System jobs run once per workstation and run in root (or superuser) mode and use the system events of the entire machine. This distinction is incredibly useful for Sleepy for Linux as we will explain. While Upstart is supported by all Linux distributions that are supported by Sleepy for Linux, these distributions have recently announced that they may stop officially supporting Upstart and replace it with a more viable alternative for event-based scripting, in their opinion.

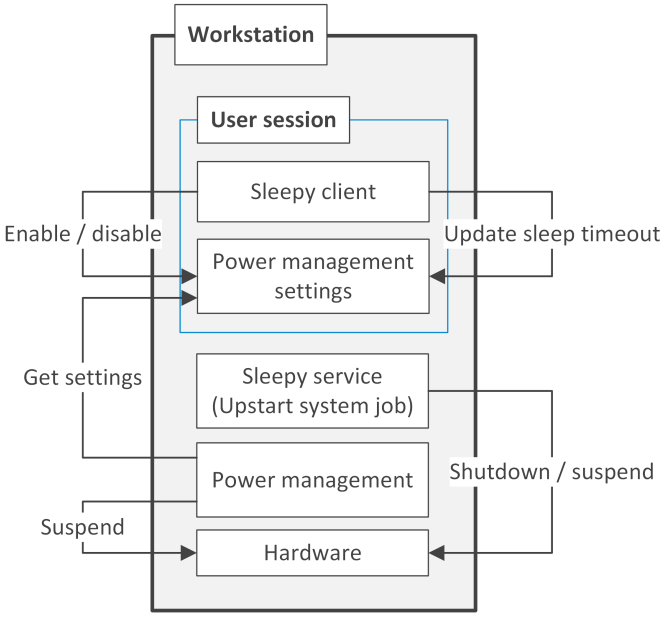

Figure 8: Architecture of the Sleepy for Linux client.

As the concept of Upstart is now known to the reader, we can continue to explain the set of components that Sleepy for Linux uses to fulfil the intended functionality. The Sleepy service is started by an Upstart system job as soon as the workstation is powered on, which is an Upstart event. First of all, this service informs the server that the workstation has been booted. Secondly, the service enters a loop that checks for pending commands at the server. When a shutdown or suspend command is received from the server, the service executes these commands on the hardware of the workstation immediately, as shown in the figure. The power management of the workstation, which is already installed and available on the supported distributions, retrieves the power management settings from a user when this user is logged on. As we have explained, each user has their own power management settings and these are kept within the user mode of the respective users on the workstation. The power management will suspend the workstation after a certain period of inactivity, based on the obtained settings. It is important to leave this responsibility to power management rather than manually accomplishing this by checking if the machine should be suspended, since the power management knows more about the system than the components of Sleepy for Linux. In the case the workstation is performing a backup, it may not be wise to suspend that workstation, which can be known by power management.

The other components of Sleepy for Linux reside in the user session, which means that these components are only available when a user is logged on. The Sleepy client is started by a so-called Autostart file which starts the application in the background at the moment that the user logs into their desktop environment. The Sleepy client has two main threads. One of these threads sends activity data to the server every few seconds and obtains the sleep and idle timeout as a response. When the sleep timeout returned by the server is different from the current value, the Sleepy client will update the sleep timeout by changing the power management settings as well. The other thread controls the user interface of the application, namely the tray icon and several menus for information about the application and disabling the framework. This thread of the Sleepy client will also enable or disable the power management settings, which is performed when the user disables or re-enables Sleepy for Linux through the user interface. Disabling the power management settings will therefore stop the workstation from automatically suspending. An Upstart user session job is also deployed to catch certain events in user session mode which are sent to the server. We will describe next which components send exactly which events.

B. Event detection

The different events that should be sent by the client software components, in order to construct the activity data, will be described now. Figure 9 shows an example work flow of two users working on a single workstation, and all of the events that are generated by Sleepy for Linux. The server is informed by the workstation when that workstation is started and shut down, with the use of status updates. The server also receives status updates when a user logs in and logs out. After a user is logged into a workstation, the server will receive a status update once every few seconds, which indicates whether the user is active or idle. When a user is not logged on at a workstation, the user is considered inactive.

Upon suspension of the workstation, a sleep status is sent to the server. From that moment on, all the users that were active on that workstation are considered to be sleeping as well, until the workstations sends a resume status to the server. Furthermore, when a workstation abruptly shuts down, it may happen that the shutdown status of the workstation is received and the logout of some client may go lost. In this case, the framework will identify that this client has indeed logged out.

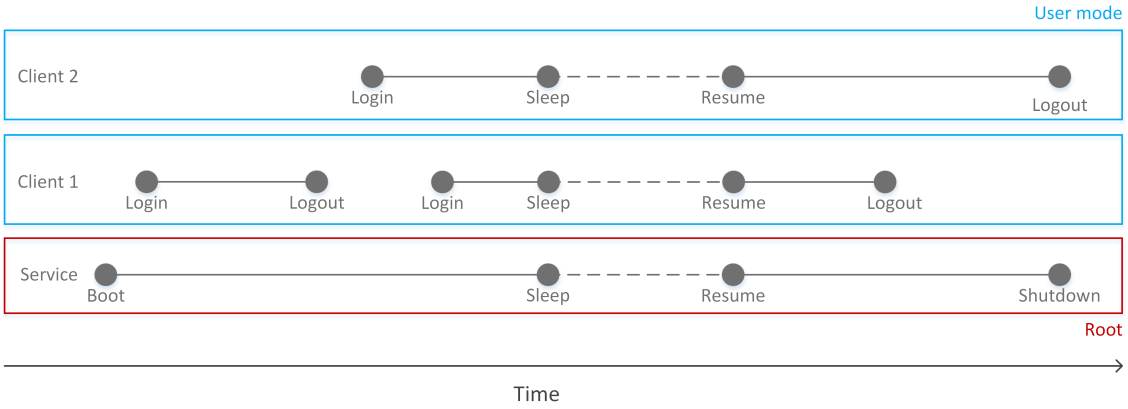

Figure 9: System events generated by Sleepy for Linux.

Figure 10 shows which components are responsible for reporting the various events. The events can be separated into two groups. One group of events update the workstation status and the other group is responsible for updating the client status. When any of the events occur, a call to the server is made to inform the server about the status update for either the workstation or a specific user.

The two components at the bottom left of the figure update the status of the workstation. The Sleepy service is an Upstart system job that updates the status of the workstation as soon as it is launched, which happens when the workstation is powered, to inform the server that the workstation is booted. When the workstation is shut down the Upstart system job sends a status update to the server indicating that the workstation is powering down. Whenever the workstation is being suspended/hibernated or resumed/thawed, the power management has an event system alike Upstart that is used to send a status update to the server, respectively a "sleep" or "resume" status. This updates the status of the workstation to either being asleep or powered, the latter being the case when the workstation was resumed. Note that hibernation may only occur manually by the user, since Sleepy for Linux only allows power management to suspend the workstation automatically.

The two components that only exist within user sessions are responsible for updating the status of the client. The Sleepy client is responsible for sending the login event to the server, which makes the server aware of the fact that the user is currently starting their session on that workstation. Moreover, the MAC address of the workstation has to be sent along with the login status, so that the server knows at which workstation the user is logged in at the moment. The second component that runs in user mode is an event script which is an Upstart user session job, which will be started on the event that the current desktop session is ending, which indicates a user is logging out. The script that is called will simply propagate the logout status to the server, as depicted in the figure.

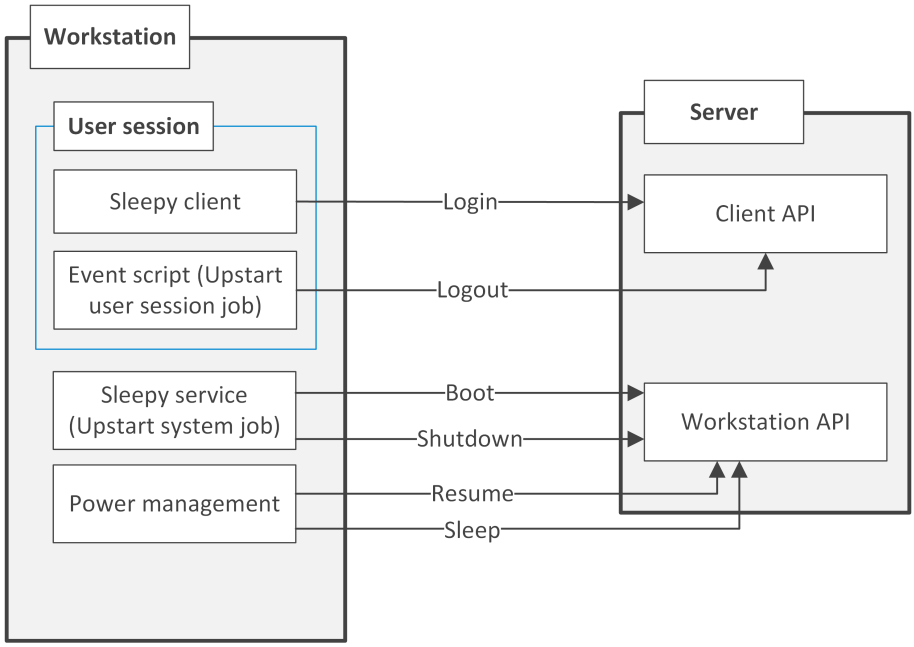

Figure 10: Responsibility of each component with respect to propagation of events.

C. Determining user activity
The X11 screen saver extension library is used to determine the current level of user activity. This extension returns the amount of time that have passed since the user has last shown activity, such as moving the mouse or pressing a key on the keyboard. When no activity has been detected for a certain amount of time, the user is considered to be idle, determined by the idle timeout. Otherwise, the user is considered to be active. This status is determined by the Sleepy client component and sent to the server at an interval of five seconds. When the Sleepy client sends the status to the server, it receives the idle timeout and sleep timeout of the corresponding client from the server. The idle timeout determines the number of seconds that must have passed before the user is considered to be idle. When either of these values change, the Sleepy client component updates the respective value on the workstation.

D. Identification and registration
Every Sleepy for Linux client, which runs within a user's environment, must have a Universally Unique Identifier (UUID). With these identifiers, activity data can be collected anonymously, while still making it possible to group the collected activity data per client. The UUID is obtained the first time the Sleepy for Linux client is started through a registration process. Whenever the Sleepy for Linux client is started, it checks whether it has already received a UUID. If it has not received, it will try to obtain one. The UUID is obtained from the server, which generates all the identifiers, by making a request to the corresponding endpoint. This request must contain the MAC address of the workstation and the user name of the user currently logged on. The user name is sent only for testing purposes and will not be sent in the final version of Sleepy for Linux, since this could void the anonymisation of the data. When the server receives the required information, it will store this data and return a new, unique, UUID for the client. The client then stores this UUID in a file on the workstation, because the UUID is needed for making requests to several of the available endpoints. The UUID is a random string of 36 characters. If the server cannot be reached for registration or the file in which the UUID is stored fails to open, or the UUID from the file is malformed, the client will immediately abort, since it cannot provide any meaningful functionality without having a UUID for the current user.

E. Remotely executing commands
Workstations that run Sleepy for Linux can execute commands that can be called upon them remotely. These remote commands can be sent to a workstation through the exposed interface for workstations, which is explained in the server architecture. When a command is placed upon a certain workstation, the workstation does not directly receive the command. The command is temporarily stored on the server so that the workstation can fetch it. The Sleepy service component on workstations regularly checks if any commands are awaiting for the workstation on which the service is running, by sending a request to the server. If the service receives a command from the server in the reply of this request, it executes the specified command on the workstation. When the server returns a pending command to the workstation, the command is removed from the server. Also, when the server receives multiple commands for a single workstation, the last command that was received overwrites a previously received command. This way, commands do not accumulate on the server when the commands are not being retrieved by the service. Otherwise, when many commands have been sent, it would not be possible to use the workstation anymore, because it will execute those commands every time the service starts running.

Furthermore, we perform a check so that commands have to be executed within a certain time frame. If a workstation is ordered to shutdown while it is already off, and it is not powered for the next hour or so, then the command becomes outdated and is no longer considered as the pending command for the workstation.

F. The user interface
While Sleepy for Linux is running on a workstation, a tray icon is displayed in the system tray of the desktop manager of that Linux distribution, to show the user that the framework is running. The tray icon also allows the user to interact with the Sleepy for Linux client, be it that the functionality at this moment is limited for user interaction. When the user clicks on the tray icon, the corresponding tray menu is displayed. From this menu, the user can get more information about Sleepy for Linux, by clicking on the *About Sleepy* option. This will spawn a window that displays information about Sleepy for Linux such as the version of the software and a few links to the website of Sleepy for Linux, where information about the software licenses, of the software used by Sleepy for Linux, is shown. The user can also choose to disable Sleepy for Linux from this menu using the *Disable until reboot* option. The process of disabling Sleepy for Linux is explained in more detail in the next section. When the application is disabled, the tray icon are faded out to reflect that the application is disabled. Furthermore, the tray menu will have the disable option replaced with the *Enable* option, which allows the user to re-enable the framework when it is disabled. See the first part of *Appendix F: Screenshots* for images of the Sleepy for Linux client application running on Ubuntu 15.04 with the Unity window manager.

Implementation wise, Sleepy for Linux uses the GTK library for the construction and interaction of all the interface components, and for displaying the tray icon. However, displaying a tray icon consistently on the large range of window managers that are supposed by Sleepy for Linux proved to be very difficult, due to the underlying differences between these window managers. With GTK alone, or another library, it was not possible to display the tray icon in a reliable and consistent way across window mangers. Therefore, an additional library, namely the libappindicator library, is used, which corrects this problem for all of the supported window managers.

G. Disabling Sleepy for Linux
It is possible for the user to disable Sleepy for Linux by selecting the corresponding option from the tray menu. When the user selects this option from the tray menu, a window is shown where the user is asked why they want to disable Sleepy for Linux. Here the user can choose any of the predefined reasons, or the user can supply a custom reason. When the user clicks the "Apply" button, the specified reason is sent to the server and Sleepy for Linux is temporarily disabled. This means that Sleepy for Linux disables the power management of the workstation temporarily, which is responsible for putting the workstation to sleep for being inactive for a time exceeding the sleep timeout. Disabling the power management means therefore that the workstation will not be put to sleep any longer for being inactive. The effect will last either until the user logs out and back in again, reboots the workstation or manually re-enables Sleepy for Linux.

H. Obtaining the MAC address
For identification purposes of the workstations, on which users are working, a MAC address is appended to the data that is sent to the web server which contains the system events. However, this address is linked to a specific network interface of the workstation, which makes it more difficult to identify a workstation. Since multiple network interfaces might be available on a workstation, the address of one of the interfaces must be chosen as the identification address for the workstation. To make a choice from one of the available network interfaces, Sleepy for Linux looks in a special directory, available on the Linux operating system, where a file is located for each of the network interfaces. The files in this folder are named according to the network interface to which the file belongs. Sleepy for Linux then tries to find a file belonging to the Ethernet interface. If such a file cannot be found, the files are traversed in the order in which they appear in the folder, and the MAC address from the first available interface is used. When Sleepy for Linux cannot find any available network interfaces, the address of the local interface is used to identify the workstation. This method allows Sleepy for Linux to obtain a MAC address to identify the workstation, and will result in the same address being used for the identification, when the MAC address is redetermined. However, if the network interface, whose MAC address is used, is no longer available or has its address changed, the address used by Sleepy for Linux will also change. Furthermore, this means that it is possible that workstations in the network use MAC addresses from different adapters, which means it is possible for two connected workstations to use the same MAC address.

4.2 User testing

This section will describe the execution of the testing phase that has been performed for Sleepy for Linux, particularly in the setting of analysing user acceptance, and will also explain how the user acceptance can be conceived from the results that have been obtained from user surveys and user interviews.

