Managing Configuration History in Domestic Networks

by

Robert Spencer BSc (Hons)

Thesis submitted to University of Nottingham for the degree of Doctor of Philosophy

May 2018

Abstract

Domestic Networks are gaining in complexity, with an increasing number and variety of devices. Increasing complexity results in greater difficulty managing configuration and troubleshooting when problems occur.

This thesis presents strategies to assist users in managing the complexity of their networks. The work is split into several parts.

First, configuration changes are tracked and users are presented with a timeline of changes to their network.

Provision of a selective undo system is the second feature. The undo facility is designed to allow any change to be undone independently of any other.

Users are also given the option of reverting to an earlier point, either before a specific change, or to a specific timestamp.

The next feature is use of notifications. Any changes that require further actions can be broadcast to users directly. Changing Wi-Fi configuration is one example.

The range of devices in use makes changing Wi-Fi configuration (and the subsequent reconfiguration of devices) a challenge, because the devices affected may be part of the infrastructure of a home (lights or thermostat for example). Because these devices have unique methods of network setup, restoring connectivity to every device can be challenging. This thesis also presents a method of changing Wi-Fi configuration which allows users a grace period to reconnect all their devices.

Each of these features was assessed by a user study, the results of which are also discussed.

Acknowledgements

I would like to thank my supervisors Professor Tom Rodden and Dr Richard Mortier whose support and insight have been invaluable throughout my Ph.D journey.

I would also like to thank my colleagues in the Mixed Reality Lab. I am particularly grateful to Anthony Brown for tolerating my constant queries and need to brainstorm.

I would also like to thank my family and friends for their support. In particular, I would like to thank my parents for always believing in me, and providing much needed motivation.

Lastly, I would like to thank my Grandma, whose support over the last year has been invaluable in allowing me to focus on my work.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

List of Abbreviations

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
A/V	Audio/Visual
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CSS	Cascading Stylesheet
DHCP	Dynamic Host Configuration Protocol
DIY	Do It Yourself
DNS	Domain Name System
DOM	Document Object Model
EAP-TLS	Extensible Authentication Protocol Transport Layer Security
HCI	Human Computer Interaction
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ISP	Internet Service Provider

JS JavaScript

JSON JavaScript Object Notation

JST JavaScript Template

LDNS Local Domain Name Server

MAC Media Access Control

NAT Network Address Translation

NIC Network Interface Card

NFC Near Field Communication

OS Operating System

PIN Personal Identification Number

POP Post Office Protocol

QoE Quality of Experience

QoS Quality of Service

RC Recovery Command

RF Radio Frequency

RI Recovery Information

RO Recovery Object

RSSI Received Signal Strength Indication

SDN Software Defined Networking

SSID Service Set Identifier

SUS System Usability Scale

TCP Transmission Control Protocol

UI User Interface

URL Uniform Resource Locator

UX	User Experience
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
WPS	Wi-Fi Protected Setup
WYSIWIS	What You See Is What I See

Chapter 1

Introduction

Home networks are an increasingly common feature [113, 130]. As technology advances, with ever-more sophisticated uses of technology that relies on Internet access, this trend is likely to continue. This trend also means that networks are increasing in complexity and the subsequent maintenance challenges harder to address. As such, the desire to find ways to address the issues faced is growing (judging by the amount of existing research, such as [13, 29, 79, 158] for example).

1.1 Background and Motivation

There has been a great deal of research ([53, 157, 159] for example) into the characteristics of domestic environments. Homes have unique characteristics which have the potential to influence the design of technology in the home, with networking [32, 114, 115] being a prime candidate for improvements.

The need for changes to networks has been shown in many different places, most notably in "The work to make the home network work" [79].

In "Unremarkable Networking: The Home Network as a part of everyday life", Crabtree et al. [53] show that tasks necessary for successful network operation are merely tasks which maintain normal operation, rather than a means to an end. At the same time, there is a need to balance this with the needs and usual routines of the household. Householders consider network management a mundane aspect of life in the home and the need to perform maintenance tasks is left to whoever is appropriate unless the necessity of a fix is urgent [53]. Relying on a single person has the potential for difficult troubleshooting phone calls if problems occur when this person is not at home.

Sventek et al. [155] have demonstrated that network maintenance tasks require considerable knowledge. With the surge in products that have an Internet connection, the range of devices connected to networks has increased [160]. Networks have increased in complexity, making networking knowledge crucial. Many of these products lack traditional user interfaces or input devices. The Philips Hue bulb[1] and the Nest products[2] are some examples.

As a result of this complexity, there have been many attempts to address the issues faced in connecting new devices to the network. Multinet [29] and ICEBox [177] are two examples.

There is also evidence that users want more information about the utilisation of their network, which has lead to research into methods of exposing bandwidth use and the perception of who can do what, on which devices and when. The Homework Router [116]—the foundation of the router presented in this thesis—is one such example.

Users are often left helpless when problems occur [79]. Restarting either the router or devices is a common troubleshooting method [178]. Making router state apparent to users and providing a method of troubleshooting appears to be a less understood area.

The amount of attention focused on improving home networking in research is extensive[3]. However, the primary focus of most research covers connecting devices or highlighting bandwidth usage.

The area of configuration management and troubleshooting is one that has been studied relatively little, even though there is evidence that users have difficulty with understanding or remembering historical changes [79]. Previous troubleshooting research focuses on using specific techniques to solve specific problems (such as ICEBox [177] or Eden [179]). There has not been much focus on the general problem of configuration management and how it relates to troubleshooting issues that may occur.

[1]http://www2.meethue.com/en-gb/
[2]Including thermostats, cameras and smoke alarms http://nest.com
[3]See Chapter 2

1.2 Aims and Objectives

This thesis aims to explore the configuration of home networks, with particular emphasis on providing users with a history of changes made to configuration, and the suitability of this information as a troubleshooting aid. The motivation behind this is because configuration changes may make sense at the point they are made, but may have unintended consequences at a later point, particularly if a large amount of time has passed.

Another method used to expose configuration changes to users builds on the work of The Homework project [116], and the use of notifications to users under certain conditions. As an example (from the Homework project functionality), users can receive notifications when a device exceeds a usage quota or used to access a restricted website (such as parents wishing to restrict access to social networks).

Designing a router that tracks changes to its configuration and allows users to view them is one way of assisting users with the process of troubleshooting network issues. Informing users of prior changes is intended to make the identification of the problem simple enough that it could be solved the first time, without the potential for recurrence, due to an incorrect diagnosis.

Another aspect of improving the management of configuration is improving the workflow of making changes to Wi-Fi configuration. Earlier research[4] has already addressed the challenge of connecting devices, so this thesis focuses on how updates are applied to aid in the process of reconnecting devices.

The overall aim is a router that emphasises keeping users informed of any changes that occur so that they can be able to address any issues that may occur.

1.3 Research Questions

1.3.1 Technical

1. Is it possible to create a history of configuration changes in a router?

2. Can changes be undone and redone?

[4]ICEBox [177] and Multinet [28, 29]

3. Is it possible to revert to an earlier point in time?

1.3.2 HCI

1. Is a timeline of changes a useful network management/troubleshooting tool?
2. Does 'undo' improve users' ability to manage home networks?
3. Is 'revert' a useful safety net when problems arise?
4. Do notifications of network changes aid users' ability to understand their network?

1.4 Outline of the Thesis

Chapter 2—Literature Review

This chapter presents related research, both to provide context and to illustrate the issues inherent in network management and undoing. Previous research influences the features and design of the router presented in this thesis.

Research areas discussed are ubiquitous computing, characteristics of the domestic environment, and home network management. Understanding the challenges that ubiquitous computing present and the characteristics of homes highlights the potential for challenges with the usage of technology. A consequence of many devices requiring an Internet connection is that networking management is directly affected by the environment and the devices contained within.

Chapter 3—Undo

It is helpful to examine the different undo methodologies to identify techniques that could apply to network configuration tracking, specifically concerning methods of storing the history such that events could be undone appropriately.

This chapter considers several different methods for undoing and considers the intended use-cases and method of operation of each. These details are used to identify advantages and disadvantages of each, in the context of network configuration.

Understanding existing approaches to undoing in a variety of situations highlights factors that should be considered when designing a new method of undoing.

Chapter 4—Design

This chapter discusses the design of the router, with particular emphasis on how the literature has impacted both the features of the system, (as identified by home networking research), and the underlying data structures used. The data structures used were influenced both by the requirements of the Homework router features in use, (to ensure they can be modified to support configuration tracking) and by the methodology of undoing. The undo methodology is developed from the result of discussions in chapter 3.

Chapter 5—Implementation

This chapter outlines implementation details for the router. The router implementation details include how the Homework Router [116] components have been modified to track changes and support undoing.

Chapter 6—Experiment

This chapter outlines the lab-based study used to evaluate the suitability of configuration history and undo as a troubleshooting tool.

Chapter 7—Discussion

This chapter is an analysis of the results of the study, to determine the effectiveness of history and undo, including a discussion of issues identified in the study.

Chapter 8—Conclusion

This chapter is a summary of the work completed and the results of the study. The result of the study is used as the basis for a discussion of potential expansion of the work to extend its usefulness, both regarding the tracked configuration, but also considering potential uses for configuration history to provide further improvements to network management.

1.5 Contributions of this thesis

1.5.1 History

While some router firmware[5] provide users with the ability to view and modify pending changes, building a history of applied changes could be used as a troubleshooting tool.

For configuration history to be used as a troubleshooting tool, it is necessary to ensure that the changes are presented clearly. Presenting a log of technical details will provide less assistance without prior networking knowledge.

1.5.2 Undo and redo of network configuration

Being able to undo changes is important in applications, especially when it may not be clear to users which action they are required to perform.

Providing an undo facility requires a history of changes. The requirements that an undo feature must satisfy will depend on the use case, but will always influence how history is stored or modified, and how the Undo command will navigate the history.

For network configuration, it is important that each configuration item can have changes applied independently, to reduce the number of undo and redo operations required.

[5]Such as OpenWRT (https://openwrt.org/)

To allow configuration history to be used as a troubleshooting tool, it is also important to ensure that the undo operation will be added to the history, such that it can be observed by users, and can also be undone.

1.5.3 Revert

Not all problems have a clear cause, especially for those with limited networking knowledge, it is important to consider the option of reverting to an earlier point.

Reverting can be achieved by extending the undo function to support changes to multiple items. However, if taking this approach, it is useful to identify the earliest change made to a configuration item after the selected timestamp, so that there are no unnecessary additional invocations of undo.

1.5.4 Multiple Wi-Fi Networks

It is usual for devices to be disconnected when Wi-Fi configuration changes are made. However, it is also possible to utilise the ability to create additional networks to provide a grace period for allowing devices to be reconfigured.

However, providing a grace period to allow devices to reconnect does not solve the challenge of how to managing network changes on each device. The effects are reduced because there is no need to reconnect all devices at the same time. However, devices do not always present available networks clearly, which will be more apparent if the Service Set Identifier (SSID) is unchanged.

1.6 Summary

The need for more transparency regarding configuration changes for home networks. This chapter introduced the problem to be addressed, with a brief explanation as to the motivations behind the project, and an overview of the structure of the thesis.

The next chapter will discuss prior research that is relevant to this thesis, to demonstrate the need for such a system. The next chapter will also highlight where the topics discussed in this thesis fit into the broader research agenda.

Chapter 2

Literature Review

2.1 Introduction

The focus of this thesis is providing a history of configuration in domestic networks. This chapter presents an overview of previous work in related fields, namely home networking, user support and ubiquitous computing. Chapter 3 covers Undo techniques.

Before exploring what is meant by configuration history, it is important to understand the constraints placed on technology by occupants of a home, the reasons for them, and how these differ from other environments. It is only by understanding the nature of homes that we can begin to understand the need to explore the issues surrounding technology in the home, and the challenges presented in use that would have a potential influence on design.

Everyday fixtures and fittings in homes, as well as existing products, are becoming network enabled, which means that an increasing number of devices are interconnected. How to manage these connections and ensure optimal performance are now receiving attention from researchers and has resulted in new companies building networking devices (Google,[1] and Eero[2] for example).

With an understanding of underlying characteristics of the home and the technology within, it is then important to understand the impact on networks specifically,

[1]https://madeby.google.com/wifi/
[2]https://eero.com/

which is important to outline the need for any of the many and varied research projects interested in home networking.

Another factor that influenced this research is the difficulties faced with technical support, both for the helper and the seeker of help. It is useful therefore to briefly discuss the issues faced when giving or receiving support and identify potential areas for improvement.

While it is often necessary for devices to be configured, in certain situations, management of the configuration can be automated.

2.2 Home Networks

As Internet use increases [113, 130], the number of smart devices in homes is ever increasing, many of which require Internet access. Examples of such devices are the Philips HUE light bulb[3], the Nest Thermostat[4] and an ever-increasing number of media devices such as TVs or products such as the Chromecast,[5] the Apple TV,[6] or the Amazon Echo[7].

There are other products which have Wi-Fi capability, where the inclusion of Wi-Fi or mobile applications may not be as necessary. The Smarter[8] kettle and coffee maker are two examples. However, these products show that there is potential for *any* electrical item to use an Internet connection, which suggests that the number of devices on networks and the challenges faced in managing them are both set to increase.

All of these devices employ a slightly different method of connecting to the network. The common factor with these types of devices is the lack of a keyboard, which results in manufacturers inventing a new way of inputting connection details. One method (used by the Chromecast and the Amazon Echo) is for the device to broadcast a Wi-Fi network to be used by another device to provide the correct details to use.

[3]http://www2.meethue.com/en-gb/
[4]https://nest.com/uk/thermostat/meet-nest-thermostat/
[5]https://www.google.com/intl/en_uk/chromecast/ (correct as at 25/09/17)
[6]http://www.apple.com/uk/tv/ (correct as at 25/09/17)
[7]https://www.amazon.co.uk/gp/product/B01GAGVIE4 (correct as at 25/09/17)
[8]http://smarter.am/

The need for home networks may be increasing (as a result of a broader range of devices which require networks), but they are difficult to configure, to the extent that users' frequently return networking equipment to stores [145] because of a lack of understanding as to how equipment should be used [148].

The range of devices and the possible uses for networks is now so great, and the network is such a significant part of life, that network management could be considered life management [53]. As a result, when designing network devices, the focus should be on user activities (such as Skype or email) rather than the technology required to implement the activities.

When studying home networks, there are two possible approaches [32]. The first is to consider the network from the perspective of its users, members of a household. When accounting for network usage, users will identify *what* they need to do, without regard to the technologies used (banking or shopping instead of web browsing, or sending messages instead of emailing). Routines play a big part in determining what resources are used, and when. Often network logs will appear to show one usage pattern, but the actual usage will be different. For example, it may appear that a website is logged in constantly all evening, but in reality, it might be open in a tab, but not used.

The other approach is to consider the network from a technical perspective. The view from the network will need to attempt to identify usage patterns, which is becoming increasingly difficult because of the use of HTTPS for services which might have used other protocols previously. It is common to access email using a web-based client which uses HTTPS, whereas historically email client applications would have used POP or IMAP. Using HTTPS instead of POP or IMAP removes the ability to identify the application from the protocol. POP and IMAP are both only used for receiving emails. HTTPS has many uses and encrypts traffic.

One aim of home networking research is to advance the available technology and how it is used, such that all the networks in a home (electricity, phone, A/V, RF, security and temperature control) are integrated. The Aladdin project [169, 170] intended to work towards those goals.

2.2.1 The Internet at Home

The Internet has a dumb middle, smart-edge architecture [144]. The protocols which underpin the Internet are designed to be generic, to support any application, whether existing or new.

Home networks have the same design and architecture as the Internet[63]. There are advantages to this approach, not least of which is maintaining compatibility with existing technology.

The problem with having the 'Internet at home' is that the network is dumb. The devices must be configured to work with the network. Where there is a requirement for configuration, there is the possibility of misconfiguration. Attempting to make the Internet at home has led to many of the usability problems [22, 63, 148] in home networks.

In general, a move toward home networks with a smart middle would move some of the management burden from devices to the network [35]. Blumenthal and Clark [22] go further and suggest that the design of the Internet could be modified to support the range of applications currently in use. However, a complete redesign of the Internet is not practical because a large-scale redesign of the Internet would require that every device on the Internet be modified at the same time [22]. Blumenthal and Clark do agree with Calvert et al [35] that updating the architecture of networks in the home, and making the router maintain compatibility with the Internet is a sensible compromise.

Using a smart middle approach is advantageous because it removes much of the configuration required. A router which is more aware of the network could also automate many tasks. Network configuration should be as automated as possible. The tasks which require user input should be set using policies. Examples of tasks which require user input are deciding which devices are allowed to access the network, or which resources they can access.

The Internet was initially designed for military use. On the basis that devices and users are trustworthy [148], low-level authentication was not included in the design of low-level communication protocols. This design assumption is not necessarily true anymore. It does not apply to home networks, so a better approach would be to assume a device is untrustworthy unless explicitly told otherwise [148].

Every device on a network must have an Internet Protocol (IP) address. Dynamic Host Configuration Protocol (DHCP) servers can provide devices with address information, but each infrastructure device (access point or range extender for example) will often have a separate DHCP server. Having multiple independent servers allocating IP addresses means there is potential for two devices to be allocated the same address, or for each server to allocate separate ranges of addresses, which means that devices will only be able to communicate directly with devices using the same DHCP server.

Home networks exacerbate another problem with the Internet. Handley argues that the Internet is 'not fit for purpose' [83], but the number of home networks means that any improvements have limited effect because of lack of support. Home networks are problematic because infrastructure devices (routers or access points) are not updated frequently [83]. Updates are only likely to happen when changing Internet Service Provider (ISP) or changing to a new package which requires a new router. Not updating means that any new protocols take a long time to be deployed sufficiently that support for old protocols can be removed. IPv4 and IPv6 are a prime example of this. Many networks do not support IPv6, even though the pool of available IPv4 addresses has been exhausted since 2011 [83]. Network Address Translation (NAT) is one method of increasing the number of available IPv4 addresses (the pool of available addresses emptied in February 2011 [85]), because each network can use the same address ranges (192.168.0/24 or 192.168.1/24), as long as the modem has an address which is unique. The router is responsible for ensuring that devices within a network can communicate to the Internet by changing data packets to use its address when sending, and forwarding replies to the original devices. NAT interferes with applications which require that source and destination addresses are not changed (such as a media server)[9].

Network management is complex enough that Brush [33] argues that many network maintenance tasks should be handled by professionals, in the same way as other maintenance tasks, such as electrical or plumbing repairs. There are tasks which can be left to householders, such as connecting a new device.

Software Defined Networking (SDN) makes networks programmable. It is possible for some tasks to be moved off the router. SDN could be used to slice a network into parts that could be handled directly by appropriate professionals, such as an

[9]Plex is one example https://www.plex.tv/

ISP [183]. For example, ISPs could manage customer networks and troubleshoot problems, without sending engineers to homes.

Networking equipment intended for use in homes is not designed to have one hundred percent reliability, so hardware failures and software crashes are common [148]. Lack of reliability is problematic because when problems do occur with the network, the edges will not be aware of them. The first indication of a problem with the network is when a device is incapable of performing a task, often without an apparent reason.

Bringing the Internet home presents security challenges. The assumption that devices are trusted means that they may not be secured, which may mean exposing an insecure device to the dangers of the Internet. Configuring NATs and firewalls so that devices can communicate presents further challenges [35].

Current protocols focus on making devices and applications work, so the assumption is that anything is permitted unless explicitly denied. Because ensuring that devices are secure is a challenge, it would be better to take the opposite approach, and block everything not explicitly permitted [35].

It may be more feasible to rearchitect home networks and have the home router maintain compatibility with the Internet [148] (rather than change all networks). This approach would allow home network issues to be addressed, without requiring a coordinated upgrade. Individual networks could be upgraded to a new architecture as is considered appropriate for each home, thereby ensuring that disruption is kept to a minimum.

Another extreme solution is to abandon the need for home networks and have networks which cover a geographical area. Not requiring networks in homes would allow network management to be carried out by experts. People would only need to consider how to connect their devices. One advantage of home networks is that devices on the network can share resources, but a large-scale network would not permit this, without virtual boundaries [148].

The IETF created an Internet draft aimed at addressing the problems with home networks [98]. The draft has now expired, but the fact that the IETF created the draft at all highlights the severity of configuration and security concerns caused by bringing the Internet home.

2.2.2 Wi-Fi Standards

For a device to be Wi-Fi certified, it must support the official IEEE 802.11 Wi-Fi standard document [87]. This version of the standard includes additions introduced since the previous version.

Home networks can be composed of many components. In [142], Bill Rose presents an overview of the standards and technologies that home networks use.

It is worthwhile considering the official IEEE802.11 Wi-Fi standards [86, 87] because these standards define how devices which can connect to Wi-Fi networks can behave. The official standards could be considered partially responsible for the difficulties faced when tackling network management tasks. Regardless of whether networking standards have caused usability problems or not, in the current form, the 802.11 Wi-Fi standards [87] limit the possible solutions to identified issues. Any significant infrastructure changes would require new standards [66].

2.2.3 Understanding Networks

Poole et al. studied the understanding of eighteen householder's networks [131]. They asked each occupant to sketch their network to remove the need for a knowledge of the correct terms [131]. Sketches are an accessible (for householders) method of identifying how people reason about their networks. Sketches can also highlight the level of knowledge of householders.

A consequence of the network infrastructure being hidden (hiding components, running cables through walls, use of Wi-Fi [49]), is that the network provides few visual clues as to its components or their topology. Hiding infrastructure components also means that they are likely to be forgotten about when attempting to understand the network, and when troubleshooting [79].

There is a difference in how network diagrams are arranged and the level of detail, between experts and novices, and between those who maintain the network, and those who do not, regardless of the level of experience with networking. In a study by Poole, Chetty, Grinter and Edwards [131], those participants who were experienced with networks arranged diagrams topologically, while those who were inexperienced preferred a spatial organisation scheme. Differences in understanding between householders who normally maintained the network and the other householders were clear from the different levels of detail provided in the sketches.

These findings, (and those found in [79, 147, 179]) suggest that if tools are to be understood by all household members (regardless of knowledge) devices should be arranged according to their location in a home. Organising devices by location is useful in helping householders understand the network if they do not normally manage it. Allowing the network to be understood by every member of the household is important in situations where the person with primary responsibility for the network is not at home when problems occur.

Devices are usually labelled based on whom they belong to (which could mean primary user if used by multiple people), or usage location, for example "Alice's Laptop" or "Kitchen Tablet".

Eden [179] is a network management tool which represents devices spatially, with devices placed in the house being able to access the network and those outside being unable to access the network. Devices can be arranged based on the location of use or to assign a relationship to a person (such as with a mobile device, because it might not have a fixed location). If a user wants more detail, Eden can display the topology of the network. One challenge when designing tools for use with a range of user experience is how to cater to all experience levels. Those with limited experience will often require basic details, and clear instructions for any actions, while experienced users may want to view more detail.

Linksys developed a network setup and management tool [68]. The network setup component explains the steps necessary to configure a network. Each stage has an overview image and also has step-by-step animations to provide users with an increased level of detail if required. The first image might be sufficient for users with higher knowledge. The animations are designed for those who need more support. The chosen approach means that the tool can be used by anybody, regardless of experience, without alienating novices or forcing experts to view details they do not need [68].

When changes to a network occur, the ability for others to understand the network decreases, because they have no way of keeping up with the changes that occur. A lack of history can cause issues when troubleshooting because constant changes are hard to keep track of, and subsequently, it is difficult to identify changes that caused problems [79].

2.2.4 Current Management Tools

There is currently a range of tools that can be utilised to perform network management tasks. These can be summarised [178] as

- Router

  Network settings can be managed on the router.

- Operating System (OS)

  Device OSs (Windows, Linux, MacOS, for instance) have tools for managing the connection of that device to a network.

- Other tools

  There are other tools available which provide additional features not found in either OSs or in Routers.

The tools used for network management vary between tasks and the experience level of users. However, the configuration interface of the router or operating system network configuration management are commonly used for each type of task [178]. Reasons for this are related to availability and cost:

- No desire to install other tools when the router or OS have tools built in
- Not wanting to pay for tools when the OS and router tools are included and have no cost
- Not aware that alternatives exist

Security (such as Firewall and Anti-malware tools) and access control are one area where commercial products are common, particularly amongst inexperienced users [178]. More experienced users were more likely to use router tools, because the security tools in routers (such as firewalls) can define the same rules for all devices at the same time, rather than repeating rules on each device individually.

Inexperienced users often rely on physical measures, such as observing Internet use, or unplugging/restarting devices. These examples apply to access control and troubleshooting respectively.

The fact that tools other than the router or OS exist (which may or may not be easier to use), but are often not used would suggest that the router and device OSs need to be as simple as possible.

In Yang and Edward's [178] study, participants—experienced and inexperienced in network maintenance—felt that built-in tools were difficult to use, and required an understanding of technical concepts.

2.2.5 Connecting Devices

The basic process of connecting devices to networks involves selecting the network name—SSID—and entering the passphrase, which is fairly easy on devices with physical keyboards and pointing devices. However, as already stated, there is an increasing range of devices that have a different form-factor, and this method of connection is either very difficult (entering a complex password on a touchscreen for example) or impossible (some devices have no means of entering text, such as some printers).

2.2.5.1 Wi-Fi Protected Setup (WPS)

The Wi-Fi Alliance has created an optional specification for one possible method of simplifying the process of connecting devices, called Wi-Fi Protected Setup (WPS) [10]. WPS allows devices to connect to a network through the use of PINs or by pressing a button on both the router and the device. Near Field Communication (NFC) can now be used to establish connections[10]. WPS ensures that WPA2 is used for security, and each device has an individual WPA2 encryption key.

WPS has a security flaw [163], especially when using PINs. Although the PIN is eight digits, it is verified as two sets of four digits, which makes them much

[10]http://www.wi-fi.org/news-events/newsroom
/wi-fi-certified-wi-fi-protected-setup-adds-nfc-tap-to-connect-for-simple-set-up

easier to brute force. More detail can be found in [188] with the addition of an alternative implementation that enhances the security.

The specification requires the PIN method for a device to be officially recognised as supporting WPS. Button pairing and NFC are both optional, even though the security issues are less severe in these cases because the router is only responsive to WPS pairing requests for a short period, which is unlike the PIN method where there is no limit on the time for PINs to be entered.

PINs are also easy to mistype, which leads to problems connecting. Also, using a static Personal Identification Number (PIN) printed on the router is another potential security problem, much like using default values for SSID or passphrase is considered less secure than changing the values (because they indicate that no effort has been made to secure the network).

In theory, WPS should be sufficient to address the problem of how to connect devices to networks. In practice, however, this is not the case, mainly to a lack of support in devices. Also, Zisiadis et al [188] suggests that support for WPS be removed from devices, or that users be advised to avoid using WPS to prevent the security flaw in the mandatory PIN method.

Push-button WPS has the potential to improve the situation because pressing buttons is a simple concept to both explain and perform, particularly if a physical button is used. Not all devices support this because the device design may make the utilisation of a button impractical, either because of a difficulty in placing a physical button, or no way to interact with a software button.

For any technology that forms part of the Wi-Fi specification, it would not be unreasonable to assume that any devices that are Wi-Fi certified would conform to the entire specification, and consequently, be able to use WPS. WPS support should mean that there would not be a need for alternative systems to configure devices on the network in a secure manner. When viewed from this perspective, the fact that there have been several attempts by researchers to simplify the process of connecting devices would suggest that WPS has failed to address the problem sufficiently. These are outlined in the next section.

2.2.5.2 Alternatives

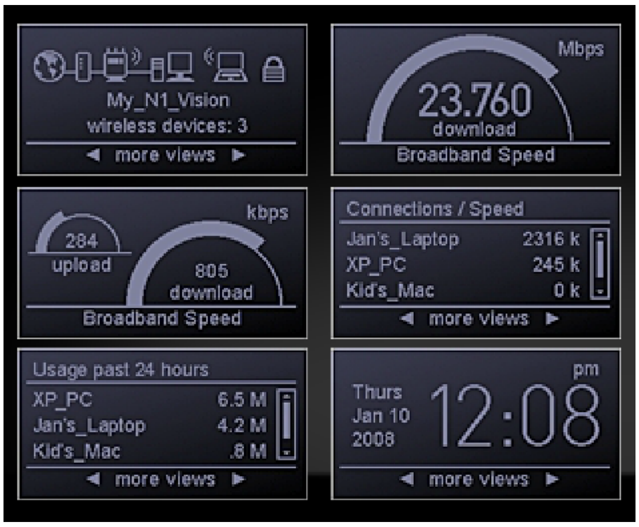
The limitations of WPS mean that it is often necessary to enter network configuration on devices manually, even if WPS was supported on the network. As a result, there is a significant research effort dedicated to improving the process of managing the devices connected to a network.

One possibility, developed by Brown, Mortier and Rodden [28, 29], is to ignore the concept of a single network to which each device must connect. Instead, it is assumed that each device is preconfigured with an SSID and WPS2 passphrase. These are displayed on devices in the form of a QR code. A mobile device with a camera would scan this code, and securely transmit these details to the router. The router would then be configured to run this network in addition to networks for other devices. Rather than having a single network that each device must connect to, it is the router that must adapt to the addition of new equipment.

Another solution is ICEbox [177], which is designed to automate the management of devices on the network and transform those actions that users must do into physical actions, such as pointing or turning a key. Connecting a new device requires pointing the device at ICEbox, which will start the connection process. As well as providing the device with network connection details, ICEbox also receives other device details such as its type, to allow it to configure network devices, such as printers, if appropriate. When new devices are added to the network, other devices are notified if appropriate. For example, devices would be notified if a new printer was connected, so that other devices can install it.

The ICEbox is a separate appliance, with a touchscreen, infrared (to connect new devices) and Ethernet (for communicating with networked devices), and a keyed lock, which restricts ICEbox's ability to connect new devices or manage the network. ICEbox is similar in style to a security alarm control panel. The intention is to ensure that ICEbox is on a wall in a prominent location, rather than being hidden, as is often the case, because doing so makes it harder to alert users to problems.

Network-in-a-Box [13] uses infrared to trigger device connections, which are ultimately secured using 802.1x EAP-TLS (enterprise-grade security). Infrared is used to share enough data so that the network can be used to transmit other data



My_N1_Vision
wireless devices: 3
◄ more views ►

Mbps
23.760
download
Broadband Speed

kbps
284
upload
805
download
Broadband Speed

Connections / Speed
Jan's_Laptop 2316 k
XP_PC 245 k
Kid's_Mac 0 k
◄ more views ►

Usage past 24 hours
XP_PC 6.5 M
Jan's_Laptop 4.2 M
Kid's_Mac .8 M
◄ more views ►

pm
Thurs
Jan 10
2008
12:08
◄ more views ►

FIGURE 2.1: The N1 Vision Display

necessary to configure devices, but have the communication be trusted as a result of using keys sent using infrared.

SDN can also be used to configure devices that wish to connect to a network [103] automatically. An OpenVSwitch controller uses a database of devices to identify a new device from packet flows and provide the device with appropriate configuration, without requiring user interaction.

2.2.6 Commercial Products

There has been a small number of commercial routers designed to address some of the issues identified in the literature.

First, Belkin N1 Vision[11]. This router has an information screen (Figure 2.1) that can show the current state of the network, such as Internet connection status, the number of devices connected, and the speed of the Internet connection. This router was an early example of a commercially available router that attempted to make it easy for users to check the status of their network. However, it is no longer available.

[11]http://www.belkin.com/n1vision/intro/

Linksys developed 'Smart Wi-Fi'[12,13], which moves the management interface of supported routers to a central location, making it possible to manage the network from anywhere by using a range of mobile applications. The main router application provides several features, such as remote network management and configuration, guest network setup, and parental controls, with other applications that support the platform. Some routers support Smart Wi-Fi, each providing a different set of features.

2.3 Ubiquitous Computing

It is important to consider ubiquitous computing research because the range of devices which could be considered to be providing ubiquitous computing is growing, which means that the need for reliable access to the Internet is increasingly important. Weiser introduced the area of Ubiquitous Computing [173] (Ubicomp) in 1991. His vision was one of 'people and environments augmented with computational resources that provide information and services when and where desired' [7]. Researchers have been attempting to achieve Weiser's vision ever since.

Weiser's vision included a range of computing devices which operate at different scales, from small personal devices to large multi-user devices. Weiser's vision of Ubicomp extends beyond the infrastructure of devices, and into the realm of applications which leverage the new infrastructure to enable new paradigms of interaction.

Having a broad range of hardware, and the ability to access data at any time means that Ubicomp devices need to support constant interaction, at any time of the day or night.