4.2.1 Execution of testing phase

The testing phase itself will first be described in detail. In first instance, we have accumulated 8 testers in total, of which 7 have used their workstation regularly in the testing phase. The testing phase lasted officially from the 21st of June to the 6th of July in 2015. However, most of the testers have been using the framework before this time frame for a few days up to two weeks, which was helpful for debugging the framework as much as possible and has contributed to the quality of the investigation into usability issues.

The testers mainly used Sleepy for Linux on desktop devices. Two testers have indicated that they used Sleepy for Linux on their laptop devices, which could potentially lead to false results of our research. Laptops will probably have a higher percentage of activity, since laptops are left unattended less often. Because laptops are used at different locations than desktops, where it is not always possible for the user to leave the laptop at that location, when the user is away. Therefore, when a laptop is not being used, the lid is usually closed, which results in the machine being suspended. However, these two testers have indicated that they used their laptops mainly as if it were a stationary desktop, which should not create any problems when the data is analysed. The test audience consisted of research graduates and students of the Computing Science department with the exception of one tester who is employed at a governmental body elsewhere in the Netherlands.

One fact to note is that the software was only tested on single-user devices, and not on networked workstations. However, we expect that, as explained in the implementation, the software should work as flawlessly on these systems due to transparent architecture design.

4.2.2 Evaluating user acceptance

As described, we will combine user surveys and user interviews in order to draw a valid conclusion about the level of user acceptance of the Sleepy for Linux framework. These two components are described in *Appendix D: User questionnaire & interview layout*. The users were asked to fill in the questionnaire and make an appointment with us to schedule the user interviews. Of these 8 testers, some were not available to take part in the surveys and interviews. The reason why not all of the testers took part in the interviews is due to the limited time frame available for this project, and the fact that the project ended some weeks into what would normally be considered holiday for most inhabitants of the Bernoulliborg.

We have determined that we will use the System Usability Scale (SUS) for determining the level of user acceptance of the software in conjunction with user interviews. When a tester has filled in the ten questions of the SUS survey, the usability score of that survey can be determined by calculating the scores for all the individual questions. For the questions with uneven numbers (1, 3, 5, 7, 9) the score is the scale position minus one. For the questions with even numbers (2, 4, 6, 8, 10) the score is calculated by subtracting the scale position from five. All of these individual scores are then summed up and the resulting number is multiplied by 2.5. [36] Subsequently, the final score is obtained, which can be used to determine the usability of Sleepy for Linux. To obtain the total score, taking into account all the surveys, all the scores are summed up and are then divided by the number of surveys.

4.3 Data analysis

This section will describe the structure of the activity data that has been gathered by Sleepy for Linux from the workstations of the test users, will quantify this data and explain the challenges that arose when transforming this data into continuous signals. An algorithm will also be described that can be used to calculate potential savings for a specific timeout.

4.3.1 Data description

When a user is logged in at a workstation, and the Sleepy client is running, the current activity level of the user is sent to the server once very five seconds. The obtained activity data is discrete, since the current status of a user is not known at every moment in time, but rather at an interval of five seconds. Besides these status updates, the server also receives updates from the client when certain events occur on the workstation, namely when a login or logout occurs. For the test users of Sleepy for Linux, the sleep timeout has been set to an infinitely long timeout. As a result of this, the workstations will not be suspended by Sleepy for Linux when the workstation is idle. This allows Sleepy for Linux to gather activity data during the entire periods of inactivity of the workstation, which helps identify the full potential to save power. If the workstation is suspended during a period of inactivity, Sleepy for Linux will not be able to determine that the workstation was idle, since no activity data is sent.

It is, however, still possible for users to suspend their workstation manually. When a workstations is being suspended, the server is informed of this through a status update of the workstation. Manual suspension of a workstation does not really influence the potential savings that are observed, since users would normally also perform this procedure and suspend their workstation during periods of inactivity, so this should not cause bias in the observed activity data.

At the finalisation of the project, we have measured over half a million events of clients and workstations together for all users combined. This is a large amount of data that allows for extensive analysis into prospected power savings with respect to the value of the timeout.

4.3.2 Constructing a continuous status signal

Analysing the activity data, obtained from the testers of Sleepy for Linux, requires locating the consecutive periods, during which the user has the same activity level. It is necessary to determine these periods, in order to determine how long each of those periods lasted. This information can then be used to answer the research questions regarding the potential savings.

The activity status of a user is only reported every five seconds, so the status signal at this point is a discrete function for which the points are on average about five seconds apart. What we actually want to achieve is to transform these discrete signals into continuous signals that approximate the activity status of the user at any given point of time, not just the points of time on which a specific status was received. This can be achieved in the following way. Given two measurement points, say A and B, which are a certain amount of time apart. If A has been recorded just before B, then in the period of time between A and B, the user will be assumed to have the same status as the activity status that was recorded at point A. This is a simple method for interpolating the data points to generate a continuous signal.

However, the length of the segment between the data points has to be considered. Due to network latency or intermittent connectivity issues, the data may not reach the server in time and may take longer than five seconds to reach the server. This may be true especially for devices which are connected to the Internet over a wireless network, where intermittent connectivity degradation is very common. [6] This creates both small and large "gaps" in the data where no activity data is recorded for a certain amount of time, significantly longer than the described five second period. For small and large gaps, the causes can respectively be appointed to network delays or routing mistakes, and a temporary disconnection between the router of the network and the workstation which is connected to that network.

Gaps that are larger than five seconds between consecutive updates need to be handled appropriately. If two consecutive measurements are within ten seconds apart, we consider it as a legal sequence of events and interpolate these points with our simple method of interpolation. If the two points are more than ten seconds apart, then there is a gap at which we cannot be sure that either the status of point A or B can be used to fill in the interval between these two points. In the next section, when the algorithm for potential savings is discussed, gaps will be further explained.

It is also possible that an activity update of the user is received after a shutdown or log out update was received, which is logically impossible, since a user can't be either active or idle after they are logged out. This can happen when the Sleepy for Linux client is not terminated yet, after the shutdown or logout update was sent. However, this issue is easy to resolve, as we can simply ignore all activity status data sent by clients that did not occur between a login and logout. Furthermore, the server will actually not even save these events to the database, since it knows that the current status of the user is that it is logged out, and therefore no activity status data by that user is regarded as valid by the server.

4.3.3 Algorithm for potential savings

When all of the activity data has been obtained from the testers of Sleepy for Linux, a method is needed to determine the potential savings for the entire data and for each user's data individually, which takes as parameters a value for the sleep timeout and the energy consumption levels of a workstation during idleness and during suspension. A simple algorithm is proposed that investigates all of the periods of inactivity and sums up the parts of these inactivity periods that extend the sleep timeout that has been provided. This results in the following pseudocode for the algorithm:

$time = 0$
for all periods where user is idle **do**
  if length(period) $>$ timeout **then**
    $time \mathrel{+}= length(period) - timeout$
  end if
end for
$savings = time * (E_{idle} - E_{sleep})$

The requirements for this algorithm are clear. It should be possible to identify all periods inactivity for a specific users. Besides uncovering these inactivity periods in the vast amount of data of each user, the lengths of each individual period must also be determined. To determine the lengths of the inactivity periods, the activity data is sorted on the time stamp at which the data was received. The algorithm will then iterate over every status update received from the user, and when the status is changed to idle, that time stamp is stored, which marks the beginning of the inactivity period. The algorithm then continues, and whenever the status changes to anything other than idle, the period of inactivity ends. The time stamp of this status is then used to calculate how long the period of inactivity lasted. This is alike the interpolation algorithm that has been described previously.

It is also possible that status updates are missing, which causes gaps in the recorded periods. Whenever such a gap is encountered, which is determined by checking if the total time between the current status update and the next is less than ten seconds, the end of the period of inactivity is marked by the time stamp of the last status update that indicates that the user was still idle. When the status update, that was received after the gap, also indicates that the user was idle, this will be interpreted as a beginning of a period of inactivity. This way, the gap is not included in the total length. We do not make any assumptions about the status of the workstations and users during these gaps, so that our results are not biased to have a higher total inactivity time than actually occurred.

For each of the determined periods, during which the user was inactive, the length is compared to a sleep timeout that can be provided to the algorithm. Whenever the length is greater than the specified sleep timeout, there is a potential to reduce the power consumption of the workstation. However, power can not be saved during the entire period of inactivity, since Sleepy for Linux first has to wait for the amount of time specified by the sleep timeout, before the workstation can be suspended. Therefore, the sleep timeout has to be subtracted from the total length of the period of inactivity.

After this process is complete, the total amount of time that the workstation could actually be suspended instead of being idle is accumulated in the running variable of this algorithm. To deduce potential power savings from this information, the energy consumption levels of the workstation need to be known. We have argued in the problem analysis that the power used by a sleeping workstation is negligible compared to the power consumption of an idle powered workstation. The total savings are therefore the amount of time multiplied by the difference in these two consumption levels. This is the algorithm that will be used in the *Results* section to construct activity graphs and figures for potential savings.

4.4 Summary

In this section, the implementation of the Sleepy for Linux prototype has been thoroughly described at various levels of granularity. The general architecture has been explained and the implementation and functionality of each component has been detailed. The exposed end-points for the APIs for both user and workstation manipulation and the way in which components interact with these end-points have been described. The two main components of Sleepy for Linux on workstations, namely the client and the service, representing respectively the user and the workstation, send certain events to the server which can be used to determine their status. We have explained how Upstart and Autostart were useful tools for executing certain scripts, commands and applications when certain system events, such as boot or shutdown, were happening.

The user testing phase lasted from the 21st of June to the 6th of July in 2015 and consisted of a total of 7 valid test users who were using the Sleepy for Linux framework. The devices that were used by the testers to test Sleepy for Linux on were mostly desktop workstations and laptop workstations used as if they were desktops, which means the data that has been gathered is relevant to our research questions, as they are targeted on the workstations of the Bernoulliborg. User surveys and interviews have been taken from part of the users which will provide interesting insights into the level of user acceptance of the application.

Finally, the testing phase was also necessary to obtain user activity data, to make assumptions about the projected power savings, and to find out whether a personalised timeout is preferable over a static timeout. In this section, we have described how the discrete data offered by Sleepy for Linux can be legitimately transformed into a continuous status signal that can be used to determine the potential savings, given an arbitrary sleep timeout and consumption levels of the workstations, using a simple summation algorithm, which requires identifying the idle periods of a user and determining their length.

5 Evaluation

This section evaluates several aspects of the research that is presented. We reflect on the development process of the Sleepy for Linux prototype, and determine the quality of the prototype and which software deficits can be improved upon. We will also discuss in particular the level of security and privacy that is ensured by Sleepy for Linux, and a comparison between Lazy Sleep (Windows) and the Sleepy for Linux prototype will be made to discuss if there is an improvement in terms of security, considering the two applications use a rather different method of communication. The testing phase and the data that has been conceived by this testing phase will be evaluated as well.

5.1 Prototype development and requirements

It has been verified that the developed prototype of Sleepy for Linux is rather robust and that it meets all of the functional and technical requirements that have been presented by the stakeholders. All but one of the non-functional requirements have been achieved as well, this one non-functional requirement is NFR-04 about Portability. While the software supports both Ubuntu and Debian with the window managers Unity, GNOME2, GNOME3, XFCE and LXDE, which is more than enough to indicate that Sleepy for Linux is compatible with the workstations of the Bernoulliborg building, we could not create a single binary package that also worked well on workstations that hosted the KDE, MATE and Cinnamon window managers.

The problem of these last set of window managers is that certain Upstart events behaved differently when these window managers were installed for some reason we are unaware of. Sleepy for Linux tracks the current state of a workstation through several of such events. Whenever these events occur on a workstation, the server is informed of this. For the last set of window managers, Sleepy for Linux does not seem to send "logout" events when it is running on KDE and MATE, and on Cinnamon both the "logout" and "shutdown" events are not being sent. This is a strange issue that will require more research time to solve, perhaps by utilising different Upstart events, so that it will run on every of the listed window managers.

The requirements for this project are not identical to those of Lazy Sleep (Windows). There are several reasons for differences to exist. The main differences are caused by the changed communication protocol and the distinction that we have made between users and workstations in the system. First of all, since the Sleepy for Linux user uses HTTP to communicate data between workstations and the server, and Lazy Sleep uses sockets to achieve this, there are of course some differences. Lazy Sleep can send commands and updates to the timeout directly, while the Sleepy for Linux server has to wait for a user or workstation to poll for updates before the modifications or commands can be sent to the user or workstation respectively. Later on in this evaluation, we will argue the differences in terms of security between Lazy Sleep and Sleepy for Linux.

Secondly, the architecture of the system was changed so that Sleepy for Linux tracks the state of workstations and users separately, where workstations and users are represented by respectively the service and client components in the architecture. This separation has been made since the activity of a user can only be determined when a certain user is logged in, and this call to the library that determines inactivity can only be made from the user mode of that user, not from the root mode in which the service runs. Therefore, a separate component needs to be running in the user mode of every user. Because we also need to execute certain statements in root (superuser) mode, another separate component was needed that runs in this mode.

Furthermore, this architecture extends well to the fact that each user on Linux systems has their own set of power management settings, so that their data is tracked separately from all other users that are using the workstations. Lazy Sleep does not have this separation, and will regard activity data from a workstation as a single entity without making a distinction which user is currently logged on. This means that activity data of different users is mixed on workstations which will not enable external modules to meaningfully use this data, since it does not reflect a single user and can therefore not be used to construct power management settings for a specific user, but rather for a specific workstation. Sleepy for Linux tracks workstations and users separately so that a personalised timeout can be offered to each user. While this does not impact the privacy of the user too badly, since the data is anonymised using unique anonymous identifiers, users are still being tracked and this data is grouped based on their identifiers, so there is a possibility that the data can be linked to specific people by malicious third parties if they are able to find out the identifier of a specific user.

5.2 Activity data and events

As we have described, Sleepy for Linux is able to determine the inactivity time for each user in the user session of those users. However, when no user is logged into a workstation, then no activity data is being logged from that workstation. As a result of this, it is not directly possible to measure how long a workstation is left powered while no users are logged into that workstation. During this time that the workstation is not being used, but is powered, the workstation consumes power unnecessarily. It is still possible to find out if a workstation is currently powered by looking at the composition of boot and shutdown events in the activity data of that workstation, as well as sleep and resume events. Furthermore, powered workstations will ask for pending commands every few seconds, which is a type of activity that can be used to determine if a certain machine is still running while no users are logged into that machine. The workstation could then be shut down using a remote command if the situation persists for a long amount of time. In the *Future Work* section we will comment on this subject again.

A different issue that occurs for multiple users on the same workstation, is that the current prototype does not track the activity level of a workstation itself, but rather the activity of the individual users. Therefore, when multiple users are logged in at the same workstation, but only one can be actively using the workstation at any time, Sleepy for Linux will record that all those users are inactive since no input events are being recorded for those users. This leads to an increase in the calculation of the potential savings, since for all of these users power can be saved.

However, since these users are actually using the same workstation, which is actively being used by one user, this is not real potential for saving power unless all users on that workstation are idle at some point in time. For our research, however, this is less of a problem, since the testers of Sleepy for Linux work on single-user systems, and they will be logged on for almost the entire time that the workstation is powered on. Furthermore, when students and staff use the workstations of the Bernoulliborg, they will usually log off entirely before leaving the workstation, especially if another individual is going to make use of the workstation in the mean time. Most of these workstations will not even support multiple users being logged in at the same time. Therefore, the impact of this issue will be minimal.

Another aspect that affects the activity data is the fact that the state of a user is not known at all times but rather is a discrete signal. The clients inform the server of the current state of the user every five seconds. While it is not that much of a problem that the state is not known at all times, the updates sent by the clients could take much longer to actually reach the server, due to network delays. It is also possible that an update is lost, or that the client is unable to send any updates, because there is no active network connection. In some cases, this leads to gaps in the data in which we cannot make any assumptions about the state of the user, so we cannot take this data into account in the analysis of the results.