Another aspect of Weiser's vision is that technology must become invisible in use (for user input). Enabling computer usage to scale to a range of form factors and to enable access from anywhere at any time requires new form factors and input paradigms. The PARCTab [172] uses the pen-based input system Unistroke [74] and is an early example of the shift towards Weiser's vision.

[12]http://linksyssmartwifi.com
[13]A list of routers that are compatible with Smart Wi-Fi is available from [105]

Ubiquitous computing inspires developers to create applications which use interaction techniques which move away from the traditional keyboard, mouse and display paradigm. There is an expectation that people will interact with their devices in the same way as they interact with other objects in the physical world, using speech, gestures or writing utensils. For products to achieve invisibility in use, they must utilise these natural interaction paradigms because traditional input methods (keyboard and pointing device) have an implicit 'I need to interact with my computer now' shift in focus. Voice-controlled digital assistants, such as Amazon's Alexa, Google Assistant[14], Siri[15], or Cortana[16] add voice control to a range of products and services, such as Hue lightbulbs[17].

Context awareness is a fundamental concept in ubiquitous computing [7]. There is more to context than location or identity. Context should focus on the "five w's", *who*, *what*, *where*, *when*, and *why*. When considering context, *who* refers to *all* people who are involved with an interaction, not just the recognised individual.

The goal of many context-aware applications is to provide users with information appropriate to their real-world actions, in real-time. Augmented reality systems which augment vision and sound with real-world interactions, and use the real-world interactions as input. Augmented reality creates a close integration between context-aware interactions and the physical world [106, 107, 152].

Because context-aware systems must interact with people, they must infer human intent [17], and act accordingly. They must be accountable to their users. It must be possible for users to understand the functionality of a context-aware system.

The requirements for ubiquitous computing (range of devices, increasing usage, natural interfaces, and easy access to information and services) have given rise to an ever-increasing variety of products which all require Internet access and all have different form factors.

Of these, easy access to information is perhaps most relevant. Being able to access information from any device at any time means that networks are increasingly important. Ubiquitous computing devices require an Internet connection if they are to satisfy the demand for information anywhere and at any time. Unfortunately,

[14]https://assistant.google.com/
[15]https://www.apple.com/ios/siri/
[16]https://www.microsoft.com/en-gb/windows/cortana
[17]http://www2.meethue.com/en-us

home networks cannot keep up with the demand placed on them, and they are difficult to install and manage [28, 177].

Many ubiquitous computing devices have non-standard interfaces, which can complicate integrating the devices into the home network. Network infrastructure devices (routers, modems, range extenders, for instance) are often designed to use connection procedures that require a keyboard and display at a minimum. Other devices are required to present a list of SSIDs and prompt for a passphrase for the selected network, which can be much harder when using devices with different interaction modes [122, 129, 131].

Ubiquitous computing device manufacturers create expectations for device use-cases. Advertisements can be produced using an ideal environment for the desired functionality [23]. The shift as mentioned above in form factor and user interface is at odds with the traditional device interfaces to networks (select the required SSID and type the passphrase). Users have networks partly because of the expectation of utility. Difficulties in use mean that user expectations are often broken [23].

If ubiquitous computing devices are to achieve a level of integration where they too are invisible in use, networks must adapt accordingly. It is useful to understand the impact of ubiquitous computing, the nature of the domestic environment, and how these influence each other if networks can be improved sufficiently to meet the changing requirements.

2.4 Domestic Environments and the Impact of Technology

Understanding how homes differ from other settings—with particular consideration of the use of technology—facilitates understanding the issues that people may face with their home networks.

Many technologies have become common in domestic settings, some of which predate the widespread use of networks in the home. Indeed, research into 'smart homes' (homes with at least one infrastructure device which has a connection to the Internet) pre-dates research into home networks by at least ten years.

Any interconnected devices, such as those with Bluetooth, present other usability challenges, particularly in light of differing levels of technical knowledge. Edwards and Grinter [62] explored some of the problems inherent in connecting Internet-enabled home infrastructure devices to a network, including usability challenges that occur when using wireless devices. Removing wires eliminates the explicit definition of connections between devices. An example suggested in the paper involves a Bluetooth stereo and speakers, and the fact that a neighbour could set up their Bluetooth speakers and find they connect to the original stereo. This example serves to illustrate the need for products to be as easy to understand as possible, to avoid such situations happening with other devices. The third challenge is of particular importance: the lack of a systems administrator. For devices to function correctly together, applications and drivers have to be kept up-to-date, or new peripherals purchased.

Another aspect to consider when designing for the home is familiarisation. Bell, Blythe and Sengers present the idea that homes are familiar [16]. As such, if designers of everyday objects make assumptions as to how devices will be used there will be an impact on usability. The act of analysing everyday objects and challenging fundamental assumptions can lead to design and usability improvements. The concept of assessing the design of mundane objects is considered in detail in 'The design of Everyday Things' by Donald Norman [121][18].

In "The Ethnography of Infrastructure" [151], Star issues "a call to study boring things". Star's definition of infrastructure is broad, but it does highlight a need to subject 'boring' things to ethnographic studies, (much like Bell, Blythe and Sengers [16]), and to study how different people view infrastructure. Based on Star's definition, networks are infrastructure, so considerations should be made at a low level regarding their design or usage.

Domestic ubiquitous computing devices depend on home networks. Chetty, Sung and Grinter [49] argue that networks should be an integral component of the infrastructure of a home. However, installing networks in a manner which satisfies this assumption is a challenge and one which must be solved if ubiquitous computing devices are to become widespread in homes [49].

[18]This is the updated version which includes items not available when the original book was published [120], so it is of greater relevance

According to Brand, six layers form the composition of homes [26], each of which plays a role in a home's ability to be networked sufficiently to support ubiquitous computing:

1. Site

   The physical properties of a home's location affect the services it can utilise, ranging from the type or speed of network connection to the availability of other services (such as television subscriptions). The site of a home can also influence the infrastructure required. If a home were subject to frequent power outages, a UPS might be required.

   The presence of a network can alter the perceptions of what 'site' means. The range of a wireless network does not match the boundaries of a home's plot, and for any given home, there may be networks belonging to others which are in range.

   An incorrectly configured network may allow devices located outside the boundary to connect, and not those inside [35].

2. Structure

   The physical structure of a home can influence networks. The presence of walls may pose challenges with using Ethernet and may hinder getting Wi-Fi signal throughout the house.

3. Skin

   The skin of a building is its outside appearance, which is of limited significance to networks (unless there is a desire for outdoor usage, in which case the presence of the outside wall may be a structural problem).

4. Services

   External services (for example, electricity, or an ISP) will be required to build a network. There will be a need for sufficient power sockets and a suitable data connection. Also, television and phone services may be required. There may also be a desire to install Ethernet cables.

   While services will often be installed and maintained by external organisations, householders often need to be responsible for wiring homes, particularly for Ethernet. In the case of Ethernet, the cost of installation is a factor, although routing cables to where they are needed and identifying them when connecting everything can be a challenge.

A challenge inherent in the use of external services is ensuring that the provision of each allows all services to interoperate correctly, which can mean that power and data connections are in a suitable location.

5. Space Plan

   The physical layout of rooms will influence the availability of networks (particularly with wireless, because walls may reduce signal strength). Chetty, Sung and Grinter [49] also identified that the network influenced the space plan by altering the purpose of rooms. They also noted that aesthetics play a role in how space is used. Where items are placed will be influenced by a desire to hide clutter. If Ethernet is used, the cables will be hidden in walls so that they can not be seen.

6. Stuff

   Stuff encompasses everything in the home, including networking devices. Stuff is changed or moved around by householders to meet changing requirements. Once again, aesthetics play a key role in determining where and how items are placed [84].

The layers of composition demonstrate how central the infrastructure of a home is to how it can be networked, which in turn influences what can be achieved with the network.

Infrastructure also has the potential to affect the usability of systems [66]. When infrastructure is designed without regard to usage, it is inevitable that technical details will be exposed to users, which is especially problematic if users are required to interact with infrastructure directly, as is the case with networks (to connect devices or troubleshoot problems).

Edwards, Newman and Poole [66] define four possible solutions to network usability problems caused by network infrastructure:

- Surface — add layers to software to shield users from infrastructure
- Interface — improve the interface between the infrastructure and the applications that use it

- Intermediate — new layer of infrastructure with better user experience, but still limited by underlying infrastructure
- Deep — make changes to the infrastructure to support an improved experience throughout

Deep modifications to network infrastructure would allow for a better User Experience (UX), but to do so requires UX specialists to be involved in the design. Large-scale changes to Wi-Fi devices to improve the UX would require changes to the 802.11 specifications [88].

Using SDN allows management of networks to be centralised, which presents the possibility of refactoring the network infrastructure [46]. In some situations, network infrastructure provides too much visibility (such as provisioning devices), and in others, not enough visibility (such as maintenance). In the former situation the infrastructure should be modified to reduce visibility, and in the latter, more visibility is required. One example of this type of modification is the Kermit study [47], in which users could understand the nature of their networks because a known image represented each device, and the image was only visible if the device was connected.

SDN can alter network infrastructure by allowing service providers to configure networks from anywhere, in a manner which meets their needs, and the needs of users [82].

Blythe and Monk [24] suggest that alternative designs for everyday devices may help to challenge the notion of who does what, which would suggest that simply changing the design of a piece of technology could affect the overall routine of the home. For example, activities such as Do It Yourself (DIY) have the potential for instant gratification. They suggest alternative designs for cleaning products to add an element of gamification to cleaning tasks, in an attempt to address the gender assignments for tasks.

The unique characteristics of homes are evident when considering the meaning of performance and optimisation. In business networks, performance and optimisation are concerned with ensuring that the network is as fast as possible and that there is no unnecessary delay. Both of these can be measured and analysed to identify issues and enable improvements to be made if necessary.

In homes, these concepts are still important, but for different reasons, and the techniques used to identify issues are different. When considering performance, reliability and trust are more important [53]. For example, can the network be relied upon to stream a film from Netflix[19]? Optimisation problems are likely to be identified by videos buffering more than usual, or a reduced speed than would be considered 'normal for the time of day'.

In both cases, issues are often worked around or ignored unless the network is considered unusable for the task required [158].

The different definitions for both optimisation and performance are prime examples of the fact that, in homes, the network is a tool to allow the routine to continue, and the network needs to function within the demands of the household [158]. It is expected that the routine should not be disturbed by the network, and households will do as little as possible to address issues [53] so that order can be restored and the household routine can continue.

The need to make technology fit into a home, and within an established routine also affects how and where items are installed [157, 158]. For example, the safety of children would be considered before optimal signal strength when deciding where to place a router. If the optimal location for a router is the middle of the room, cables will be needed to connect the device to a power socket and data connection (ADSL or Fibre); having trailing cables might be considered a safety concern. Devices may be hidden (to avoid safety issues), at the expense of performance and coverage area. The orientation of a networked device has a significant impact on its connection [125], even in an optimum location. Because aesthetics are a factor in positioning equipment [84], network performance and coverage are likely to be suboptimal, especially compared to a location identified to give optimum performance or coverage using a site [125]. Site surveys are unlikely to be carried out due to the costs involved [84]. If a site survey were to be carried out, it is likely to suggest that the optimal position of an access point is in a location which is at odds with other considerations (safety, the location of utilities, available storage locations),

Making devices or applications interoperate correctly is a challenge because no single application or device can know about all possible devices or programs with which it may need to be compatible. It might be necessary to install software or

[19]http://netflix.com/gb

device drivers, which may or may not work with the hardware and software already in use. The tools required to install new technology are many and varied, and there is still potential that items will be missed, or that they would be incompatible [157]. Universal Plug and Play can help with device discovery and installation, but there may still be a need for software installation.

Technology in homes is being adapted to the environment and devices which were once considered novel are now an unremarkable part of the routine of everyday life [124].

Homes are constantly changing, and the technology used in homes must adapt to changes as they occur [141]. The kinds of changes that occur could be household members, new requirements for facilities, spatial or structural. In addition to adapting to changes in the home, technology may be responsible for some changes. Technology can either be a direct cause of change, or it can bring about the desire for change (such as a desire for a technological upgrade).

The addition of new technology can result in the original devices being re-purposed, such as older computers being used by other householders, or old A/V equipment being moved to a different room [79].

As long as the technology is functional, there is no need to change it, and the perceived benefits would need to be significant if an upgrade were to occur at another point [84]. Reluctance to change technology is partly due to the inconvenience of changing something which has become a normal part of home life.

When installing new devices, a primary concern is ensuring that neither the presence of the new device or the tasks necessary for installation (referred to as digital plumbing by Tolmie et al [157]) have an impact on household routines. The new devices must be made 'at home' immediately [157]. Being at home immediately means that the space plan, services and household stuff layers [26] of a home all influence where new items can be placed and how they can be connected.

The act of installing any new device in a home will be considered another household chore that must be completed [158].

Any devices in the home are accountable to the broader issues in the household, not the other way round [158]. When installing a wireless network, the access point will be located according to these concerns, even if its location means a reduction in range. The needs and functions of a home should not have to be modified to fit within the constraints of the network [78, 79, 158]. One participant in the study outlined in [158] demonstrates this. They always sat in the same chair, and the position of the router meant that they had to sit in a way which meant they could get a signal. Moving the router or rearranging furniture was not considered an option.

Because installing new devices can be disruptive, advanced planning is required. The installation must be planned so that it can be completed within a set timescale, to minimise the impact of the installation on the routines of others. Completing the work in stages may be necessary.

2.4.1 Routine

When examining home life and how any technology fits into such context, the existence of routine and the importance of this is something identified time after time in research on a range of topics linked to the use of technology or when analysing domestic routines. In the case of [159], the impact of having a routine was not originally considered but was soon identified as significant enough to be studied in some detail. Of particular note was the observation that if people were observed, they would not comment on events or actions that were part of the routine, not because they were oblivious to them, but because the activities were considered mundane enough not to draw attention to them.

2.4.1.1 The effect of technology on routines

Research into the use of the Roomba Vacuum in homes [71] suggests that technology has the potential to modify routines by simplifying tasks or changing the perceptions of who is responsible for any given task required for the running of the house. In some homes in this particular study, the person tasked with vacuuming changed when the Roomba was introduced into the home. Also, the presence of the Roomba led to more men undertaking the vacuuming, which is a change from historical events [52]. This study into the use of the Roomba also notes that how technology is introduced into a home affects how householders perceive it.

Use of technology is shaped by the normal, everyday activities which are part of life. Technology can also influence routine. Television use demonstrates this. The available programmes and the timings of those to be watched can influence what happens when, and where.

2.4.1.2 Activities

Routines can also affect how activities fit within the home. The handling of mail is a prime example of this [54]. When mail is collected, by whom, how it is sorted and where it is placed, are all aspects of the routine of mail handling. They are designed to be recognised by other members of the household. Where items are placed can serve as a signal to the identity of the intended recipient, or serve as a reminder for action.

Domestic technology should be designed with the same goals in mind. How and where technology is used or located should provide an implicit message regarding its intended usage[54]. A digital calendar should be viewable and editable by anyone, in the same way as a paper calendar.

System design should consider the same action points or messages. For example, how could a digital bill be handled such that the bill payer would be reminded of the need to pay the bill when they leave. In the case of paper bills, this situation would be handled by locating bills in a specific location. Just placing items in a location is enough to remind the bill payer to pay the bill [54].

2.4.1.3 Room Usage

Routines will also change the use of individual rooms, or their perceived ownership, depending on the task at hand [84]. The use of technology is central to the routines of individual householders, to the extent that there will often be multiples of the same devices, to accommodate differences in routine [84]. TVs are one example of this phenomenon. Homes will often have multiple TVs to accommodate clashes in the scheduling of programmes that different household members may wish to watch. Having multiple TVs caters for the scenario that there may be content that is unsuitable for children [49].

2.4.1.4 Changes in routines

Changes in routine, such as the presence of guests, can result in behavioural changes. Even though watching television can be considered a shared activity, the presence of guests would often mean that the TV is not used because watching TV is considered antisocial [84].

Technology usage is central to household routines to the extent that routines can often be described by referring to the technology used. TV usage often shapes routines, with activities scheduled around TV schedules (for example, watching the same programme while eating lunch [84]).

Some actions are defined by the space available, or its character [54]. Use of available rooms may need to be flexible to account for changes in routine, or where someone else wishes to use a particular room for a different task [84].

2.4.1.5 Workplace

The importance of routine can be seen clearly in homes which are also used for work. The household routine takes priority over working [159]. Work can happen at any time, but other members of the household must leave at set times. For example, getting children to school on time takes priority over work tasks.

2.4.1.6 Maintenance

Routines affect the installation and use of technology. New devices must be installed in a way which does not interfere with any established routines. It is not enough to make new items fit into an established routine but to "make usage of a new device routine" [158].

Keeping the devices on the network operational is a requirement if the network is to blend into the routine of the home. Certain technical tasks are themselves routine (such as backups), but the fact that they are routine is not enough to mean that they are part of the routine [158]. Problem-solving tasks are not predictable and cannot be made part of the routine. The best that can be done is to schedule fixes to best fit within established routines. Maintenance is easier if the network is stable because there is no need for large-scale changes to the network, which might require considerable disruption to address.

Any activities in the home are accountable to the routine of the home, including network maintenance [53]. Any network management tools must account for this constraint. In a technical sense, maintenance tasks are considered an end goal. In homes, however, a network maintenance task is just one of the many tasks required to ensure that the household routine is not disrupted.

2.4.2 Social Context

When designing devices that are intended for use in the home, there is another area where the home presents a different set of challenges. Namely, how any product would affect the social aspects of home life, regarding who is responsible for what, how it is ensured that tasks are carried out, and the implications that the items in the home have on how social occupants of a house are with each other. Use of technology may change from the norm depending on who is present. For example, the presence of guests might mean that the television is switched off [84], even though in other similar circumstances, the television would be central to the household routine.

2.4.2.1 Additional Technology

New technology in the home can change what is considered acceptable. For example, consider the concept of good parenting [62]. New technologies mean that the decisions parents must make about what is acceptable are changing. The television is one example. Parents must determine what is appropriate for their children to watch, and for how long.

2.4.2.2 Access Control

Access control is considered a social concern, in the sense that providing Internet access to guests is considered part of being a good host [53]. Also, determining who or what can access the network is not restricted to devices. Those devices which are allowed to connect to the network may be subject to restrictions in *when* access is allowed, and *what* is allowed [53]. Parents may not want their children to access Social Media (such as Facebook) at certain times because they should be focusing on completing homework.

Although attempts have been made to provide homeowners with the facility to police network usage as part of network configuration, (such as the comic strip policy interface in the Homework Router [116], or Ponder2 [161]), homeowners often just identify other devices which use high bandwidth and ask the person using the device to wait. Indeed, high bandwidth tasks such as uploading video to YouTube might be delayed until the network is not being used for other purposes [53].

Any rules that do exist to define acceptable use apply to people, and not their devices. It is not sufficient to police network usage per device because devices are often shared [143]. Brown, Mortier and Rodden investigated whether it is possible to identify a person based on their usage, without regard to the devices used [30]. If so, it would be possible to define rules entirely linked to people, and allow the network to identify who was responsible for what data and enforce rules, without needing to define different rules for each device. Users were asked to tag network usage with the likely reason for the usage, or the person who would cause it. The tagged dataset was used to train the analyser, which was used to test another set of usage data. The accuracy of the analyser was between 33% and 88%. Errors occur when multiple users have the same interests. Automated systems (such as backup) were included in the sample, and where multiple users accessed the same sites, the same traffic was logged each time. Using multiple devices also added complexity. Websites sometimes serve a different site to users based on which device they used for access. The differences between data for the same site caused by changing the device, suggests that using separate classifiers for each device might improve accuracy. In general, while the approach has potential, accuracy would need considerable improvement if it were to be applied in all situations.

2.4.2.3 Bandwidth Usage

Bandwidth usage is another area shaped by social concerns. What usage is deemed acceptable will depend on usage, what is required, who requires access and why (using the network for work vs children watching videos for entertainment, might require different bandwidth, but have different priorities). Attempts to inform users of bandwidth usage will typically focus on device usage. If it were possible to reliably match bandwidth usage to users, any tools which present bandwidth usage could consider both device or user usage.

One reason knowledge of bandwidth usage is important is that ISPs are increasingly imposing data caps on customers. Without an understanding of how much data is being used, when, or by who, the likelihood of exceeding a usage cap is increased. Exceeding caps results in decreased performance increased cost or loss of service. Tools such as uCap [48] or Home Watcher [45] are designed to show people how much data devices or websites are using. Showing people their bandwidth usage might enable them to alter behaviours to reduce bandwidth usage. In the case of Home Watcher, exposing bandwidth allowed householders to reason about bandwidth use in the same way that they might reason about hot water, and how much is available.

A problem with the use of data caps is that ISPs define what types of usage are appropriate [182]. ISPs may limit the bandwidth for certain applications with the aim of fairness to all users of their service. Allowing ISPs to decide usage might cause problems. Yiakoumis et al [182] argue that users should be involved in the decisions on what services they use. If one home never streams video but uses Skype frequently, they will face bandwidth shortages if their ISP has used different Quality of Service (QoS) or Quality of Experience (QoE) rules, which favour other usage patterns.

One consequence of showing people how much bandwidth a device is using in real-time and allowing bandwidth to be prioritised is that it may not be clear how activities should be prioritised [45]. It is not sufficient to state that one device always gets priority over all others at certain times (for telecommuting for example), or that all use of certain traffic types is bad. For example, use of YouTube by children could be considered low priority most of the time. Stating that YouTube usage for children should always be a low priority when allocating available bandwidth might cause problems because there may be a genuine need to use YouTube for a homework assignment.

During a trial of Home Watcher [45], it was shown that knowledge of bandwidth usage could confirm assumptions about bandwidth, as well as highlight other usage patterns which were previously unknown. Being aware of bandwidth usage also had implications for household routines. It was possible to know if children were awake after they should have gone to bed because their device was still using bandwidth.

2.4.2.4 Policing Network Usage

Network usage will be subject to certain household rules. The rules that apply to the use of a network are often informal, and not written down. They are also defined in the context of the social order of the home [55]. Rules exist to enforce social rules, such as good parenting, or being a good host. Rules are about people, and what they can do and when.

The presence of a rule does not guarantee compliance but does force people to account for their actions. Rules are often subject to negotiation and are often enforced by having conversations. For example, although a rule of 'no network access after 9 PM' could be enforced automatically, it is often considered better to have a conversation [55], to hold people to account for their usage.

While specific rules may exist, there is often a reason for them. It may be that children are not permitted to access certain sites between set times. The enforcement of the rule will be defined based on the intention behind it. If the reasoning for the rule is that homework should be completed, getting homework completed is more important than whether or not a rule is followed. The policy system in the Homework Router [55] supports notifications to enable users to make others aware when actions occur.

When policies are defined formally, such as with the Homework Router policy system, they will be defined by who can do what, when, and what the consequences of an action would be.

In the case of the Homework Router, enforcement of policies was considered a parenting tool. In one specific case, a group of students did not consider it acceptable for any one person to decide who could do what. They had a rule that no one could download anything during periods of gaming, but the rule was never enforced. The router's device control panel was used to identify that a device was downloading, which signalled the need for a conversation. It was never considered acceptable to define a formal policy or to use the control panel to prevent people from accessing the network.

Any network which allows for the provision of QoS must present the facility to users in a manner which facilitates social interaction. It must be possible to express rules that apply to people, and to consider what actions are considered acceptable, and should focus on providing information to users, to enable social enforcement.

Although bandwidth usage and usage rules are not directly linked to network configuration (at least, not in the social sense in which they are applied), having an understanding of how these aspects of network usage are viewed and understood, is useful to understand how central social context is to technology usage.

When devices or media are shared, collaboration is a factor [79]. Audio/Visual equipment is one type of network which is shared. Computers are often assigned ownership.

A/V equipment is often used in a central location so that its use can be a group activity.

Providing network access is considered necessary to be a good host [53], but this concept is sometimes taken further, and networks are deliberately insecure so that neighbours can access them. Providing network access is sometimes considered part of being a good neighbour [79].

Organising photos demonstrates the clash between individual devices and shared media. Photos will often be taken—and subsequently saved—on personal devices. The use of personal devices means that photos may be organised differently on each device, making it difficult to collaborate on the task of photo organisation [79]. Views of the network which provides visibility to its structure are not necessarily enough to solve problems such as those which occur with photo management, because it may be possible to see photos on other devices, but it is not necessarily clear what to do. Grinter et al [79] suggest that if tools focused on tasks rather than devices, these problems might be resolved.

Integration of devices and services is also a challenge. People like to interconnect devices but sometimes prefer separate systems [79]. Remote controls are an example of this. Using a universal remote control reduces the number of remote controls required, but there are sometimes tasks which are easier on separate remotes, because of the range of buttons available [79].

2.4.2.5 Maintenance

Any task which must occur to ensure the smooth running of the household may be the responsibility of a single occupant [53]. In the case of networks, the person responsible for the network will often have no concerns fixing it if they identified the problem, without requiring consultation with others. Other people are often reluctant to do any task for which they do not have the responsibility, so will always ask the appropriate person, even if the fix is as simple as rebooting the router.

When issues do occur fixing them is a matter of being able to complete the current task, such as sending an email or making a Skype call, and not concerned with permanent resolution [53].

In addition to defining socially acceptable practices, policies can be used to manage some aspects of network management, such as QoS [137] but more tools exist for enforcing policies in business networks [25, 96, 137].

2.5 User Support

User support is another area affected by the increase in smart technologies and the methods of configuring everything to make different technologies work together.

It is not realistic to expect that an application or device can be designed to operate with any combination of software versions or with different hardware combinations. Making new software or hardware work correctly presents a challenge if a user presents a use-case that does not fit the use-case of the original application or device designers. There is a possibility that user expectations of a product's potential use would break in that case [23]. These broken expectations could then extend to a request for technical support.

Bly et al. [23] provide examples from a study that demonstrate how expectations could be broken such as having a wireless network which uses Wi-Fi Protected Access (WPA) when a new product only supports Wired Equivalent Privacy (WEP) or incorrect operating system versions installed on computers. Another example is that of a computer with a graphics card and Wi-Fi card that were incompatible, meaning that the computer must use Ethernet to remove the incompatibility. This example is particularly significant because it serves to illustrate that the combination of devices or software in use in that situation is not one tested for by the manufacturers.

In the situations described above, the problem is not that the products are broken or that they have not been installed correctly, but that users sometimes have unrealistic expectations about the potential uses for the devices or software they buy. Complex or time-consuming steps may be required to resolve the problem (such as upgrading the software on a computer or buying a new computer), or where there is a need to identify a suitable solution if one is available.

2.5.1 Formal Support [132]

The first type of support which users can request is a formal support request to an expert, either to request that an expert visits them to troubleshoot the problem or remotely.

There are challenges when troubleshooting problems from technical support organisations, for both the support person and the person requesting help. Support can be somewhat simpler for in-person support because support staff can see the devices in question and how they are connected or configured.

Remote assistance presents a bigger challenge because support staff cannot always know the exact configuration of every device (both hardware and software), or how they are connected. Users often have difficulty explaining the problems, or the details regarding their setup to support staff [133]. In this paper, Poole, Edwards and Lawrence analyse twenty-one calls to technical support for a manufacturer of network equipment, specifically for problems related to connecting devices. Users frequently don't use correct terms for items, and they have their way of describing components. This lack of understanding of terms is problematic for both sides because support staff need to understand the problem and the setup of the user, but the user cannot provide the details necessary.

2.5.2 Informal Support

Receiving support from known sources such as friends or family members can sometimes address some issues with official support [132].

There are potential benefits to this for both the requester and the helper. If the provider of support is young, the provision of support can lead to a source of income and useful experience [132].

For those requesting support, approaching a friend or family member can be less intimidating than requesting official support. Finding contact details for relevant support departments can be difficult. Requesting support requires setting aside a considerable amount of time. Requesting support from friends and family can remove or reduce the problems sometimes faced with remote support [132].

There are advantages and disadvantages to informal support for those providing the support. For young people, providing support can be used to gain experience with problem-solving, which can be helpful for future careers. It is also a source of income.

Those providing support seldom need to advertise, because awareness of their computer knowledge spreads amongst friends and family.

Time is a factor which, unlike recipients of help, can be a disadvantage. If the request covers hardware or software which is unfamiliar to the supporter, time is required to investigate the problem. Regardless of knowledge, time must be set aside to provide support, which can be problematic if requested by a significant number of people.

Various techniques were used to lessen the impact of providing support. It is necessary to decide *if* to help, and *how* to provide the help (in person, phone, email, or referral to a third party) and how much to charge [132]. In some cases, charging more was intended as a deterrent, but instead was considered an indicator of knowledge.

2.6 Automatic Configuration Management

There is a possibility that configurable systems could be configured incorrectly [35]. There have been some attempts to automate configuration management for PCs such as Strider [171], PeerPressure [168].

PeerPressure stores registry keys for several Windows PCs in a central database, and can identify potential configuration problems from differences in registry key values. If one machine was misconfigured, and a group of devices all had the same registry values, except for one machine and one value, PeerPressure could identify the mismatch as a likely cause for problems.

Strider examines the registry of a PC and identifies a set of registry keys which changed, and where the result of the change could have resulted in the observed error.

Strider and PeerPressure are designed to detect invalid configuration in PCs, so are not directly applicable to Wi-Fi configuration. However, they do demonstrate what is possible with knowledge of the configuration. PeerPressure is a demonstration of how the configuration of multiple machines can be combined to diagnose configuration problems, by determining what is different between devices. The concept of PeerPressure is generic enough to be applied to other domains, such as troubleshooting network misconfiguration.

ResFi [185] is a configuration management system for Wi-Fi networks. It is designed to address problems caused by large numbers of Wi-Fi networks in a small area, particularly in neighbourhoods. Many networks in a small area can cause QoS problems, because of interference between networks [9]. Organisations can resolve this problem by using a central management system to configure access points such that interference is not an issue (BIGAP [189] is an example). Home networks are managed by the relevant householder, which means that there is potential for interference. ResFi is an attempt to connect networks using the Internet, to share knowledge of the networks that are within broadcast range of each other. ResFi can use this to coordinate channel selection between networks so that each network is on a different channel.

Another feature of ResFi is the ability to create virtual networks. In some areas with large numbers of networks, it is possible that devices that should connect to one network would get a better connection if connected to another network. By creating virtual networks on routers, devices can connect to the virtual network if it has a stronger signal, and have traffic routed via the network to which it belongs.

WiFiProfiler [42] is another network configuration tool. It can diagnose configuration problems by comparing the configuration on a non-functional device with the configuration of a correctly functioning device.

Devices can communicate with other devices on the network to request details of their network configuration. The peer-to-peer connection requires devices to have two Wi-Fi adapters or to create a virtual Wi-Fi adaptor [41]. Using either method would allow hosts to connect to each other for fault detection while maintaining Internet access.

By collecting data from multiple hosts, it is possible to narrow down the possible causes of errors. Even if it is not possible to directly identify the cause of a problem, the collected data should be enough to point users in the right direction.

Devices can infer some information about the network status themselves. A device will know what networks it can see, whether it was able to associate with a network, and whether it obtained an IP address. This information can help to identify possible causes, but more information is often needed. For example, a client would not be able to identify that its WEP key was incorrect because the use of invalid WEP keys does not cause an error.

WiFiProfiler can use a range of configuration data to identify possible causes of misconfiguration:

- Wireless Layer
  - NIC model, name and driver version
  - BSSID list
  - SSID list
  - RSSI list
  - Security Protocol
  - Passphrase
  - Beacon loss rate
  - Interface queue length
- Network Layer
  - IP address/subnet/mask
  - IP mode (dynamic or static)
  - DHCP info (server address, lease start and end)
  - LDNS information (IP address of the local DNS server).

- Transport Layer
  - failed connection attempts
  - packet retransmissions
  - server port numbers with successful TCP connections
- Application layer
  - web proxy setting

By testing items at each layer, it is possible to identify areas that may be incorrect based on values seen on other hosts. If one device was functional and another was not, and they both used the same security protocol but had different passphrases, an incorrect passphrase is likely to be the cause of the problem.

Performance is another area where problems can occur. HomeMaestro [93] is a per-device performance monitor. Performance data is gathered from devices because not all traffic will be visible to the whole network. HomeMaestro monitors the network performance of applications on participating devices and can identify performance problems by monitoring network flows and using time-series analysis to infer performance problems caused by multiple flows or applications competing for bandwidth. QoS rules can be applied to address the competition between flows or applications.

Monitoring performance from the router must not affect the performance of the network [187].