5.3 Ensuring user privacy

The state of a workstation and the state of the users working on that workstation are currently tracked separately by Sleepy for Linux. The system consists of two parts, one that tracks the state of the workstation and the other tracks the state of a single user. Both of these send data to the server independently of each other, and the data is also stored separately on the server. This separation means that the activity data is directly linked to a specific user, rather than to a workstation. However, since the user is identified with a unique identifier, which is randomly generated by the server, it is not possible to directly trace the gathered data back to a specific user. Furthermore, the communication between the client and the server happens in a secure way, which makes eavesdropping very difficult.

However, there are several ways for the privacy of the user to be broken regardless. The identifiers are stored on files in the user space of the respective users. Therefore, if an individual has access to these files, either because they are the owner of those workstations and can see the files of other people, or someone gains access in case someone does not log out of their workstation, they can obtain the identifier of those users. If in addition to that, they also have access to the activity data of Sleepy for Linux through some external module, (though this is unlikely to be the case in the first place) then they can view the data of specific individuals who use Sleepy for Linux. For some of the functionality regarding remote commands, we also store the last MAC address of the workstation that a user logged in from. If a certain user only logs into a certain workstation, and the MAC of this workstation is known, then third parties with access to the data could view the data that was sent while specific users were logged into that MAC address, and uncover the activity behaviour of a user that way. All of these methods are very unlikely and require access to two sources of information that are rather secured and hard to access in the first place.

5.4 Communication layer and security

One of the research questions will be answered in this evaluation based on the implementation details that have been discussed and earlier comments on the security of Lazy Sleep compared to Sleepy for Linux.

Sleepy for Linux uses the HTTP protocol for the communication layer between the workstations and the server, as well as between the servers and the external modules. The server runs a web service, consisting of several REST endpoints, that can accept incoming requests from the workstations. Workstations send requests that includes their activity data to the server. As a reply to these requests, the modifications in power management settings and remote commands are propagated back to the workstations. This approach couples the communication between the workstations and the server, which means that the server cannot directly reach the workstations, but can only send updates to the workstation as a response to a request made by a workstation. Lazy Sleep uses IP sockets to transfer information in both directions between workstations and the server, meaning that the communication is not being coupled, so that the workstation can send more or less requests to the server and vice versa. Both the client and the server can send instructions directly to the other, which means there is no delay when remote commands or modifications to power management settings are sent to workstations.

While the coupled communication of Sleepy for Linux means that the server cannot directly send instructions to a workstation, it also increases the security of the overall system, since it is possible to enforce authentication at the server, when instructions are sent to a workstation, since the server has a security certificate that is used to verify the contents of the replies that are being received by the workstations. When IP sockets are used, as is done by Lazy Sleep, it is much more tedious, and technically involved, to enforce authentication on the messages, since anyone could potentially send instructions to a workstation at any time, not just as the reply to a request initiated by the workstation itself. Authentication is therefore more important for Lazy Sleep than Sleepy for Linux, while it is implemented for Sleepy for Linux, but not for Lazy Sleep. This is a serious security loophole.

The use of the HTTP protocol also has the advantage that no additional ports have to be opened on a network for the communication to be successful. Opening additional ports on a network could potentially lead to a less secure network, since attackers gain more potential entry points to the devices within the network. Since many networks already allow HTTP traffic, which is on the ports 80 (insecure) and 443 (secured), no changes have to be made in the network configuration in which Sleepy for Linux is active. Furthermore, workstations could be located in different subsections of the same network, or they could be located within entirely different networks than the server and each other. In these cases, the use of IP sockets requires the user to forward ports on the network, which is not necessary when the HTTP protocol is used. Forwarding ports is a rather technical process, and we cannot expect that the users of Sleepy for Linux, which may be non-technical, understand how this process works. The installation of Sleepy for Linux should require as little manual effort as possible, which cannot be accomplished if IP sockets are used.

Since the communication between workstations and the server should be secure, some form of encryption should be used. There are many solutions readily available to secure the communication that work on top of the HTTP protocol. This makes it relatively easy to secure the communication between the workstations and server, and between the server and external modules. Sleepy for Linux uses the Transport Layer Security (TLS) protocol to secure the communication. Besides encrypting the data that is being sent, the protocol also enables the clients to verify the identity of the server, through the use of certificates. This makes attacks such as an impersonation attack much more difficult for malicious third parties. When IP sockets are used instead, it is also possible to transmit data securely, with the use of encryption, which is done by Lazy Sleep. However, this requires much more development effort, and also does not provide the possibility to verify the identity of the server.

Weighing all the benefits and risks of using HTTP communication over the use of IP sockets, we can conclude that, while IP sockets have the benefit of sending data directly to certain workstations, allowing for faster updates, this update is only delayed by a few seconds when the HTTP method of communication is used where workstations and clients poll the server for updates every so often. The benefits of enhanced security and ease of implementation of security and authentication of HTTP communication are much more important and relevant for Sleepy for Linux. We conclude that Sleepy for Linux indeed does have a better level of security than the Lazy Sleep framework for Windows.

5.5 Testing phase and testers

The activity data that is gathered by Sleepy for Linux and that is used to answer the research questions, is not obtained from actual workstations of the University of Groningen, with the exception of a single unmanaged workstation present in the Bernoulliborg. Due to strict regulations, it is not possible to deploy Sleepy for Linux on managed workstations and therefore we cannot gather activity data from the users of these workstations. Instead, we have asked several students and staff members if they wanted to install Sleepy for Linux on their personal workstations. During the testing phase, we have gathered activity data from these workstations, which is also used to formulate an answer to the research questions on power consumption and usability.

The usage patterns of these user devices will likely differ from the patterns of the workstation at the University of Groningen, and this difference will also be reflected in the obtained activity data. However, since these devices belong to the people who also use the workstations of the University of Groningen, we believe that the usage patterns of these personal devices will show similarities with the usage patterns that would otherwise have been obtained from the workstations of the University. These personal devices will probably be idle less often, though, which can result in a smaller potential for saving power. However, if opportunities are discovered for the potential to save power, it will probably mean, even more so, that power can be saved on the workstations of the University of Groningen. The same is true in case evidence is found that personalised timeouts offer a benefit over static timeouts for the test audience.

Finding students and staff members who were willing to install Sleepy for Linux on their personal devices proved to be a rather difficult process. Many were not comfortable with installing Sleepy for Linux on their devices, since it gathers data about the usage of their device. This discomfort was also caused by the fact that they were not able to verify exactly what information is actually collected. Because of this, the amount of testers, from which the activity data is collected, is rather low. In the future, it may be wise to improve upon the platform website that has been made so that it contains more detailed information what kind of data is captured by the application. The software could also be made open source, so that prospective users can look at the source code of the application to find out what it does exactly.

While the number of testers is limited, we have argued that this amount of testers is enough to cover the usability studies, and we believe that we have obtained enough testing data to analyse in order to answer the research questions about power reduction.

5.6 Network presence and sleep proxies

Not all of the workstations that are left powered on by the user when they do not use it can be suspended. In some cases, it is necessary that the workstation remains powered on, because the user wants to remotely access the workstation or may need to interact with a service running on the workstation remotely. In such cases, the workstation could be idle for most of the time that it is on, but it is actually required that the workstations remains on, because if the workstation would be suspended it would lose its network presence. When this happens, it is no longer possible to reach any of the services that were running on the workstation. This could pose a serious problem for users relying on the fact that those services are available at any moment on their workstation. In these cases, the use of Sleepy for Linux would be inapt.

There are various techniques that avoid this particular problem, and with these techniques it is still possible to save power by suspending these workstations. One of the techniques, which was already mentioned in the *Related Work* section, is the use of a proxy server. Such a server will maintain the network presence of any workstation within the network, when that workstation has been suspended. All the incoming network data that is destined for a suspended workstation will be routed to the proxy server. The proxy server can then handle the request on behalf of the workstation or it might need to wake up that workstation that should handle the request.

However, the proxy server itself also contributes to the total power consumption, and this should be taken into account. If the proxy server uses too much power per workstation for which it maintains network presence, it would be more cost effective to just keep the workstations up and running. It should also be noted that, when only a few workstations require such a technique to be implemented, it might not be worth deploying such a technique, since it can be quite complex, especially when the solution should be highly efficient. Instead, we should enquire with the users of such workstations if it is really necessary to have their workstation powered almost the entire time just to run these services, since this is arguably not an example of responsible power usage by the employees, rather a counter example.

5.7 Frequently updating sleep timeout

The Sleepy for Linux system will continuously monitor user activity and provide the users with updates on their power management settings through external modules. The question arose whether this was an effective method of providing remote power management. Is it not enough to simply calculate the personalised timeout once per user over a couple of weeks of activity data, rather than infinitely monitor the workstations and the users that are using them? Perhaps, once this sleep timeout has been determined for all users, there is no longer a need for a remote framework that controls the workstations, which would also reduce power consumption. Do we even need a whole framework to decide the timeout of users, can we not make an educated guess or can users not decide their own dynamic timeout?

There are obvious and subtle drawbacks to this issue. The workstations are controlled by humans who display a certain type of behaviour when they are using their workstation. However, there is no reason to assume that this behaviour will always be the same when looking at extended periods of time. The usage patterns of a user could change, which could mean that the set sleep timeout causes too much inconvenience for the user, or it could be the case that much more power could be saved, if the sleep timeout was adjusted. Furthermore, from testing experience, we have determined that users are very bad at estimating when and for how long they will be idle during their working day. Letting users pick their own sleep timeout will not generate the right values as a sleep timeout for these users. Users also tend to set the timeout higher than it needs to be to remove all inconvenience that they may experience from power management.

The final point that is brought up is the fact that we can determine the sleep timeout for all users once and that this would give a decent power savings potential. However, the University of Groningen has a large amount of users and new users are constantly being added to the network. It is not possible to manually track the activity of all of these users or manually provide a valid sleep timeout for each of these users. The presence of a continuous remote power management framework is necessary to ensure each user has a timeout that is personalised to their needs. Furthermore, a continuously present framework allows for remote commands to be executed on workstations.

6 Results

In this section the results of our research will be presented, and the three remaining research questions will be answered based on the obtained results. Several users have been using Sleepy for Linux, providing us with activity data about their activity behaviour. This activity data is processed and is used to determine if there is a potential to reduce the power consumption of workstations. The data that was obtained from any user will also be compared to the data of other users, to answer whether it is sufficient to have one sleep timeout for all the users, or if every user should have their own sleep timeout. Furthermore, the data obtained from the user surveys and interviews will be used to determine the level of usability (or user acceptance) of the Sleepy for Linux framework.

6.1 Potential power savings

The gathered activity data is used to determine if there is a potential to reduce the power consumption of the workstations, which is the focus of the first research question. The activity data is processed to discover all the periods during which workstations were left idle. For each of these periods, the duration was calculated, and the results are visualised in a histogram, which is shown in Figure 11.

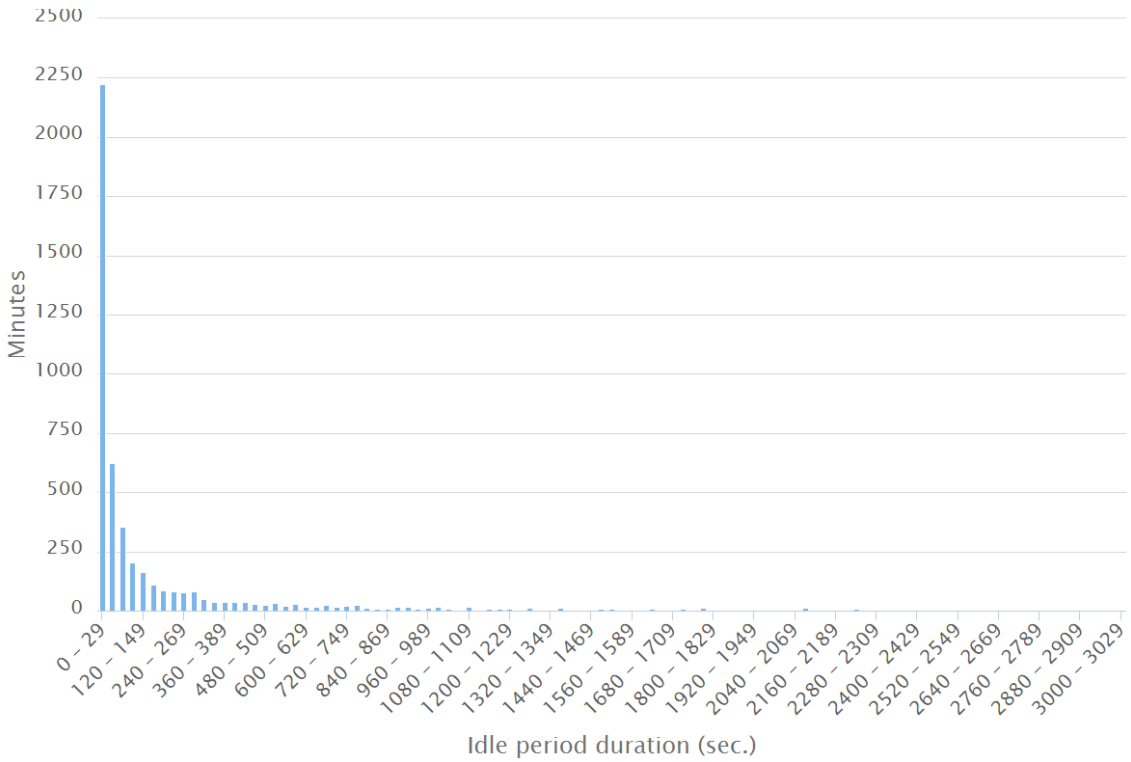

Figure 11: Histogram of the lengths of the idle periods for all testing data of all users.

The histogram shows how many times an idle period of a certain duration was recorded for the combined data of all test users. Because there are many idle periods, and many of these idle periods have a different duration, the data has been grouped in intervals of thirty seconds. For this same reason, the histogram also does not show all the intervals in which idle periods were observed. There are some idle periods observed with a duration that was longer than the maximum duration shown in the histogram. However, due to limited space and relevance, these points have been omitted from the histogram. If these points were included, it would have been hard to see where they are located, because the total number of idle periods with such a long duration is very small.

From the histogram it becomes clear that the duration of most of the idle periods is very short. These short idle periods are not a good candidate for reducing the power consumption of the workstations, since that would require a very low sleep timeout, which would drastically increase the user inconvenience. However, there are also quite a few idle periods with a longer duration, which indicates that quite a few workstations are left powered on, while they are not actively being used. These idle periods with a longer duration are the most suitable candidates for the reduction of the power consumption of the workstations, because power can be saved during those periods, while not causing the user too much inconvenience. The fact that there are quite a few of these longer lasting idle periods observed in the activity data, indicates that there is indeed a potential to reduce the power consumption of the workstations, and with that, to save power.

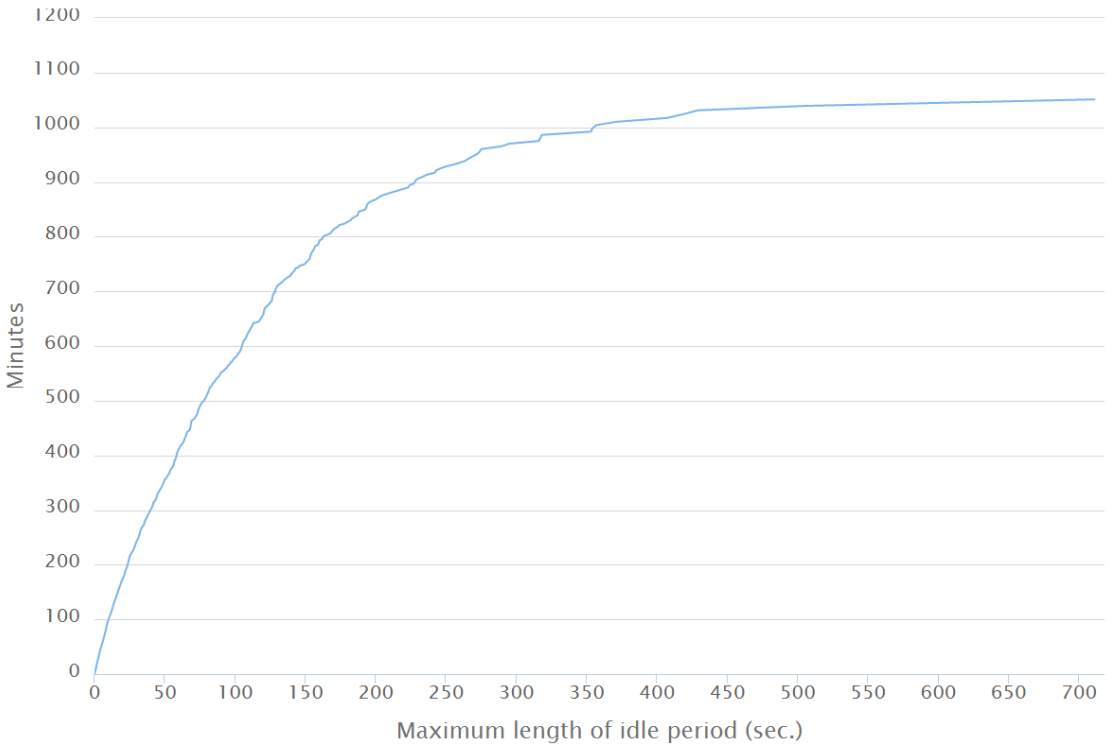

Figure 12: Graph showing the accumulated savings for maximum lengths of the idle periods.

To verify this more accurately, another graph has been constructed which is shown in Figure 12. The graph in this figure shows the accumulated savings for a specific length of an idle period. This means that the y value for a certain length on the x axis displays the savings for that length plus all the savings for all the lengths that are shorter. The steepness of the curve thus indicates how much a certain idle period length contributes to the total savings that can be achieved.

From this graph it becomes clear that even though there are many idle periods with a short length (a minute or less), these do not contribute greatly to the total savings. The idle periods with a longer duration contribute to a significant amount of power waste, and this is a useful property, because it is much harder to save power during the short idle periods. Suspending a workstation during those periods usually lead to a great deal of inconvenience for the user. However, since this is only a small portion of the total savings that can be achieved, this is not a problem, and significant power savings are therefore possible by suspending the workstations during the longer idle periods.

6.2 Comparison of activity data

The idle periods, used to determine the potential savings, are also used to answer the research question about how the sleep timeout should be set. Together with the algorithm that was presented in the *Realisation* section, the potential savings for every user are calculated separately. The algorithm is applied on the idle periods of a specific user, using several different sleep timeouts. These sleep timeouts were picked because of the fact that these users used these timeouts by themselves or that these were often timeouts that were picked as global static timeouts for institutional networks. The savings that are calculated indicate how many minutes the workstation can be suspended, given a certain sleep timeout, which is also shown in minutes. We did not calculate the savings in terms of energy, since we do not know the consumption levels of the different devices that have been used by the testers. The calculated savings for every user are shown in the table below.

Sleep Timeout							
0	19905	1820	1028	727	3829	5307	65
1	19172	1597	463	426	3085	4804	6
2	18518	1424	298	321	2672	4466	1
5	16771	1028	112	186	1952	3785	0
10	14368	617	25	88	1359	3104	0
20	10865	264	0	23	716	2195	0
40	6371	55	0	0	148	1291	0
60	3694	2	0	0	30	860	0

The table shows how long the workstation of a specific user can be suspended in minutes, given a certain sleep timeout, using the activity data that has been obtained for that user in the testing phase. The calculated savings show that there are large differences between users. For some of the users, the workstation can be suspended for quite some time when the sleep timeout is set to a high value. This indicates that the duration of most of the idle periods of these workstations are very long.

For other users, the time that the workstation can be suspended decreases very rapidly when the sleep timeout is increased. This happens when the average length of an idle period for these users is very short in comparison. When the duration of the idle periods are short, setting the sleep timeout too high means losing any potential to reduce the power consumption of the workstation, since we can only save power on idle periods of which the length exceeds the value of the sleep timeout.

Because of the large differences in the potential savings shown in the table, it is probably better to set a sleep timeout per user, and not to choose a single sleep timeout for all of the users. When all users are given a static global sleep timeout, either some users are inconvenienced too often due to the sleep timeout being low and not taking into consideration the large amount of power saved by some users, and other users may lose any potential to save power by means of automatic suspension of their workstations due to their average idle periods being very short to begin with. Setting the sleep timeout too short would mean that a workstation is also suspended during many of the shorter idle periods. While this might not be problem for all of the users, since some users need a lower sleep timeout to be able to save power, also enforcing a lower sleep timeout for other users could give rise to usability problems. Users who can also save power with a higher sleep timeout would be inconvenienced much more often with a low sleep timeout, since the workstation would be suspended more often, which means that the user has to manually resume the workstation each time this happens. Therefore, it is probably better to set a higher sleep timeout for these users, while using a lower sleep timeout for the users who cannot save any power otherwise. For some users it might not be possible to save any power, because the workstation is simply not idle very often.

Choosing a sleep timeout means finding a balance between reducing the power consumption of the workstation, while limiting the user inconvenience in relation to the proportion of power that is being saved. The user inconvenience, like the potential savings, is highest when the sleep timeout is set to zero. When the sleep timeout is increased, the user inconvenience will decrease. How fast the user inconvenience decreases depends on the duration of the idle periods of the corresponding user. When the duration of most of the idle periods is very short, the user inconvenience will drop very quickly. Otherwise, this decrease will progress more slowly. However, when many of the idle periods last fairly long, the user inconvenience could still be rather high, but it will probably be less tedious for the user, since the user is not using the workstation for quite some time.

The data shown in the table is visualised in Figure 13, depicted below. The graph in this figure shows the potential savings of all the sleep timeouts as a percentage of the maximum amount that can be saved. This maximum is used as a reference for the other sleep timeouts, and is obtained by setting the sleep timeout to zero. We can see clearly that the slope of these functions is different for each user that has been testing the software. This validates the assumption that users should have a personalised sleep timeout even further. For the yellow line in this graph, a low sleep timeout would be preferred, since no power can be saved for higher timeouts for this specific user. However, for the blue line, we see that this user can also save a considerable amount of power when high values for the sleep timeout are used. Therefore, for this user, a high sleep timeout may be more appropriate, so that the amount of savings by this user weighs up against the amount of times that the user is inconvenienced.

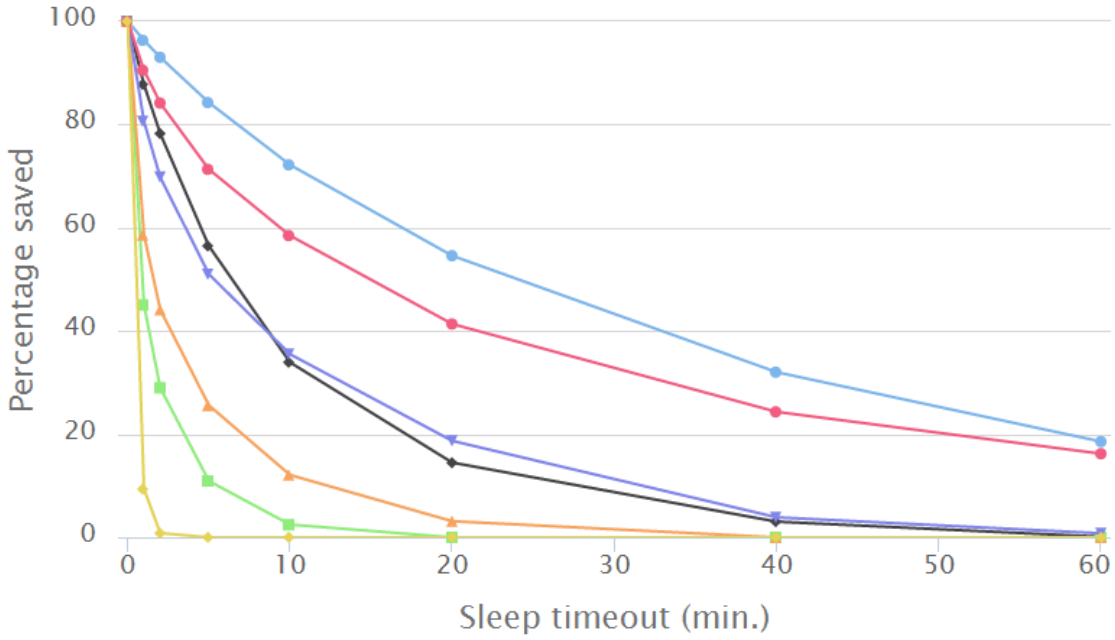

Figure 13: Power savings for different sleep timeouts for all separate users.

6.3 Analysing user patterns

The activity data has also been used to analyse the behaviour of the users. A graph is made for every user, showing how long their workstation was powered on, and how long the workstation was left idle. This is calculated for every hour of every day, and the data from the same intervals is then summed. This gives the total time that the workstation was powered on during the two weeks within an interval, and also how long the workstation was idle during that same interval. This information can be used to determine during which parts of the day the user was idle the most. The graph can also show parts of the day during which the workstations was never powered on in the two weeks. The graphs for all of the users can be found in *Appendix E: User behaviour*.

When looking at the graphs, it is clear that the behaviour of the users varies greatly when time of day is considered. Some workstations are only left idle for a short amount of time, with respect to the total time that those workstations were powered, while other workstations are idle for almost the entire period. This supports the fact that there is indeed a potential to save power, and also supports the claim that it is probably better to set a potential different sleep timeout for every user, instead of using one sleep timeout for all the users.

Apart from answering our research questions, we wanted to look at the possibility to use this behavioural data to identify periods of time during a day where most of the users displayed similar patterns in terms of the percentage of activity. We predicted that during periods such as lunch time or at night, the idle time of the workstations would be much higher or the workstations were simply turned off.

The graphs show that most users have in fact turned off their workstation during night time or are idle during that time. Especially for user 1, whose workstation has never been turned off during the testing phase and only occasionally looks at their workstation for a few minutes, has a large amount of power saving potential here. We could not identify any other visible patterns from the limited amount of testers that we tracked. We do see that some users experience a peak in idleness during what appears to be lunch and dinner time, but there is no way to verify this and it is not present in all behavioural graphs. More research will have to be performed to uncover such patterns in the data. If such patterns can be found, then these can be used by an external module which will remotely suspend the machines during the start of the time frames of these patterns.

6.4 User surveys

All testers of Sleepy for Linux have been asked to fill in a survey. The questions of the survey originate from the system usability score (SUS) and the results will be interpreted using that method as well. The users were asked to rate ten question on a scale from one to five, and the results of the survey are presented in the table below.

Question nr.	**Entry 1**	**Entry 2**	**Entry 3**	**Entry 4**	**Entry 5**
1	4	5	5	2	4
2	2	1	1	1	3
3	5	5	5	4	5
4	2	2	1	1	2
5	3	5	4	3	3
6	1	1	1	2	2
7	4	4	5	4	4
8	2	1	1	2	3
9	4	4	4	4	3
10	1	1	1	1	2
Score	80	92.5	95	75	67.5

The table shows how each of the users rated the ten questions of the survey. While the individual ratings of each of the questions do not give much information, they are used for calculating a score that indicates how usable the system is, as perceived by the user. The resulting score for each of the entries is shown at the bottom of the table. The steps involved in obtaining this score have been explained in previous sections.

By taking the average of the scores of all the individual entries, an overall score is obtained that indicates the usability as perceived by the users of Sleepy for Linux. The average score calculated from the surveys shown in the table is 82. The scores obtained with the SUS range from zero to one hundred, with a higher score indicating a better overall usability of the system. This means that the overall usability of Sleepy for Linux is considered to be quite positive by the users, since the calculated value exceeds the average that is given to most software systems as described by various studies that research the impact of the SUS scores. [35]

6.5 User interviews

The users of Sleepy for Linux were also asked several open questions in an interview, which are not only used to assess the user acceptance of Sleepy for Linux, but are also used to determine if the prototype at this point contains any bugs, misses any functionality, or contains serious usability problems. Sleepy for Linux collects sensitive data about the user, and the interview questions help to discover potential improvements that could be applied to the system, to improve the user acceptance of Sleepy for Linux. The user interview questions can be found in *Appendix D: User questionnaire & interview layout.*

Almost all of the users of Sleepy for Linux indicated that they have used their device in a regular way, and that they have not been more active or idle than usual in their experience. This is important for the obtained activity data, because different usage of the device can give other results regarding the potential savings. If the results were measured during a time where usage can be regarded as regular, the conclusions on these data are more usable than in the opposite case.

None of the users have reported any issues while running Sleepy for Linux, both regarding the usability of the system and the performance of their device. However, many of the users reported that they experienced discomfort with the use of Sleepy for Linux. The main reason for this is the fact that it is not entirely clear what data is actually recorded by the software, and also that it cannot be verified what data is obtained, since it could easily gather more data about the user that has not been described on the platform website. Therefore, it is very important to thoroughly explain to the users what data is recorded, and for which purposes this data is used. It might also be a good idea, as proposed by one of the users, to make the software open source. This allows the users of Sleepy for Linux to look at the source code, and to verify what data is being recorded by it.

The platform website, that was developed to provide potential users with information about the software and to help them install the software, was considered to be beneficial to the use of the system. The users indicated that the installation instructions provided here were clear and easy to perform. None of the users experienced any problems with the installation of Sleepy for Linux. However, it was noted that more information could be provided on the platform website. The information that the users were missing mostly relates to what data is collected by the application and also information about how this data is obtained by Sleepy for Linux. One of the users mentioned that is unclear what the effect of disabling Sleepy for Linux had. It was not clear what the application does when it is being disabled, and which parts of the application are actually disabled.

The users were also asked if they could come up with any improvements for Sleepy for Linux. An improvement that has been proposed is extending the user interface, such that it provides more information for the user about what the program is doing and also shows the user the data that has been obtained by Sleepy for Linux. Additionally, the interface could also provide the user with information about when the workstation has been suspended or shut down, and also the amount of idle time that has been saved by Sleepy for Linux so far for the current user. A final improvement that was mentioned is making Sleepy for Linux available in the software libraries, which would increase the ease of installing Sleepy for Linux even more.

7 Conclusion

This section will conclude our research and give an answer to the research questions, deduced from the results that we have obtained through our research. The following research questions have been defined at the start of this paper:

- Does the activity data of users indicate that Sleepy for Linux will be successful in decreasing power usage, and how is the amount of energy that is potentially saved calculated, based on this activity data?
- Are there significant benefits, in terms of decreasing power consumption, to determining a sleep timeout for each individual user of the workstations, rather than setting a global static timeout for all workstations, no matter which user account is currently logged in?
- Does the architecture of the Sleepy for Linux software attain the security and privacy standards as described, and is there an improvement with respect to the level of security of the Lazy Sleep software?
- What is the level of user acceptance of the software, is the software suitable to be implemented on both student and staff workstations?

The research questions can be separated into two different groups. The first group, consisting of the first two research questions, involve the level of power reduction that can be reached through the use of the Sleepy for Linux framework and a personalised timeout over a global static timeout. The second group involves software qualities of the Sleepy for Linux framework, namely the level of usability and security of the framework.

The methodologies and results of our research into answering these research questions will now be briefly described and the research questions will be answered in the order in which they are presented based on the results.

7.1 Software prototype

Sleepy for Linux is a remote power management framework which will track the activity data of users of the workstations in the Bernoulliborg. It provides personalised power management settings that can be remotely influenced through the framework. The most important of these parameters is the sleep timeout, which influences the amount of inactivity time after which a workstation will automatically suspend. Furthermore, Sleepy for Linux allows external parties with the capabilities to remotely command workstations to suspend or power down. All of these tools are provided to external parties in an attempt to reduce the power consumption within the Bernoulliborg. The Sleepy for Linux framework is constructed in such a way that it can be easily applied on a different network than the one present in the Bernoulliborg and can also be used for workstations that are not located in the institutional network of the University of Groningen.