Where performance is poor, there are many possible causes, such as low signal strength or interference (between networks, devices on the network or other radio frequency devices, such as cordless phones). However, interference is more likely to cause performance problems than signal strength[128]. The 2.4GHz spectrum must support a larger number of devices than 5GHz [80]. The 2.4GHz spectrum has more frequency bands than 5GHz, and only three frequency bands have no overlap.

People often use online resources—such as forums—to aid with troubleshooting problems [8], in the hope of finding others who have had the same problem, and might point them towards a solution. Netprints [8] is a tool which works on the same premise, but which is automated. Netprints uploads configuration to a central server. When problems occur, the current configuration can be compared with the configurations of other networks, to find potential causes of the error.

Another method of using automation to aid in troubleshooting is to log changes that occur. An aggregated log of changes can be used to help identify potential problems [34].

In addition to QoS, policies can be used to define rules which will optimise network configuration as required [127]. For example, policies can be defined to alter the network's channel if performance is reduced.

2.7 Summary

This chapter explored research areas that are most relevant to the aims of this thesis and is used as the basis for the features of the router, as described in Chapter 4.

Home networks are built using the same technologies as large commercial networks. Commercial networks are maintained by specially trained staff, which is often not the case in homes, resulting in a need for technical support. The disparity between technological design and technical ability of users presents challenges to researchers to identify methods of simplifying network management. Household routines affect which technologies are used, when, how, and who by. Any technologies must fit into the routine. Network maintenance and troubleshooting are considered necessary chores to ensure that disruptions to routines are minimised.

As a result, two significant focuses of home networking researchers are device connection management and revealing bandwidth usage. However, although networks are often considered a challenging technology to integrate into the home, there is limited attention to considering how to assist with troubleshooting. The difficulty with troubleshooting and user understanding of networking was first identified by Grinter et al. [79].

The next chapter is a review of history storage and undoing techniques. Both Undo and history are considered in a separate chapter because history storage and undoing are fundamental to the router, and all other features of the router should be designed to be compatible with the storage of history. There are many potential solutions, and it is important to evaluate each of them to determine the suitability for storing the history of configuration changes. Undo and redo methodologies require similar treatment.

Chapter 3

Undo

3.1 Introduction

This chapter outlines considerations that must be made when designing a recovery, or user-centred command undo facility for an application or system.

To design an undo function for any system it is useful to consider what it means to Undo, as well as the various examples of specific implementations of undo. It is possible to learn from existing undo systems and ensure that the resulting system is fit for purpose.

The chapter first outlines what it means to 'undo', first generally, and later with formal definitions. Specific implementations of 'undo' follow the definitions.

A recurring theme in Chapter 2 is that non-technical members of households have difficulty in developing a sufficient understanding of networking technologies to allow their network to blend-in to the home. When the network behaves differently than expected, it is not always clear why, or how to address the problem. One possible solution is to provide users with lists of changes, which would allow users to see potential problems.

Having a list of changes is of limited use without a mechanism to reverse undesirable changes.

3.2 Definition of Undo

A simple definition of undo when applied to computer systems is 'to restore a previous state in a user's program or application' [104]. Another definition, (which takes into account that systems view Undo as a function, but for users, it is an expression of intent [6]), is that *undo is a system function that allows the user to reach some prior state in the history* [108].

Reachability means that from a given system state it is possible to reach *any* state by executing commands, both previous states, and future states (that are not currently defined). Undo could be considered as a special case of reachability [61] because it is only possible to transition to pre-existing states.

Each potential application of an undo function has a unique combination of requirements, meaning that a significant portion of undo and history literature is focused on a particular application or problem domain. However, it is possible to compare Undo implementations by how the Undo method meets the criteria defined by the particular problem domain.

Cole, Lansdale and Christie [51] suggest that any user should be able to answer the following questions: *Where am I? How did I get here? What can I do? Where can I go next?* The ability to undo can assist with answering these questions by lowering the risk associated with experimentation. However, undo can also make application behaviour nondeterministic [109].

The ability to undo commands is a very focused method of reverting state to an earlier time. An undo function provides an application with the ability to restore a previous version of some portion of its state.

The notion of Undo is complex[1] because there are several decisions to make when designing an Undo function. Undo may initially appear simple: revert to a previously seen state. This is a reasonable—but simplistic—definition. There are some edge cases to consider, such as the command to undo is itself an Undo [108]. Can Undo commands be undone, or is a Redo command required? Will subsequent Undo commands move further back through history, or oscillate between two adjacent states [108]? Another complexity is that, depending on the context, the state after an undo may not be what the user was expecting, which might be caused by

[1]Yang [181] expanded on the definitions by Gordon et al. [76] to provide clarification of the original definitions.

the presence of many interpretations of undo. Confusion should not be a surprise if a user uses many applications, each of which behaves differently [108].

3.3 Definition of Terms

There are some often repeated terms used when describing command histories and undoing.

Primitive Command An individual user-issuable command.

Composite Command A series of primitive commands that are issued together.

Macro Command A named command which is composed of a combination of primitive and composite commands.

Meta Command A command which operates on other commands for recovery and command re-use. The command history will not normally include Meta Commands.

Action Any user-initiated procedure.

Event A change of state of any object, at the time it is indicated to the user.

Object-oriented applications perform operations on *objects*, which are containers for data. User actions in a system occur in an *interactive dialogue*, which consists of a sequence of *interactive cycles* [181].

In the context of user support, there are three sets of properties that *action*s can possess (for object-oriented systems) [181]:

1. Effectual or Ineffectual

   Effectual commands modify the data structure of an object. An ineffectual command does not modify an object.

2. Reversible or Irreversible

   A reversible command can have its effect on an object reversed.

3. Have operational effects/non-operational side-effects

   Only operational effects should be undoable.

Based on these properties, the domain of undo in the context of object-oriented systems is the set of all commands which are reversible in respect of their operational effects.

Implementations of undo have four components [181]:

1. Presentation

   How a recovery command is unique to the user, and how the result of the commands is conveyed.

2. Internal Interface

   How other components can support undoing.

3. Data Structure and Algorithm

   How to design the data structure so that it can support relevant (possibly multiple) undo situations.

4. Recovery Techniques

   Different commands need different techniques to reverse their effects.

3.4 Notation

Each implementation of an undo function is described using a different notation. Comparision could be simplified if all implementations used the same notation. However, each implementation of Undo uses different concepts, so will require new notation.

The notation described in this section describes notation concepts which can apply more broadly, such as notation for presenting initial states of history. The notation used is based on notation defined by Leeman Jr [104].

Sequences of actions are represented by $\langle x_1, x_2, \ldots, x_n \rangle$. Action sequences can also be defined as $\sum_{i=1}^{n} s_i$. Ω denotes the null sequence.

The example shown in [104] and shown here in (3.1) illustrates the use of this definition.

$$\langle \Omega, w, wx, wxy, wxyz \rangle \tag{3.1}$$

(3.1) represents the commands used to edit lines in a file. Each command inserts one letter, and Ω represents an empty file.

3.5 Formal Definitions of Undo

The concept of undoing can mean different things depending on the context. The environment in which an undo command is requested, as well as the potential side-effects of the command, can result in differences in whether a command is considered undoable or not. How history is stored, and the mechanism for undoing can also influence what can be undone.

Smith and Mosier [149] outline three design guidelines for presenting user recovery in UIs: *any user action should be reversible*, *an undo action should be able to reverse more than the most recent command*, and *an undo action itself should be reversible*.

Another factor which can distinguish between implementations of Undo facilities is what happens when executing successive Undo commands.

Formally defining what 'undo' means has three benefits:

1. There is a definitive definition
2. It is possible to establish properties that recovery systems and undo commands must have.
3. There is a single point of reference to be used when comparing recovery techniques and implementations.

Gordon, Leeman, Jr and Lewis present a formal definition of undo and suitable properties [76], which can be used to compare other actual undo implementations.

3.5.1 Basic Requirements for Undo

Any undo function must fulfil certain basic requirements, regardless of how it operates.

User-issued operators o, belong to a set O. Let K be the subset of O consisting of editing commands k which operate on item states. In addition to k, O will include other operations such as undo. The letters f and g are used to represent individual commands in K. u is an undo command.

3.5.2 Basic undo command

This section is a reproduction of [76]. The definitions are reproduced here for clarity because the undo definitions and properties will be referred to in the context of concrete undo techniques.

Given the set S of states, and the set K of editing commands with domain and range S, a simple definition of an undo command is:

$$uk(s) = s \qquad \text{for any} s \in S \quad \text{and} \quad k \in K \tag{3.2}$$

Commands are applied from right to left. This undo definition assumes that it is possible to identify the previous state to return to uniquely. However, commands do not have unique inverses [76]. It is, therefore, necessary to refer to an expanded state of the system. Gordon et al. [75] use the term 'history' to define one possible solution.

Regardless of how it is accomplished, editing commands have a one-to-one mapping to a state if the state is extended to a set E with a mapping $c : E \to S$, which gives the associated state of the item, then applied to any extended state [76].

How E is constructed does not matter, as long as each editing command k can be extended to E in such a way that if $\mathbf{k} : E \to E$ is the extension of k, then

$$c(\mathbf{k}e) = kc(e) \tag{3.3}$$

and $\mathbf{k}$ is one-to-one. $\mathbf{K}$ is redefined to denote the set of extended editing commands. The use of bold letters highlights user commands in the extended space.

The definition of a new undo command $\mathbf{u} : E \to E$ is:

$$\mathbf{uk}(e) = e, \forall \mathbf{k} \in \mathbf{K} \tag{3.4}$$

This definition of undoing will permit returning to the state before the last command **k** as if **k** had never been issued.

The definition in (3.4) is restrictive because the entire extended state of every item is not important, just the state of the item in question. A less restricted definition is

$$c(\mathbf{uk}e) = c(e) \forall \mathbf{k} \in \mathbf{K} \tag{3.5}$$

Any definition of undoing which does not satisfy (3.5) should be rejected, because (3.5) states that the minimum requirement for an undo facility is to remove the effects of editing commands on the state of the application.

3.5.3 Properties of Undo

There is a set of expectations usually associated with undoing. Any undo command must have certain properties if it is to meet these expectations.

The most important expectation is that an undo should itself be undoable. The definition of Basic Undo in (3.5) does not specify the effect of applying undo to the restored state $\mathbf{uk}e$. If $\mathbf{uk}e$ is identical to e, for any $n \geq 1$ the property of thoroughness should hold. Thoroughness is defined as

$$c(\mathbf{o}_n \ldots \mathbf{o}_1 \mathbf{uk}e) = c(\mathbf{o}_n \ldots \mathbf{o}_1 e) \tag{3.6}$$

for any $\mathbf{o}_i$ defined on E, including **u**.

u should itself be invertible. For **u** to be invertible there should be a redo command **r** such that

$$c(\mathbf{ru}e) = c(e) \forall e \in \mathbf{E} \tag{3.7}$$

There is a special case of invertibility, where **u** is self-applicable. That is, it can invert itself. Without self-applicability, **u** is restricted by the fact that it can only undo editing commands. Self-applicability is defined as

$$c(\mathbf{uu}e) = c(e) \forall e \in \mathbf{E} \tag{3.8}$$

If an undo command is invertible, it cannot also be thorough [76]. That is, it is not possible to be thorough and recover from a series of editing commands. The property of thoroughness cannot hold for any system which has more than two states [60, 61], because the additional state required to store history makes it impossible to return to the exact state the system was in before, without losing history. Without history, it is impossible to undo any further actions.

The property of unstacking overcomes the limitation of thoroughness.

$$c(\mathbf{u}^n \mathbf{k}_n \dots \mathbf{k}_1 e) = c(e) \forall n \quad \text{and} \quad \mathbf{k}_i in \mathbf{K} \tag{3.9}$$

Self-applicability is incompatible with unstacking, just as invertibility is incompatible with thoroughness. An undo operation can be invertible and unstacking [134].

Having incompatible properties is not problematic because there may be situations where one approach is preferred. In some contexts, it would be desirable for multiple undo commands to move further back in the history while, in others, it would be expected that subsequent undo commands undo the effect of the previous undo command. In other words, some situations are better suited to unstacking, and others to self-applicability.

Both COPE [12] and POLITE [134] use an undo command with the property of unstacking. P-EDIT [95, 97] and Bravo [100] both have self-applicable undo.

3.5.4 Higher-level definitions of undo

Expressing undo as a general concept is useful, in part because it allows for a definition which does not influence the implementation details.

However, the difference in abstraction level between the Undo definitions shown here vs actual implementations of Undo (which may or may not satisfy the requirements and properties of undoing described above) is quite significant.

Leeman Jr. [104] has also formally defined undo in the context of programming languages. The definitions of undoing presented at the level of programming languages are at a higher level of abstraction than the already presented definitions, but they are still sufficiently removed from the implementation details to apply to many different approaches to undoing.

3.5.4.1 $\mathbf{undo}_b(t)$ [104]

Assume the presence of a time parameter where each unit of time corresponds to one command. The operation $\text{undo}_b(t)$ moves back t units of time.

If $\text{undo}_b(1)$ were applied to (3.1), the result would be:

$$\langle \Omega, w, wx, wxy \rangle \tag{3.10}$$

Both the file and the command history have the last line in the file removed.

Applying $\text{undo}_b(3)$ to (3.10) would result in $\langle \Omega \rangle$ and the file would be empty. The same result can be achieved by applying $\text{undo}_b(4)$ to (3.1).

In general,

$$undo_b(t_2)undo_b(t_1) = undo_b(t_1 + t_2) \tag{3.11}$$

always holds for non-negative t_1 and t_2.

The command $\text{undo}_b(t)$ is destructive because it removes items from the history. It is a meta command. It has the property of unstacking. Multiple invocations move further back through history. It is not possible to undo the effects of the undo because the history has been truncated.

3.5.4.2 $\mathbf{undo}_f(t)$ [104]

The function $\text{undo}_f(t)$ also undoes t commands. Unlike $\text{undo}_b(t)$, the result of the t commands are undone, but the history is appended to, not truncated. Applying $\text{undo}_f(1)$ to (3.1) gives:

$$\langle \Omega, w, wx, wxy, wxyz, wxy \rangle \tag{3.12}$$

The file has three lines. The history shows that the file once had four lines. If $\text{undo}_f(3)$ is applied to (3.12) the result looks the same as applying $\text{undo}_f(2)$ to (3.1). Although the file looks the same, the processing required to achieve the result is different. The general case can be defined as

$$undo_f(t_2)undo_f(t_1) \iff undo_f(t_2 - 1) \tag{3.13}$$

where $\iff$ signifies that although the result of the operations is the same, the computation required to achieve the result will be different. The general case applies when t_1 is non-negative, and t_2 is positive. If one t_2 unit of time is expended, the result is undoing the undo. The function $\text{undo}_f(t)$ is therefore self applicable and not unstacking.

3.5.4.3 Limitations

The commands $\text{undo}_b(t)$ and $\text{undo}_f(t)$ represent two different general models for undoing. However, they do not scale to the undoing of either compound or macro commands. Yang [180] defines general undo functions suitable for such cases.

Firstly, sequential undo, which operates on primitive commands.

The sequence $\langle A_1, A_2, \ldots, A_n\rangle$ can be undone sequentially by executing undo commands in the sequence $\langle A_n, A_{n-1}, \ldots, A_1\rangle$.

Partial undo has two forms, one for compound commands and one for macro commands. In both cases, the subset of commands to be undone are undone sequentially.

Finally, pattern undoing will allow undoing of commands which match a pattern.

Redo commands can be categorised in the same manner.

Multi-user systems have other challenges which require different approaches to storing history, selecting commands to undo and how to undo. They also require specialist support if using programming frameworks [58, 126]. The primitives described in [58] are based on the single-user primitives in Suite [57]. Without the facility to build applications which support multiple users, the only way to create a multi-user system is to rely on screen sharing (NLS [69] for example) or window sharing (such as Rapport [70] or VConf [101]). The limitation of both screen and window sharing systems is that applications are collaboration-unaware [102] (the application is unaware that it is being used collaboratively).

3.6 System Recovery

One method for providing history and undo is to perform snapshots and allow users to choose to revert to an earlier one. Routers could use snapshots to create backups of configuration before applying new changes, which would then allow changes to be reverted.

However, routers can store configuration in files or databases, so it is also useful to consider historical approaches to versioning and data backup for both to be aware of considerations which must be made when designing an undo system.

3.6.1 Checkpoints

The ability to recover from errors is important when creating fault-tolerant systems. There are two approaches to recovery. Firstly, continue in an invalid state. Secondly, to revert to a previous valid state. In situations where actions are distinct (such as router configuration), these fault tolerance approaches are largely irrelevant. However, because they demonstrate some of the first techniques for undoing or state recovery, it is important to acknowledge them. Further discussion on the topic can be found in [37].

A more recent example of a historical data storage mechanism is described in [138], which stores historical versions of files, creating the possibility of reverting files to older versions.

Reverting to an earlier state using system snapshots is too broad to be considered a useful recovery technique suitable for small independent changes, such as those in configuration changes.

3.6.2 Configuration Files

One aspect of the UNIX philosophy [72] suggests that applications store any configuration in plain-text configuration files, along with using plain-text to store data.

Applications which are designed to conform to the UNIX philosophy can be provided with a basic history of configuration changes by storing multiple versions of the configuration files and providing a method of restoring a previous version [138]. A method such as this is more suited to backup and recovery of a system or application than to reversing the effects of individual changes because individual actions may appear to be a single atomic action when they are composed of multiple low-level changes. Also, to be able to identify changes easily, data must be in plain-text configuration files. If the application uses a proprietary format, it must be responsible for its history and cannot use third-party tools such as version control systems[2] or `diff`[3] because these only work with plain-text data.

Relying on configuration files to track changes is overly simplistic, for several reasons. First, not all configuration is stored in configuration files, meaning that configuration would either be tracked in different ways or duplicated (to store configuration in a central location). Another alternative solution to this problem is to store all configuration in files and create new applications to update the real configuration.

Second, each application has different requirements for configuration, so each would need to be edited differently. For example, Hostapd has different options to DNSMasq. IPTables can store rules in files, but these files do not contain rules, but the commands to insert the rules.

Third, if the only record of configuration is the configuration items, and the changes are managed with Git, the changes are low-level, and information presented to users could be technical.

[2]Git (https://git-scm.com) or Subversion (https://subversion.apache.org/)
[3]The file comparison tool documented at https://linux.die.net/man/1/diff

3.6.3 Databases

As with files, it is possible to use recovery techniques with databases. Verhofstad analysed database recovery techniques, to identify how recovery techniques can be used to recover from the loss of data [162], as shown in the following list of possible recoveries:

1. Recovery to correct state
2. Recovery to an earlier correct state (a checkpoint)
3. Recovery to another previous state
4. Recovery to a valid state
5. Recovery to a consistent state
6. Crash resistance

In this context, ‘correct’ state means that the database contains all user changes, and does not include any data that has been deleted. ‘Valid’ state means that the data in a database is correct when considered alongside other data. Consistency means that the data is valid, and meets user requirements for consistency, which will be different for databases [77].

Crash resistance is provided when data processing algorithms operate on data in such a way that the system will always be in a correct state, even when errors occur [162].

3.7 Specific Techniques

3.7.1 System-Wide Undoing

For some systems, it may make sense to undo changes at the operating system level, rather than the database or application level. Email is an example of a system that might benefit from a system-level undo. An email store must be able to recover from a configuration problem quickly, but preferably without data loss. The Three Rs (Rewind, Repair and Replay) Recovery technique [31] was developed as a recovery system for email. The rewind step reverts to a previous system snapshot. The repair phase takes steps to prevent the problem from recurring. The replay step performs actions that were undone by the rewind step, but with modifications so that the repair is not undone. Rewind can jump to a previous state in one step, but replay must restore state incrementally; otherwise, the incorrect state would be reapplied.

Joyce [123] is another system-wide undo system, which models events based on the relationships between them, and not purely chronological. It provides a logging system, which applications can use to log changes to their data. It uses checkpoints and rollback on a per-application basis. Because the application defines the history and constraints for rollback, checkpoints can be defined frequently enough to be considered a form of user-undo system, rather than a system recovery system.

Using a system-wide undo process may be applicable to routers as a tool to provide the functionality to restore a snapshot from an earlier point, which would be useful if all a user knew was that the router was working some period previously. It is not useful as a troubleshooting tool, or to reverse the effects of a single change.

3.7.2 Linear Undo

The simplest forms of undoing operate on a history which is linear. Any linear undo method supports navigating through history starting with the most recent item. It is necessary to undo the $n-1$ more recent items to undo the *nth* item. There are two variations of linear undo on which other linear undo implementations are based. They are history undo/ undo and linear undo/ redo.

History undo/undo has a single recovery command: Undo. The Undo command is a self-applicable primitive command.

Interlisp [156] uses history undo/undo. Lack of a redo function means that it is not possible to redo previously undone commands that are not the most recent command. Interlisp can support redoing, but it is necessary to modify the history to place the undo commands corresponding to the commands to be redone at the end of the history.

If a user entered the commands $\sum_{a=1}^{n} s_a$, and subsequently wanted to perform commands b_1 and b_2 before a_2, commands $a_n, \ldots, a_2$ must be undone. If b_1, b_2 were executed, it would not be possible to redo $a_2, \ldots, a_n$.

History Undo can be used to provide Undo support in databases [94] (based on History Undo as defined by Prakash and Knister [136]) so that applications can support user interactive undo simply by using the database. Databases which use transaction logs can support user-level undo by inserting boundary markers around distinct user actions within the transaction log. Without database support, user undo must be an application function. The need for dedicated support is important because reverting to a previous snapshot of the data may undo a change which the user intended to keep.

Linear Undo/ Redo is the type of recovery used in the COPE programming environment [12]. Unlike History undo/ undo, linear undo/ redo only operates on system commands. It does not work on recovery commands. A redo command executes a command that was previously undone by an undo command.

Primitive Undo [181] has two history lists, the first contains all commands, and the second contains those for which the effect is apparent. There is only one recovery command: undo. The primitive undo model is self-applicable, so cannot have the property of unstacking. It can only operate on primitive commands. As self-applicability is a special case of inversibility, primitive undo must also hold the property of inversibility.

Meta undo [181] also has two history lists, a list of effective commands, and a list of undone commands. There are two recovery meta-commands: undo and redo. Undo undoes user-initiated commands, and redo reverses the effects of undo commands. Any recovery system which operates using undo and redo commands has the properties of single and multiple reversibility and inversibility [181].

The Triadic undo model [181] also has history and undone lists. There are three recovery meta commands: Undo, Redo and Rotate. The Rotate command rotates the undone list so that the desired undone command is the one which would be undone. The triadic undo model has the properties of reversibility and inversibility. The undo command has the property of unstacking, so cannot have the property of self-applicability.

In object-oriented applications, there are two ways to make application objects support undoing, namely command-objects [112, 118, 186], or event-objects [167].

Event objects track events triggered by interactions with the system. Because of encapsulation, the system only knows when events occur, and not how they are handled. It is, therefore, the responsibility of each object to define its undo semantics (what changes are undoable and how the Undo should be performed) if the system makes an undo request.

In the event object model, each UI object has a related Recovery Object (RO). Each RO has a set of Recovery Information (RI) objects and Recovery Command (RC) objects. RIs are history lists. Each RO has a pointer to the location where the next RI will be added. The pointer is moved using a recovery command (undo, redo, skipback, skipforward).

Command objects represent changes to object state. In all three cases of command objects, the model has a three-level hierarchy, with each object having an individual history. In [118], commands issued on one object can be reused on other objects, and commands are represented as DO actions tied to UI elements. In [186], references to objects modified in each interactive cycle are stored together, with the changes made to each object stored in an independent history.

The Unidraw framework [165, 166] has support for undoing commands, with an `unexecute` operation to reverse the effect of an `execute` operation. Repeated `unexecute` commands will not be effective, unless commands are stored in an ordered list, in which case, repeated `unexecute` commands will move along the list and reverse all reversible commands.

Because linear undo methods traverse history linearly, they are best suited to undoing the most recent change. Because router configuration changes are not always linked, there may be multiple unrelated changes which occurred after the one to be undone. It would be necessary to undo all interim changes and redo them again, which may have unintended side-effects.

3.7.3 Selective Undo

Selective Undo strategies allow users to undo any command in the history without having to undo subsequent commands. The concept of selective undo was first introduced by Berlage [19] in 1994. The term 'Direct selective undo' is used by Berlage to show that his selective undo does not rely on a script ([6, 11, 135] are script undo implementations). He highlights five benefits that a command object based undo method has over script undo [19]:

- Allows an application to model undo in a manner which is a better match for user intentions. Script undo will not always match user intentions because executing the scripts may result in the execution of unexpected commands that remove the effect of the undo.
- The script model assumes that undo is always possible (more formally: every possible sequence of commands can be interpreted and has a logical opposite, for undoing). Direct selective undo allows applications to signal to users when an undo is not possible because it does not make sense in the current context.
- Direct selective undo is easy to use. Undoing a single action is the only option. Multiple undo commands are required to undo a series of actions.
- To make selective undo worthwhile for users, as an alternative to manually creating the desired state. To facilitate this, direct selective undo allows users to search for the desired command. When possible matches are found, Selective Undo can be attempted until the desired command is found.
- Direct selective undo is simple for application programmers to implement. Selective undo for a single command class can be developed in isolation, without affecting other components, which improves modularity of code.

Direct selective undo stores commands in a linear history and duplicates the command to be undone at the end of the history list before reversing its effect. Selective redo also copies the original command to the end of the history but does not reverse its effect. The presence of a redo command shows that direct selective undo is invertible, instead of self-applicable.

Myers and Kosbie [118] designed the Amulet UI framework with a recovery system which had the same semantics as Berlage's. The use of Command Objects allows user actions to be organised into a hierarchy. Lower-level commands can invoke Higher-level commands. In addition to Selective Undo, Amulet allows commands to be reused on new objects.

The GINA framework [18, 150] (a class library to aid the development of interactive graphical interfaces) is an implementation of direct selective undo.

Multi-user systems require a selective model of undoing so that the appropriate actions can be undone [40]. There are many collaborative undo systems based on selective undo (such as [43, 136, 139, 154]). Some selective undo models assume either no dependencies between actions or that dependencies will be handled separately. Script undo [11] is one such model, which works in the following manner. Given n commands $\langle a_1, \ldots, a_n \rangle$, the state after undoing command a_i will be equivalent to executing the commands $\langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \rangle$. Script undo acts as if the undone command had never happened, which completely ignores any dependencies between actions. The script must be modified to undo commands.

Cascading selective undo [38, 39] determines whether any actions besides the requested one must be undone to satisfy dependencies.

Cass, Fernandes and Polidore [40] compared script undo and cascading undo to determine which one was better understood by users, by asking them to undo actions in a paper-based system. Using a paper-based system facilitated identification of what users would choose if they were not influenced by the functionality of any particular application. Another disadvantage of using an application is that it may bias user choices to how specific applications work. The goal of the study was to determine how users would *like* undo to work.

Sun [154], Prakash and Knister [136] and Berlage [19] do not consider linear undo to be a natural concept for users, because it places unnecessary restrictions on users. Cass et al [40] expected that their study would show that cascading and script undo would be chosen with greater frequency than linear undo. The results show that cascading undo was chosen with greater frequency than script undo which was chosen with greater frequency than Linear Undo. Cascading Undo (on average) took less time than script undo. These results suggest that cascading undo is a more naturally understood concept. As cascading undo respects dependencies between actions, and script undo does not, the results suggest that any concept of undoing which respects dependencies would be easier to understand.

Undo, Skip and Redo (U, S&R) [164] is designed to address the limitations of both History/Undo/Undo, and linear undo/redo. As well as undo and redo commands, a skip command is introduced. The skip command does not change what the user perceives the history to be but can change the effect of later skip or redo commands.

U, S&R stores history in a tree. The skip command can move between branches of the tree, and change the effect of redo commands.

If the starting state was

$$\langle A_1, A_2, A_3, \ldots, A_n \rangle \tag{3.14}$$

and the user wanted to insert B_1 and B_2 between A_1 and A_2. The user would issue $n-1$ undo commands, to reach $\langle A_1 \rangle$. A_1 would be the root of the history tree, with the other commands in a branch. Executing B_1 and B_2 would result in a new branch, and the history would be

$$\langle A_1, B_1, B_2, \rangle \tag{3.15}$$

The user can then issue $n-1$ redo commands, and choose the original command each time, resulting in

$$\langle A_1, B_1, B_2, A_2, A_3, \ldots, A_n \rangle \tag{3.16}$$

To undo the first n commands, while leaving the following m commands intact, issue $n+m$ undos, followed by n skips, and m redos, choosing the first option each time, so that the m commands are redone.

U, S&R has the properties of invertibility and unstacking.

Little-JIL [174, 175] is a programming language for process-programming. Little-JIL programs are hierarchies of steps that are instantiated at run-time and placed on the agendas of agents acting in the process. A Little-JIL program represents a controlled walk through a design space, and the run-time system maintains a representation of a walk. Any potential Undo function must operate in a manner that allows the run-time system to maintain an accurate representation of state information.

An undo mechanism for Little-JIL must allow a primitive action that was not the most recent to be undone. Programs can specify actions for multiple agents so multiple actions could occur concurrently. The minimum requirement is for the undo of one agent's most recent action (a linear undo for individual agents). Completing actions can result in new actions becoming available. If the action completed were the last of the children of a sequential step, the sequential step would also be completed.

Steps in a program have control and data dependencies with other steps. Any undone actions must also ensure that the undo is meaningful given the dependencies. For example, if the first child of a sequential step were undone, the other children would also be required to be undone, to maintain valid state.

Given the dependencies between items, determining the semantics of a cascading undo is a challenge. Undo can either be classed as a meta command or as a user command [164]. If viewed as a user command, the undo would be undoable, but a meta command is not undoable. The dependencies between components make undoing an undo command difficult.

Although the Little-JIL undo system is limited in scope, it does serve to highlight areas to consider for any other undo system, with particular emphasis on the dependency of components and the effects of these dependencies on the ability to undo user actions.

Techniques for undoing which are selective require consideration as to how users will be able to select commands to be undone. The Amulet [117, 118] presents commands in a text-based list. The alternative [111] is to present the history graphically.

One limitation of many systems which provide a history of actions is that recovery commands (such as undo or redo) are the primary method of navigating the history [99]. A graphical representation of History allows users to examine the history without making changes. Allowing users to edit the history without relying on explicit undo and redo operations is one way of extending this concept [99].

Selective undo techniques can be better suited to routers than linear undo because such techniques simplify the process of undoing changes other than the most recent. The biggest differentiator of techniques is *how* the history is navigated, and entries selected for undo. However, selective undo techniques may be an improvement over linear undo techniques, but none of them provides the flexibility offered by Undo functions in collaborative software.

3.7.4 Collaboration

Software designed for use by multiple people simultaneously has characteristics which influence how changes can be undone. Regardless of implementation, any undo facility will be based on selective undo, because a user may wish to undo changes that they made, but changes made by others are later in the history.

Collaborative undo research often focuses on document editing, usually text or images, but the concept of multiple users having individual changes could be applied to any situation where changes are distinct.

Items of network configuration are distinct. Each item can be changed independently of others. The concept of a user could be mapped to network configuration, and treated as a multi-user system, regardless of who the person is who makes changes.

Young and Whittington carried out a knowledge analysis of undo [184], which highlights four questions that users must be able to answer to use an undo facility effectively.

1. What stream of activity is relevant? — for a single user system, this would be the user's activity stream. For a collaborative system, it can either be a stream of actions for an individual user, or a global stream of actions.
2. How is the stream articulated into units — how low-level are the actions?
3. Which unit is affected by an Undo?—what is relevant? An Undo command may operate on individual user actions, but only on destructive actions.
4. What is the definition of undo?—how is the underlying data structure affected by undoing, and how is this presented?

The first question is most important when considering collaborative undo systems because it directly concerns whether undo should be global or local to an individual user.

Questions two and three are just as relevant in single-user systems and multi-user systems [176].

The script method of undoing [11] can be extended to support multiple users [6], using both Global Undo (the entire history of commands from multiple users) or

Local Undo (only the commands from an individual user). A global definition of undo would be identical to the single-user variant of script undo because it allows changes to be undone regardless of who made the change. The effect is the same as only allowing a single user to make changes. However, for a local undo, the command to be undone would be dependent on *who* requested the undo. The command to be undone will be the last one made by the person who requested the undo, which may not be the most recent command.

A global undo would be simpler to implement, but may not fit a user's perception of undoing in many contexts, and may be expected to undo any of their changes. A global undo method may make sense for a text editor with a shared cursor.

If configuration replaced the concept of a user, the same principles would apply. In general, for any unique configuration item, any changes should be undoable at any point.

Dix proposed the handle space model [59–61], for multi-window systems, which used the metaphor of a multi-user system, and has since been extended to model multi-user undo [6], with the goal of determining an appropriate solution for the disparity between local and global undo. If considered in the context of configuration, it provides a useful model in the context of managing the state of unrelated configuration items.

The set C denotes the set of user commands. Each command is tagged with the user that issued the command. U denotes the set of user handles. For two users, this would be $U = user1, user2$. H denotes a history of actions, as

$$H = (U \times C)^* \tag{3.17}$$

S denotes the set of possible system states. The state transition function *doit* maps the initial state s_0 to the current state. *doit* is defined as:

$$doit : (S \times H) \rightarrow S \tag{3.18}$$

Each user sees only part of the state of the system. $_u$ denotes a user's tag and display functions map from system state S to a displayed state D and are denoted by

$$\forall u \in U \bullet display_u : S \to D \tag{3.19}$$

For a collaborative editor, the state includes cursor and selection information, which is not part of the document, so the model also defines a result set R, which includes state which is a part of the document. It is a mapping from the state to the result, without regard to user information and is represented by

$$result : S \to R \tag{3.20}$$

There must be a degree of independence between interactions. Total independence is not always desired, such as in a collaborative editor. The two types of independence are result independence and display independence.

Result independence can be expressed such that for any state s, two commands — a issued by *user1* and b issued by *user2* are result communicative if

$$reult(doit(s, h_{ab})) = result(doit(s, h_{ba})) \tag{3.21}$$

$$h_{ab} = \langle (user1, a), (user2, b) \rangle \tag{3.22}$$

$$h_{ba} = \langle (user2, b), (user1, a) \rangle \tag{3.23}$$

Display independence means that the results of an action performed by one user are not perceivable by other users, even if the commands are result-communicative. If the relative order of commands issued by different users do not affect the display, the commands are display-communicative.

Consider h_{ab}, with a issued by user 1 and b issued by user 2. If user 1 changes their mind, the result depends on whether the undo is local or global. The result of a local undo is that command a is undone, but a global undo would b, which was issued by user 2.

If commands are result-communicative, the history can be rearranged to order commands such that they appear to be local but allows a global undo. That is, commands a and b would be swapped so that the undo would undo a whether local or global. For non-communicative commands, local undo does not make sense, because the state cannot be determined.

From the user's perspective, the ordering of the commands in the history is irrelevant if the result is as they expect. That is, if $\langle a, b \rangle$ and $\langle b, a \rangle$ yield the same result and appear to have the same history, the actual order is not significant.

There are several strategies to ensure communicative commands. Coarse-grained collaboration methods such as locking, roles, or copying, prevent multiple users from editing the same object or part of an object.

There are fine-grained options, which allow collaboration, but ensure that two commands would have the intended result regardless of the order they were made. The Grove group editor [67] uses this type of technique.

Commands can also operate using dynamic pointers [61], which can be updated automatically as the underlying state changes.

When considering router configuration, changes made to distinct configuration items should be undone in a local manner (only one configuration item should be changed). More complex changes would need a global undo, or grouping related primitive commands into a single composite command.

Some multi-user undo systems are extensions of other single-user systems. Prakash and Knister [135] extended History Undo, by allowing history entries to be tagged. Tags could be users, dates, the reason for the change, or any other criteria for selecting the command to undo.

Undo facilities in collaborative systems must be able to handle conflicts [135]. Tagged History [135] checks for conflicts when undoing. Any implementation must also be able to transpose commands. That is, for any two commands which do not conflict, the order of execution should not matter.

In its simplest form, Tagged History cannot handle multiple undo operations correctly. If two commands conflict and an undo removes the conflict, it is not possible to undo any additional commands, because the subsequent undo commands are unaware that the conflict has been resolved. The solution is to track undone operations and link them to the original operation, which would highlight that conflicts have been resolved.

When undoing operations in a collaborative system, operations can be selected for undoing using several possible criteria, such as the user (so that each user can only undo their changes), the region of the document, or treat primitive commands as a composite command, and undo the composite command.

The GINA framework [18, 150] has been extended to support multi-user systems [20, 21]. History is represented as a tree of command objects. Collaboration is achieved by replicating the command objects between application instances. Command objects represent state transitions, which can also be achieved using the undo and redo meta commands. The undo model in multi-user GINA has the property of invertibility, and the undo meta command has the property of unstacking.

Synchronous cooperation can be accomplished by broadcasting new command objects to all participants [15, 140].

If What You See Is What I See (WYSIWIS) [153] properties are required, selection and scrolling must be implemented as commands, to keep the scroll position synchronised.

For a multi-user system, GINA supports different degrees of coupling, ranging from WYSIWIS to separate, but remote edits.

Rhyne and Wolf suggest that it is harmful to use only the terms synchronous and asynchronous to categorise time. They designed the WeMet [140] framework to use both categories as appropriate.

According to Rhyne and Wolf, all applications have an application component and a viewer component. The viewer component is the part that facilitates user interaction.

The WeMet architecture adds additional components to support collaboration and history. Any application which uses this framework is required to support Undo and Redo operations at the application level. The presence of a redo operation shows that the undo command has the properties of invertibility and unstacking, but not self-applicability.

Commands are added to the history before being executed so that the commands can be shared between instances by sharing the history.

HistoryCursor objects navigate through the history. UNDO actions move backwards, and DO actions move forward. HistoryCursors and their corresponding histories are the mechanisms by which state is changed, rather than a log of changes.

The single-user linear undo model developed by Wang and Green [167] has been used as the basis for a multi-user undo system by Choudhary and Dewan [50]. All executed commands are shared between users. Executing an undo command or a redo command will perform the command for all users.

The limitation of this model is the assumption that all copies of the history have the same commands listed in the same order, which requires an atomic broadcast facility or restricts changes to one user. If ordering is not guaranteed, commands may appear in a different order in each instance. If one user performed an undo command on the latest command in their history, the command to be undone would be different in each instance where the order of the history differs from the one where the undo command was issued. If each command were assigned a unique identifier, it would be possible to request the undo of a command using its ID, meaning that the correct command could be undone even if it is in a different place in other History instances.

Only allowing the most recent command to be undone can be problematic in a multi-user system, because of delays in copying commands between instances. Commands can be marked to be undone.

A multi-user system has two categories of commands, collaboration (to share state between users), and computation commands (which request computation).

Undoing a collaboration command means that shared state will no longer be shared. The extent to which computation commands can be undone depends on the specifics of the command, and any side-effects.

In this undo model implementation is shared between the system and applications. Applications which only use undo-aware system components may not need any specific undo implementation.

Another technique for developing collaborative software with an undo facility is to use multiple timelines, such as in [64, 65, 89, 119]. Flatland is of particular interest because it not only stores changes in multiple timelines, but it allows timelines to be merged.

Flatland is a whiteboard system. Separate sections of the whiteboard can have distinct timelines, with a global history of all changes. Using separate timelines for each section means that changes in each can be undone independently of others. Also, changes can occur in other sections in the interim.

The global timeline is unique in that it is a shadow data structure which references the local timelines, rather than a separate copy of the data.

As demonstrated with Flatland, large-surface input devices (such as whiteboards) present a different form of collaboration, where it is a single application used by multiple people at the same time. In collaborative systems where there are multiple instances, Undo commands can be local (single user) or global (all users) in scope. For shared spaces, an undo system which operates on input regions might make more sense [146]. Flatland uses multiple timelines, one for each region. Regional Undo can be interpreted in different ways, such as field of view, or a group of items, although the field of view has been shown to be more easily understood [146].

Collaboration software is designed for multiple people. This means that undo functions must be able to handle undo or redo of any item in the history, without affecting other unrelated items. Router configuration changes may not be related. As such, using techniques suitable for collaborative software provides the greatest level of flexibility for allowing users to select which change they wish to undo.

3.8 Summary

This chapter presented an overview of different techniques for undoing changes and the requirements for storing the history of the changes.

There are four main techniques for undoing, with each having different requirements and different optimal use-cases:

1. System-Wide

   Including checkpoints, backups, and database data redundancy.

2. Linear

   History of changes is stored as a list. Only the most recent item can be undone. It is necessary to undo multiple changes if the desired change to be undone is not the most recent.

3. Selective

   Selective Undo removes the limitation that only the last item can be undone. The mechanism for how this is done varies, and the selectiveness of the techniques also varies, depending on the history storage technique.

   Storing history linearly, with items labelled still requires later history entries to be undone first, but this can be handled automatically.

   An alternative is to store history in a tree, with branches for unrelated changes, such that only changes in a given tree need to be navigated linearly. This means that changes to unrelated items can be skipped when navigating the history.

4. Collaboration

   Software which supports collaboration must be able to handle changes made by multiple users and must be able to allow users to keep separate histories, which can be navigated independently. Depending on the nature of the software or the changes made, it may also be necessary to store a global history, and apply some undo commands globally. Applications must be able to allow an individual user to undo their own changes without affecting other users in situations where there are no overlapping changes. This means that other users do not need to be aware that another user made changes which were subsequently undone if such changes would not affect them.

Another consideration is the meaning of an Undo command in an application, and how it affects the history. Undo commands can be self-applicable (an Undo command can undo itself) or not, in which case, a 'redo' command is required. For an undo command to be self-applicable, the 'undo' must be appended to the history. If history is truncated when a change is undone, it is not possible to redo changes unless a separate history of undone changes is also maintained. If the results of the undo command are added to the history, the undo could be redone.

The next chapter builds on both this chapter and the literature review to define the requirements and system design of the router, on which history and undoing have a considerable influence. The reason for this influence is related to the importance of storing historical information, to the extent that the technical implementation of both History and Undo underpin the implementation of all other components.

Chapter 4

Design

4.1 Introduction

The research questions (defined in Section 1.3) can be answered by building and deploying a router. The additional features required to answer the research questions are also influenced by existing research. The router features are therefore influenced by research presented in Chapter 2 and Chapter 3.

The previous research which had the most significant influence on the design of the router presented here included investigations into home networks but the characteristics of a home regarding routine and how the use of technology fits into this routine are also important. The specific issues raised, and the relative importance of these were discussed in Chapter 2. The research focused on 'undoing' methodologies and techniques is not considered to influence the functional requirements because, in general, the focus is geared towards defining how to perform an 'undo' in the context of the specific environment being studied. As such, the need for an undo function stems from home networking research (Chapter 2). The requirements for an undo function are technical.

The technical requirements exist to define the demands of the technologies used to implement the system, such that the implementation will address the functional requirements. In this case, the primary factor in determining how the system should satisfy the functional requirements is the method used to store the configuration changes. All the features are linked to the configuration, so the needs of the history storage will have a significant bearing on the overall system architecture and the resulting implementation details of the other components.

4.2 Issues to be Addressed

Home networks are often managed by people who inexperienced and reluctant. However, the technology is the same as all other networks, and particularly the Internet, which means that householders are required to understand technical details if they are to successfully configure a network and connect all their devices [63].

Another difference in homes is that networks have to fit-in with the home and its fixtures and fittings, along with the routines of householders. Network management is not a full-time task, and is just another household chore [157, 158].

The literature review (see Chapter 2) highlighted two issues that could potentially be addressed. The first is network usage and how this impacts on troubleshooting, and the second is device management. These may appear to be entirely separate issues—with two entirely distinct solutions—however, configuration changes can result in devices being unable to connect or, conversely, allow a device that previously could not connect to do so.

Also, the devices that a user wishes to connect to a network will impact both on the physical placement of network infrastructure devices (modem, router, range extender or switches), and on the configuration chosen.

For example, WEP encryption has a well-known security flaw ([36] for example), and would only succeed in making it more difficult for a neighbour to connect to a network. In homes, however, devices may not be upgraded until they are no longer functional, or a change in circumstances means that the existing device may not be fit for purpose. If there are devices old enough, then the use of WEP may be required, whereas a house with only newer devices could ensure they support WPA2. Following what would be considered best practice, (using WPA2 instead of the less secure alternatives), would be problematic in the first situation.

The other link between device management and router configuration is that changing configuration can have an effect (either positive or negative) on the ability for devices to connect to the network. As more devices gain the capacity to connect to Wi-Fi, this has the potential for additional complexity.

4.2.1 Network Usage and Troubleshooting

As discussed in chapter 2, there are some areas where users have issues in using their network, with troubleshooting being an area of concern.

In homes, routines define what functionalities networks must provide. If the network is able to contribute to the normal routine in a home, the network is behaving normally, and abnormal behaviour is identified by a disruption to the routine caused by the network [53].

Network management activities are often considered as merely one task that must be accomplished to maintain the routine of the household. As such, modifications tend to occur at times when disruption will be reduced unless immediate action is required for normal household routines to continue.

Furthermore, when users encounter issues, their primary concern is restoring order as quickly as possible, which often means finding a solution quickly at the expense of finding the best long-term solution. The obvious downside to these temporary fixes is that there is a high likelihood of the issue recurring.

Also, the way in which problems are identified or considered may be different from that of a commercial network managed by experts, even though the actual requirements may be the same. For example, performance is often a concern, regardless of the network. In a large network, issues would be found by analysing performance metrics and attempting to optimise for best performance.

Methods applied in homes are different because the primary concern is making sure the household routine is maintained. A performance issue might be identified by constant buffering of an online video. In this case, there is a need to improve performance. However, the focus will be on getting the video working, and not on achieving optimal performance. To do this might mean identifying other devices that are also using large amounts of bandwidth, and prioritising the video, often by stopping the other activity, or by asking the person responsible for the bandwidth use to stop what they are doing.

The action that supposedly fixes the problem may in fact not do so, because the user may be incorrect about the reasons for the problem. Users often rely on tools built into the router or the operating system of their device for information [178], rather than use other tools that might present the information in a way which is easier to understand. The use of interfaces built-in to routers and operating systems suggests that there is a need for more transparency regarding the state of the network and its use.

As a result, the focus of this research is changes made to the network configuration, with the hope that any issues caused by changes to configuration could be addressed quickly, and in a manner that permanently addresses the problem, rather than a temporary quick fix solution.

4.2.2 Device Management

The trend for interconnected devices is one that also increases the complexity of management tasks. The biggest challenge presented by these devices, such as Internet-connected thermostats, light bulbs or TVs is that they often have a unique type of interface, which makes the action of configuring them for network access difficult. Many devices lack the traditional keyboard and mouse, which are still necessary to facilitate entering network details given the implementation of networking equipment that is currently available. Each of these devices will have a different method of addressing this challenge. Some use on-screen keyboards, with varying levels of usability. Others require the use of a mobile application to configure the network. For example, Chromecast[1] asks users to connect their device to the network that it broadcasts, to use that device to send the actual Wi-Fi connection information to the Chromecast. Once done, users can switch back to their network, and the Chromecast will connect to the network.

There is a significant body of research focusing on improving the process of connecting devices, such as IceBox [177], or Multinet [29], but less on the specific issues surrounding updating network settings on devices when the configuration changes. These types of system could be used in such situations. The more significant problem is that all devices are disconnected when the configuration is changed. As the number of Wi-Fi devices increases the work required to reconnect everything will be considerable. Another consideration is that any network maintenance must fit in with the routine of the household [53]. A consequence of the current brute-force update required when settings are changed is that all wireless devices must be reconnected, which can be a daunting prospect in itself. When combined with the

[1]https://www.google.com/intl/en_uk/chromecast/

fact that the tasks must be completed for the usual routine to resume, the problem is more significant still. If a goal of designing networking products for domestic environments is to make the products fit with routines (which it could be argued should be the case), then simplifying large-scale maintenance tasks (initial setup, reconfiguration or relocation, for example) is essential.

Modifying the process by which the network configuration is updated is one possible solution. The objective is to update the configuration in a manner that does not disconnect any devices because that would allow for the provision of a grace period for users to update the settings on all their devices, with an initial suggestion of one day. By not forcibly disconnecting devices, there is the possibility that network configuration could be modified at a time when the device is not being used, but that still allows full use of the network in the meantime.

Another potential improvement is to ensure that users are aware of the need to reconnect all of their devices, which should be accomplished by sending notifications to users for each device connected, to inform them that they need to reconnect that specific device. In addition to informing other members of a household that they need to reconnect their devices, notifications would also serve as a reminder to reconfigure devices which are part of the fabric of the home. There are types of devices which are used all the time, but it might not always be evident that they use the network and would need to be reconfigured.

4.3 Requirements

The high-level requirements must be expanded into functional requirements, with greater detail as to what functions would be provided and how they would be expected to function.

4.3.1 Improved Network Updates

When making changes to SSID or WPA password, any devices which are connected will be disconnected, because the details they are using are no longer valid. This means that users would have to update the configuration to reconnect their devices. As the number of devices connected to a network increases, the complexity of reconnecting devices will also increase. To reduce the impact of a configuration change, the router should create an entirely new network using the new settings, and leave the old network operational for a short period, after which the old configuration would be removed.

The usefulness of such a feature could be improved further by notifying the person responsible for each device that the network settings were updated, which would also serve as a reminder to ensure that all the devices are reconnected, where they might otherwise be forgotten about until it was realised they were no longer connected. This situation could occur for devices that are part of the home infrastructure, such as Nest thermostat[2] or Nest Protect[3]. Because these devices are a normal part of the home and have the potential to blend into the background, there is a possibility that users would not think to reconnect them. They might do so with their computer because it would be immediately apparent that it was disconnected when used. A device in the background may not be used enough to notice that it needed to be reconfigured or had disconnected.

Further enhancements could be achieved by adapting notifications to the circumstances. For example, if network settings are changed back to the originals before the grace period had ended, and some devices have not been reconfigured, they should receive notifications telling them that they have nothing to do. Other devices should trigger a notification alerting them that the old settings will be restored, meaning that they will need to make sure they use the original settings again.

4.3.2 Configuration History

The router should track changes to network configuration. User-initiated changes are particularly important but system changes should also be included where doing so could provide useful information, or be used for troubleshooting problems, and might include a notification that Internet access is currently unavailable, and state why e.g. DNS error.

In general, it is not useful to log when DHCP leases are assigned or expire, for example, because underlying technical details are relatively meaningless to users, especially if they are not familiar with networking terminology [178].

[2]https://nest.com/uk/thermostat/meet-nest-thermostat/
[3]https://nest.com/uk/smoke-co-alarm/meet-nest-protect/

Including system-initiated events should be considered as a potential extension because doing so needs to strike a balance between providing information, but doing so in a manner that does not present users with unnecessary technical details.

4.4 System Design Considerations

Having identified the problems to be addressed, it is then necessary to examine the potential technology that could provide a solution. There are several factors to consider when choosing system components or architecture or the software that will be used. Each possibility has positives and negatives, and these influence the decisions made. The purpose of this section is to provide an analysis of these details and explain the reasons why each decision was made.

4.4.1 Platform

The first design decision to be made is whether to add desired features to an existing router firmware such as OpenWRT or to build an entirely new system.

The advantage of using an existing router firmware designed for router hardware is that initial setup and deployment is simplified because it uses readily available hardware, which might already be present in a home. The problem with most routers designed for use in a home is that they must be small and have minimal power usage. This combination usually means that such routers often have slow processors, limited RAM and only a small amount of storage. Tracking changes to router configuration over time is a task that would quickly fill all available storage, and would also use enough system resources, so as to limit the performance of the network.

Building a new system means that hardware can be chosen such that hardware performance is not a limiting factor. The main negative of doing so is that all router components have to be created from scratch, or use existing UNIX programs to perform common tasks, such as Hostapd to create a wireless access point and Dnsmasq for DNS requests and DHCP leases. In this situation, this presents the problem of how to track changes to the configuration, because each program uses a different configuration format and location, making the use of version control technologies (such as Git) difficult.

As a result of prior work in the Homework project [116] in which I developed the notification system, and a resulting high level of understanding of the system functions and architecture, the use of the Homework router as the basis for a custom router is a sensible approach.

The main aims of this thesis are not the same as those of the Homework project, so a subset of the features created in the original project are included. The Homework project is complete enough that it offers an excellent starting point for development, without the need to implement the entire system from scratch, but without the limitations of a traditional router, even with custom firmware.

4.4.2 Data Structure Considerations for Undo

The simplest way to store configuration is to store individual items of configuration in separate documents. Each item of configuration can be tracked individually.

There are two possible implementation strategies for this type of undo, given the use of CouchDB. The first is to generate a list of events based on the document versions, which has the advantage that there is only one reference to a configuration option. The downside is that presenting a list of events to users requires regenerating the list every time it is requested, which would be slow for a long history.

The other, preferred option is to create another document for every change that represents the event that occurred, as displayed to users, with a link back to the specific version of the specific document for which the event was created. There is no need to recreate the list of events every time it is accessed, and this still allows the notion of undoing based on the stored revisions of the documents.

The other advantage to using separate documents for events is that it is not necessary to keep track of when configuration changes occur in the same document as the configuration itself. It also better supports the notion of selective undo because it separates the history of data changes and the data. Generating the history of changes would make it difficult to undo changes while keeping the rest of the history intact. This implementation would be better suited to a more linear undo where the entire history is modified when a change is undone.

4.5 History, Undo and Revert

4.5.1 Properties of Configuration

Network configuration has certain characteristics which influence the techniques used to track configuration changes and how changes can be undone or redone.

First, each component in a router has a separate configuration, meaning that any configuration is distinct from any other, and changes to one will not affect the others. For example, changes to Hostapd configuration will neither affect, or be affected by, changes to DNSMasq configuration, in the sense that changing one program's configuration should not prevent the other from running.

It is, however, possible that changes to one application's configuration could affect the running of another application.

Consider the combination of `Hostapd` and `DNSMasq`. If the `DNSMasq` DHCP server were used, the DHCP lease configuration could limit the number of devices that could connect to a network created by `Hostapd`. In this scenario, the configuration does not affect the applications but could affect their usage.

Because of the independent nature of configuration, any recovery technique which is a form of linear undo would be inappropriate. Operating in this manner would mean either that the temporary undos should be ignored, or the configuration of applications would be modified several times, each of which would be unnecessary, and could cause unintended side effects.

4.5.2 Possible Solutions

One solution to the problem of correctly handling distinct configuration items is to treat each distinct configuration item as if it were a user in a multi-user system.

Multi-user systems can undo changes in two ways. The undo operation could be local (only consider changes made by a user), or global (all changes, regardless of the user). Depending on the context, either could be appropriate. In some applications, it may make sense to use both in different contexts.

In the case of configuration, local undo operations would be appropriate to undo most changes. Global undo would only be required if a user wanted to undo multiple changes at the same time, such as reverting to an earlier point in time.

The undo mechanism in Flatland [64] must handle changes to distinct areas of the whiteboard independently of other areas. The Flatland Undo system would serve as a useful starting point for developing an undo system.

4.5.3 Undo system

This section presents the model used for history storage and undoing in this router.

The model has been influenced by other research into undoing techniques in other contexts[4]. The Flatland undo system is the most relevant.

Changes to each distinct item of configuration are tracked in separate histories. There is also a global history which is used to present changes to users, and where all user interaction takes place. The reason for this approach is so that each item of configuration can have changes undone directly, without needing to undo changes to other configuration items. Using separate histories, and a global history which serves to drive the visual representation of the history means that the timing of the change to be undone is irrelevant.

The global history uses timestamps to order the events from each local history in the order they occurred, but the ordering of events and the timestamps are not required by the undo system.

Each configuration item's history is formed by storing each version of the configuration as changes are saved. Each version is tagged with a version identifier. When individual configuration items are modified, a new version is created, and a new entry is added to the global history. The global history contains a user-friendly description, the timestamp, and the version number of the corresponding change so that the undo system can request the correct version from the appropriate individual history. Specifying the version number facilitates Undo a change which is not the most recent.

The user-friendly information is required because it may not be clear exactly what changed between two versions. If the process which updated the configuration

[4]See Sections 3.7.4 and 3.7.3

initially is also responsible for updating the global history, it can provide this information based on knowledge of the underlying change.

Timestamps are not part of the individual histories because the undo system only operates on the individual local history, which neither knows nor cares, *when* a change occurred, merely that it occurred. On the other hand, a user may be interested in when changes occurred.

The history depicted by (4.2) represents changes to an individual device and (4.1) represents changes to Wi-Fi. (4.3) shows how they could be interleaved. Because the global history stores a timestamp with each entry, the entries are sorted reverse-chronologically so that the most recent changes are presented first.

$$\langle A_1, A_2, A_3 \rangle \tag{4.1}$$

$$\langle B_1, B_2, \ldots, B_{n-3}, B_{n-2}, B_{n-1}, B_n \rangle \tag{4.2}$$

$$\langle A_3, B_n, B_{n-1}, A_2, B_{n-3}, B_2, A_1, B_1 \rangle \tag{4.3}$$

All changes to configuration will be stored in the local history. However, it does not always make sense to include a change in the global history. An example of a local change which would not be included in the global history is when a device requests a new DHCP lease. The DHCP server will store the current state of a DHCP lease, and the leased IP address with other device details. However, it does not make sense to present the activity of the DHCP server to users, because of the technical complexity, and because they would not be undoable (so that the DHCP server is the only component which can modify DHCP lease information). This example also illustrates why the global history requires a reference to the version number of each entry. If a user wishes to undo the most recent change to a device in the global history, there is a significant likelihood that the change will not be the most recent in the corresponding local history.

4.5.4 Undo

A side-effect of not storing all changes from local histories in the global history is that each configuration type interprets the concept of undoing differently.

The undo algorithm must behave differently depending on what the configuration is that is being undone, and the state of the configuration being undone.

In all cases, the undo system will identify an older version of the configuration to be reapplied. A new version of the configuration will be created based on the old version. The reason for creating a new version is that the undo will be appended to both the local and global histories.

The criteria used to select an older version and how the changes are applied are different for each configuration type.

A device will have a local history which contains changes not found in the global history, caused by the DHCP server. The undo system would need to choose a previous version from the local history which is also in the global history (which means that a user made the change). However, DHCP settings should not be modified, so the new version only updates user-modifiable settings (such as name, owner, or if the device can access the network), to the old values, and ignores the other data. The new version is a mix of both the old version and the most recent version at the time the undo command was issued (which could be newer than the command being undone).

4.5.5 Redo

Undo commands are appended to the history. They can be undone by using the same undo process starting with the version created by the undo. The previous version would be the change that was undone so that a new version would be created from the original change, and the undo would be undone. The redo would also be appended to the history, and can also be undone.

4.5.6 Revert

There may be scenarios where it makes sense to revert state to an earlier point in time, or to before a particular change occurred.

This model follows that pioneered in early systems which provided an ‘undo’ facility. In those situations undoing a change means to revert to a timestamp before the change occurred.

In the case of this router, the method used to revert is based on the undo method.

When a user wishes to revert to an earlier point, the system will retrieve a list of *all* events that occurred on or after the specified timestamp, that are undoable. The revert feature is the only part of the undo system which uses the timestamp in the algorithm, and that is not merely providing a criterion for sorting the Global History.

Given a list of events to undo the simplest solution would be to iterate over the list and undo each change, using the existing undo mechanism.

The problem with such a simplistic approach is that it has the potential to trigger multiple undo requests for the same configuration.

When performing an individual undo, the global history must specify the version of the configuration to undo changes to, which may not be the most recent. The need to specify the version of the configuration can be exploited in the revert process.

(4.4) shows a sample local history, with the most recent change first.

$$\langle A_6, A_5, A_4, A_3, A_2, A_1 \rangle \tag{4.4}$$

Assume that the revert event selector determines that A_6, A_5 and A_4 must be undone. The result of applying three Undo commands is shown in (4.5), which demonstrates the effect that the configuration appears to have the same state it did in A_3.

$$\langle A_3, A_6, A_5, A_4, A_3, A_2, A_1 \rangle \tag{4.5}$$

An alternative approach to obtain the result shown in (4.5) is to undo A_4. The new state obtained from undoing A_4 will be the same as A_3.

The revert system uses this enhancement when selecting document versions to undo. First, a list of all events in the global history starting with the first with a timestamp greater than or equal to the timestamp to revert to, and ending with the most recent.

Hash tables store data as a series of key-value pairs. The key must be unique. If the ordering is maintained, it is possible to identify the oldest version of any individual configuration item by attempting to insert each change into the hash table, with the ID as the key. If the key does not exist, it will be added to the hash table. Documents will only be included once, with the oldest version number of each.

The revision list is reversed so that the most recent event is stored first, and the events are undone one after the other. Because configuration is independent, the order does not matter. The reason for starting with the most recent is so that the individual Undo commands are presented to users in the reverse order to which they occurred.

The current revert algorithm does not support undoing of the revert operation because the revert operation does not store the affected configuration items or the resulting version numbers from the original undo operations that resulted from the revert request.

With the necessary information available, a revert could be undone, by simply iterating over the list of modified configuration items and undoing the change at the specified version number.

4.6 Summary

This chapter used issues identified in the literature to determine features of a router with configuration history, and the need for an easier method of reconnecting devices.

This list of features was then used to create a list of specific requirements that the router must be able to satisfy to realise the features defined.

A discussion of technical constraints is also included because technological constraints influence the technologies used, particularly for data storage, and how the needs of an undo system would determine the data structure, and which database server would be the best approach for the identified requirements.

The chapter also outlined the design of the undo and revert algorithms.

The next chapter expands on the design decisions in this chapter and provides an overview of the system components and how everything fits together.

It will provide a more detailed description of the features of CouchDB, with particular reference to those required to implement the features of this router. The use of CouchDB influences the architecture and implementation of the router. It is necessary to understand how CouchDB works to understand the implementation details. Using HTTP for data access, and serving static files, are both useful features, but the effect these have on the implementation is significant. The two methods of data access (just using the CouchDB web server or using an external web server) are an example of how the database features can change the fundamental design of the system.

Chapter 5

System

5.1 Introduction

This chapter is a detailed description of the software designed as part of this research. The chapter includes a description of the system architecture, as well as a more detailed description of each of the components.

5.2 Technical Design Considerations

5.2.1 Configuration File Management

Some router features are controlled by separate processes to those used to track user changes and apply changes. For example, Hostapd manages Wi-Fi and Dnsmasq is responsible for DNS. Both of these store configuration in files. Two main strategies could be used to manage such a configuration. The first option is to track changes to files in version control and save the commit identifier in the configuration database when changes are made. Moving through history could be done by manipulating the version control system to ensure the correct version of the file is the one used by the applications.

New changes would be made as a result of changes to the datastore, and a new commit would be created.

There are problems with this approach. The behaviour is different depending on whether the change is new, or an undo of a previous change. Also, some configuration, such as device settings or notifications do not use configuration files, meaning that such a technique is unsuitable unless SQL database queries to change the data are used and tracked, but doing so adds complexity.

Relying on version control also adds extra steps to the process of configuration management, and increases the complexity.

The alternative approach is to recreate (or edit, if appropriate) the configuration file every time the configuration is changed. This option could be considered less efficient if an update would create an older version of the configuration file that was in use previously. However, using this approach means that the method used to update the file is the same regardless of whether the change is new or an undo of a previous change. It also does not require configuration files and databases to be treated differently.

Another potential option considered is a compromise between these approaches, and takes advantage of CouchDB's ability to attach binary files to documents. Each configuration file version would be part of the document that created it. The configuration file could then be updated on disk by downloading the appropriate version, and copying it into the appropriate place. The downside to this approach is that doing so requires more HTTP requests. As well as PUTting a new document version, the attachment must also be PUT on the document. Attachments are retrieved with GET requests. For configuration files where the name is known, this can be retrieved directly, because its URL will be the document ID/attachment. However, the ID of the document must be requested first, which means performing a query, which also requires multiple HTTP requests. As a result of HTTP overheads, attaching configuration files to documents is less efficient than just regenerating configuration every time. Storing the file as an attachment is also subject to the same limitations as apply to using version control, namely that any configuration not in files is not track-able in this manner.

An example of configuration not stored in files is notification details. Notifications only need to be able to map between a name and a username, so only requires a database.

5.2.2 Database

The main requirement of a system used to track configuration changes is that of a suitable data store. The requirements that such a data store are the ability to undo any change, at any point in the history, including the ability to undo these undone changes. It should also be possible to undo several changes together to roll configuration back to a previous state. Many potential databases could be used for such a purpose, some of which are already present in the Homework router.

The requirements for history storage and undo mechanism, both functional and technical, mean that there are arguments for and against any database. Presenting these arguments will explain the decisions that were made. The Homework router uses two databases. The first is HWDB. HWDB does not store data in a persistent manner, and data will be lost when the router is switched off. An additional database is required for persistence. Because they are already installed and usable, they are a suitable starting point. When choosing a database, many points raised which are specific to MySQL would also apply to other relational databases that use SQL. As such, the alternatives considered here are NoSQL databases, because they use an entirely different approach to storing data and are suitable for use in different situations and types of data than relational databases.

HWDB is a database created to store data used by each component, and all OpenVSwitch data flows. All data is stored with a timestamp. The database is also used to pass messages between components, and trigger the running of other components, using a publish-subscribe model.