Requirements for the implementation of this framework have been obtained through requirements elicitation with the stakeholders of the project. These requirements entail the functionality of the system, both functional and technical, and describe the non-functional software qualities that the framework should adhere to.

Since the research questions demand for a prototype of Sleepy for Linux to be implemented, as the data gathered by Sleepy for Linux is essential for the research questions about power reduction, usability studies and security, these requirements were especially important in the process of validating the functionality of the prototype. A working prototype has been implemented that runs on the targeted workstations. Several benefits and risks of the software have been identified throughout this paper and will be discussed further on in this conclusion.

7.2 Power savings potential

Workstations in the Bernoulliborg are often left idle while no employee or student is currently using the workstation. This means that power is wasted, as the workstation could have been suspended to reduce the amount of time that the workstation is fully powered. Sleepy for Linux will suspend workstations when they are inactive based on the sleep timeout. We will now summarise the results that we have found to two of the research questions involving power savings, by analysing activity data that we have gathered in a testing phase with real users.

Does the activity data of users indicate that Sleepy for Linux will be successful in decreasing power usage, and how is the amount of energy that is potentially saved calculated, based on this activity data?

The activity data shows that there are a large amount of periods where a specific user is idle and is not currently using their workstation, during which the workstation could be suspended. While there is a large amount of idle periods with a short length, during which real power reduction cannot and should not take place, (regarding the level of user convenience) there are in fact extended periods of idleness that make up a large portion of the total time that these workstations are powered. There is plenty of potential for power reduction by using automatic power management in these cases, so we can assume that if Sleepy for Linux is implemented, it will be successful in saving power.

To calculate the amount of power savings for a specific user, we proposed a simple algorithm that takes as parameters an arbitrary value for the sleep timeout and values for the consumption levels of the workstation during suspension and while it is idle. For each idle period, if the idle period is longer than the sleep timeout, then we can save the amount of time that exceeds the sleep timeout. If all of these time segments are summed, and multiplied by the difference of the consumption levels of suspension and idleness, the total amount of energy savings for a given timeout can be determined.

Are there significant benefits, in terms of decreasing power consumption, to determining a sleep timeout for each individual user of the workstations, rather than setting a global static timeout for all workstations, no matter which user account is currently logged in?

We have compared activity data from several users to find out the differences and similarities between these sets of data. It has become clear that the distribution and length of idle periods is very different between each set of two users. By applying the algorithm as described above to different users for different timeouts, we see that the amount of power that can be saved varies greatly between users for different timeouts.

For some users, an increase of the timeout leads to a quick degradation of total savings, while a slower degradation is visible for others. This is due to the fact that in the first case, the average idle period is very short, while in the latter case the average length is much higher. Therefore, we can apply different timeouts for these users, so that the amount of experienced user convenience is relative to the amount of power that has been saved by that specific user. Since users with averagely long idle periods can still save significant power at higher timeouts than users with averagely short idle periods, we can apply a longer timeout for these users. The data thus shows that a personalised timeout is indeed favourable over a global static timeout, since the latter option does not take into account these differences in user behaviour.

Furthermore, we have performed a short study to see if there are any patterns that are similar between users depending on the time of day during the limited testing phase. We have found that while workstations are often powered off or inactive during night time, no other patterns can be found using the data that is available to us at this moment. More research will have to be performed to establish the existence or non-existence of behaviour patterns between users.

7.3 Communication layer security

The Sleepy for Linux framework is based on its Windows counterpart, the Lazy Sleep framework. This framework has questionable design decisions that lead to a degradation of the overall level of security and authentication that is applied by this platform. The architecture of Sleepy for Linux provides a benefit over the architecture of Lazy Sleep in this regard, which will now be summarised.

Does the architecture of the Sleepy for Linux software attain the security and privacy standards as described, and is there an improvement with respect to the level of security of the Lazy Sleep software?

Lazy Sleep uses IP sockets to transfer data between the workstations and the server of the framework in both directions. There are benefits and risks to this communication model. The benefits are that sending messages between the components is fast and direct, the components can send data to each other at any time and are not dependent on the communication of other components. Furthermore, by using IP sockets to transfer remote commands to workstations, it is possible to implement Wake On Lan (WOL) in an easy way, since the server is located in the same network as the workstations. However, the risks of this method are that security and authentication has to be ensured manually and this can be a difficult task. Furthermore, the server cannot easily be places outside of the network in which the workstations run, and a different server instance is possibly needed for each network.

Sleepy for Linux uses HTTP communication between the different components of the network, which has several advantages. First of all, security and authentication are to a high extent readily available by using the Transport Layer Security protocol. Secondly, no port forwarding is necessary, since the system does not require additional open ports, and all workstations in different networks can be easily connected to the same server instance, unlike the Lazy Sleep server architecture.

Thirdly, since the server replies with power management settings and remote commands for workstations, it must be verified that these replies are actually originating from the server, which is done through a security certificate. This is more difficult in the Lazy Sleep architecture. A deficit of the Sleepy for Linux communication protocol is that remote commands and power management settings are not directly applied on workstations and must be polled first, though we have argued that this does not create any significant delays or issues.

7.4 User acceptance studies

Finally, the level of user acceptance of the Sleepy for Linux framework has been researched. This is an important requirement to the system, as Sleepy for Linux does not create any direct benefits to the work flow of the user. The software should therefore be as transparent as possible and the user should be aware of the usefulness of the application in the long term. We will summarise how the level of user acceptance has been measured.

What is the level of user acceptance of the software, is the software suitable to be implemented on both student and staff workstations?

The level of user acceptance has been measured by asking all the users of Sleepy for Linux to fill out a survey. The survey that was used for this was obtained from the system usability scale (SUS), and requires the participants to rate the system in question on several different aspects. Combining the results of all of the surveys gives an indication of the perceived level of usability from the perspective of the user. It was concluded that Sleepy for Linux was perceived as very usable by the users. This was concluded from the fact that Sleepy for Linux received an average of 82 on a scale from 0 to 100. A higher value means a higher level of usability, and since Sleepy for Linux scored very high on this scale, the usability is also high.

User interviews were also conducted to obtain additional information from the users of Sleepy for Linux regarding the level of user acceptance and usability problems that may be present in the application. The questions that the users were asked during these interviews were mainly focused on the level of user acceptance of Sleepy for Linux. During these interviews, the users indicated that they did not always feel comfortable while using Sleepy for Linux. The main reason for this, is that it was not abundantly clear which types of data were recorded by the Sleepy for Linux framework. They would have felt more comfortable if they would have been better informed. Another issue that was addressed was the fact that it was not possible for users themselves to view their own activity data, which users indicated to be interested in. Therefore, to increase the user acceptance of Sleepy for Linux, it is vital that the users are informed in the best possible way about what the framework actually does and how the framework is affecting them. Finally, Sleepy for Linux could be made open source, which allows users to verify what the software actually does and what data is recorded.

7.5 Summary

The research questions have been successfully answered and in this paper we obtained the insight that a remote power management framework such as Sleepy for Linux has a large potential for reducing the power consumption of office buildings such as the Bernoulliborg, especially when a personalised timeout is favoured over a global static timeout for all users. The software has a high level of security, privacy and user acceptance, and is compatible with the workstations in the Bernoulliborg. We will finalise our paper by addressing future research that can be performed, and additions and refinements that can be made to the framework.

8 Future Work

This section discusses the challenges and opportunities that we can see so far for Sleepy for Linux (and to some extent Lazy Sleep). First of all, we will discuss some opportunities for improvement of the implementation of the Sleepy for Linux framework. Secondly, we will talk about deployment on the workstations of the University of Groningen, particularly in the Bernoulliborg building. Lastly, we will discuss some external modules that could be connected to the Sleepy for Linux (or Lazy Sleep) framework to enhance its functionality.

8.1 Software improvements

We will explain the software improvements in three sections. Technical opportunities deal with software technicalities that have not been implemented yet due to time constraints or a lack of need of such technical behaviour for this particular project. Functional opportunities discuss how the software can be extended or changed so that it adheres better to the non-functional requirements and will propose changes to some functional requirements that could make more sense in light of what the framework is supposed to accomplish. Lastly, we will discuss the compatibility of Sleepy for Linux and Lazy Sleep.

8.1.1 Technical opportunities

Sleepy for Linux modifies the sleep timeout of workstations in the power management settings of the user session. Each user has their own set of persistent power management settings and these settings can be changed between user sessions independently. The power management of the workstation will then decide based on these user settings when the monitor of an idle workstation is dimmed and when a workstation should automatically suspend due to inactivity. However, when the workstation does not currently have an activated desktop session and is displaying the login window, when no user is logged in or if a user just locked their screen, the power management system will only dim the screen after inactivity, but not suspend the system.

While this is mostly an upstream issue of the power management of Linux systems for multi-user workstations, there could be a simple solution to this problem. The service is the component of Sleepy for Linux that has a single instance per workstation and polls the server for commands that should be executed. It also transmits the boot and shutdown events of the workstation. This service could be extended to keep track of the last status of all users that are logged in. If all users have been inactive for a certain amount of time that exceeds the individual sleep timeout of all these users, then the workstation itself will be suspended. This means that workstations without active user (either everyone is logged out or some users are logged in on a locked workstation) will be automatically suspended by means of inactivity, giving more opportunity for reducing power consumption by workstations.

The second technical issue that we would like to point out is that the clients will not save any collected data locally, but immediately try to send it. While there is a retry mechanism in place for the REST calls that are being made to the server, this may not ensure delivery in cases where the device on which Sleepy is running does not have access to the Internet for some time. While the device cannot establish a connection with the server, all of that activity and event data from the clients and workstations is simply lost.

Admittedly, this problem is not so significant that the software cannot function while it exists. The reason for this is that Sleepy for Linux is intended to be used on networked workstations that are connected to the network of the University of Groningen directly via Ethernet cables. Therefore, it is not likely that there will be prolonged periods of loss of connectivity at any time while the software is running. The chance of event data or activity data getting lost is thus very small. However, if Sleepy for Linux is ever extended to work on wireless connectivity devices of employees, or wireless devices owned by the University of Groningen, then this kind of problem will have to be fixed first. These kind of wireless devices will often experience a loss of connectivity due to having to rely on wireless networks, where traffic is inherently more likely to get lost or damaged compared to wired networks. [6]

To mitigate this problem in both the cases of (less importantly) networked stations and (more importantly) wireless devices, the client applications of Sleepy for Linux could cache events and activity data locally, including the time at which they were measured, in case of a delivery failure due to a lack of connectivity. This way, the events and data can be retransmitted at another time when connectivity has been established. Activity data and event data will then always reach the server.

Moreover, using this method can have some interesting benefits for networked workstations as well. For example, when a user is heavily using their connection to the network or the bandwidth or CPU performance of the workstation is very low, the user may experience minor (yet frequent) connectivity or performance degradation due to Sleepy for Linux sending activity data every few seconds. Sleepy for Linux could instead temporarily cache the data from the client if the user is busy, and send this data later when the user is idle and not actively using their network connection. This could potentially lead to a minor increase in performance for the user in some cases.

A problem with the caching method, both for wireless devices and networked workstations, is that the client application will have a maximum amount of messages that can be cached due to data storage requirements, so behaviour has to be implemented that decides what will happen in case the cache is full. Needless to say, more research and user testing will be needed to fully understand and resolve this problem.

Right now, the software needs to store the MAC address of the Ethernet adapter of the connected workstations, so that they can be woken up using WOL packets by an external module if necessary. Furthermore, to allow external modules to wake up single-user devices or devices for specific users, we also keep track of the last MAC address that each client application of a user session connected from. This could pose a serious issue regarding the privacy requirements of the system. It should potentially be possible to abstract from MAC addresses and create a sort of identifier for workstations, just as we have with users (using the UUID system). However, if that is the case, then it will not be possible for external devices to wake up workstations, since WOL packets require the MAC address of the targeted workstation to be known. This is a dilemma without a clear solution. See also the next section for our comments regarding the wake-up requirement of the framework.

8.1.2 Functional opportunities

The Sleepy for Linux prototype that was made for this project provides the user with a limited control over the application. Furthermore, the interface that is offered to the user is very limited in terms of functionality and information that is displayed. We will briefly describe some opportunities for improving the interface and control of the client application here.

When disabling the Sleepy for Linux application, a user must pick a reason as to why they are disabling Sleepy for Linux. They can pick from a preset list of reasons or simply type a custom reason up to a certain amount of characters. This gives the user enough sense of freedom and usability. However, it is only possible to disable the application itself until reboot, and the application has to be manually re-enabled. It would be interesting to give the user the option to disable Sleepy for Linux for a custom amount of time, or to let the user disable Sleepy for Linux for a certain time period each day due to having to perform tasks or decreasing inconvenience due to sleep timeouts. This scheduling information could be shared with the server, so that external modules can take this data into account for the kind of functionality that they provide, if necessary. Furthermore, the user could specify that they want to disable certain functionality of Sleepy for Linux, such as the remote command structure (where the server can remotely order the workstation to suspend or shutdown at any time) in cases where the user would like to immediately stop the inconvenience that is caused by this part of the functionality.

The interface of Sleepy for Linux only provides information about the current status of the application (enabled/disabled) and some basic information about the software itself. To raise the level of perceived usefulness of the application by the user, the interface of the application could show part of the activity data that has been obtained for the current user, possibly including the amount of power that has been saved through Sleepy for Linux, so that the user can see that Sleepy for Linux performs its intended task. The user may also be interested in the sleep timeout and the idle timeout, these values could be displayed in the interface as well.

Right now, Sleepy for Linux determines activity based on events from input devices such as mouses and keyboards, which generate events as they are being used. An interesting research approach is to find out if and how extra information about the activity of a user can be deduced for special cases. For example, power management can already detect if the user is actively watching a video and the system will in that case be considered active. As a counter example, when the user is performing a job such as inspecting a page of a document, which may in some cases take longer than the sleep timeout, and the user does not touch their keyboard or mouse, then power management will falsely claim that the user is inactive and suspend the workstation. In these special cases, it is seemingly impossible to determine if the user is currently using their workstation based on the information that power management receives. More research is needed to find out if, in such special cases, we can distinguish between an active and inactive user, possibly by using external sensors such as motion sensors or webcams, but this would of course risk the privacy of the user.

The requirements as they are describe the existence of Wake On Lan (WOL) functionality where external modules should be able to remotely wake up workstations through WOL magic packets. As mentioned in the previous section, this requires the MAC address of the clients, which is a risk to the privacy requirements, as it can be used to identify users through the knowledge of which MAC address belongs to which workstation. This is especially true for single-user workstations used in the offices of the employees of the University of Groningen. Furthermore, does WOL functionality for a power management framework really make sense? We identify here three cases:

- Some module wakes up a workstation *before* an employee is present to use it.
- The module wakes up the workstation *at the exact time* that it will be used.
- The module "wakes up" a workstation *later* than the employee intends to use it, which means the employee has to turn on their workstation manually.

In the first case, the power consumption actually increases unnecessarily due to the fact that no one is using the workstation yet. In the third case, it does not improve the level of convenience for the user. The only case where WOL functionality matters (from a perspective of power management and user convenience) is the second case. However, we are not convinced that the deterioration of the privacy requirements weighs up against saving the user a press of a button in the best case scenario. For future development, the potential benefit of this requirement should be investigated thoroughly.

8.1.3 Architecture of Lazy Sleep (Windows)

Sleepy for Linux is the Linux version of Lazy Sleep, which is only available for Windows. However, as has been duly described in this paper, the communication principles of Lazy Sleep and Sleepy for Linux are entirely different and not compatible with each other. Therefore, if the goal is to integrate the two server architectures, so that all existing client applications can run under the same server instance, there are two choices that can be made.