HWDB is populated with data on every boot from data stored in MySQL. The original Homework architecture required that all main configuration options be stored in a central configuration file, which would be applied to the actual configuration or the HWDB database after the data is initialised from MySQL.

The problem with using HWDB to store configuration changes is that there is not a single optimal way of storing the data or reverting the data to an earlier version. It would be necessary to store the configuration in one set of tables, and use another set of tables to store the actions to be applied to the configuration.

Using MySQL as the configuration history data store would remove the need to duplicate history between two databases, as it is already persistent. There is also an inbuilt mechanism for applying changes to data, namely saved queries. Queries could be generated that would apply a change and also a corresponding query to undo the change. The problem with this is that there needs to be a way of storing these change files. Another limitation to using a specific query for each change is that for each change, it must be assumed that the change will always be the most recent when undoing. Without this, there would be no way of knowing the effect of the change on the content of the database, meaning that undoing one change could also undo others. Splitting data between tables (as is the case for the Homework Router for HWDB and MySQL persistence layer) expands the problem. For example having DHCP leases in one table, NOX data for a given MAC address in another, and other tables for each of device name, device type, and user, each of which linked to the Leases table by IP Address. The MAC address linked the Leases and NOX tables. Queries that undid changes would have to account for the fact that any record in any table could be modified independently of others, so a query may be created assuming each table has a set of known values, which in reality have different contents, so the result of the query may not make sense.

NoSQL databases such as MongoDB or CouchDB are another possibility. Both of these databases store data in JSON format, with no concept of relations between different data. Both refer to a single item of data as a document, and both have HTTP APIs.

MongoDB has a similar problem to relational databases in that it is necessary to track manually data changes. However, many MongoDB libraries provide a mechanism to enforce a schema to each document based on the document's collection.

CouchDB has a similar data storage mechanism to MongoDB. The main difference is that CouchDB is designed to be used on multiple physical servers at the same time. All changes to documents create a new version so that server instances can be synchronised. When making a change, it is a requirement to specify the revision of the document being updated, and attempting to update a document that isn't the most recent version is an error. Multiple versions of documents mean that tracking of configuration changes is built into the database, and does not need to be implemented manually.

The fact that CouchDB also keeps every version of every document also supports the idea that each item of configuration has a separate history. The history presented to users would be based on these separate histories, in a similar manner to that of the undo system in Flatland.

These features are the reason that CouchDB is the data-store of choice for tracking configuration changes.

5.3 Architecture

The system presented here is based on work undertaken by the Homework research project [114, 116]. The Homework Project created the device management, packet handling and notification components, and these have been modified for this research to support change history. The packet routing system (using OpenVSwitch, a software switch which uses the OpenFlow Protocol [110]) is a direct port of the original C++ NOX controller [2] into the Python POX controller [3].

The router built by the Homework project used a time series database, HWDB[1] to store configuration details, but because the data is not trackable, HWDB is replaced by CouchDB.

5.3.1 Overview

The router has ten components shown in Figure 5.1.

1. CouchDB Database.

   CouchDB is the database used to store configuration because it has a built-in revision system.

2. OpenVSwitch.

   OpenVSwitch and POX are used to capture how devices utilise the network. Custom POX components route packets, handle DHCP requests and interface with other parts of the system.

3. POX Controller.

4. Hostapd.

   Convert a Wi-Fi device into an Access Point.

[1]Based on the Stanford Stream database http://infolab.stanford.edu/stream/

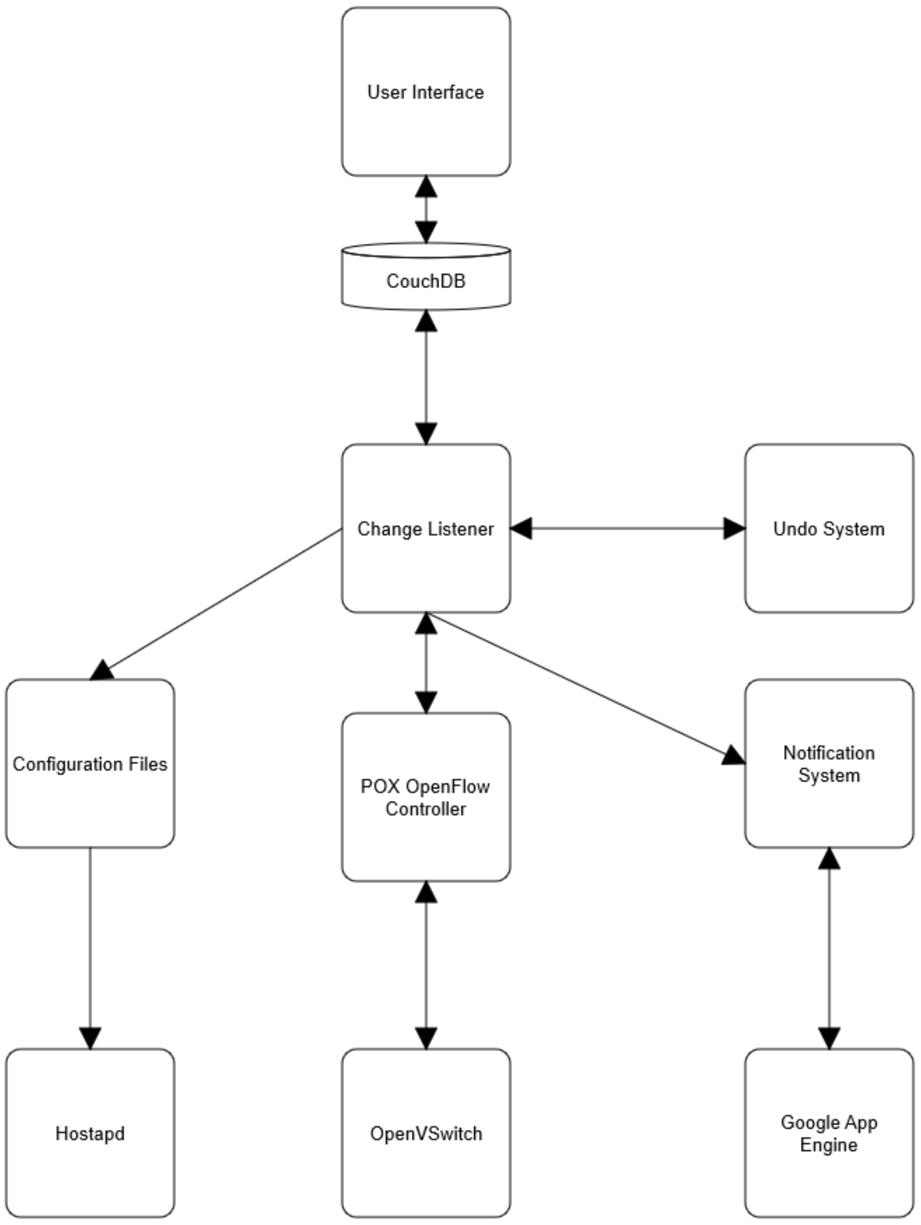


FIGURE 5.1: System Architecture

5. Configuration Files.

   The configuration files used to configure the components described in this section.

6. ChangeListener.

   Python scripts used to update router configuration in response to changes made in CouchDB.

7. Notification System.

   Sends notifications to alert users to network events or required actions.

8. Google App Engine.

   The Notification System uses Google App Engine to send notifications.

9. UI.

   The Timeline is a key component of the router because it presents changes to users. Also, all configuration changes happen in the UI which makes the UI the starting point for all interaction sequences.

5.3.2 Interaction Sequences

5.3.2.1 Modifying Configuration

Configuration changes follow the same overall process for any configuration type, but each will have slightly different actions depending on the storage medium used. The steps below are the process for changing Wi-Fi configuration.

1. Update Configuration on the Wi-Fi screen
2. Validate the change and save the new version of the configuration
3. Access the change from the `_changes` feed
4. Update the Hostapd configuration
5. Restart Hostapd
6. Update the document with the status
7. Report the status in the UI

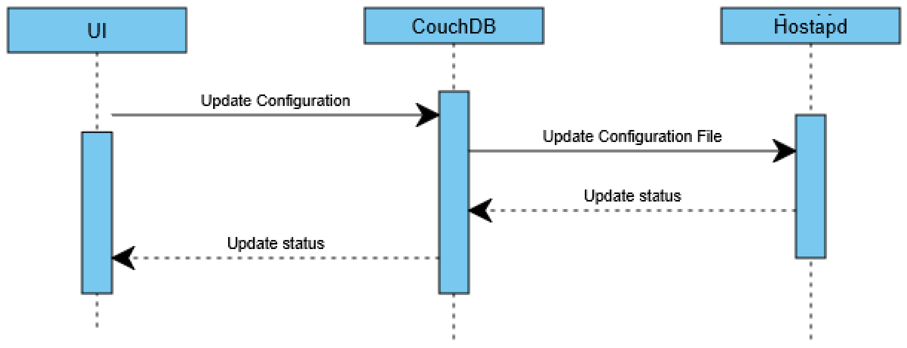


FIGURE 5.2: Wi-Fi change sequence diagram

5.3.2.2 Undoing

The process for undoing a change depends on the configuration type. Due to the asynchronous nature of making a change, the previous revision is often not the correct one to choose (because documents have a `status` field). It is also not entirely sensible to allow users to roll a configuration change back to a known bad state because they would then need to move further back through the history.

Not all configuration changes are undoable. The data store for configuration for a device includes the DHCP lease status. Users should not be able to modify an active DHCP lease unless they intend to remove the device from the network (otherwise the device will have a different view of the status of its DHCP lease to the DHCP server. Non-undoable changes made by central system components during normal operation are included in the individual histories but not the global history.

Because some changes are not undoable, it is necessary to undo the corresponding user change, while preserving the state of non-undoable actions.

The following steps are common to all cases of undoable actions:

1. Click the 'Undo' button for the change to undo
2. Fetch the document which stores the changed configuration, at the version referenced in the event
3. Request the list of revisions for the document
4. Find the version with the previous configuration values

5. Save a new version of the document based on the previous version (so that the update is appended to the history)

6. Create a new event document which is labelled as an 'undo' of a previous change

7. Reload the timeline with the new 'undo' event at the top

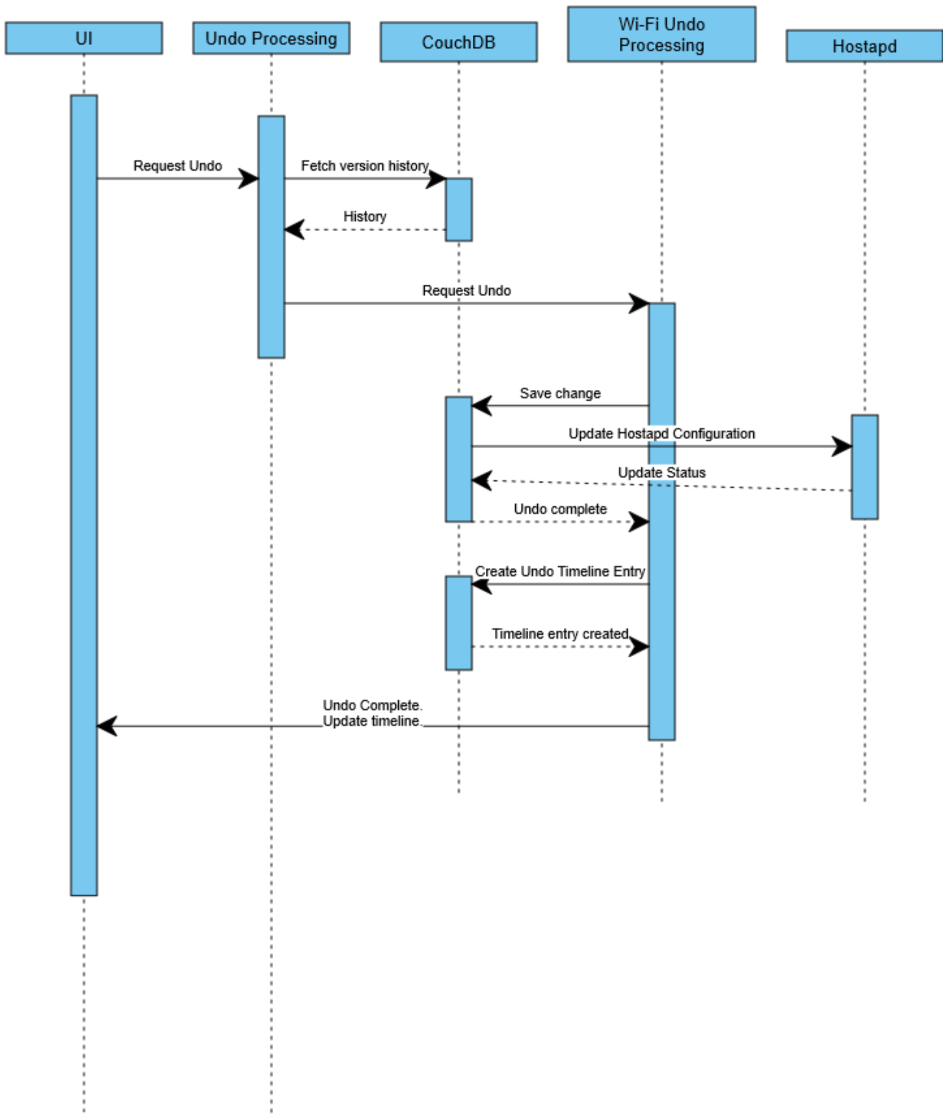


FIGURE 5.3: Wi-Fi Undo Sequence Diagram

5.4 User Interface (UI)

Other than permitting and denying devices, the features ported from the Homework Router did not have a UI. As such, the entire UI was created for this research. The UI for device management is not a direct port of the original Homework design because the techniques used are not the subject of this research.

CouchDB has a builtin webserver, which can host static files, such as HyperText Markup Language (HTML), Cascading Stylesheet (CSS) and JavaScript (JS).

The UI was created using some libraries and tools. The main structure of the HTML, CSS and JS is defined by the use of Backbone JS[2]. Backbone JS provides a suggested structure for web application code but does not dictate any specific methods of structuring code, or what frameworks to use. The only exception to this is that Backbone JS depends on Underscore JS[3]. jQuery[4] is used for manipulating the Document Object Model (DOM).

Marionette JS[5] is an extension to Backbone JS. It imposes greater restrictions on implementation details but does so to provide useful features.

Most of the screens are rendered by passing data to a template function, together with HTML, and Marionette JS provides default renderers for rendering these templates. Marionette JS provides an event mechanism to allow the rendering process to be customised if required.

Twitter Bootstrap[6] is used to style the UI. It provides an easy way to layout pages without having to implement custom CSS. Also, there are some test devices, each with different screen sizes. Twitter Bootstrap has built-in support for scaling a UI to the viewport size. The timeline makes use of Bootflat[7] for styling.

5.4.1 Backbone JS and Marionette JS

Backbone JS provides a number of component types. These are:

[2]http://backbonejs.org/
[3]http://underscorejs.org
[4]https://jquery.com
[5]http://marionettejs.com/
[6]http://getbootstrap.com
[7]http://bootflat.github.io

- Model
- Collection
- View
- Events
- Router

Marionette JS provides additional functionality. The functions in this list are used instead of the default Backbone JS implementation apart from Model and Collection, which do not have an equivalent in Marionette JS. The Marionette JS components listed are implementations of the Backbone JS features listed above. This list is not a complete list of components in Marionette JS, just those that are used.

- ItemView
- CollectionView
- CompositeView
- LayoutView
- Regions
- Application
- AppRouter
- Events

These are combined to provide all features necessary. The backbone-couchdb[8] library is used to provide implementations of the Backbone JS data synchronisation functions that correspond to requests in CouchDB.

Model Each model instance is a JavaScript object representation of a CouchDB document. Manipulating a model would update the appropriate CouchDB document on save. Backbone-Couchdb makes sure the model URL property is set to the document's `_id` field.

[8]http://janmonschke.com/projects/backbone-couchdb.html

AppRouter This component handles changes to the URL and triggers the loading of the correct view (Backbone JS), based on the URL. It also tracks which view is active so that views can be removed when they are no longer needed. It supports navigation history, making it possible to use the browser history manipulation features, such as back, forward, reload or bookmark.

This Marionette JS implementation of the Backbone JS router provides more structure to the definition of actions assigned to routes (URLs) than the standard Backbone JS router.

Collection A Backbone JS Collection is a collection of models. Collections are populated from CouchDB views. Backbone JS automatically creates models for each document retrieved from the CouchDB view, based on the model property of the collection. The default CouchDB used is byCollection, and it uses the Uniform Resource Locator (URL) property of the collection as the key to use when requesting all documents in one collection. If a db property is specified, it is possible to supply another view and to specify that the collection is updated when the database changes (using _changes). In this case, the appropriate filter to use must be provided.

Views Backbone JS views represent a portion of the UI. The view is responsible for rendering the HTML with the appropriate data passed to the linked JavaScript Template (JST) template function. Views are also responsible for handling both collection and model events such as add, change or reset, and to events triggered by HTML elements such as button click, or select element change.

The event system in Backbone JS uses the view instance as the context of the event, which makes it possible to manipulate the model or collection that the view is displaying in the event handler directly. In the case of Wi-Fi, a view with the context of the collection is sufficient because there is only one model in the collection. Where a collection can have multiple models, it is not possible to determine which model caused the event if the view rendered the collection. For this method to work, the mapping between HTML elements and the underlying data must be tracked manually.

There is an alternative approach that overcomes this limitation, and this is the one that is used. Collections are rendered as a series of nested views. The view representing the collection is the HTML container, such as `table`. Data for an individual model is presented in a separate view, with each view responsible for managing its events. For a given sub-view instance, it is possible to manipulate the underlying model directly. Using multiple views requires multiple template files and multiple view instances for each page, but it removes the need to keep track of which model was being interacted with when a button was clicked on. This simplification provides more benefits than the drawback of splitting the code into many smaller chunks.

This approach is simplified by using the Marionette JS components listed here.

LayoutView The Marionette JS LayoutView defines `Regions` which contain other view types. Regions simplify the process of managing UI components because LayoutView handles management of events and view removal.

The UI is contained in a LayoutView with two regions. The first region contains the menu items in separate `ItemViews`.

The other region contains the main content for each screen.

ItemView This type of view represents a single model. It allows the definition of UI and events that apply to an individual model.

CollectionView A collection view simply iterates over all items in a collection and creates `ItemViews` for each model in the collection. This parent view does not have its own UI. Changes to models in a collection are automatically applied to the corresponding `ItemVIew`.

CompositeView CompositeView is very similar to CollectionView, except that a CompositeView has a template, and model change events are not automatically applied to ItemViews. This type of view can be used for creating tables, to specify the column headings.

Application The Application is used as a central component to reference all other components. It is also responsible for starting all other components when necessary.

Events Events are handled in a similar way to Backbone JS, except for the handling of Model and Collection events, many of which are handled automatically, meaning that model changes are automatically applied.

5.4.1.1 Screens

This section describes each screen and the Backbone and Marionette JS components that they use.

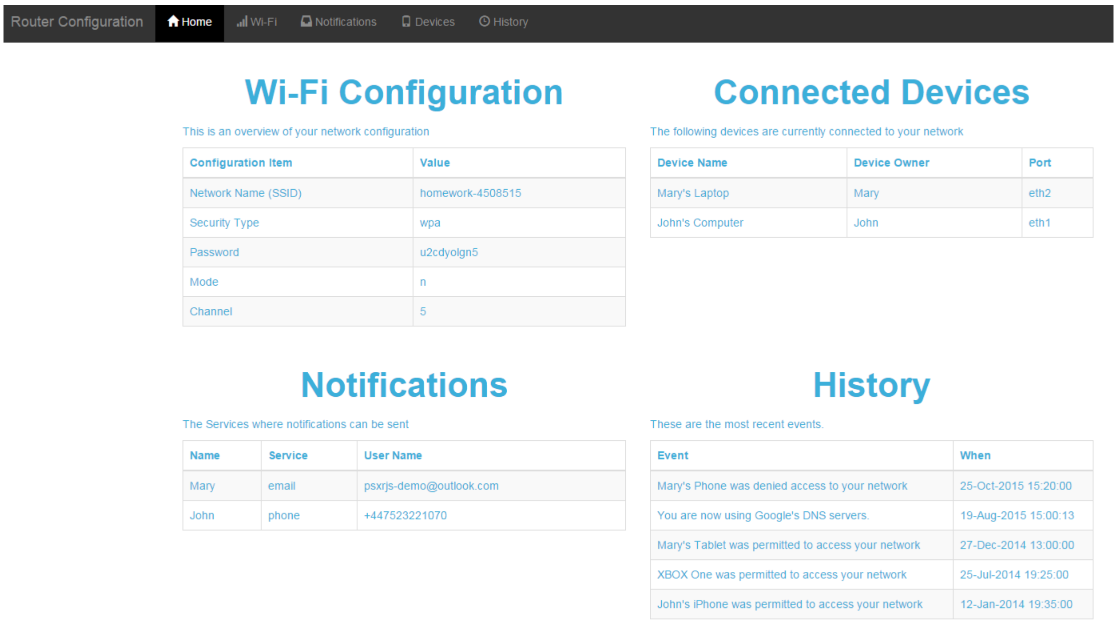
Home This screen is an overview of network configuration. Each section is a link to another screen with more details. There is a view that contains placeholders for other components, and separate views for each section. These subviews use the same models and collections as the views to which they link. The `Connected Devices` section is the exception because it uses a different CouchDB View, so needs a different Collection.

The four sections are arranged in a LayoutView, with each section in a separate region. The Wi-Fi section is a CollectionView (because there is only one model in the collection so it can render the whole table directly in the ItemView), the others sections are Composite Views (because the table header must be rendered separately from the models).

An example of what this looks like is shown in figure 5.4.

Wi-Fi This screen is used to update Wi-Fi configuration. Validation is performed by CouchDB when saving documents, so error messages are displayed at the top of the screen, and do not highlight the problematic fields. The collection used uses the default `byCollection` view because that is all that is necessary to retrieve the document in its entirety. The collection listens to changes using the `wifi_ui` filter function. The status field is changed when changes are applied to hostapd.conf, and subsequent changes need to refer to the latest version. By listening to changes, the latest version will be available automatically.

As there is only ever one document for Wi-Fi, the entire content region can be created by a `CollectionView` that loads an `ItemView` for the document. There is no ambiguity as to which model a UI event applies to.



Wi-Fi Configuration

This is an overview of your network configuration

Configuration Item	Value
Network Name (SSID)	homework-4508515
Security Type	wpa
Password	u2cdyolgn5
Mode	n
Channel	5

Connected Devices

The following devices are currently connected to your network

Device Name	Device Owner	Port
Mary's Laptop	Mary	eth2
John's Computer	John	eth1

Notifications

The Services where notifications can be sent

Name	Service	User Name
Mary	email	psxrjs-demo@outlook.com
John	phone	+447523221070

History

These are the most recent events.

Event	When
Mary's Phone was denied access to your network	25-Oct-2015 15:20:00
You are now using Google's DNS servers.	19-Aug-2015 15:00:13
Mary's Tablet was permitted to access your network	27-Dec-2014 13:00:00
XBOX One was permitted to access your network	25-Jul-2014 19:25:00
John's iPhone was permitted to access your network	12-Jan-2014 19:35:00

FIGURE 5.4: Home

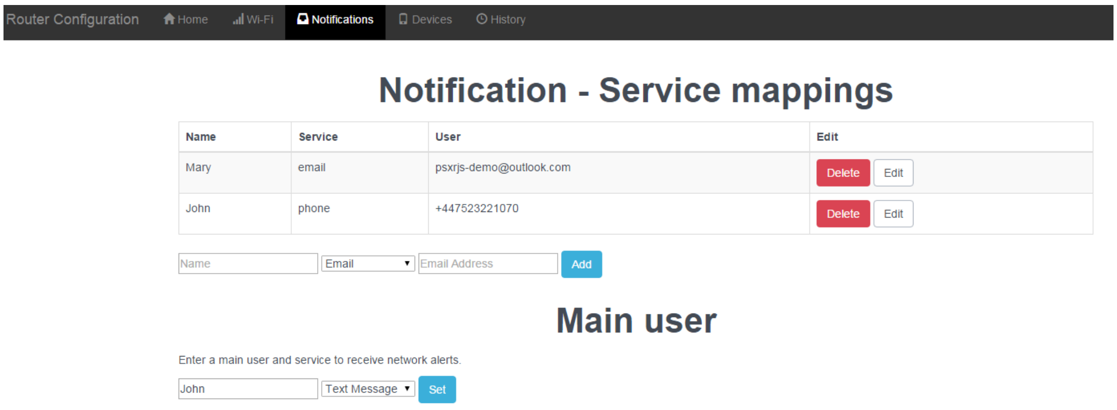
Configuration is displayed in a table, with configuration options shown in the left column. These are static HTML. The middle column `<td></td>` elements each refer to a field in the Wi-Fi document, and the table is populated with the data when the view is displayed. The `ItemView` template also includes the table header.

An example of what this looks like is shown in figure 5.7.

Notifications This screen is for the management of notifications. A `LayoutView` has separate regions for notification mappings and `MainUser` configuration.

The main user region references a single document, displayed in an `ItemView`. There is also a form to allow the screen to be updated. Both the form and the data are children of a `CompositeView`.

The notification–service mapping region is made up of multiple Marionette JS views. The `table`, `thead`, and empty `tbody` tags and add mapping form are in a `CompositeView`, that is also responsible for fetching the entire collection. Each table row is in a separate `ItemView`, with the data for the view being an individual model. Using separate views means that click events for the edit and delete buttons are automatically linked to the correct row, which will automatically update the correct CouchDB document.



Notification - Service mappings

Name	Service	User	Edit
Mary	email	psxrjs-demo@outlook.com	Delete Edit
John	phone	+447523221070	Delete Edit

Main user

Enter a main user and service to receive network alerts.

FIGURE 5.5: Notification Screen

The collection is updated with changes, using the `notifications_ui` filter function. It uses the default `byCollection` view.

When the edit button is clicked, the user field is hidden, and a text input is shown, along with both save and cancel buttons. Clicking either of these will restore the UI to its original state, and save will also update the corresponding document.

An example of what this looks like is shown in figure 5.5

Devices This screen is for device management. Devices are grouped according to their state (pending, permitted or denied). Within each group, they are sorted first by connection state, followed by MAC address. Sorting in this manner has the effect of grouping connected devices together.

The screen is made up of two view types. The structure of the page is one `CompositeView`, which has no user functionality. It is only responsible for fetching data and monitoring the collection for changes. This uses the `devices_ui` filter function and the `control` CouchDB view.

The other `ItemViews` are representations of individual devices, with different HTML templates depending on the state of the device. The template used to render the row will depend on the value of the state property.

Using separate templates is the simplest way of providing different HTML for each state. The other option would be to use JavaScript to show and hide different parts of the template depending on whether it is rendering a pending, permitted or denied device. The difference between permitted or denied is only the text on the permit/deny button, and the function called to handle the click. However, using a single template and updating the values is more complex than using separate templates. Pending devices need a separate template, because there is the addition of the hostname field, as well as replacing the edit button such that the device has both permit and deny buttons. The edit button is not necessary because these devices must have user values set when permitting them. Other states hide the text view and display text boxes and select elements when the edit button is pressed. The edit and permit/deny buttons are replaced by `save` and `cancel` buttons.

An example devices screen is shown in Figure 5.10

History The history screen presents a timeline of events that occurred on the network. The main history view is only responsible for loading the events and rendering the placeholder template, in a `CompositeView`. The events and the timestamps are in separate `ItemViews`, to make it easy to identify the event when the undo or rollback actions are triggered. The collection uses a custom CouchDB view, to request the documents ordered by timestamp, and also makes the request with descending=true, so that the most recent event will be displayed at the top. One side effect of sorting the events in this manner is that the addition of new events requires that the entire collection must be reloaded and the entire screen completely re-rendered, so that the new event is at the top, with the pattern of events from left to right also maintained.

Figure 5.11 shows an example of the display of the timeline screen.

5.5 CouchDB Overview

CouchDB is a key component in the router. It is used to store configuration history and to trigger changes to underlying configuration when changes are made in the UI. It is also a key component of the Undo and Revert functions. Because it is used by all other functions, it is important to understand basic concepts of CouchDB operation.

CouchDB is a document-oriented database. Each document is stored in JavaScript Object Notation (JSON) form. Its Application Programming Interface (API) uses

Hypertext Transfer Protocol (HTTP), and all queries use MapReduce [56] and are defined in JS functions.

CouchDB is suited to storing configuration changes because documents are versioned. Every change to a document creates a new version. The History and Undo technique defined in Chapter 4 maps directly into CouchDB concepts.

Full details of CouchDB's features and operation are available in the Wiki [1] and the CouchDB Guide [90]. The relevant details are presented here to explain the router features.

CouchDB documents all have `_id` and `_rev` fields. The `_rev` field is the version of the document, and the `_id` field is the unique ID of the document. Most documents have an `_id` value generated by CouchDB because the actual `_id` value is not significant, only the fields representing configuration data. It is sufficient to store the `_id` value in clients after it is generated. The `main_user` doc has `_id` set to `main_user`, to signify that the document should both always exist, and there only be one of them. Documents that represent devices use the MAC address as the `_id` because it simplifies the process of extracting a specific device data when dealing with packets and connection state changes.

5.5.1 Document Types

Each type of configuration has predefined properties that are dependant on what the configuration is. Devices have different properties to Wi-Fi for example. Each configuration type has a unique category based on the different configuration types. The `collection` property defines the category of a document.

5.5.1.1 Wi-Fi

There is only one document in the wifi collection, which contains details of the current Wi-Fi configuration.

The properties in the wifi doc are (with validation rules in brackets):

- _id — the ID — set to 'wifi' (there should only be one doc)
- _rev — the version number

- ssid — network SSID (1–32 characters)
- encryption_type — the encryption used (must be WPA)
- password — the WPA passphrase (8–63 ASCII characters)
- mode — either G or N
- channel — 1 to 11
- status — either 'done', 'pending' or 'error'
- with_bss — optional — true is the default
- collection — set to 'wifi'

All fields are required. The `_rev` field is auto-generated by CouchDB. The `status` field is used to request that the Hostapd configuration file is updated (when set to pending) or to update the UI (done or error). All other fields represent configuration parameters in `hostapd.conf`.

Validation for Wi-Fi includes rules that ensure that the Hostapd configuration file will be valid. For multiple networks to function correctly, invalid encryption keys must be rejected by the router, to prompt users for the new passphrase. The presence of this rule is for purposes of usability. WEP does not reject invalid keys. The only indication that a key is incorrect is that devices will not connect.

The intention of creating a new network is to simplify the process of managing devices if changing network settings. If devices were not able to connect, any potential benefits of creating a new network would be negated if there were no errors.

Not supporting WEP simplifies validation because the WPA passhrase must be 8–63 ASCII characters. If allowing WEP, it would be necessary to verify the passphrase is correct depending on the format (5 or 13 ASCII characters for WEP with text passphrase, or 10 or 26 hexadecimal digits for WEP with hexadecimal key), or 8–63 ASCII characters for WPA. Supporting WEP also requires ensuring users are informed of the requirements for the currently selected options.

5.5.1.2 request_notification

This type of document is used to trigger sending a notification. It contains the following fields:

- _id — auto generated ID
- _rev — version number
- name — name of recipient or ‘everyone’
- service — what service (email, growl, text, twitter)
- message — the content of the message
- collection — set to ‘request_notification’
- status — ‘done’, ‘pending’ or ‘error’

All fields are required. Setting status to pending would trigger sending of the notification. If the user field was set to ‘everyone’, the details for all users using the specified service is requested, but if a name is provided, just that person’s details would be used. Details of how users are identified in each service are stored in separate documents, and retrieved by the notification system so that it only needs to be done in one place. The details of that data are described next.

5.5.1.3 notifications

These documents contain one username for one person for one notification service. The fields used are:

- _id — auto generated ID
- _rev — version number
- name — the name of the person
- service — service to use (email, growl, text, twitter)
- user — the username for the specified service

- collection — set to ‘notifications’
- status — set to ‘done’, ‘pending’, or ‘error’
- hidden — optional — if true removes document from views and filters
- suid — set by Google App Engine

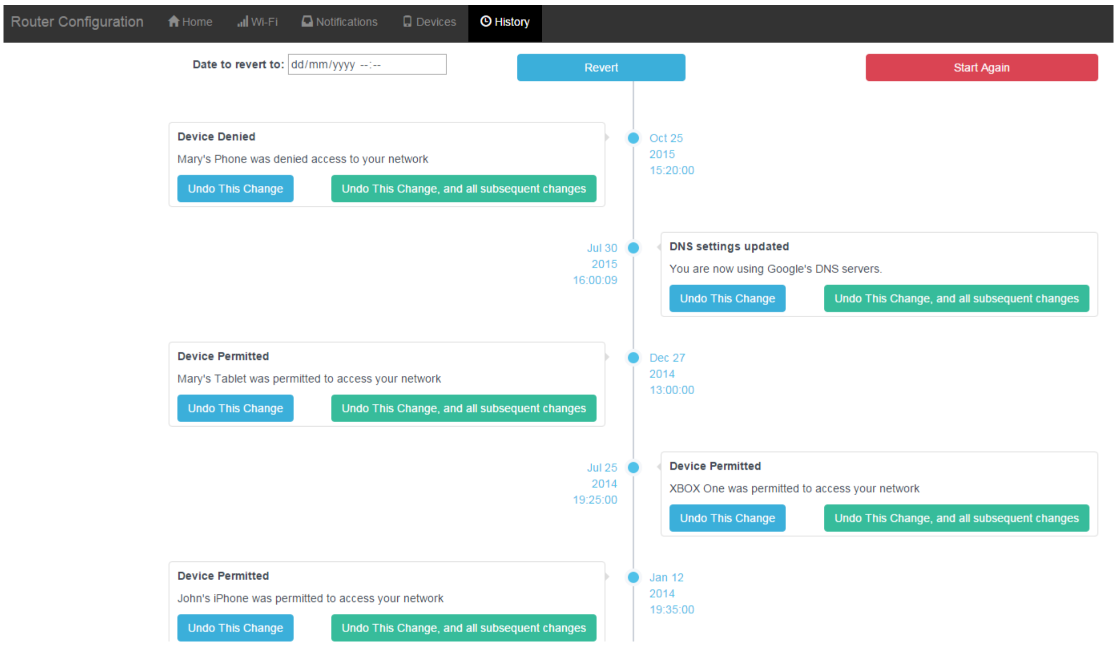
All fields other than `hidden` and `suid` are required. Changes to these documents are sent to the Google App Engine application that is used to send notifications if the status field is set to pending. If successful, the status field would be set to ‘done’, or ‘error’ otherwise. Sending details to the App Engine application allows it to verify that the specified user wanted to receive notifications from that router when a notification is sent.

Each record of such mappings in App Engine has a `suid` field, which can be used to access a particular record for edit or delete. The `suid` is the Service Use ID. Edit would generate a new `suid`, and deleting a record would remove the `suid`. The current `suid` generated for any given document is stored in the document to allow the record to be edited or deleted.

As part of the process of updating the Homework Router notification system to work with CouchDB, the App Engine App also received updates. Firstly, adding the ability to edit and delete notifications. Facebook and Push were also removed from the list of available services. Facebook does not allow applications to send messages to users, so the only option is to email them, and there is already an email option, so Facebook is redundant. Push notification is not supported because there is no need for a mobile application, which would be required to receive push notifications.

When the notification system is sending a notification, it searches documents in the notifications collection for a name that matches the recipient name and service and uses that to send a notification. If ‘name’ were set to ‘everyone’, the search would just use ‘service’ as the search criteria, and then remove duplicate names. The resulting list of unique user and service usernames would be used to send a notification.

This look-up of usernames means that other components that wish to send notifications can do so without needing to store the usernames. To do so would be



Device Denied
Mary's Phone was denied access to your network
Oct 25 2015 15:20:00

DNS settings updated
You are now using Google's DNS servers.
Jul 30 2015 16:00:09

Device Permitted
Mary's Tablet was permitted to access your network
Dec 27 2014 13:00:00

Device Permitted
XBOX One was permitted to access your network
Jul 25 2014 19:25:00

Device Permitted
John's iPhone was permitted to access your network
Jan 12 2014 19:35:00

FIGURE 5.6: History Timeline

unnecessary duplication that could also mean that users could have multiple usernames defined for the same service if the individual components were responsible for staying up-to-date.

The `hidden` field is used in the same way as the built-in `_deleted` field. The reason for using a custom field is to simplify finding documents that appear to have been deleted when undoing the deletion. It is not possible to open a document that has `_deleted` set to true without specifying its `_rev` property, and doing so means that the revision list cannot be accessed. The revision list is required for undo. Using a custom field that removes the document from the views and filters means that the document appears to be deleted, but it is still available to access revisions or undo.

5.5.1.4 events

These documents contain data for one event shown in the history timeline (such as that shown in figure 5.6).

The fields used are:

- _id—auto-generated ID
- _rev — version number

- collection — always set to events
- title — Overview of event
- description — more detailed description of the event
- timestamp — Date and time event occurred, in ISO8601 format, in UTC timezone
- docs — an array containing the modified document details, an array of objects with the structure:
  - doc_id — the _id field of the document that was modified
  - doc_rev — the version of the document that caused this event
  - doc_collection — the collection property from the document that caused the event
  - action — the type of modification that occurred (add, edit, delete)
- undoable — whether the change can be undone
- perform_undo — triggers the undo of the action that caused the event

These documents are used to provide data necessary to display a timeline of events, so that presentation data does not need to be added to documents containing configuration. Updating the timeline of events would be inefficient because every document would need to be analysed to determine the details of changes to present them to users.

Most events would be undoable, except for items that do not relate to user actions or that exist purely to inform the user that an event occurred. An example of such an event would be stating that the router was unable to connect to the Internet.

5.5.1.5 request_revert

These documents are used to signal that configuration should be reverted to the state it was in before the date specified. The fields required are:

- _id — Autogenerated ID
- _rev — version number

- collection — always request_revert
- status — should be done, pending or error
- timestamp — in ISO8601 format with UTC timezone

Creating a document in this collection with all the required fields and a status of pending would start the process of reverting all changes made after the date specified.

5.5.1.6 main_user

This single document is used to store the name of the person who will receive notifications regarding events that don't affect individual devices (notifications are sent to each device owner separately for these), and the service that should be used as the endpoint for notifications. When a new device connects to the network for the first time, this person will receive a notification to inform them of this. Removing a mapping is achieved by setting name and service to empty strings.

The fields required are:

- _id — must be set to main_user
- _rev — auto-generated version number
- status — must be set to done, pending or error
- name — the name of the user
- service — the service to use to send notifications

5.5.1.7 devices

This collection contains documents describing devices. Each device has a separate document. This document is created when a device connects to the network for the first time and requests an IP address. Once the DHCP server has assigned an address, it creates a new document with the following fields:

- _id—set to the device MAC address
- _rev — version number
- hostname — the hostname supplied the the DHCP discover packet
- ip_address — the IP address assigned to the device
- lease_action — commands used to signify state of DHCP lease, either add or del
- mac_address — the MAC address
- port — Which port the device is connected to
- name — the name of the person to whom this device belongs, or 'everyone' — set by users
- state — whether or not the device is allowed to access the network, (initially set to pending but can be changed to permit or deny)
- device_type — set by users and has the following possible values:
  - desktop
  - laptop
  - phone
  - tablet
  - other
- notification_service — where the person in name would like notifications that affect the device to be sent — set by users
- timestamp — updated whenever the lease is updated, value is seconds since midnight 1st January 1970
- collection — must be devices
- changed_by — whether the document was changed by DHCP, a user, or to update which devices are connected
- connection_event — either connect or disconnect, set whenever a device connects or disconnects and used to create a list of currently connected devices

- device_name — the name set by the user to identify this device
- action — set by users to signify a change in state to either permit or deny

Every time a device requests an IP Address, it is searched for in the database. If it is found, it is assigned the same address, and the timestamp is updated. The timestamp value is used for lease expiration. If it is a new device, a new document will be created, with user-defined fields set to empty strings. Changing the state field requires changing the action field. Doing so triggers a change to the OpenVSwitch routing system that then updates the state, and clears the action field to signify that the change was made correctly. Further details regarding devices will be provided in the discussion of the POX OpenFlow controller in Section 5.8.

5.5.2 Design Documents

Design documents are a special type of document that contains textual representations of all database search functionality. Documents can also have additional file attachments. The HTML, CSS and JavaScript files utilised in the UI are attached to a design document. Any file attached to a document can be accessed by a GET request to `document/attachment`. To aid organisation of both UI code and configuration data, the UI is stored in a separate database, which contains a single design document.

Search, filtering and validation are implemented as JavaScript functions, which are included as fields in the design document. Couchapp [91] facilitates writing these functions in separate files and having the design document generated from the contents of the files. All the files in the database directory in the source directory are combined into the design document, with subdirectories for each type such as views and filters.

5.5.2.1 validate_doc_update

This function contains validation rules to use when saving documents. The function accepts three parameters, the old document version, the new document version, and the user context object, which contains details of the user who saved the document, or null if no one is logged in.

All documents (other than design documents), must have a collection property, which is used both to group related documents for searches and validation of documents as appropriate for the collection.

Rules used for validation of each document type in validate_doc_update are specified in the document structure above.

Validation occurs each time a document is saved. If an error occurs, an error will be thrown, along with an appropriate HTTP error code (401 for unauthorised or 403 for forbidden). If the function returns without an error being thrown, the document is considered valid and is saved to the database.

5.5.2.2 Filters and the _changes API

The _changes feed contains every change to every document in the database. Filter Functions restrict the documents that are included in the _changes feed to the documents which match the filter criteria. Restricting the output in _changes means that changes for each type can be processed separately, and without having to test the included data to test for relevance to a particular process. The details of each process are described in Section 5.7.

All processes (except POX) use a permanent HTTP request, into which a valid JSON string is inserted per document change. The structure of this is shown in Listing 5.1[9]. For a deleted document, a `_deleted` field will be included and set to

```
{
    "seq": 10,
    "id": "wifi",
    "changes": [{
        "rev": "1-abcd"
    }]
}
```

Listing 5.1: Changes Feed example

true.

POX polls for changes once per second because a permanent HTTP connection interferes with packet handling. The structure of such a _changes feed is the same

[9]Documentation is available in the CouchDB Guide (http://guide.couchdb.org/draft/notifications.html)

except that each change line is included in an object with a results property with a value of all the changes that occurred. There is also a `last_seq` property in addition to the results array that is equal to the highest `seq` value from all the changes included. The `since` parameter is sent with the request to ignore all previous changes. For the continuous long-term changes, this is set when the process starts, and for POX, this is initialised in the same way but is updated after every request with the value of `last_seq`.

5.5.2.3 Filter Functions

This section lists all filter functions, their purpose and the criteria they use to decide if a document should be included in the _changes feed.

devices_pox The `devices_pox` filter is used to detect a change in device state so that the change can be applied to the OpenVSwitch routing system.

The definition of this filter is shown in listing 5.2:

```
function (doc, req) {
    return (doc.collection === 'devices'
    && doc.action !== ''
    && doc.action !== doc.state
    && doc.changed_by === 'user'
    && !doc.hasOwnProperty('hidden')
    && !doc.hasOwnProperty('_deleted'));
}
```

Listing 5.2: devices_pox

devices_ui The `devices_ui` filter is used to update the UI automatically when the state of a device changes, typically after a user has requested a change, and the routing system has been updated.

The definition of this filter is shown in listing 5.3.

When the state of a device is changed, `action` is set to an empty string, and state is updated. The check for both is required to prevent the _changes feed from being populated by the same requests repeatedly.

```
function(doc, req){
    return (doc.collection === 'devices'
    && doc.action === ''
    && doc.changed_by === 'system'
    && !doc.hasOwnProperty('hidden')
    && !doc.hasOwnProperty('_deleted'));
}
```

Listing 5.3: devices_ui

edit_devices The `edit_devices` filter is used to monitor changes to devices that do not alter the state, but that do change other metadata, allowing an event to be created. The _changes feed is used for this because using the UI to manage events means that business logic is mixed with presentation details, and also means that the same logic would be duplicated if there was more than one interface, for example, mobile applications and a website.

The filter definition is shown in listing 5.4.

```
function(doc, req){
    return (doc.collection === 'devices'
    && doc.action === ''
    && doc.changed_by === 'user'
    && !doc.hasOwnProperty('_deleted'));
}
```

Listing 5.4: edit_devices

history The `history` filter is used to update the UI whenever new events occur. Without this, it would be necessary to refresh the event timeline manually if a new event occurred because a previous change was undone. The definition of this filter is shown in listing 5.5.

```
function(doc, req){
    return (doc.collection === 'events');
}
```

Listing 5.5: history

main_user The `main_user` filter is used to create events when the main user is edited.

The filter definition is shown in listing 5.6:

```
function (doc, req) {
    return (doc.collection === 'main_user'
    && doc.status === 'pending');
}
```

Listing 5.6: main_user

notification_request The `notification_request` filter is used to trigger sending a notification. Notification request documents define the recipient and service to use to send the notification. The creation of the document will trigger the notification.

The filter definition is shown in listing 5.7:

```
function(doc, req){
    return (doc.collection === 'request_notification'
    && doc.status == 'pending'
    && !doc.hasOwnProperty('hidden'));
}
```

Listing 5.7: notification_request

notifications The `notifications` filter is used for changes to documents in the notifications collection, facilitating the mapping of notification changes to the Google App Engine Application used for sending notifications. It also creates an event document for each change. The criteria used is shown in listing 5.8:

```
function(doc, req){
    return (doc.collection === 'notifications'
    && doc.status === 'pending');
}
```

Listing 5.8: notifications

notifications_ui The `notifications_ui` filter is used to update the UI when notification documents are updated. Google App engine updates documents by setting a `suid` field. The UI must show the most recent version of a document at all times, to avoid conflicts when saving changes. The value of this field is just used internally, and never presented in the UI. The criteria used is shown in listing 5.9:

```
function(doc, req){
    return (doc.collection === 'notifications'
    && !doc.hasOwnProperty('hidden'));
}
```

LISTING 5.9: notification_ui

revert The `revert` filter is used to trigger the rollback of configuration to the date provided. The criteria used is shown in figure 5.10:

```
function(doc, req){
    return(doc.collection === 'request_revert'
    && doc.status === 'pending');
}
```

LISTING 5.10: revert

undo The `undo` filter is used to trigger the undo of a configuration change, as defined in an event. The criteria used is shown in listing 5.11:

```
function(doc, req){
    return (doc.collection === 'events'
    && doc.perform_undo === true
    && doc.undoable === true);
}
```

LISTING 5.11: undo

wifi The `wifi` filter is used to trigger a change to the Hostapd configuration file, which would result in new Wi-Fi configuration. The criteria used is shown in listing 5.12:

wifi_ui The `wifi_ui` filter is used to keep the UI updated when Wi-Fi configuration changes, so that any errors can be displayed. The criteria used is shown in listing 5.13:

```
function(doc, req){
    return (doc.collection === 'wifi'
    && doc.status !== 'pending'
    && !doc.hasOwnProperty('hidden'));
}
```

Listing 5.12: wifi

```
function(doc, req){
    return(doc.collection === 'wifi'
    && !doc.hasOwnProperty('hidden'));
}
```

Listing 5.13: wifi_ui

5.5.2.4 Views

The CouchDB term for MapReduce is `view`. Each map function iterates over all documents in the database, and the map criteria are used to decide whether or not to include the document in the results. If a document matches the criteria, it will be included in the result, as a key-value pair, as defined by the `emit` function.

Each document that matches the map criteria will be an object in a results array. The results array is itself part of an object that also contains the number of documents that match the view criteria, and how far through the result set the current rows are (this offset allows for pagination, which is not used.)

The individual result rows are an object containing the document's `_id` field, and both a key and value as defined by the map function.

Reduce functions are not defined, because the documents are required, rather than summaries.

The `key` field orders results, followed by the document's `_id` field. The key field can also be used to filter the set of matching rows. The DHCP server specifies the Media Access Control (MAC) address of the device making a DHCP request, to find its IP address (if available). If the view does not return any rows, the device is new and needs an IP address. Otherwise, the DHCP server assigns the device its previous IP address.

The `notification_with_service` view only returns the `user` property because the `name` and `service` are already known. In all other cases, the full document is returned by the `emit` function of the view, because all fields are needed.

Other query parameters can be requested (see the views chapter of the CouchDB guide [92]), that can affect the view result. Specifying `key` reduces the view result to only those documents where the `key` matches the provided criteria. `descending=true` will output the results in reverse order. `start_key` and `end_key` can be used to specify a range of possible values. Specific details any parameters used will be included in the discussion of each view below.

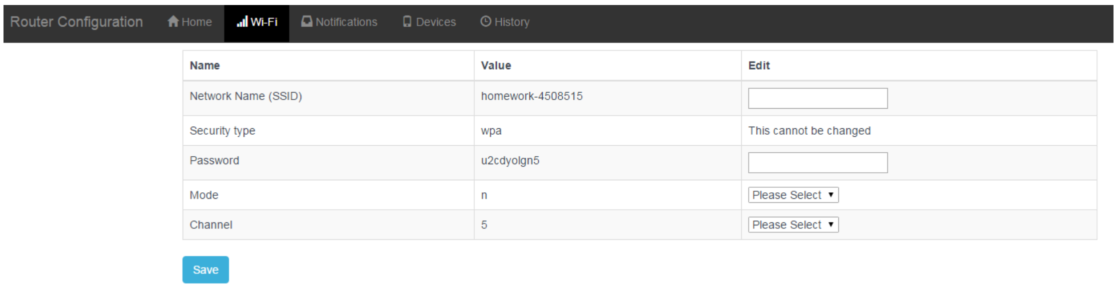
byCollection This view returns all documents in a collection and is used to request data to display in the UI. Specifically, it is used to generate the Wi-Fi screen and the user details table on the notifications screen.

The view is defined in Listing 5.14. The Wi-Fi screen is shown in Figure 5.7, and

```
function(doc){
    if(doc.collection && doc.hasOwnProperty('hidden') === false){
        emit(doc.collection, doc);
    }
}
```

LISTING 5.14: byCollection view

the user details table is shown in Figure 5.8.

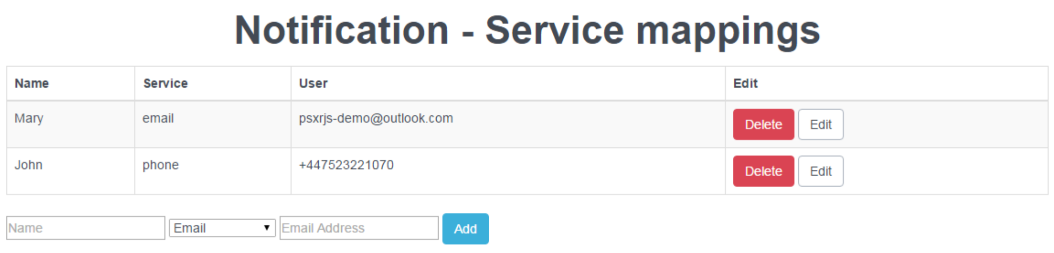


Name	Value	Edit
Network Name (SSID)	homework-4508515	
Security type	wpa	This cannot be changed
Password	u2cdyolgn5	
Mode	n	Please Select ▾
Channel	5	Please Select ▾

FIGURE 5.7: Wi-Fi Screen example

connected_devices This view requests a list of connected devices.

It is used to display a list of connected devices on the home screen, shown in Figure 5.9.



Notification - Service mappings

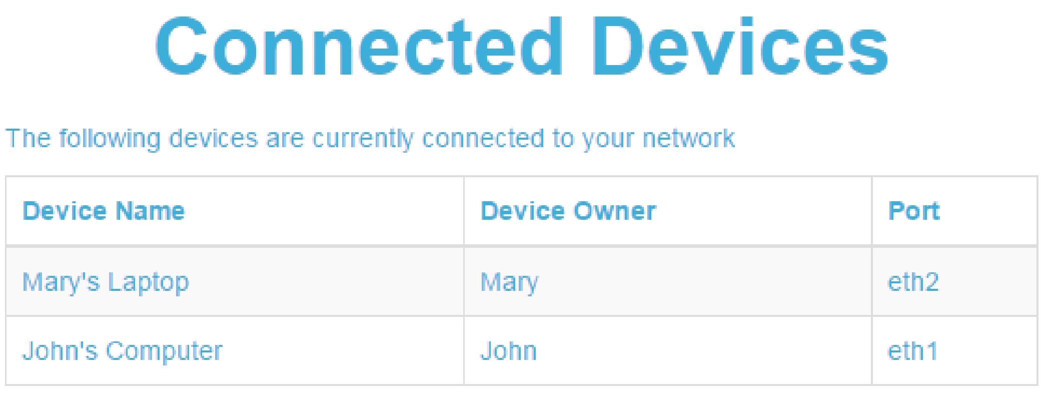
Name	Service	User	Edit
Mary	email	psxrjs-demo@outlook.com	Delete Edit
John	phone	+447523221070	Delete Edit

FIGURE 5.8: Notification user details table

The view function is shown in Listing 5.15.

```
function(doc){
    if(doc.collection === "devices" && doc.connection_event === "connect"){
        emit(doc.mac_address, doc);
    }
}
```

LISTING 5.15: connected_devices view



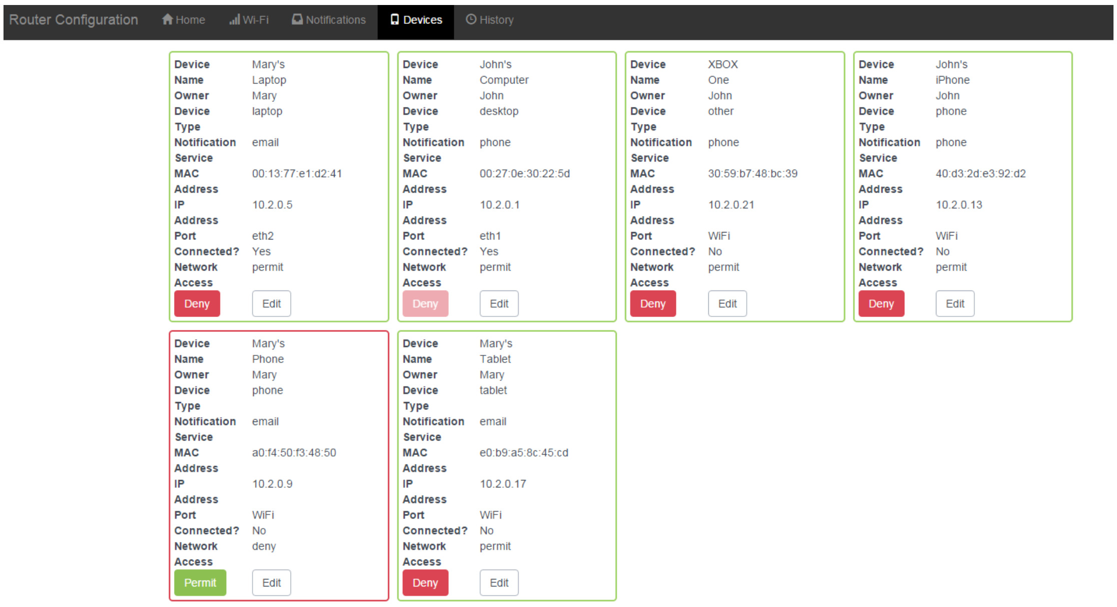
Connected Devices

The following devices are currently connected to your network

Device Name	Device Owner	Port
Mary's Laptop	Mary	eth2
John's Computer	John	eth1

FIGURE 5.9: Presentation of connected_devices view

connected_devices_for_notification This view is used by the notification system to find connected permitted devices to send notifications for events which will affect devices (such as changing Wi-Fi configuration. The criteria used is similar to the `connected_devices` view, with the addition of checking the state property is set to ‘permit’. Pending devices do not have notification recipients defined so cannot receive notifications. However, by limiting the list of connected devices to



Device Name	Mary's Laptop	John's Computer	XBOX One	John's iPhone	Mary's Phone	Mary's Tablet
Owner	Mary	John	John	John	Mary	Mary
Device Type	laptop	desktop	other	phone	phone	tablet
Notification Service	email	phone	phone	phone	email	email
MAC Address	00:13:77:e1:d2:41	00:27:0e:30:22:5d	30:59:b7:48:bc:39	40:d3:2d:e3:92:d2	a0:f4:50:f3:48:50	e0:b9:a5:8c:45:cd
IP Address	10.2.0.5	10.2.0.1	10.2.0.21	10.2.0.13	10.2.0.9	10.2.0.17
Port	eth2	eth1	WiFi	WiFi	WiFi	WiFi
Connected?	Yes	Yes	No	No	No	No
Network Access	permit	permit	permit	permit	deny	permit
	Deny Edit	Deny Edit	Deny Edit	Deny Edit	Permit Edit	Deny Edit

FIGURE 5.10: Device Configuration

those which are permitted, there is no need for the notification system to search for notification details which are known to not exist.

connected_via_wifi This view is also used to obtain a list of connected devices to send notifications to, but specifically for changes to Wi-Fi configuration. Because Wi-Fi changes do not affect devices connected via Ethernet, sending notifications to owners of these devices is not necessary. The additional criteria required is that the port must be set to "wlan0*".

control This view returns all documents in the `devices` collection and is used to populate the devices screen, (shown in Figure 5.10). The view criteria is shown in Listing 5.16. Using `connection_event` as the key will group all connected devices together.

```
function(doc){
   if(doc.collection === "devices"){
     emit(doc.connection_event, doc);
  }
}
```

LISTING 5.16: control view

device_notification_service_mapping This view requests a list of devices that have a notification service specified (specifying a service to use to receive network notifications for a device is optional). The view is defined in Listing 5.17.

```
function(doc){
    if(doc.collection==="devices" && doc.notification_service !== ""){
        emit(doc.mac_address,
        {name: doc.name,
        service: doc.notification_service});
    }
}
```

LISTING 5.17: device_notification_service_mapping

dhcp The DHCP server uses this view to request the details for a specific device. The view is shown in Listing 5.18.

```
function(doc) {
  if(doc.collection === "devices"){
    emit(doc.mac_address, doc);
  }
}
```

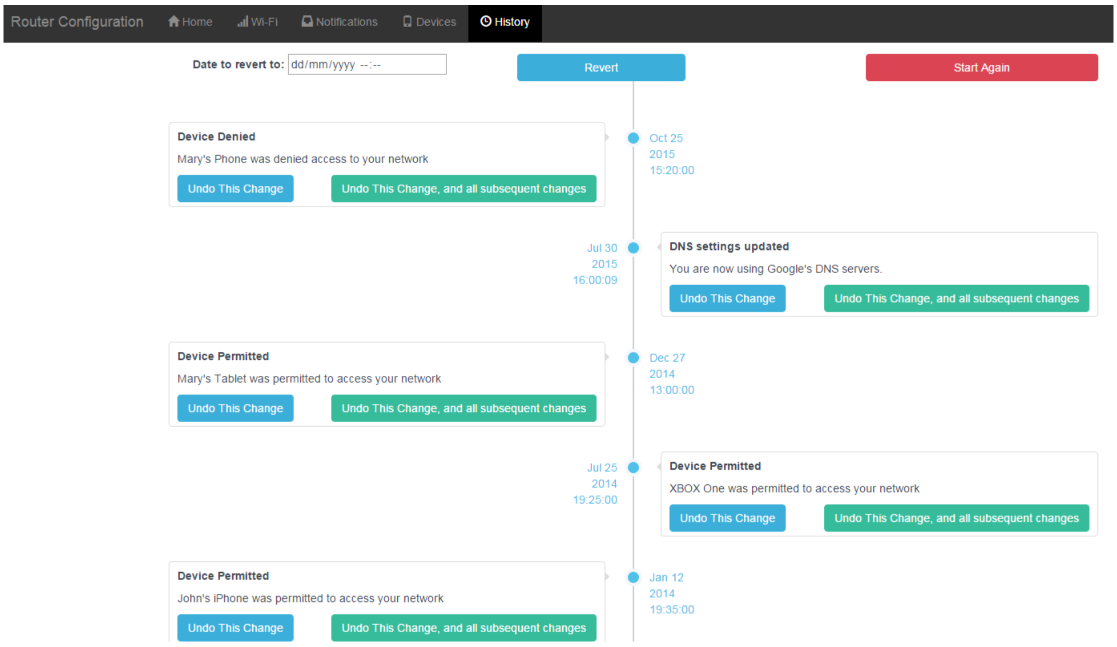
LISTING 5.18: dhcp view

events This view returns documents in the events collection, with the key set to the timestamp so that events are in chronological order. The query parameter `descending=true` orders events reverse chronologically.

This view is used for the history screen. Figure 5.11 illustrates how the results of this view are presented and the view definition is shown in Listing 5.19.

```
function(doc){
    if(doc.collection === "events"){
        emit(doc.timestamp, doc);
    }
}
```

LISTING 5.19: events view



Device Denied
Mary's Phone was denied access to your network
Oct 25 2015 15:20:00

DNS settings updated
You are now using Google's DNS servers.
Jul 30 2015 16:00:09

Device Permitted
Mary's Tablet was permitted to access your network
Dec 27 2014 13:00:00

Device Permitted
XBOX One was permitted to access your network
Jul 25 2014 19:25:00

Device Permitted
John's iPhone was permitted to access your network
Jan 12 2014 19:35:00

FIGURE 5.11: History Screen

mac_from_ip This view is used in the timeline screen in the UI (Figure 5.11). The view finds the MAC address which is assigned the provided IP address. In the timeline, any event which relates to the current device MAC address is marked as not undoable. It is not sensible to allow the current device from being denied, because the UI would break on the device. The view is defined in Listing 5.20.

```
function (doc) {
    if (doc.collection === "devices") {
        emit(doc.ip_address, {'mac_address': doc.mac_address});
    }
}
```

LISTING 5.20: mac_from_ip view

notification_with_service This view returns documents in the notifications collection that are not hidden. The notification system uses this view to look up usernames for services by recipient name. The value for this view is the username. The key is a compound key containing both the service and the name. All documents can be returned if no key is specified. To search for a specific user and specific service, `start_key` and `end_key` should be provided, both with the same values. The owner of a device can be 'everyone'. This requires that notifications are sent to all users, and requires a wildcard name. CouchDB wildcards use an empty string as the start_key and {} as the end_key. The view is defined in Listing 5.21.

```
function(doc){
    if(doc.collection === "notifications" && doc.hasOwnProperty('hidden') === false){
        emit([doc.service, doc.name], doc.user);
    }
}
```

LISTING 5.21: notification_with_service view

Ports This view is used when detecting the connected devices. ethtool can be used to determine which Ethernet ports are in use. This view is used to find the device currently connected to a port. The mapping between a device and a port is created by the DHCP server when a new DHCP request is received. The view definition is shown in Listing 5.22.

```
function(doc) {
  if(doc.collection === "devices"){
    emit(doc.port, doc.mac_address);
  }
}
```

LISTING 5.22: ports view

undoable_events This view lists events that can be undone. Being able to request a list of undoable events simplifies the process of reverting configuration to an earlier time because there is no need to check if a change could be undone when processing the revert. The view is defined in Listing 5.23.

```
function(doc){
    if(doc.collection === "events" && doc.undoable === true){
        emit(doc.timestamp, doc);
    }
}
```

LISTING 5.23: undoable_events view

wlan0 This returns documents representing devices connected to either `wlan0` or `wlan0_1`. `iw dev wlan0 station dump` will list all devices currently connected via Wi-Fi. Devices that are not connected are not listed. Without a list of all known Wi-Fi devices, there is no way of identifying devices which have disconnected since the last check. The view is defined in Listing 5.24.

```
function(doc){
    if(doc.collection === "devices" && (doc.port === "wlan0" || doc.port === "wl
        emit(doc.mac_address, doc);
    }
}
```

Listing 5.24: wlan0 view

5.6 Configuration History

All documents in CouchDB are versioned. Storing each configuration item in a separate document means that every item will have a built-in local history.

The user-facing global history requires metadata such as a title or timestamp. There are two options for building a global history display:

1. Include metadata in each document and build the global history on demand
2. Create a separate document to represent a single entry in the global history and reference the entry in the appropriate local history

The first option has simpler data storage requirements but does not scale. First, the global history will take longer to generate as the history expands. Second, the metadata must be updated when the document is originally changed to prevent the creation of additional items in the local history. Third, non-undoable changes should not be included in the global history.

The second option addresses the scaling problems with the first approach, at the expense of requiring more data storage. Creating an additional document to display change metadata makes it possible to display the global history from the results of a MapReduce query. The data required to display the global history is also not required in the local histories. If a new document is not created for a change, it will not be included in the global history.

The result is a series of histories (one for each configuration item) and a global history. The global history has pointers to specific locations in the other history, containing references to both the _id of the document and the current _rev of the document. The individual histories can be traversed independently.

5.7 Document Change Processors

User changes are performed in the UI, which triggers an update to the database. It is then necessary to synchronise the router configuration with the database to ensure that the active configuration matches what is defined in CouchDB. This functionality is provided by Python scripts.