The first choice would be to add two ways to communicate with the server, both through REST services and through IP sockets. This way, no modifications have to be made to the client applications and existing users do not have to be informed about this change. Requests by external modules are handled differently for each type of client: for a Linux client the requests to change the timeout or status are put in a database where the client can pull them through REST services; for a windows client the requests are sent directly through the IP socket channel. However, this method requires that the server runs in the same network as the workstations and is subject to all of the risks of the Lazy Sleep software that have already been described.

The second choice is to port the Sleepy for Linux application back to Windows, by changing the Lazy Sleep software so that the communication happens in the same way as its Linux counterpart, through the use of the REST services. We have shown in this paper that the REST approach carries significant benefits to the level of security and privacy of the application, as well as its ease of implementation and architecture. Furthermore, there is no need for the server to be in the same network as the workstations for this approach. It does however take development time to change the Windows client application, and existing users will have to reinstall the software in case the original server is taken down.

8.2 Deployment on targeted environment

The targeted environment for the Sleepy for Linux software is, first of all, the network of the Bernoulliborg, at the Zernike campus of the University of Groningen. This building contains some hundreds of networked workstations that all run Linux, specifically Ubuntu 10.04 and up and Xubuntu 14.04 and up, using the window managers Unity, GNOME and XFCE. Since Sleepy for Linux supports any combination of these distributions and window managers, we do not see any compatibility issues.

However, the software has only been tested on a small subset of potential users and has not been tested on networked workstations themselves. Testing on this system could reveal unforeseen issues that could not have been possibly uncovered by the testing phase of this project. Furthermore, there may be special restrictions on network traffic or restrictions to functionality of certain packages that may require minor adjustments to the client applications. Since the server implements a REST interface, network restrictions should not get in the way of the application, since the standard ports for HTTP and HTTPS are naturally exposed for traffic in almost every network.

8.3 External modules

Sleepy for Linux is to be seen as a framework that allows external modules to remotely influence power management settings of the connected workstations and to remotely control the status of those workstations. We can already predict that two of these modules will have to exist in order to make Sleepy for Linux successful. These two modules, the machine learning module and the status control module, will be discussed here. We will also discuss two other systems that could help improve the user acceptance of the application, one of which has already been implemented as part of another Bachelor project.

8.3.1 Machine learning module

Sleepy for Linux has no included functionality for manipulating the sleep timeouts of users. Since eventually we want to setup personalised timeouts for every user of the framework, an external module should manipulate these timeouts on the server, which are then propagated to the clients by means of HTTP polling. This module can also request the activity data from the server to construct this timeout by means of machine learning algorithms.

As part of a Master's project in Computing Science at the University of Groningen, such a module is currently being developed. We know so far that it implements algorithms that attempt to minimise two parameters, namely the idle time and the amount of negative feedback submitted by users (when they temporarily disable the framework). Three different models are being tested in this project which use machine learning techniques that have as input one or more of the following: activity probability, negative feedback, and idle time of a specific workstation. The output of each of these models is the new sleep timeout for that respective workstation. [4] At the time of writing this paper, this project has not yet been completed and experiments are still being performed to judge its effectiveness. When it has been implemented, it can greatly contribute to the Sleepy for Linux framework, as we have argued that personalised timeouts are beneficial over global timeouts.

8.3.2 Status control module

The Sleepy for Linux server has endpoints that allow external modules to send shutdown, wake-up and suspend commands to specific workstations. These endpoints could be exploited by a module which, somewhat alike the machine learning module, will suspend, wake up or shut workstations down at appropriate times. For example, this module could take into account the working schedule of an employee of a specific workstation, or more generally, shut down workstations that are still powered after a certain time of day that have for some reason not suspended automatically yet. This module by itself would allow for a significant decrease in power consumption with minimal user inconvenience.

8.3.3 User interactivity

In the projects related to the Green Mind, some components are mentioned that visualise the data that is gathered from the users in terms of power consumption of their offices. [3] While this is an interesting data metric to be showing to users, users may also be interested in seeing their periods of activity and inactivity and see how Sleepy for Linux has intervened in the behaviour of their workstation by means of power management or remote commands. Furthermore, the Green Mind proposal also refers to the ability to compare power consumption data between users. Users could potentially compare their activity data to other users and see if there are any common patterns, etc. However, this will deteriorate the level of privacy, so a survey has to be performed first if this is a feature users would find useful.

8.3.4 Energy Smart Office

During the Sleepy for Linux project, several other projects were conducted that modify or track behaviour of programmable devices in order to reduce power consumption in the Bernoulliborg. One of these projects was the Energy Smart Office. This project tries to create a harmonised collaboration between an amount of different sensors that are present in offices, namely presence or illumination sensors for lighting and a Sleepy for Linux sensor. These sensors by themselves give very little information about user presence: if the user is sitting behind their desk and does not move for a while, the Sleepy for Linux sensor will probably describe the user as being active, while the presence sensor in the lighting may not pick up any relevant motion by the user. By actually combining the data outputs of each of these sensors, the authors claim to have a more accurate way of determining whether the user is currently present in their office. [5]

The authors make use of the Sleepy for Linux sensor to check the status of a certain workstation that is connected to the framework. With this information, they are allegedly able to accurately determine the user presence. When this information is known, it is possible to remotely turn off the power to lighting, workstations, and other appliances in the office that are currently running. Since Sleepy for Linux has functionality to remotely suspend a workstation and to remotely shut it down, the authors have successfully utilised this functionality of Sleepy for Linux during the process of powering down devices with. [5]

8.4 Summary

The prototype of Sleepy for Linux was made as a proof of concept of the Lazy Sleep principles, implemented on a Linux environment. Furthermore, it was used as a data acquisition tool to determine the feasibility of Sleepy for Linux in practice. However, this does not mean that the current version of Sleepy for Linux is as stable and versatile as it can be. We introduced various opportunities for improvement in this section.

We do not yet know what to expect of deployment on the networks of the University of Groningen. While it is clear that the software is compatible with the identified distributions of Linux and desktop environments, we do not have a full overview of the other software and processes that are running on those workstations or how exactly the networked workstations and the network itself are set up. Therefore, unforeseen issues may still occur.

Finally, some modules have been discussed that we believe may prove the usefulness of the Sleepy for Linux framework and extend and use its current functionality. These modules show that the Sleepy for Linux framework is extensible and offers many opportunities for smart applications in order to reduce power consumption, or increase user-awareness of energy consumption and energy waste.

9 Acknowledgements

We would like to thank Faris Nizamic, prof. A. Lazovik (Distributed Systems research group) and dr. A. Ampatzoglou (Software Engineering and Architecture research group (SEARCH)) of the University of Groningen for providing the project proposal and for providing feedback on our progress and this paper.

For providing information and advice on the requirements and implementation of Sleepy for Linux, we thank Brian Setz. We also thank both Brian Setz and Ruurtjan Pul for contributing to the original Lazy Sleep framework for the Windows platform.

Finally, we give our thanks to the testers for all of their testing efforts regarding the Sleepy for Linux application: Brian Setz, Jelle Bakker, Ugo Moschini, Gerrit Spieard, and all of our anonymous testers. Without their effort, we could not have performed any usability analysis or analysis on power saving potential for Sleepy for Linux.

10 Appendices

See the full text for references to and explanations of these appendices.

10.1 Appendix A: Table of functional requirements

FR-01	**System tray indicator**
The end-user can see an application indicator in the system tray on any window manager that is supported according to the technical requirements.	
FR-02	**System tray menu**
The end-user can click on the application indicator to bring up a system tray menu that presents all of the controller options available to the user: showing the about window, showing the disable window, re-enabling Sleepy for Linux. The first two options are available when the application is enabled. The first and third options are available when the framework is disabled.	
FR-03	**Information about the framework**
The end-user can open up a window that shows the following information about the framework: application name, application version, the application logo, a copyright notice, links to an online information source containing help, contact and licensing information.	
FR-04	**Temporarily disabling the framework**
The end-user can open a disable window, where they may specify a reason for disabling Sleepy for Linux from a predefined list, namely "I'm performing tasks while idle.", "The sleep timeout is too short." or "It slows down my machine."; they may also submit a custom reason with maximally 250 characters. When this process is completed by the end-user, Sleepy will be disabled until the user logs out and back in again, until a reboot is performed or until the user manually re-enables Sleepy.	
FR-05	**Disable notification**
When the end-user submits a reason for disabling the framework and confirms the disabling procedure, they will receive a notification when Sleepy for Linux has been disabled.	
FR-06	**Duplicate window handling**
The end-user is not capable of opening more than one window of the same type. If there is already an about window or a disable window, and the user attempts to open respectively a new about window or new disable window, the existing relevant window will be activated. If there is a disable notification active, the user cannot interact with the application until the notification is closed.	
FR-07	**Window deactivation**
The end-user can, at any time, close the about window, disable window, or disable notification, without consequences or damage to the state of the model within the application.	
FR-08	**Distinction between activated/deactivated framework**
The end-user is capable of deducing from the application indicator in the system tray whether the framework is currently enabled or disabled.	
FR-09	**Re-enabling the framework**
The end-user can re-enable the framework by selecting the re-enable option from the system tray menu.	

FR-10	Online information overview of clients
Administrators and authorised third party members are able to view the following data on remote registered client applications: UUID (unique identifier), last received status, current deduced status (on/off/sleeping), last MAC address, time of registration.	
FR-11	**Online information overview of workstations**
Administrators and authorised third party members are able to view the following data on remote registered workstations: most relevant MAC address (preferably from Ethernet adapter, otherwise the next existing adapter in the list of adapters is used), last received status, current deduced status (on/off/sleeping), MAC address, time of registration.	
FR-12	**Remote workstation status manipulation**
Administrators and authorised third party members are able to send remote commands to specific workstations, these commands are: "shutdown", "sleep". They respectively shut the workstation down and suspend the workstation.	
FR-13	**Remote idle timeout manipulation**
Administrators and authorised third party members are able to change the idle timeout of a specific registered client application. The idle timeout is the amount of time after the last activity event that has to be passed at least for an end-user to be considered inactive.	
FR-14	**Remote sleep timeout manipulation**
Administrators and authorised third party members are able to change the sleep timeout of a specific registered client application. The sleep timeout is the amount of time after the last activity event that has to be passed at least for an end-user's workstation to be automatically suspended.	
FR-15	**Access for external modules**
Administrators and authorised third party members can access all of the status data that is gathered from the client applications and the workstations, namely, access their status information at any point in the past that has been recorded.	

10.2 Appendix B: Table of technical requirements

TR-01	Connectivity between server and clients
Any Sleepy for Linux server and any Sleepy for Linux client application are able to communicate with each other in terms of character messages using some communication principle and/or protocol, independent of the network configuration of the client applications.	
TR-02	**Power management activation settings**
The Sleepy for Linux client application is capable of activating and deactivating the power management of the current user session.	
TR-03	**Power management timeout settings**
The Sleepy for Linux client application is capable of changing the timeout after which an idle system will suspend in the power management settings of the current user session.	

TR-04	Determining user activity
The Sleepy for Linux client application can determine the amount of time that has passed since the last event from an input device (mouse, keyboard, etc) or some other to be defined source has been observed, signifying the last activity of the end-user controlling the user session.	
TR-05	**Processing remote command events**
The Sleepy for Linux client application can shut the workstation on which it runs down or suspend it at any given time, if respectively a remote shutdown command or remote suspend command is received by the server.	
TR-06	**Catching user session events**
The Sleepy for Linux client application is capable of detecting when an end-user logs in/out of their account and must send these events to the Sleepy for Linux server.	
TR-07	**Catching workstation events**
The Sleepy for Linux client application is capable of detecting when the workstation is booted, when it is shut down, when it is suspended or hibernated and when it resumes from suspension or awakes from hibernation, and must send these events to the Sleepy for Linux server.	
TR-08	**Identification and authentication of end-users**
For each user account on each workstation that Sleepy for Linux runs on, there is a unique identifier that is known both by the client application and the server, which uniquely identifies an end-user in the database of the server.	
TR-09	**Abstraction of workstation types**
The Sleepy for Linux client application works on single-user workstations, multi-user workstations and networked multi-user workstations. In the latter case, the end-user of the user session is uniquely identified in the same way, no matter on which workstation they log into their account.	
TR-10	**Obtaining MAC address**
The Sleepy for Linux client application is capable of extracting the current MAC address of the current workstation. The order of preference in network adapters should be the following: any Ethernet adapter, any adapter that is not the loop-back adapter, the loop-back adapter.	
TR-11	**Obtaining username**
For the prototype, the Sleepy for Linux client application is capable of extracting the username of the current end-user user session for debugging purposes.	
TR-12	**Platform**
The Sleepy for Linux client application is available on the Ubuntu and Debian Linux distributions.	
TR-13	**Desktop environment**
Sleepy for Linux supports the following desktop environments on Ubuntu and Debian: Unity, GNOME (Ubuntu GNOME, Debian), XFCE (Xubuntu), LXDE (Lubuntu), KDE (Kubuntu) and MATE (Ubuntu MATE).	
TR-14	**Installation and deinstallation**
The installation and deinstallation processes for the Sleepy for Linux application is automated and involves minimal user intervention.	

10.3 Appendix C: Table of non-functional requirements

NFR-01	**Privacy**
The data that is sent from a specific client application to the server, which incorporates the UUID (unique identifier) of the client, cannot be traced back to the personal identity of the user. Furthermore, no data is sent between the clients and server that could be used to identify specific users or user behaviour to personal identities. An exception to this is that in the testing phase of Sleepy for Linux, the name of the end-user, controlling the user session where a client application registers itself from, is sent to the server for debugging and research purposes. This functionality will be removed in the production version of Sleepy for Linux.	
NFR-02	**Security**
No data between the client applications and the API on the server travels over an insecure connection. All data is sent over a secured connection. Furthermore, the architecture of the application does not in any way (make demands that will) deteriorate the security of the network that the respective networked workstations are a part of. (In this case, this would be the networks of the University of Groningen.)	
NFR-03	**Scalability**
The architecture of the client application and server application allows a single Sleepy for Linux server instance to handle at least 100 connected user sessions at once.	
NFR-04	**Portability**
The Sleepy for Linux software is available for both the Ubuntu and Debian Linux distributions, and supports at least the following desktop environments based on availability on the workstations of the University of Groningen buildings: Unity, GNOME2, GNOME3, XFCE, LXDE. Optional desktop environments to support are: KDE, MATE, Cinnamon.	
NFR-05	**Performance**
The Sleepy for Linux client application does not hinder the user in terms of performance. The average CPU usage over time is below 1% while the application is running. The application uses less than 5MB of RAM at all times. The application uses less than 5MB of hard disk space. The data send/receive rate of the application is less than 1 Kb/s.	
NFR-06	**Usability**
The level of usability is crucial to the adoption of the software by any end-user or by any production environment. The application therefore ensures the highest level of usability in terms of user interface and user experience, and provides enough functionality for the end-user to control the application.	
NFR-07	**Interoperability**
The server maintains an API that has exposed functionality for external / third party modules. Examples of this exposed functionality include: obtaining data from specific workstations and end-users, modifying the sleep and idle timeout of end-users, sending remote commands to workstations.	

10.4 Appendix D: User questionnaire & interview layout

This section concludes all of the components that were used for research into various software qualities of the software.

10.4.1 System Usability Scale questionnaire

The following survey [31] serves both as a validation of the usability, as well as the ease of installation of the Sleepy for Linux application:

	< **Disagree**				**Agree** >
1. I think that I would like to use this system frequently.					
2. I found most of the system unnecessarily complex.					
3. I thought that the system was easy to use in general.					
4. I need the support of a technical person to use this system.					
5. I found the various functions were well integrated.					
6. I thought there was too much inconsistency in this system.					
7. Most people would learn to use this system quickly.					
8. I found the system very cumbersome to use in general.					
9. I felt very confident using the system most of the time.					
10. I needed to learn a lot before I could get going with this system.					

10.4.2 User interview layout

In the user interviews, the following open questions detail the main layout of the interview:

Question 1. Have you experienced any usability issues while working with the application?