Most continuously monitor the _changes feed, with an appropriate filter, to restrict the documents they see, but ethtool_monitor and iw_monitor detect when devices connect and disconnect and update the corresponding CouchDB document accordingly.

All modules make use of the couchdbkit package [44], as a convenience wrapper around CouchDB data requests. CouchDB has an HTTP API, and all data is transmitted and stored as JSON, so it is possible to interact with CouchDB without an extra package. Using couchdbkit removes the need to request all the different resource types, and manually create modules that can parse the data obtained from the database.

Any components that need to refer to the _changes feed consist of two modules, which are discussed together below. The runner component subscribes to the continuous _changes feed using two threads and a queue. One thread reacts to changes and adds them to the queue. The other thread pulls the change out of the queue and processes it.

Two threads are used so that multiple changes of the same type can occur near simultaneously, without processing of a change risking changes being missed.

Processing requires opening the changed document and passing the changed document to a function in the other module where the change will be applied. Using separate components means that components have the option of interacting with each other directly, without needing to rely on the _changes feed. The main place this is used is when processing a request to undo an event. In this situation, it is necessary to both update the document content and apply the new content to the configuration. However, it is more efficient to save the document with a status of done, (so the changes feed is not triggered), and apply the change directly. In the case of documents in the notifications collection, for example, the document content can be changed, and the function that would apply the change to App Engine can be called directly. Importing modules that listen to _changes requires infinite loops, which makes importing them difficult.

The components edit_device and edit_main_user are used to create event documents as required. Details of such an occasion are provided below. The _changes feed is used for this because the only other way to create the events would be for the process that made the initial change to create the event. In practice, this means that the UI must perform this action. It is not considered appropriate for the UI to require knowledge of such activities because doing so would mean duplicating the requests for every possible way of interacting with the database. The benefit of CouchDB using HTTP for its API is that it can be used anywhere HTTP can. The downside to this is that it is not possible to control what is used to interact with CouchDB, and it may be the case that a change was made, without the corresponding event being created. Events are a crucial component of the system, so it is not considered appropriate to make creating events optional, just because a different HTTP client was used.

All of these components are run using the upstart [4, 5] system in Ubuntu, which simplifies process management. Each process needs to run in the background (so that all can execute at the same time), but each creates two threads that each starts an infinite loop. Stopping the process requires either creating a PID file on startup and killing the process from that or using `ps` to find the PID to kill. Creating a PID file is not optimal because it must also be deleted when a process is killed. Using upstart makes it possible to start each process on startup and stop it at shutdown. It also makes it easy to start and stop processes manually. Using upstart also means that logging for a process involves nothing more than outputting to stdout. Any such output is logged automatically.

5.7.1 add_history_item

This module contains a convenience function used to create new documents in the events collection. It does not respond to the _changes feed but is used by other components that do. It maps the function parameters to document fields and creates a new document based on those fields.

The parameters are:

- title —used as the title of the event in the timeline
- description—the event description
- undoable—if the event is undoable—optional, defaults to true
- ts—optional, defaults to None—if supplied uses the timestamp, otherwise uses current time
- docs—an array of modified documents

The docs array is composed of dictionaries with the following structure:

- doc_id—the modified document _id
- doc_rev—the modified document _rev
- doc_collection—the modified document collection
- action—how the document was modified—'add', 'edit', 'delete'

In most cases, there will be a one-one mapping of an event to a document, so the docs array will only contain one document. The only current exception to this is the event for reverting, which lists all documents that were updated.

5.7.2 change_notification

This module contains a function that creates a request_notification document from the supplied parameters. It is used by other components to trigger sending notifications, and it does not respond to the _changes feed.

The parameters used are:

- to — the recipient of the message
- service — which service to use to send the message
- message — the message to send

5.7.3 couchdb_config_parser

This module returns a reference to the configuration CouchDB database. It does not make use of the_changes feed but is used by all components that need to read or write to the database.

It uses the Python built-in ConfigParser module and the ConfigParser class. The ConfigParser is initialised with CouchDB default configuration, of no administrator username or password (known as admin party), and running on port 5984. The database name defaults to "config".

Assuming `$HOME/couchdb.conf` exists and is readable, the actual configuration used will be read from this file. The differences are supplying administrator credentials, and using a different port, currently 8000. `$HOME` is an environment variable that represents the home directory of the user running the script.

Once the configuration file is parsed, a connection to the CouchDB server is established and the database loaded. This instance is returned. This module does not create a singleton because having multiple HTTP connections to the same database does not cause an issue. If it were the case that multiple sources attempted to update the same document at the same time, CouchDB has a mechanism for detecting conflicting updates, and any attempt to update a document the _rev is not the most recent version is an error, and HTTP status code 409, Conflict, would be returned. As such, any document update that could result in conflict passes the `force_update` parameter with a value of true to the couchdbkit document save function. The most recent _rev will be fetched before making the save, making the chance of conflict smaller.

5.7.4 edit_device and edit_device_runner

These two related modules are responsible for creating an event any time a user changes the metadata for a device, but where the state is not changed. An example of this would be changing the owner of a device, or the notification service used. This component is responsible for creating an event, to keep this centralised.

5.7.5 edit_main_user and edit_main_user_runner

This component creates events for changes to the main_user document. It exists to centralise the process of creating events.

5.7.6 edit_wifi and edit_wifi_runner

These two modules are responsible for applying changes in Wi-Fi configuration from CouchDB to the Hostapd configuration file.

The entire document is treated as one entire piece of configuration. As such, every field in the document must be valid for the document to be saved and have it included in the _changes feed. By ensuring that the content of the document is valid, it makes it possible to ensure correctness in the Hostapd configuration file.

The configuration file is read to obtain current values. Many of the configuration parameters do not change, and so rewriting the entire file every time is inefficient. It is also necessary to know the original values for SSID and wpa_passphrase, as a new BSS is used to supply the new values.

Channel and mode are only defined once, and so are updated by splitting the file into a list of keys and a list of values. The keys are searched for the index of the string 'channel'. The corresponding index in the values list is updated. The mode is always set to 'g'. If `80211n=1` is present, the mode will be 802.11n instead of 802.11g. This value is added or removed as appropriate.

If an additional Basic Service Set (BSS) is defined, it is removed from the keys and the values, so that it can be updated. At this point, both lists are converted back into a list of strings in the form key=value.

The new SSID and wpa_passphrase are defined as a new BSS, and the lines are written to the file.

It is also possible to update the configuration without using a new BSS, as is the case when the router is initially configured with an initial configuration. In this situation, the configuration file would not have existing values, so would need to be created, and a BSS would not be required. Using a new BSS is the default option.

After updating the configuration file, an up-to-date list of connected devices is obtained using the `connected_devices_for_notification` view, (so that only permitted devices are included) and notifications are sent to device owners to inform them of setting changes. At this point, Hostapd is restarted to bring the new configuration into effect. An event is created, which will be undoable in all cases except the initial configuration setting. If the configuration was created using a BSS, the advanced Python scheduler package's Background scheduler is used to schedule the removal of the BSS in 24 hours.

5.7.7 ethtool_monitor

This process is responsible for maintaining the list of devices connected via a router's Ethernet ports. The `ethtool` command is polled for every Ethernet port in the router except `eth0`, which is the gateway port.

Each time `ethtool` is run it outputs a large quantity of data for the specified port. The only relevant line in the output is the "Link detected" line. If this is "yes", then there is a device connected to an Ethernet cable on this port. The ports database view is used to identify the device connected to the port. The OpenVSwitch flow used when a device sends a `DHCPdiscover` packet includes the port on which the device was connected, and this data is stored in the device document when the DHCP lease is assigned. Without this, detecting what device was connected to each port would be difficult. Once the device is identified, its document's connection_event field is updated, unless the value is already correct. There is no need to update the document every time `ethtool` is polled (approximately once per second), only when the state changes. Also, device connection and disconnection is considered an event and recorded as connected and disconnected. The event that occurred most recently for a device determines whether or not the device is connected.

The first time a device connects, this process will recognise that, and attempt to update its `connection_event` property. However, there won't be a document until after the DHCP server has assigned the lease, so this is not possible. However, the DHCP server will set the connection_event field whenever a lease is assigned, renewed or expires so would have the same result.

It would be preferable to monitor the system logs to identify when an Ethernet link is connected or disconnected because it would not require regular polling, but this information is only available reliably for the gateway.

5.7.8 iw_monitor

This script has the same purpose as ethtool_monitor, but for devices connected via Wi-Fi.

The command, `iw dev wlan0 station dump` will list all devices currently connected to wlan0. If it exists, the same test is performed with wlan0_1, to ensure the list is accurate.

Unlike `ethtool`, the output from `iw` will contain, amongst other items, the MAC address of every device connected to the corresponding port. There is no need to look up the MAC address for devices associated with every port but makes it harder to identify which devices have disconnected. The purpose of the wlan0 database view is to identify all wireless devices that are currently known to the router, regardless of whether or not they are connected. This script creates a list of MAC addresses that are not currently connected from the difference between the view output and the output of iw. At this point, the process updating device documents is the same as for ethtool_monitor, except that each list is processed separately.

As with ethtool_monitor, it is necessary to rely on polling rather than monitoring system logs, because device disconnections are not logged in all situations. One example of a situation of a disconnect would not be logged is if Wi-Fi is switched off on a device.

5.7.9 notification_registration_client and notification_registration_client_runner

These modules are responsible for mapping changes in documents from the notifications collection into records in the Google App Engine Application used to send notifications. App Engine will not send notifications if it does not recognise the provided username for the service used on the router sending the notification.

The App Engine App provides `register`, `edit` and `delete` endpoints. Register sends both a service and a username in the POST request. The new `suid` (unique ID for the registration) is returned and included in POST requests to edit and delete. Edit also requires the new username and will generate a new `SUID`, which replaces the original `SUID`. Delete is only passed the `SUID`, and, because the data is deleted, the `SUID` should be removed from the CouchDB document too.

Event documents are also created for each change, making the change undoable.

5.7.10 perform_rollback and perform_rollback_runner

This process acts on requests to revert configuration to an earlier time. It collects all events after the specified timestamp and builds a list of documents changed by these events. This process starts with the oldest change and progresses to the most recent. The doc_id field is used to find modified documents. The list of documents is implemented as an OrderedDictionary, with the document_id as the key, and the event as the value. Events are only added to the dictionary if the doc_id is not present. The result will be a dictionary of events that contain the oldest version of every document changed after the timestamp. Events that are not undoable are not included.

This dictionary is then iterated over, and the event document has `perform_undo` set to true, to pass the event to the undo process. Undoing the most recent change after a timestamp will result in each document being restored to the version in use when the timestamp occurred.

5.7.11 remove_vlan

This process is scheduled to run one day after Wi-Fi configuration is updated, assuming a BSS was used to change the configuration. This script updates the Hostapd configuration file and sets the original SSID and wpa_passphrase values to the values defined for the new BSS. The BSS line and all lines following it are removed. The result of this is that the old network is no longer available, and the new configuration is the only configuration.

5.7.12 Notification System

The Notification System is a group of modules that work together to send notifications via the App Engine Application. This system is a port of the notification system from the Homework project. The original system used HWDB to request notifications and the requirements of this system influenced the implementation of the system as presented here. It has been ported from Java to Python because Python has more robust support for CouchDB, with many Java CouchDB wrappers either only supporting an old CouchDB version, an old Java version (or both), meaning that it would have been necessary to write a custom CouchDB library.

The original version also supported push notifications and Facebook as notification endpoints. These are no longer used because of a lack of support by the service providers.

Google App Engine is used because it makes it possible to avoid storing application credentials and OAuth tokens on the router.

5.7.12.1 notification_service_runner

This component connects to the _changes feed with the notification_request filter and coordinates sending notifications.

`notification_request` documents have a name, a service and the message to send. The script first uses the notification_with_service view to lookup the corresponding username if there is one. If so, the notification will be sent to all matching usernames.

The supported services are Email, Growl, SMS (referred to as phone internally, but displayed as 'text message' to users) and Twitter direct messages.

Each service has different requirements for sending notifications, and different criteria for determining if the notification was sent correctly. The rules for each are defined in separate modules. These are dynamically imported and have the same name as the internal representation of the service in CouchDB. Each module has a `sendNotification` function that is passed the document `_id`, the username and the message. The document `_id` field is used to as an identifier for the notification being sent.

Using separate modules that each implement the same function makes it very easy to change which services can be used without having to modify existing code. To support a new service, all that is required is to create a new module that implements the `sendNotification` method. The App Engine application would also need to be updated to support the new service.

5.7.12.2 Notification Service Modules

Email Emails are sent using App Engine, and the only verification of success is whether the App Engine app processed the request successfully, and is the simplest to implement.

Growl Growl is a notification system for OS X or Windows. It supports the ability to send notifications to other devices over the network. It needs the destination address and is only available on the local network. Growl notifications are sent from the router directly to the device, with App Engine just used to log that the notification was sent.

SMS This service uses Tropo[10] to send SMS messages. The response from Tropo is stored in App Engine, which responds with an ID. This ID is then used to request the result of the notification.

Twitter Direct Message This service works in the same manner as sending SMS messages, especially with regards to how status is checked. It is also necessary for users to follow the `@HomeworkRouter` Twitter account so it can send them direct messages. The Homework Router did not provide a UI for managing notifications, so there was no way for users to authorise their account to receive messages from their router.

remote_notify This component is a wrapper around the urllib Python functionality. It is in a separate module so that there is less code duplication, as it can just be imported and used.

[10] https://www.tropo.com/

notification_response Twitter and SMS both require a separate response object, which needs to be parsed to determine the status of the request to send the notification.

5.8 POX OpenVSwitch controller

Using OpenVSwitch provides the possibility to add history to configuration of devices and how the network is used. The functionality that OpenVSwitch provides is determined by the controller used.

The Homework research project used OpenVSwitch with a “NOX classic” (see [2, 81, 114] for more information and usage related to Homework) controller written in C++. This router has the same features from that implementation. However, it was necessary to make changes to NOX to use CouchDB instead of HWDB. NOX-Classic does not compile on Ubuntu 12.04 or newer. To enable installation of packages, Ubuntu 12.04 or newer is required.

POX is a Python implementation of the newer NOX. The library support for CouchDB in Python has more features than those for C++. The Python couchdbkit library wraps CouchDB functions in Python functions and also handles converting between JSON and Python datastores. In C++, the only library listed in CouchDB documentation (pillow talk), is simply a cURL wrapper, which lacks some more advanced features. It is because of this disparity between libraries, combined with the fact that I have a greater knowledge of Python that POX is considered more appropriate than using the newer version of NOX in C++. Another advantage of Python during development is that it is not compiled. Due to the way NOX is structured, it is necessary to recompile NOX every time a component is changed. Due to the complexity of NOX, and combined with the limitations of the EeePC hardware, this would result in significant amounts of time spent waiting for the compilation.

The underlying functionality of these components does not need modification. The only changes necessary are those required to support storing changes to devices in a database. The details of these changes are discussed later. The only other difference other than how the data is stored, is the interface to POX, compared to the original NOX interface. The NOX implementation used both HWDB and a web service as interfaces to modify devices. Because the web service is an implementation detail and not itself a part of the configuration, it is considered unnecessary to port from NOX to POX.

Using CouchDB also requires modification to the original NOX components to support using CouchDB as a datastore. One area where functionality was changed (instead of a direct port) was how devices are permitted/denied. CouchDB is used as a replacement of the Homework webservice which runs the Homework device management interface.

The three components are:

- homework—dhcp
- homework—routing
- homework

The names used are the same as those used originally. The following is a brief description of the components. It is included because it helps to explain the design of other parts of the system, especially any component which manages device configuration.

5.8.1 homework-dhcp

This component is the DHCP server. DNSMasq is used for DNS requests, but the DHCP server is disabled. The DHCP server creates addresses in the 10.2.0.x/16 subnet, but once an address has been identified as available, it is changed to 10.2.0.x/30. IP addresses are assigned such that there is a gap between addresses large enough to ensure that each device is on a separate Subnet. When a lease is assigned, the router IP address is defined as the next address after the device requesting an IP address. For example, if a device were assigned 10.2.0.1, the DHCP reply would identify the router as 10.2.0.2. The next device would receive an IP address of 10.2.0.5. Using separate subnets means that all data must pass through the router, even if it is destined for another device on the same network.

DHCP lease information is stored in CouchDB documents, one per device, in the Devices collection. When a device requests an IP address, the database is searched for a document for that device. If one exists, it reuses the originally assigned IP address; otherwise, it finds the next available IP address and creates a new document accordingly.

To reduce the number of places that the database is read from and written to, the DHCP server also keeps the document updated whenever the homework-routing component requests a state change.

5.8.2 homework-routing

This component identifies different packet types and from OpenVSwitch flows and modifies the flow with the appropriate action.

The main criteria for handling flows is whether or not devices are permitted. If a packet was sent from or to a device on the network, the MAC address is checked to see if it is in the list of permitted devices. If so, it processes the flow in the way most appropriate for the type of packet. Otherwise, the flow is removed, and packets are dropped.

5.8.3 homework

This component listens for changes to data in CouchDB broadcast in the _changes feed using the `devices/pox` filter. When a change matching the filter is found, it is included in the results when the feed is polled. Unlike other processes that listen to _changes, this cannot use the continuous feed and must use regular polling. The feed is checked once per second because the permanent HTTP connection prevents POX from handling packets, due to threading problems. When a change is detected a POX event is triggered to pass the request to homework-routing to update the state lists and corresponding CouchDB document.

5.9 Undo

The ability to undo changes is a key feature of this router. The undo procedure was outlined in Section 4.5.3. This section presents a method of implementing the design from Section 4.5.3.

Undoing a change to a document will result in another document version. The undo command will be self-applicable.

Each configuration type requires bespoke processing to determine the previous value of the configuration. It is not sufficient to find the previous version of a document to undo a configuration change. There are different undo implementations for each configuration type.

Every document type requires a list of previous revisions to be able to undo changes. There is a base document which provides this function, with Python subclasses to extend this function to satisfy document specific requirements for undoing.

The Undo processor uses the Python dynamic import system to load the appropriate handler for the configuration type being undone. The classes used for the specific implementations are all stored in the same directory and are named after the collection, with the first letter capitalised.

5.9.1 Devices

Devices can be changed by the system (usually the DHCP server), or by users. The revision list is first filtered to remove all system changes. The result is a list of previous user changes.

The undo procedure varies depending on if there are previous revisions available. If there are previous revisions, the existing document has `device_name`, `device_type`, `name`, and `action` fields set to the values from the first available revision. The `changed_by` field is set to 'user'. The other fields will only be set by system processes so are not changed. The document will be saved as the most recent version. The result will be that the user-editable options for a device will be undone.

If there are no previous revisions available, the device was new to the network. In this situation, the undo procedure is to delete the document, which will mean that the device would be treated as a new device if it were reconnected.

5.9.2 Wi-Fi

The Wi-Fi document has a status field. When undoing changes to Wi-Fi it makes sense to choose from successful changes when selecting previous versions (otherwise the undo command would break the configuration). The Wi-Fi specific undo command filters the list of previous revisions to remove versions where the status is `pending` or `error`. The first in the list (the most recent) is used as the previous version. The undo system does check whether there is an additional network active. If there is, the selected version of the document is used, but explicitly set to not use a BSS. The presence of a BSS in the version of the document to be undone means that the change was made within the last day. If the change is to be undone, the old settings would still be active, so would not need a new BSS to switch to them. If the undo happened when the old network was removed, the undo process would behave in the same way as a user request to change the configuration and create a new network with the updated settings (even though the old settings were the previous version).

5.9.3 Notifications

Notification documents store mappings between users, services and their identification on that service. These mappings can be created, edited or deleted. Each action needs to be undone differently. When users remove these documents they are not deleted, they are just excluded from views by the presence of the `hidden` field. Deleting a document from CouchDB would be a difficult change to undo because the undo process requires the list of revisions of a document. The revision list is not accessible for deleted documents. It is necessary to know the previous version and request that version explicitly.

Notifications are managed entirely by CouchDB, and the Notification System uses the same documents when sending notifications. As such, actually removing the documents is not necessary. Excluding the documents from the views which generate the UI and control the Notification System has the same result as actually deleting the document, but in a way which is easy to undo.

Undoing a ‘deletion’ can be achieved by removing the `hidden` field from the document. Undoing a creation, can be undone by adding the `hidden` field.

Undoing an edit requires accessing the previous version with a status of `done`. The previous version would then be saved as the new version (so that it can be undone).

5.10 Revert

The ability to revert to the state at an arbitrary point in time is another important feature of this router, and a natural extension of an Undo function. The revert algorithm uses timestamps to trigger the undo system for multiple events at once.

There are two methods for triggering a revert. First, use the revert button on an event, or specify another timestamp.

There is no difference to how the algorithm functions. The only difference is how the timestamp used for the revert is selected. Reverting an event triggers the revert with the timestamp from the event document. Manually selecting the timestamp provides greater flexibility as to when changes can be undone.

The revert algorithm is shown in Algorithm 5.1.

Algorithm 5.1 Revert Algorithm

```
$history \leftarrow$ SEARCHDB(timestamp)
$collection \leftarrow \{\}$
for all $item$ in $history$ do
    if  then$item$ not in $collection$ $collection \leftarrow item$
    end if
end forREVERSE($collection$)
for all $item$ in $collection$ do UNDO(item)
end for
```

The implementation of revert has an optimisation for situations where there are multiple changes to the same document to undo. Only the oldest change (since the timestamp) to undo will be undone. Listing 5.25 shows how the list of documents (and which versions) are to be undone.

The value returned from the function in Listing 5.25 will be a list of event documents which contains at most one of each other document type.

Once the list of events to include in the revert has been generated, the events are undone in the same way as triggering the undo function directly.

```
def get_docs_to_revert(self, timestamp):
        #get list of documents which could be undone, oldest first
        events = self.get_events_after_timestamp(timestamp)
        #Hashtable with known order of keys
        doc_list = OrderedDict()
        for event_val in events:
            #The value field in the view result
            event = event_val['value']
            for ed_doc in event['docs']:
                if ed_doc['doc_id'] not in doc_list:
                #add to list of changes to undo if
                #this is the first change to a document
                    doc_list[ed_doc['doc_id']] = event
        pprint.pprint(doc_list)
        return doc_list
```

Listing 5.25: Revert Document Selection

5.11 Network Transition

It is also important to not require devices to disconnect and be reconfigured. This section explores how this can be achieved using Hostapd, and the multiple BSS feature.

5.11.1 Hostapd

Hostapd is an application which allows a Wi-Fi device to operate as an access point. The network configuration is defined in the application configuration file. Changing configuration is achieved by modifying the configuration file and either restarting Hostapd or issuing a `SIGHUP` signal.

5.11.2 Basic Service Set (BSS)

Generating a new network can be achieved by adding a `bss` section to the configuration file, which has its own `ssid` and security parameters.

5.11.3 Validation

There are rules for defining the fields in the Hostapd configuration file. SSIDs should be at most 32 ASCII characters long, and cannot be empty. Security parameters depend on the type of security used. WEP passphrases can be either 64 or 128 bit long. They can be specified as ASCII or hexadecimal. If using ASCII passphrases can be either 5 or 13 characters. If using hexadecimal, they must be 10 or 26 digits long. WPA passwords must be 8–63 characters long. Valid values for the radio mode or channel depend on the facilities of the device and the country of operation.

Hostapd will fail to start if the configuration file is invalid. When using CouchDB to change the Wi-Fi configuration, the `validate_doc_update` function will ensure that document changes adhere to the rules. If there is a problem, the update will not be saved and the update to the configuration file will not be triggered.

5.11.4 OpenVSwitch

Creating a new BSS creates a new virtual Network Interface Card (NIC) which must be added to the OpenVSwitch managed switch. Hostapd does not support OpenVSwitch so cannot dynamically create new virtual NICs. Support for OpenVSwitch must be added to Hostapd by modifying the source code to use OpenVSwitch commands instead of Linux Bridge commands.

5.12 Summary

To build change history and undo functionality into any system, it is important to consider which data storage mechanism will be the best fit for the requirements of the system. Network configuration consists of many items which must be independent of all others. As such, it is useful to be able to maintain separate histories for each configuration item, as well as a global history which links to specific entries in local histories. This will result in the history appearing to be linear, but with support for many independent configuration items.

Undo algorithms which alter network configuration should be restricted to operate within the history of a specific configuration item.

Care must also be taken to accurately define the meaning of a valid previous version to use as the basis of the result of the undo function. In this router, many configuration items have status flags. Because configuration is tracked separately from how it is applied (for example Hostapd uses CouchDB and a configuration file), it is important to select a previous version which has a ‘pending’ state, but which is *not* the most recent ‘pending’ version. Selecting a known broken version will cause problems, and selecting a version which has been applied will mean that the undo will have no effect.

Some configuration types have some data which is undoable and some which is not. Devices are an example. User changes can be undone, but DHCP leases cannot be undone. To undo a change to a device, the undo function must select the appropriate previous version, and create a new version which combines the current DHCP lease with the previous user-provided data.

Reverting to an earlier point in time requires the same consideration as with undoing an individual change. However, there is the added complexity of needing to determine the configuration items which must have changes undone. In addition, if any configuration item had multiple undoable changes in the period to undo, only the oldest change should be undone. If this optimisation is not applied, there would be unnecessary undo operations required.

Converting a Wi-Fi antenna to an access point in Linux can be achieved using Hostapd. It is possible to define multiple networks using the same adapter, with different SSID or security configuration. This ability can be exploited to create a new network when a user wishes to make changes to their Wi-Fi network. Users could then have a small period to reconfigure all their devices before the old settings would be disabled.

The next chapter describes the experiment designed to evaluate the usability and usefulness of both multiple networks, and configuration history and undoing.

Chapter 6

Experiment

6.1 Introduction

The features of the router described in this thesis have been defined based on literature[1] that leads to the theory that such features would be useful additions to a router.

The next stage is to study the usefulness of History as a troubleshooting tool and both Undo and Revert as recovery tools.

6.2 Experimentation Methods

Two main methods could be used to evaluate the usefulness of configuration history. The first is a deployment in homes, and the second is a lab-based study. There are positives and negatives to each approach. As such, it is useful to consider these in the context of testing the usefulness of the new features in the router presented in this thesis.

[1]see Chapter 2 and Chapter 3

6.2.1 Home Deployments

The biggest advantage to deploying routers in homes is that the usage patterns are real. The new router features could potentially be used to fix actual network problems.

However, networks are often configured and utilised in a manner which means they are a part of a home's infrastructure. Interaction with infrastructure only occurs to manage devices or to fix problems [53]. There is no guarantee that users would be in a position to need to interact with the router at any point, other than to connect new devices. The features used would also depend on the characteristics of the particular network, which would mean that different houses could each use a distinct subset of these, leaving little to compare between individual deployments. Not being able to compare directly would also make it harder to draw conclusions from the data obtained.

Another negative of deployment in homes is that to get the full benefit; the deployment would have to be over a very long period (at least six months, but preferably a year, if not longer) to generate enough history for occupants to find them useful. One potential use for configuration history is to remind people what they changed in the past that could be causing problems. The potential usefulness decreases for recent changes, because the more recent an event, the higher the ease of recall, mainly because there would be a reduced chance that other changes would occur in the intervening period.

One method of addressing some of these issues is by providing users with a set of configuration changes to make, or challenges to solve. The problem with doing this is that the forced interaction might not make sense for all networks. It would be necessary to ensure the prescribed tasks are possible on all the networks, to both ensure that the test will work and to ensure that the presented challenge will not inadvertently cause other issues with incompatibility between devices. Asking participants to perform a series of predefined tasks means running a lab experiment in participant's homes.

6.2.2 Lab-based experiment

The main advantage of a lab-based experiment is that it is possible to control all aspects of the experiment. It is possible to ensure each participant does the same tasks, which means that there is a point of comparison between participants. It also means that the devices used in the experiment can be chosen specifically to ensure interoperability. Another benefit is that it is possible to plan the tasks such that users would be required to use specific features to perform the tests.

Another advantage of a lab-based experiment is that the time necessary for the experiment will be condensed. It should be possible to collect enough data for analysis within a timeframe measured in days, and the time for an individual participant would be no longer than an hour.

A downside to a lab-based study is that it is hard to emulate precisely the conditions that would be found in a home. A home is a particular environment, the very fabric of which can affect the network and the way it is structured or how it performs. In this respect, a controlled, lab-based experiment is not realistic.

Another potential challenge is making sure that for each particular experiment appears as if the router had been in use for an extended period. With only one router and a single set of devices available, it is necessary to start each experiment from a known state but to do so in a manner that ensures that the router appears to have been running successfully for an extended period.

6.2.3 The selected option

After considering the pros and cons of both home deployment and a lab-based experiment, a lab-based experiment is the best choice. An important deciding factor is the length of time required. Planning and running a deployment would require a period of study far longer than that which would be available.

Asking users to commit to a small amount of time is much more feasible, and also increases the potential for an increased number of participants.

The ability to control an experiment means that the issues faced are ones that have been defined by the tasks presented, and not by issues of incompatible devices or software, as could be the case for home deployment.

6.3 Scenario

This section describes a fictitious network belonging to John and Mary. They live alone. Their network is used to enable their devices to share an Internet connection. The devices they use, how they are identified, as well as device type and how they connect to the network, is shown in table 6.1.

TABLE 6.1: Experiment Devices

Device Name	Device Identifier	Type	Port
John's Computer	Ballard	Desktop	eth1
Mary's Laptop	Callison	Laptop	eth2
Mary's Phone	mrldesx2	Android Phone	wlan0
Mary's Tablet	mrltablet6	Android Tablet	wlan0
John's iPhone	mrliphone1	iPhone 3GS	wlan0
XBOX One	XBOX One	Games Console	wlan0
Unknown Phone	rsxperia1	Android Phone	wlan0

The device name is the user-assignable device name. The device identifier is the label on the device. There is a separate labelling system to provide users with the Device Name values, which are intended to be more user-friendly. Devices with system identifications 'mrltablet6', 'mrldesx2' and 'XBOX One' have predefined hostnames (which is how devices are identified before being labelled by users). The other devices are using the defined identifiers as the hostname.

The purpose of 'rsxperia1' is to emulate the scenario where a new, unrecognised device connects to the network. As such, users will not be shown this device.

All devices are pre-configured with the router's default settings. The router has a scripted history to emulate the scenario that John and Mary obtained their devices over a period, starting in January 2014. It emulates initial router setup, the configuration of devices, and configuration of the notification system. It also includes other events relevant to the experiment, such as mrldesx2 being denied access to the network, to prevent it from connecting, and setting the scene for the first two tasks.

6.4 Notifications

The experiment makes use of the router notification system to alert users of new devices, and that network configuration has changed. Notification settings are preconfigured to use an email address and phone number created specifically for the experiment. Mary's tablet has been configured with the experiment email account.

6.5 Tasks

As discussed in Chapter 2, the features of the router were derived from findings in previous research. As well as influencing the features and the software design of the router, this previous research also facilitated the definition of a hypothesis as to how the system might be useful. The assumption is that the router might help troubleshoot problems, either by users directly, or provide them with enough details to allow a source of technical support to troubleshoot the issue.

The tasks that form the experiment have been chosen both to provide a realistic simulation of configuration changes and the types of problems that may occur as a result of these changes, as well as being able to determine if the hypothesis given above is correct.

The tasks provided here are meant to simulate the types of changes that could be made to the configuration of an actual network in a home environment. The tasks chosen range from simple changes, such as change Wi-Fi or permit a device, to more complex changes. Changes to DNS or firewall configuration are not changes that would be expected in households where there is a low level of networking knowledge. They are, however, tasks that might be found in homes with at least one occupant with a higher level of knowledge. Such changes may cause issues when the person who made them were not present, creating a situation where other occupants could not use the network correctly, until either the change initiator was available, or rely on remote support.

The data that lead to the router design would suggest that the most complex tasks, such as DNS or firewall changes, would not be solvable by inexperienced users on most existing routers ([178] suggests that technical detail is confusing for users, regardless of ability). For changes such as these, it is believed that a list of configuration changes should be enough to allow users of any ability to correct the problems.

Participants chosen for this experiment should have a range of knowledge, to test for any differences in the potential usability of this router with different levels of networking experience. It is expected that users with more experience would be able to use the system to remind them of what was changed when, even if they could fix issues without undoing changes or rolling back. However, less experienced users would need the prompt of being able to undo changes to fix problems.

The second task presents a challenge because the router provides two ways of identifying and fixing the problem. Either the device management screen or the timeline can be used to complete the task.