Question 2. Have you experienced any performance issues while the application was installed on your machine, or have you ever had to forcefully kill the Sleepy for Linux processes due to high resource usage?

Question 3. Have you experienced any connectivity issues or impact in the period that you were using Sleepy for Linux?

Question 4. Did the platform website motivate you in a positive or negative way to install the application?

Question 5. Would you describe the period that you tested the application in as regular, or did you spend more/less time on your machine than usual?

Question 6. Do you experience any level of discomfort due to the fact that anonymous activity data is gathered from your device, and if so, what could we do more to alleviate this discomfort?

Question 7. Were the installation and deinstallation instructions clear enough, and did you encounter any problems during the installation/deinstallation that were caused by any step mentioned in these instructions?

Question 8. Did the online platform offer enough information, was the intended functionality of the application clear? Did you miss any information that you would like to see on the website in the future?

Question 9. Do you have any suggestions for functionality that you would like to see added, if you were to use this application for a longer period of time?

10.5 Appendix E: User behaviour

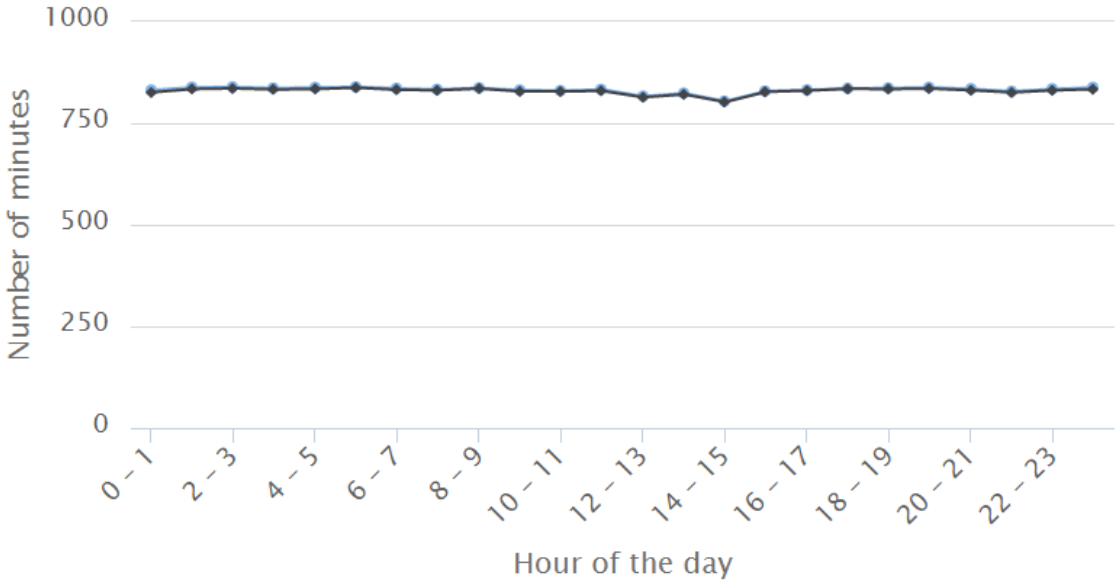

Figure 14: Behaviour of user 1

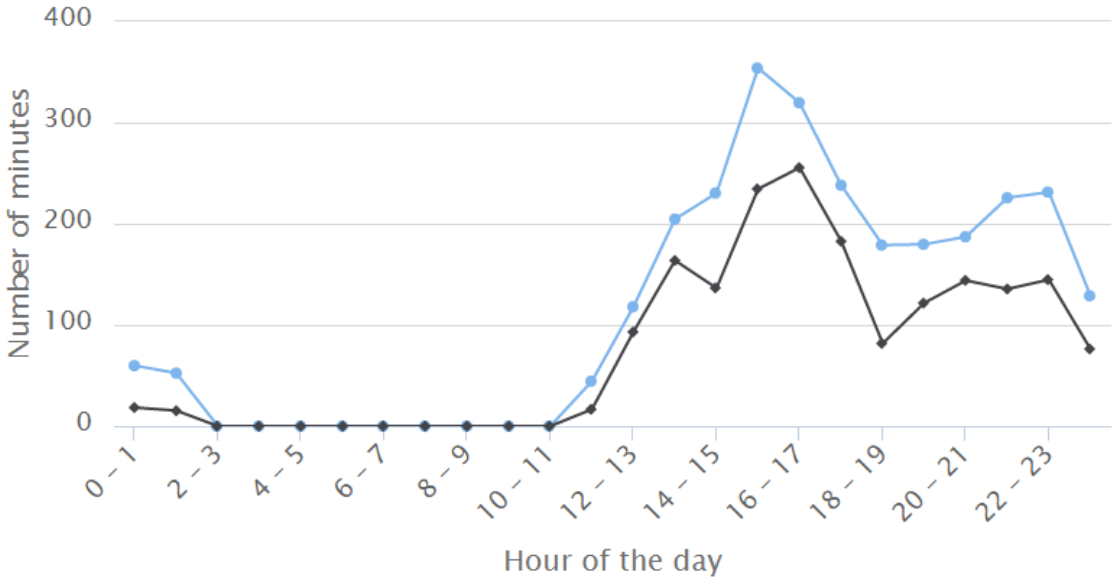

Figure 15: Behaviour of user 2

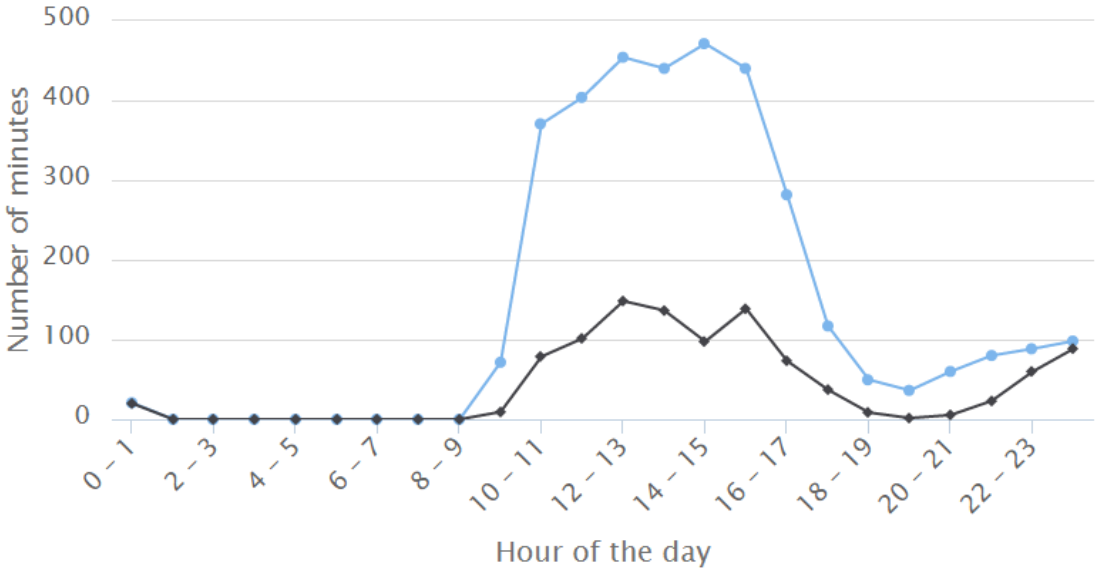

Figure 16: Behaviour of user 3

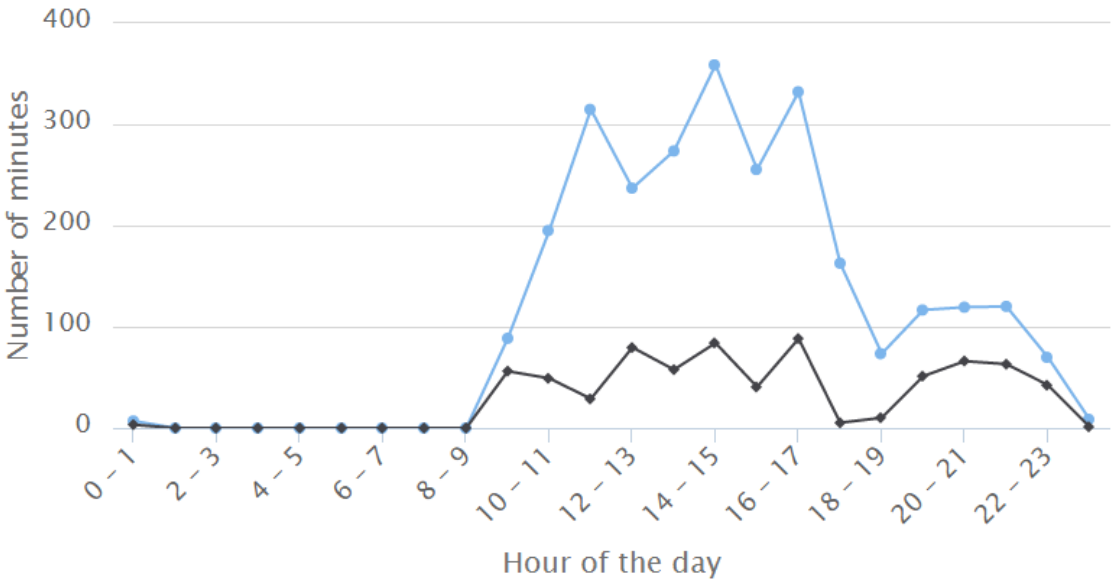

Figure 17: Behaviour of user 4

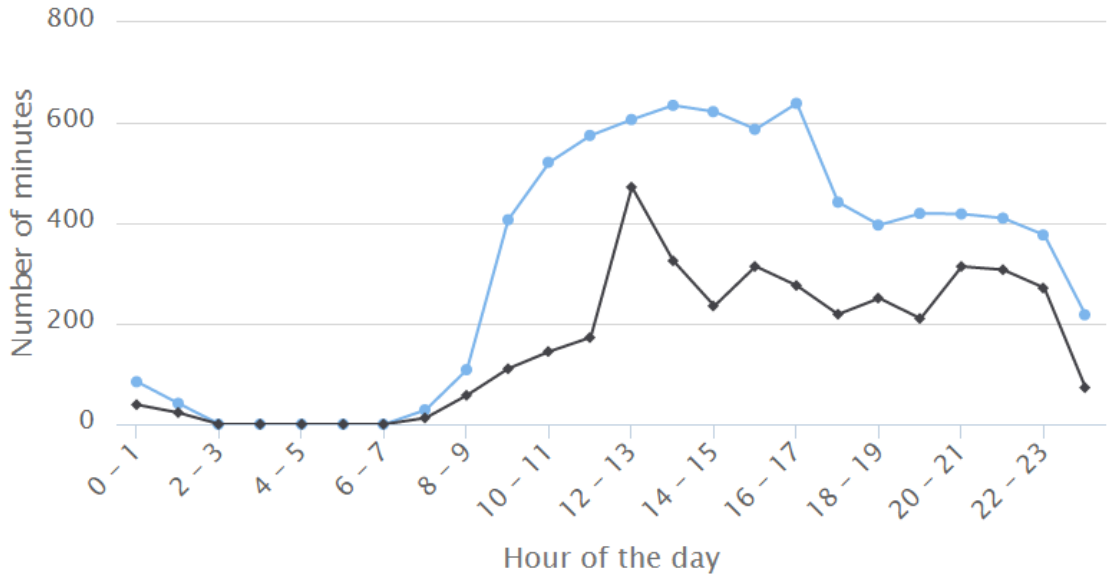

Figure 18: Behaviour of user 5

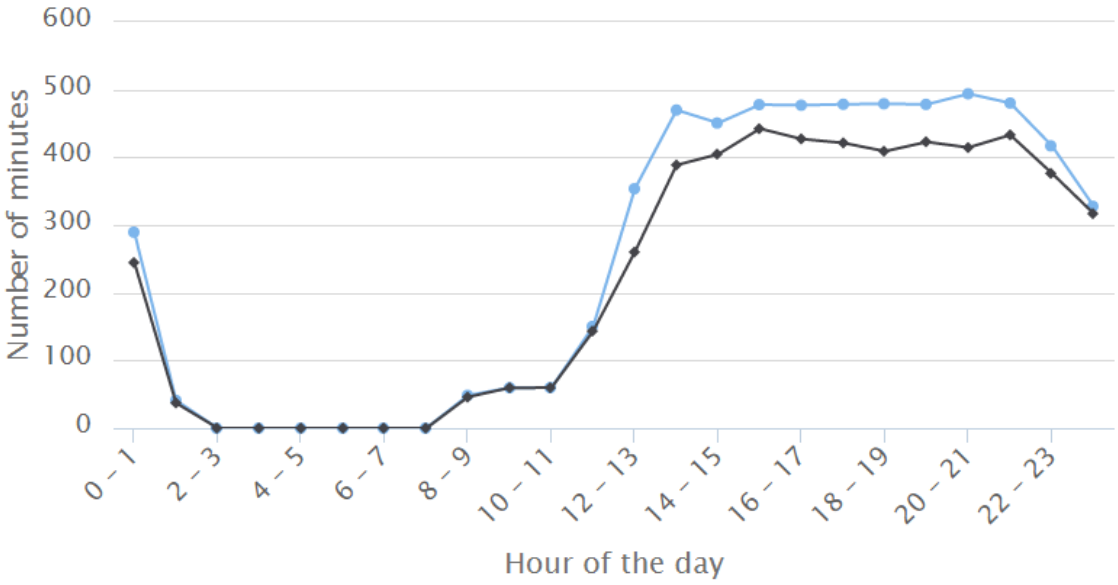

Figure 19: Behaviour of user 6

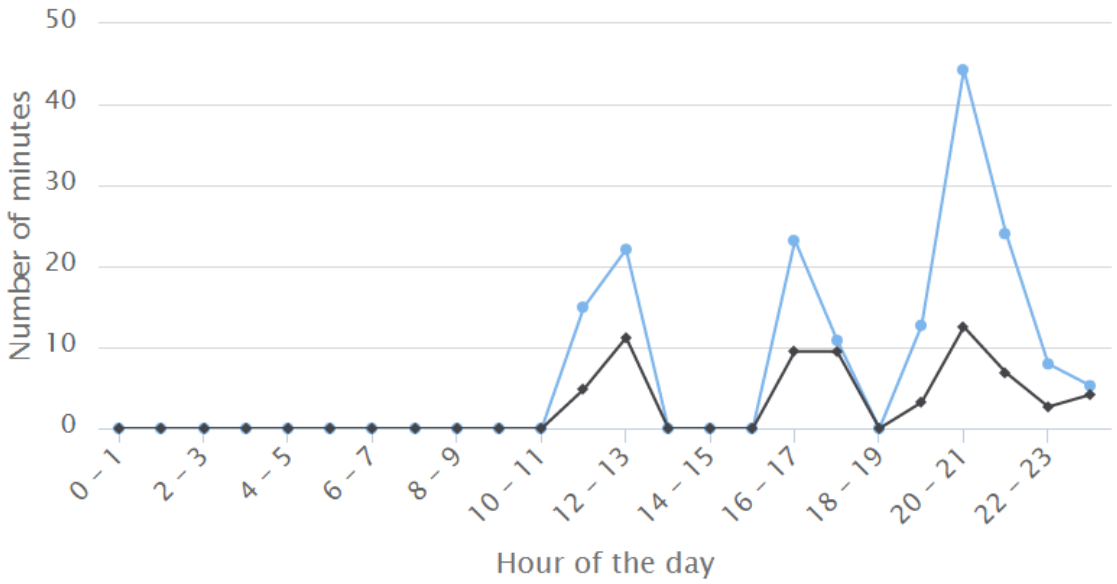

Figure 20: Behaviour of user 7

10.6 Appendix F: Screenshots

Screenshots of the Sleepy for Linux client running on the workstations of test users and part of the platform website that has been constructed will now be shown to give the reader of this paper a brief impression of the software and website.

10.6.1 Sleepy for Linux client on an Ubuntu 15.04 machine

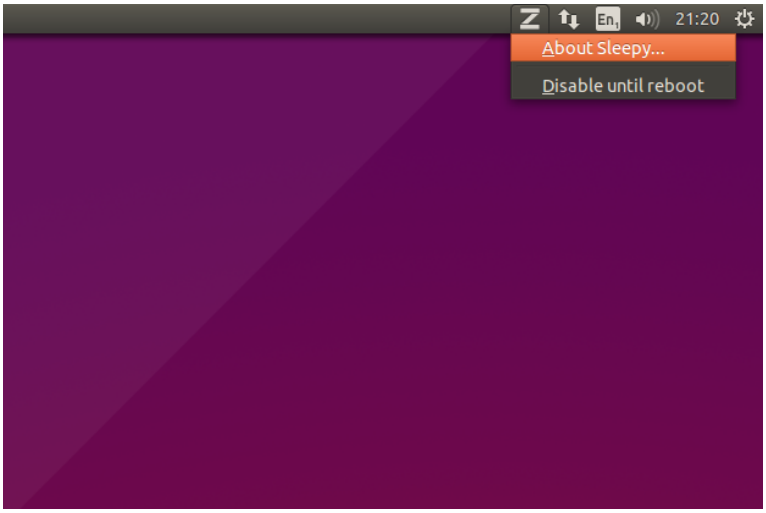

Figure 21: The tray icon and tray menu.

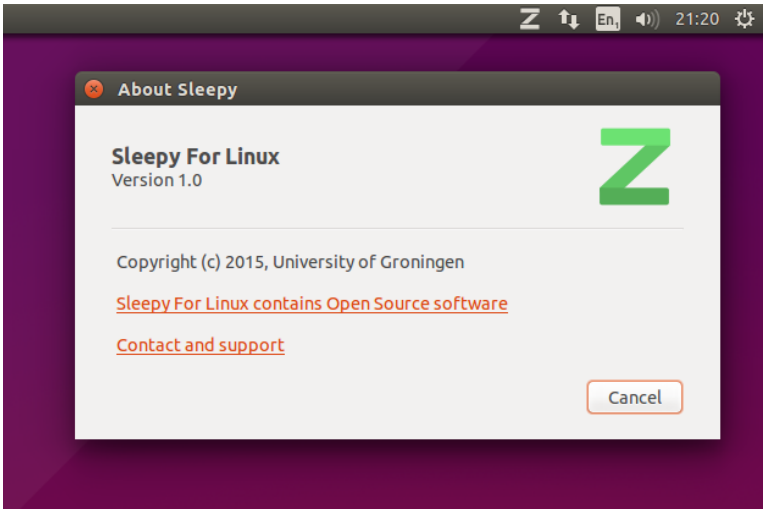

Figure 22: Information window about Sleepy for Linux.

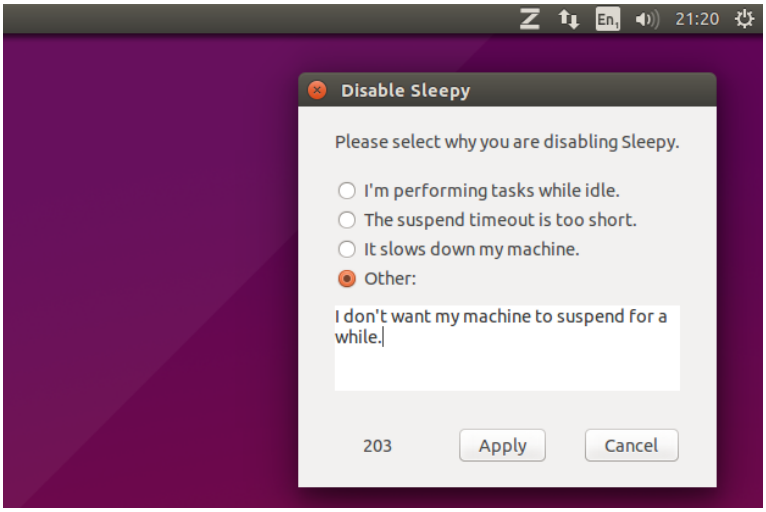

Figure 23: The window for disabling Sleepy for Linux.

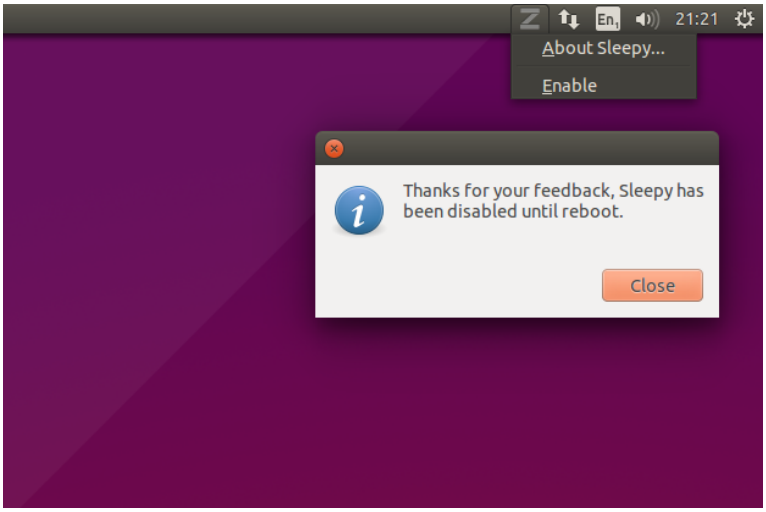

Figure 24: Confirmation of disabling Sleepy for Linux, including modified tray icon and menu to reflect disabled state.

10.6.2 Platform website

The website and software will be available for a limited time after the finalisation of the project at `http://www.sleepyforlinux.nl/`. For future readers, this domain may no longer host content related to Sleepy for Linux.

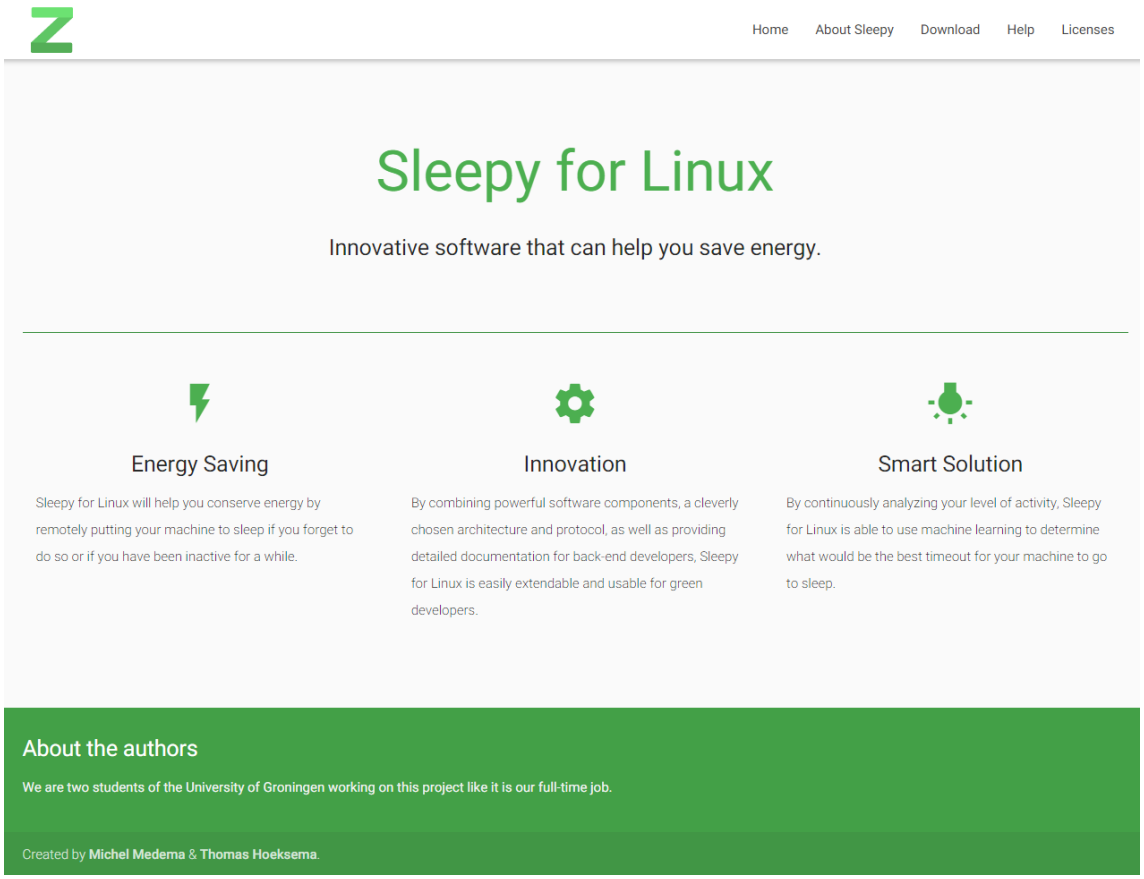

Figure 25: The platform website to improve on the perceived usefulness of the framework by users.

10.7 Appendix G: Source code

The source code of the framework is available through a private git repository for the authors and supervisors of this project. If the reader of this paper would like access to this source code for educational or reviewing purposes, they are advised to contact the authors of this paper.

11 References

[1] Setz B., Pul R. (2013), *Lazy Sleep - a Green Mind project*, University of Groningen, July 12.

[2] Nizamic F., Anh Nguyen T., Lazovik A., Aiello M. (2014), *GreenMind - An Architecture and Realization for Energy Smart Buildings*, Atlantis Press, ICT4S 2014, p. 20-29.

[3] Nizamic F., Anh Nguyen T. (2013), *Bernoulliborg - The Building of Sustainability*, Distributed Systems Group, JBI, FWN, University of Groningen - GreenMind competition 2013.

[4] Setz B. (2015), *Context Aware Power Management based on User Behaviour*, University of Groningen, Msc thesis.

[5] Schaaf R., Kollenstart M. (2015), *Energy Smart Office*, University of Groningen, July, Bsc thesis.

[6] Balakrishnan H., Seshan S., Amir E., Katz R. H. (1995), *Improving TCP/IP performance over wireless networks*, ACM, Proceedings of the 1st annual international conference on Mobile computing and networking, p. 2-11.

[7] Lanzisera S., Nordman B., Brown R. E. (2012), *Data network equipment energy use and savings potential*, Buildings, Energy Efficiency, vol. 5, nr. 2, p. 149-162.

[8] Bray M. (2006), *Review of Computer Energy Consumption and Potential Savings*, White Paper, Dragon Systems Software Limited (DssW), December.

[9] Kawamoto K., Shimoda Y., Mizuno M. (2004), *Energy saving potential of office equipment power management*, Energy and Buildings 36, p. 915-923.

[10] Safran J. (2009), *Power Management for Networked Workstations: Going Green at the College of William and Mary*, College of William & Mary, April 24.

[11] Nedevschi S., Chandrashekar J., Liu J., Nordman B., Ratnasamy S., Taft N. (2009), *Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems*, NSDI, vol. 9, p. 381-394.

[12] Reich J., Goraczko M, Kansal A. Padhye J. (2010), *Sleepless in Seattle No Longer*, USENIX Annual Technical Conference.

[13] Agarwal Y., Savage S., Gupta R. (2010), *SleepServer: A Software-Only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments*, Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIX Association, p. 22-36.

[14] Sung W. P., Tsai T. T., Chen K. S. (2010), *Assessment of the energy-saving and carbon reduction efficiency of an air-conditioned office in Taiwan*, ICRM2010, Green Manufacturing, Ningbo, China, p. 128-133.

[15] Das T., Padala P., Padmanabhan V. N., Ramjee R., Shin K. G. (2010), *LiteGreen: Saving Energy in Networked Desktops Using Virtualization*, USENIX annual technical conference.

[16] Sato K., Samejima M., Akiyoshi M. (2012), *A Scheduling Method of Air Conditioner Operation using Workers Daily Action Plan towards Energy Saving and Comfort at Office*, IEEE, Emerging Technologies & Factory Automation (ETFA), IEEE 17th Conference, p. 1-6.

[17] Fischer M., Wu K., Agathoklis P. (2012), *Intelligent Illumination Model-Based Lighting Control*, IEEE, Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference, p. 245-249.

[18] Kazandjieva M., Heller B., Gnawali O., Levis P., Kozyrakis C. (2013), *Measuring and analyzing the energy use of enterprise computing systems*, Sustainable Computing: Informatics and Systems 3, p. 218-229.

[19] Kitagami S., Matsushita M., Kaneko Y. (2013), *An Energy-Saving Office Lighting Control System Linked to Employee's Entry/Exist*, IEEE 2nd GCCE, p. 440-444.

[20] Kaneko Y., Matsushita M., Kitagami S., Kiyohara R. (2013), *An energy-saving office lighting control system linked to employee's entry/exist*, IEEE, 2nd Global Conference on Consumer Electronics (GCCE).

[21] Singh S., Woo M., Raghavendra C. S. (1998), *Power-aware routing in mobile ad hoc networks*, Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking, p. 181-190.

[22] Smailagic A., Kogan D. (2002), *Location sensing and privacy in a context-aware computing environment*, IEEE, Wireless Communications, vol. 9, nr. 5, p. 10-17.

[23] Harris C., Cahill V. (2005), *Exploiting user behaviour for context-aware power management*, IEEE, Wireless And Mobile Computing, Networking And Communications 2005, IEEE International Conference, vol. 4, p. 122-130.

[24] Krause A., Ihmig M., Rankin E. et al (2005), *Trading off prediction accuracy and power consumption for context-aware wearable computing*, IEEE, Wearable Computers 2005, Proceedings Ninth IEEE International Symposium, p. 20-26.

[25] Harris C., Cahill V. (2007), *An empirical study of the potential for context-aware power management*, Springer Berlin Heidelberg, p. 235-252.

[26] Ravi N., Scott J., Han L., Iftode L. (2008), *Context-aware battery management for mobile phones. In Pervasive Computing and Communications*, IEEE, PerCom 2008, Sixth Annual IEEE International Conference, p. 224-233.

[27] Wood A. D., Stankovic J., Virone G. et al (2008), *Context-aware wireless sensor networks for assisted living and residential monitoring*, IEEE, Network, vol. 22, nr. 4, p. 26-33.

[28] Herrmann R., Zappi P., Rosing T. S. (2012), *Context aware power management of mobile systems for sensing applications*, Proceedings of the ACM/IEEE Conference on Information Processing in Sensor Networks.

[29] Davis F. D. (1986), *A technology acceptance model for empirically testing new end-user information systems: theory and results*, Massachusetts Institute of Technology Archives, December 20.

[30] Davis F. D. (1989), *Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology*, MIS Quarterly, vol. 13, nr. 3, September, p. 319-340.

[31] Brooke J. (1996), *SUS-A quick and dirty usability scale*, Usability evaluation in industry, 189(194), 4-7.

[32] Spool J., Schroeder W. (2001), *Testing web sites: Five users is nowhere near enough*, ACM, CHI'01 extended abstracts on Human factors in computing systems, p. 285-286).

[33] Woolrych A., Cockton G. (2001), *Why and when five test users aren't enough*, Toulouse, France: Cépadèus, Proceedings of IHM-HCI 2001 conference, vol. 2, p. 105-108.

[34] Faulkner L. (2003), *Beyond the five-user assumption: Benefits of increased sample sizes in usability testing*, Behavior Research Methods Instruments & Computers, vol. 35, nr. 3, p. 379-383.

[35] Bangor A., Kortum P. T., Miller J. T. (2008), *An empirical evaluation of the system usability scale*, Intl. Journal of Human-Computer Interaction, vol. 24, nr. 6, p. 574-594.

[36] Lewis J. R., Sauro J. (2009), *The factor structure of the system usability scale*, Springer Berlin Heidelberg, Human Centered Design, p. 94-103.

[37] Nielsen J. (2012), *Why you only need to test with 5 users, 2000*, Jakob Nielsen's Alertbox

(Available online at `http://www.useit.com/alertbox/20000319.html`)