6.5.1 Task 1

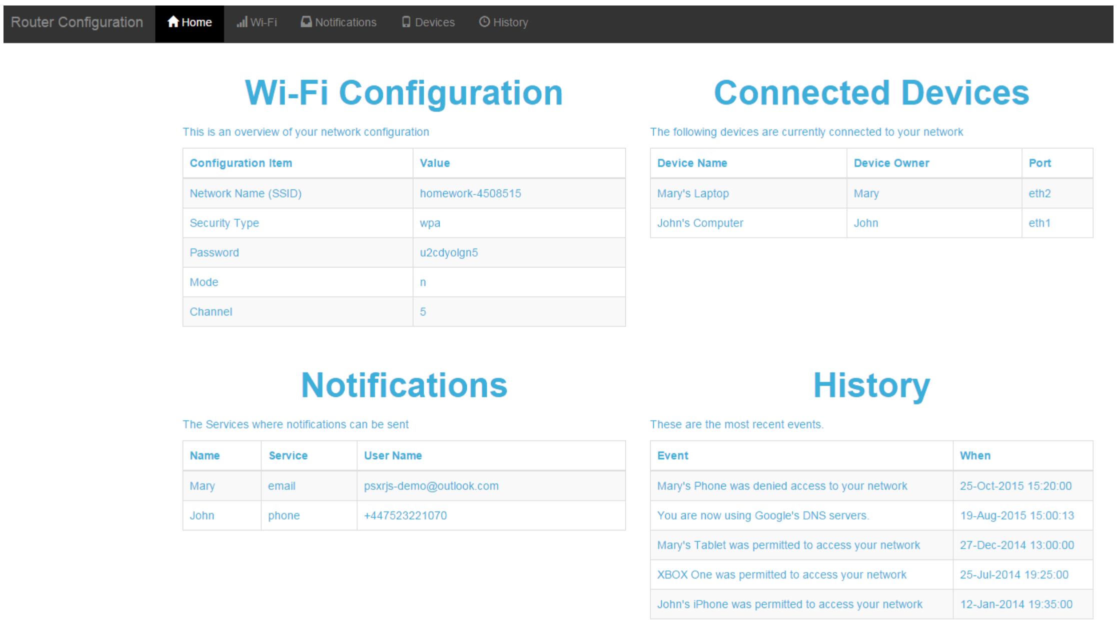
The first task is a familiarisation task. All devices will be connected, except Mary's phone. They should verify that all devices are connected, and have Internet access. In doing this, they should be able to identify that Mary's phone is not connected.

Users can identify connection status in three different ways. Firstly they can check individual devices to see if the device is connected to the network and possibly confirm that it can access the Internet. The second option is to look at the screen shown in figure 6.1.

The panel on the top-right lists each connected device, with basic details, including how it is connected. The final option is to use the screen that is presented when the panel on the right is clicked on, shown in figure 6.2

6.5.2 Task 2

The second task is to determine why Mary's phone cannot connect and correct the problem. The scripted history finishes in September 2015 when Mary's phone is denied access to the network. Being denied is what prevents mrldesx2 from connecting to the network because it is added to the Hostapd blacklist, so its association will be rejected.



Wi-Fi Configuration

This is an overview of your network configuration

Configuration Item	Value
Network Name (SSID)	homework-4508515
Security Type	wpa
Password	u2cdyolgn5
Mode	n
Channel	5

Connected Devices

The following devices are currently connected to your network

Device Name	Device Owner	Port
Mary's Laptop	Mary	eth2
John's Computer	John	eth1

Notifications

The Services where notifications can be sent

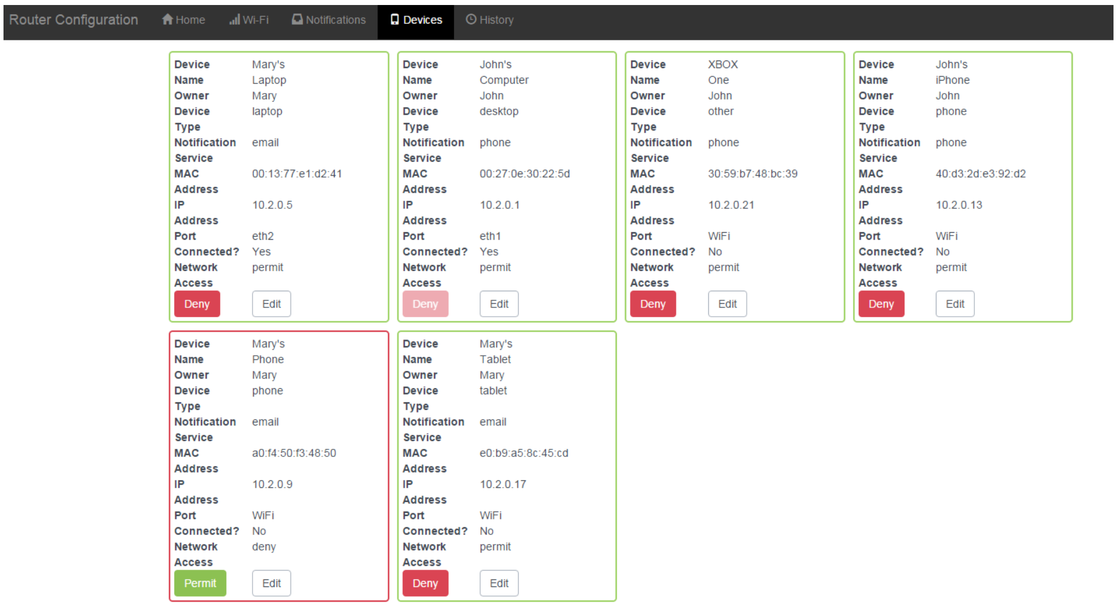
Name	Service	User Name
Mary	email	psxrjs-demo@outlook.com
John	phone	+447523221070

History

These are the most recent events.

Event	When
Mary's Phone was denied access to your network	25-Oct-2015 15:20:00
You are now using Google's DNS servers.	19-Aug-2015 15:00:13
Mary's Tablet was permitted to access your network	27-Dec-2014 13:00:00
XBOX One was permitted to access your network	25-Jul-2014 19:25:00
John's iPhone was permitted to access your network	12-Jan-2014 19:35:00

FIGURE 6.1: Home Screen with Connected Devices List



Device Name	Mary's Laptop	John's Computer	XBOX One	John's iPhone	Mary's Phone	Mary's Tablet
Owner	Mary	John	John	John	Mary	Mary
Device Type	laptop	desktop	other	phone	phone	tablet
Notification Service	email	phone	phone	phone	email	email
MAC Address	00:13:77:e1:d2:41	00:27:0e:30:22:5d	30:59:b7:48:bc:39	40:d3:2d:e3:92:d2	a0:f4:50:f3:48:50	e0:b9:a5:8c:45:cd
IP Address	10.2.0.5	10.2.0.1	10.2.0.21	10.2.0.13	10.2.0.9	10.2.0.17
Port	eth2	eth1	WiFi	WiFi	WiFi	WiFi
Connected?	Yes	Yes	No	No	No	No
Network Access	permit	permit	permit	permit	deny	permit

FIGURE 6.2: The device screen shows which devices are connected, and whether they are permitted or denied.

On the screen illustrated in figure 6.2, mrldesx2 will have a network connection state of denied, and will have a red outline.

6.5.3 Task 3 (Hidden from Users)

At the start of the experiment, users will have been informed that at some point during the experiment, a text message will be sent to John's iPhone to inform them that a new device has connected to the network. Users will be required to decide the most appropriate action. The expected response, given the features of the router, and the notification sent, is that this additional device would be blocked.

6.5.4 Task 4

The addition of a new device on the network suggests that the network is insecure and that the network configuration will be changed.

As such, the next task is to change the network configuration from the defaults to something more secure. Doing so will trigger the multiple network feature, and allow all devices to reconnect to the network with original settings. Each device will trigger a notification to either John or Mary, which will be either a text message or an email. This notification serves to inform users that they need to update the configuration settings within 24 hours.

6.5.5 Task 5

After this, there will be a scripted change to the data that injects an event that corresponds to a change of Domain Name System (DNS) settings. This scenario is one that could happen in homes where the person who is responsible for network maintenance has is knowledgeable about networking technologies. The rationale behind this task is that when such a change occurred, an inexperienced user would have to contact the person who made the change, and requests that they fix it. It is assumed that providing people with information that a change to DNS occurred in the past, together with an event stating that there is an Internet connection problem caused by a DNS error, should be enough to suggest to people that the earlier change was the one that caused the error. The purpose of this task is to test the validity of this assumption.

6.5.6 Task 6

The final task is designed to force users to revert the settings to the way they were before the experiment started.

Participants will be asked to revert configuration to before Mary's Tablet was permitted to use the network, which can be achieved by clicking the Revert button on the "Mary's Tablet" permit event. As a result of the event timestamps used, reverting events after that point will remove Mary's Tablet and rjsxperia1. It will change the state of Mary's Phone back to deny, and reset the Wi-Fi configuration to the default values.

6.6 Initial Room Setup

When users enter the area used to run the experiment, they will see a desk or table on which all the devices used will be arranged, on sheets of paper showing their user-defined name. The only exceptions will be John's Computer, which will have the name label attached, and rjsxperia1, which does not have a label, and will not be on the table.

The router management system will be running on both Ballard and Callison. They will both be running Google Chrome. Callison will have a USB mouse attached, to reduce the risk of problems being identified that are caused by the use of the trackpad. All other devices will also be switched on, and most will be connected to the network.

A labelled illustration of this is shown in figure 6.3.

6.7 Observations

Two methods will be used to collect data regarding the usability and usefulness of the interfaces provided. First, participants will be interviewed after they complete

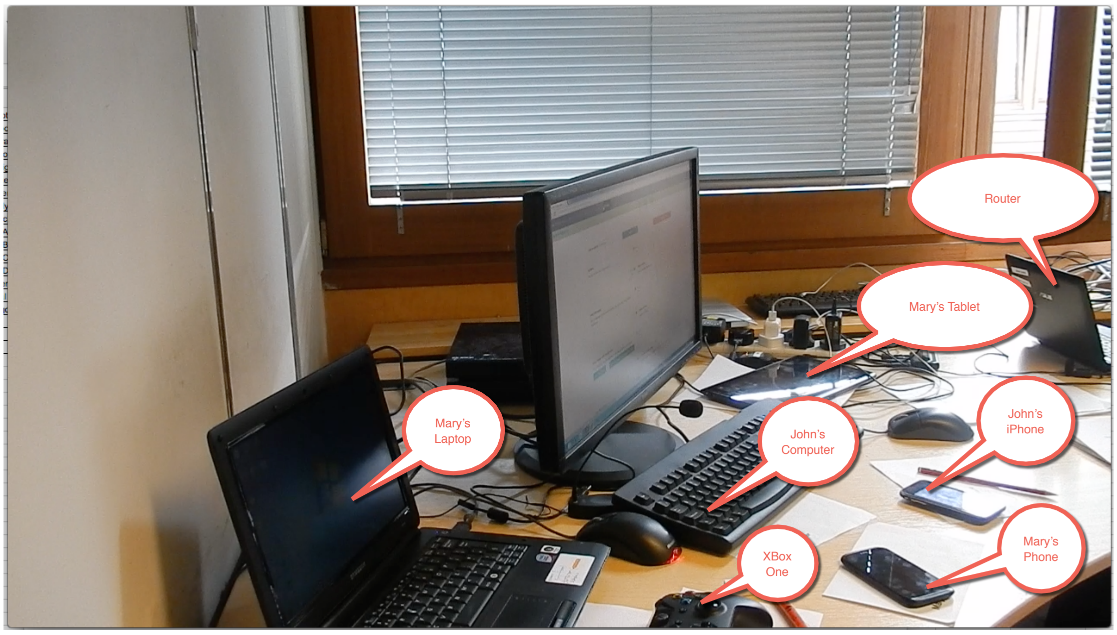


Figure 6.3: Layout of the experiment

the required tasks. Interviews allow collection of user's perceptions of the system and the usefulness of interfaces provided.

The experiment will also be recorded, which will make it possible to identify issues surrounding usability of the system, and any areas where there is a discrepancy between interview answers and actual usage.

Camtasia will be used to record the screens of both Ballard and Callison. These recordings will provide a detailed view of how users interact with the interface and show any difficulty in performing the required tasks.

Part of the experiment requires users to interact with other devices. It is necessary to film users as they perform the requested activities to capture this interaction, and how they react to anything that they experience. Due to the positioning of each device, this will require at least two cameras.

Filming the entire experiment will facilitate keeping track of questions asked by participants during the experiment, or any hints given. It will also allow differentiation between challenges using the router, and challenges when setting up devices.

A further benefit of both filming the experiment and recording the screens of Ballard and Callison is that it will highlight usability issues which users do not necessarily notice. It may be that users do not see issues with one component, but the recording shows that they do have difficulty completing the task.

6.8 Questionnaire

Participants were asked to complete the System Usability Scale [27] to get an overview of the usability of the router. The System Usability Scale allows easy identification of the *presence* of usability problems (but not the causes). The scale can be used to gain an initial insight as to the usability of the router.

6.9 User Experience

One area for comparison is the effect of networking knowledge on the ability to complete the tasks.

The hypothesis on which this experiment is based is, in part, related to knowledge of users. It is expected that the use of the router would reduce the difficulty of completing tasks, to the extent that knowledge would not be a factor.

To compare task performance between participants using this measure it is necessary to identify the knowledge level of each participant. Participants will be asked questions which relate to their use of the network and configuration management in particular. The questions asked will relate to their experience in the domains of access control, network configuration, device management, and troubleshooting. These topics have been chosen because the router has features which aim to assist in these areas.

Participants will also be asked to draw a diagram of their home network. It is expected that those with a higher level of knowledge would provide a more detailed drawing. Also, participants who are primarily responsible for their network are likely to have a greater understanding of the network compared to others in the household, in part because the network infrastructure is invisible (because it is hidden, or because of a lack of wires [79]). It is therefore expected that whether or not participants manage their network will influence their ability to perform the tasks in the experiment.

It is necessary to rely on questions because the configuration options available on this router are those which enable other features. For example, Wi-Fi changes are tracked, because of the method used to change the network configuration. Device changes are tracked because device management is a central component of the Homework Router, on which this router is based. The features of this router are unique, and it is not possible to compare its ease of use (for these tasks) to that of other routers because the features of this router are not directly comparable to other routers.

If the router had feature parity with other routers, such that the only difference between this router and another was the addition of both configuration history and multiple networks, it would be possible to ask users to complete the tasks on both routers and compare the results.

Instead, comparisons are made between participants' performance when completing the tasks, and participants' knowledge levels will be considered to determine if the hypothesis of the research is correct.

Another factor in analysing knowledge level is that of the types of devices with which they are familiar. By asking users to identify the devices with which they are familiar, consideration can be given to potential issues caused by a lack of familiarity with a platform or device.

The questions that participants are asked are listed in Section 6.10. The rationale behind each question is also provided.

6.10 Interview

The system usability scale can show that usability problems exist, but not identify the cause of the problem. The scale is also incapable of identifying what is an actual problem, versus problems perceived to exist by participants, which are the result of mismatched expectations.

Also, there are some potential issues with the design and functionality of the router's user interface. One aim of the experiment is to identify whether they are issues, or if they are not considered significant by participants. It may also be the case that participants identify other areas that have the potential to improve the router.

Interviewing participants is one way of addressing the shortcomings of the system usability scale, particularly in the case of identifying problems with usability. Open-ended questions facilitate identifying *why* participants had problems using the router to complete the tasks, and what changes they feel would aid them in completing the required tasks.

6.10.1 Interview Questions

6.10.1.1 Preliminary Questions

It is useful to determine participants' backgrounds and existing level of technical knowledge.

1. Who do you live with / what kind of accommodation?

   This question is to determine whether participants live alone, with other students, or other people who are not students, or live in other types of accommodation (such as a hall of residence, in which case, they are asked about their non-term address).

2. Do you have a network at home?

   Knowing this will give a clear indication of lack of experience if they do not have a network at home.

3. Are you responsible for maintaining your network? If so they may have needed to perform some of the tasks presented in the experiment.

4. Can you draw a diagram of the devices on your network and how they are connected?

   Users who manage their network would be expected to have a higher level of knowledge than those who do not configure any aspect of their networks.

5. If you needed to change your network's name or passphrase, how would you do that? Are there other tasks that you would need to perform? If so, what are they?

   The purpose of this question is to identify if users could change SSID or passphrase, *and* if they know that all Wi-Fi devices would be disconnected and would need to be reconnected using the new details.

6. How do you control which devices can connect to your network, and when? How would you prevent an unwanted device from connecting?

   The answers to each of these questions are not likely to be technical, especially for those participants who lack technical knowledge.

   The answer to this question also relies on the features available in a participants router, and on the participant's knowledge that they exist, and how to use them.

7. How would you troubleshoot the problem of a device not being able to connect to your network, assuming that you want it to be able to connect?

   The purpose of this question is to establish a baseline for the participant's ability to identify that Mary's phone has been denied access to the network.

   It also allows for comparison with the information provided in different commercial routers.

8. How would you approach the problem of a lack of Internet connection?

   This question will depend on both the knowledge of the participant and on the information provided either on the router or via other means (such as technical support).

9. This network uses an XBOX One, an iPhone, 2 Windows 7 machines, and 2 Android phones. Which of these are you familiar with?

   The purpose of this is to be aware of any potential difficulties caused by participants not being familiar with a device.

6.10.1.2 Follow-up Questions

These questions are intended to identify participants' thoughts concerning the usability and usefulness of the features being studied.

1. Do you think that a new network would simplify the process of managing devices when network settings are updated?
2. Is there anything regarding making Wi-Fi changes which was confusing?
3. Did you find configuration history helpful for finding/troubleshooting problems?

4. Were there any specific aspects of the router you liked or found useful?
5. Were there any particular aspects you did not like or didn't find useful?
6. Was there anything presented by the router which was unclear?
7. Based on the information provided, was there anything that you expected to happen which did not happen?
8. Following on from the previous question, was there anything that you were not expecting to happen?
9. Is there anything that you feel would help you complete tasks?

6.11 Participants

One participant is a member of staff at the University of Nottingham. All other participants are students, also at the university. The study was advertised by placing posters at various locations around the university.

Two participants indicated that they live in university halls of residence. The networks in these properties are centrally managed, so would not require any configuration by the participants. These participants referred to the network in their non-term address. In both cases, this means their parents' home.

Details of the participants in the study are shown in Table 6.2. Level of technical knowledge is expressed relative to other participants in the study and was based on the preliminary interview, network diagram and relative level of confidence in completing tasks.

Table 6.2: Participants

	Home Network?	Admin?	Technical Knowledge	Household
P1	Yes	Yes	High	Shared
P2	Yes	Yes	Low	Shared
P3	Yes	No	Low	Shared
P4	Yes	Yes	Low	Shared
P5	Yes	Shared	High	Shared
P6	Yes	Yes	Medium	Shared

P7	Yes	Yes	Medium	Shared
P8	Yes	Yes	Medium	Parents
P9	Yes	Yes	Low	Parents
P10	Yes	Yes	Medium	Shared
P11	Yes	Yes	Medium	Shared

None of the participants in the study has a high level of technical knowledge (although all are highly educated). P1 and P5 were more knowledgeable than other participants. Although it is true that a greater range of technical knowledge (for networking specifically) would provide a broader range of comparison, there is a range of technical knowledge. Having no highly knowledgeable network users is a preferable situation to only having knowledgeable participants. If all participants had a high level of networking knowledge, it would not be clear whether they could complete the tasks because of existing knowledge, or because the router was helpful. By not having any participants with a high level of networking knowledge, it is more likely that the router would assist them in completing the tasks, and more likely for the difference between their knowledge and the use of the router to be apparent.

In most cases, participants used similar techniques for network management as highlighted in [178]. They often used tools that were part of the OS on their computer for troubleshooting, and resorting to rebooting the router and/or devices when problems occur.

6.12 Results

6.12.1 Overview

The overall outcome of the experiment was somewhat unexpected, especially with regards the areas of difficulty.

Many participants assumed that they could reconnect devices using the router, which was unexpected. It was not surprising that many participants had difficulty identifying the change which could be the cause of the DNS error. However, the reason for the difficulty was unexpected. Most participants assumed that the error was caused by something they did, which resulted in several attempting to undo the change to the Wi-Fi. Others assumed that because they witnessed the error message appear, that they could revert to just before the error message. Only one participant (P10) realised that the change was not caused by something they did.

The unexpected behaviour is interesting because it highlighted areas for future research and demonstrated the challenges of dealing with a system where time is a factor.

The text message notifications were not sent reliably, which meant that some participants were not aware when a new device was added. In the case of P2, the text message was received immediately before the notifications of the network change, meaning that the new device notification was missed. As P2 put it “I did see the black box though, and I could see that someone was trying to request, but that was when I was trying to work out the DNS”.

Many of the difficulties that were expressed with the use of the router were small details which affected the usability, such as how the history timeline was presented.

Table 6.3 outlines the overall outcomes of tasks.

Table 6.3: Task Status

Task	Status
1	Success
2	Success
3	Partial Success
4	Success
5	Partial Success
6	Success

6.12.2 System Usability Scale

Participants were asked to complete the System Usability Scale (SUS) [27]. The SUS is a measure of the usability of the system being tested. The questionnaire is shown in Section 6.12.2.

These are the participant responses for the system usability scale [27] questionnaire.

The lowest SUS score was 60, the highest was 97.5, and the mean score was 86.25. According to metrics by Bangor, Kortum, and Miller [14], for the router to be considered to have good usability, a SUS score higher than 70 would be required. A score in the high 80s or 90s would be considered excellent. In this study, eight participants gave the router a score above 80, with one under 80 and only two under 70.

SUS scores do not highlight areas where improvements could be made, but they do allow direct comparisons of perceived usability between participants.

Table 6.4 presents the overall SUS score awarded to the router by each participant. Each participants' skill level is also included to reflect how skill level affects the assigned score.

Table 6.4: Overall SUS Score for participants

Participant	Skill Level	Score
P1	High	97.5
P2	Low	92.5
P3	Low	65.0
P4	Low	97.5
P5	High	92.5
P6	Medium	95.0
P7	Medium	87.5
P8	Medium	80.0
P9	Low	60.0
P10	Medium	82.5
P11	Medium	77.5

The SUS values demonstrate that the router had a reasonable degree of usability, but that there is room for improvement.

6.12.3 Network Diagrams

During the experiment, participants were asked to draw a diagram of their home network. These diagrams are included here. The details shown in the diagrams are an indication of the level of networking knowledge. All participants were at least partially responsible for managing their networks, except P3. All participants are students or staff at the university. Most live in private accommodation in Nottingham, except for P8 and P9, who live in halls of residence. Their network diagrams are for their non-term address. Asking P8 and P9 to draw a diagram of their network, or answer questions related to network management would not be an indicator of knowledge if applied to the halls of residence because the networks are centrally managed, so participants would only be able to list the devices on their network. Basing the diagrams and questions on non-term addresses allows comparison between users.

The network diagrams can be found in Appendix D.

6.12.4 Tasks

6.12.4.1 Task 1

This task was designed to introduce participants to the network and familiarise them with the devices used. All participants were able to understand the network and were able to identify that Mary's phone was not connected.

6.12.4.2 Task 2

All participants were able to reconnect Mary's phone. Most tried different troubleshooting techniques with the phone, including rebooting and re-entering the passphrase. However, once participants switched from the device to the router, they were all able to identify the cause of the problem. Some relied on the timeline but were able to find the 'Device Denied' event and undo it. Others found the solution by exploring all screens and noticing a difference on the Device screen.

6.12.4.3 Task 3

All participants completed task 3 successfully. However, the notification was delivered at a random point in the experiment, which caused confusion when completing other tasks. P2 only realised there was a new device when completing task 5.

6.12.4.4 Task 4

All participants were able to adjust Wi-Fi configuration and identify that devices were still connected.

Adjusting device configuration was more problematic, particularly for the XBox. Many participants could not understand how to reach the settings screen. However, as those who had difficulty also stated that they had never used an XBox before, this was expected.

6.12.4.5 Task 5

Only P10 was able to complete the task successfully. All other participants had difficulty identifying the cause of the error. The design of the experiment may have contributed to this factor because all participants witnessed the modification I made to the router to cause the error. All participants except for P10 assumed that the problem was caused by either them changing the Wi-Fi or they realised that I caused the issue. In both scenarios, participants attempted to revert to a point which was too recent to fix the problem.

However, once participants checked the task instructions (in Appendix A), they were able to identify the task.

For this task, participants were required to use the timeline to solve the problem, which provides 3 options for a solution.

1. The undo button on each undoable event.
2. The 'undo all' button on each undoable event.
3. The date picker at the top.

Participants who assumed I caused the problem used the date selector, as it is the only method of choosing an arbitrary point. Participants who assumed the problem was caused by them changing the Wi-Fi used a mixture of all three options to undo the change to the Wi-Fi. Once participants realised that they needed to undo an event earlier in the timeline, most participants used option 1 or 2 to fix the issue.

6.12.4.6 Task 6

All participants were able to complete task 6 successfully.

6.13 Summary

This chapter described the lab-based experiment used to evaluate the router. An outline of the data collected for the experiment is also included.

The fact that participants were able to complete the tasks successfully suggests that the router satisfies its primary objectives. The next chapter discusses the results in more detail.

Chapter 7

Discussion

7.1 Introduction

This chapter examines the results of the experiment described in Chapter 6, to be in a position to answer the research questions.

7.2 Themes

During the experiment, the participants' experience with the tasks was similar.

While reviewing data from each participant, patterns emerged regarding the difficulties faced, what was successful, and how challenges were overcome. This was most apparent in the use of the timeline, particularly for task 5. During this task, the similarities between participants' understanding of time and inter-event relationships could be witnessed in almost every case. P10 was the only participant who did not have issues.

As a result, participants' experiences with using the router during the experiment can be categorised into four themes:

1. Using History
2. Perspective
3. Relationship Between Events

4. Network Transition

7.2.1 Using History

Overall, feedback on the usefulness of history as a troubleshooting tool was positive. However, there were issues faced with using the history.

7.2.1.1 Successes

All participants were able to identify the events in the timeline which corresponded to changes which they had made. All participants also understood the concept of returning to an earlier point in time and could do so successfully, either by entering a date or by using the revert button associated with an event.

Participants could also identify the difference between reverting to an earlier time and undoing a single change.

7.2.1.2 Confusion

Although participants had no issues identifying changes they made, all but one ignored any event which was not made by them. P10 was the only participant to identify the event which caused the DNS error message without confusion.

Many issues faced by participants in using the timeline relate to how the changes were presented in the timeline. There were various issues which impacted on participants ability to complete the tasks.

Several participants noted that it would be helpful if error messages were highlighted as such, and not simply like any other event. The fact that some events did not have Undo buttons was also confusing for some participants, with one assuming that the error message was something that could be undone.

It may make sense to display error messages at the top of the screen in a persistent state, and only remove the error when it has been fixed. Many participants assumed that reverting to just before the error appeared would resolve it, and the router gave no indication that this was not the case, which is something P11 found particularly confusing. Other participants were also confused, but only P11 made the confusion explicit.

When the DNS error was displayed, its wording prompted many participants to search other screens to find what to change. It would appear that considering how events are displayed and having a greater degree of differentiation between event types would be beneficial. The wording used to describe events may also affect understanding, and help guide people towards related events.

There were occasions where participants could not locate events that they had previously seen, especially after an undo or rollback, which will add events to the timeline and move the others to make room. One participant highlighted this specifically when locating the DNS error message. It has also been suggested that events representing errors could be highlighted to indicate them more clearly.

One observation made during the experiment was that although new events are added to the top of the timeline as they are added, many participants did not notice, and required a prompt to scroll back to the top to see what new events were added (if any). When participants used the undo or rollback buttons associated with the change to DNS, one or more new event was added (depending on the choice of Undo or Revert), which would have indicated that the change had been undone. Events being rearranged (to ensure that events alternate from left to right around the central timestamp list) is perhaps too subtle.

Some participants also appeared to be reluctant to undo changes without confirmation that they had chosen the correct change to undo. Reluctance may be due to a lack of understanding of what the error meant, or what the changes did. Because the router provides the ability to undo changes, if participants had chosen the wrong event to undo (rollback is not undoable), they would have been able to undo the undo.

7.2.2 Perspective

Task five required participants to identify the event in the timeline which could have caused DNS lookup failures. The event in question was in the timeline at the start of the experiment. The error event was added (along with breaking DNS lookups) at the appropriate point in the experiment. By having the change to DNS settings in the timeline at the start of the experiment, the difficulty of the task was increased. Adding both events at the same time, or specifying a timestamp for the event which was closer in time may have resulted in users being able to identify the correct event more easily because the events would be closer together. By not having the change be the most recent, and having it there from the start meant that users had to look for it, and could not simply identify what was new.

One side-effect of running the experiment in this manner was that several participants demonstrated a shorter perception of time than the router presented. A common occurrence was the belief that the error had just occurred, so rolling back by a short period should fix the problem.

The short-term perspective of users differs from the long-term perspective that the router provides. The difference in perspective could be attributed to the experiment methods. However, the presence of such a difference could also indicate a lack of understanding of the long-term effect of changes. Several participants assumed that changing the Wi-Fi configuration must have caused the error because it was the last thing to change.

It may be the case that a change made now would cause a (possibly unrelated) change at some point in the future.

Another way in which participants' short-term perspective was apparent was a belief that the DNS error must have been caused by something they did, with several participants assuming that the change to Wi-Fi must have caused the error because it was the last thing that changed.

Differences in how time is perceived suggest that the presentation of a history of changes must include a reference point as to how much time has passed.

A limitation of a lab-based experiment is that participants make several changes in a short time-period. While this may add to the confusion with time perception, if changes were made over a long period, it may not be apparent from the timeline what caused the problem. The most recent change may have occurred several weeks or months before an error, and rolling back to just before the error may not provide a fix, depending on the cause of the error.

Perception of time may be affected by the fact that the router did not provide a clear link between related events.

7.2.3 Relationship Between Events

Some participants explicitly referred to the fact that it was not clear which events were related. Given that people appear to have a short-term perspective of time, this observation is not surprising.

In fact, those participants who made explicit reference to the fact that the link between events was not clear also suggested that the problem could be addressed by highlighting related events. In the case of a DNS change being the cause of an Internet connection failure, an explicit reference to the fact that the original DNS change could be a possible cause of the error may have helped them make the connection between the events. This also suggests that it would make the notion of the passage of time explicit, and may mean that users become aware of the fact that changes could cause issues at a later date.

7.2.4 Network Transition

The majority of participants in this study suggested that the use of multiple networks for changes to Wi-Fi configuration would be helpful. However, many participants did experience some level of difficulty in completing the tasks.

7.2.4.1 Presentation

The only indication from the router that there is a new network, and that the original settings are still in effect is the notifications that are sent to those devices connected via Wi-Fi. It was never explicitly suggested by participants, but it may be beneficial for users if the router UI indicates that the status of both networks, not just the new ones.

P3 did not understand why devices were still connected to the old network, despite indicating that leaving devices connected would be useful. This confusion may be partially due to a lack of understanding the instructions, as they were written in English, which is not her first language. Rephrasing the question was necessary to gauge whether they found the feature useful.

Several participants ignored the notifications for the devices and merely noted that they had received them. Many participants were unclear as to which devices needed to be reconfigured, with many attempting to configure devices connected via Ethernet, despite notifications being sent for each device which needed to be reconfigured. Although the router displays how devices are connected, it would appear that specifying the connection as Wi-Fi or the Ethernet port (eth1, eth2 or eth3) may not be sufficient.

Also, it may not be appropriate to rely on notifications sent to devices (text messages and emails in this case), because the notifications were not always received. The router always sent the notifications as necessary, but text messages were often not received, possibly as an anti-spam measure by the phone network. Notifications are only sent for devices which are active on the network. If a device is sleeping, it will disconnect, and no notifications will be sent. One participant expected notifications to be sent directly to each device which requires reconnection, stating that doing so would have made the list of devices to reconnect more obvious.

It may be more appropriate to make the router UI display both active networks, with a countdown till the old network is removed. It would also be appropriate to show a list of devices which must be reconfigured, whether they are currently connected or not. Making required actions explicit is one possible method of reducing confusion.

It may also be appropriate to label each device on the Devices Screen when a new network is active, but a device is still connected to the old network. By labelling devices that must be reconfigured, it would be explicit when actions are required. The Devices Screen was often used to attempt to reconnect devices, so by stating whether or not the device is connected to the old or new network, or Ethernet; it would be clearer what must be changed.

7.2.4.2 Migration

Migrating devices to the new network was another source of confusion. Unfamiliarity with the devices used may have been a factor, but this was addressed by asking participants which devices they were familiar with before the experiment. Instructions for each device were provided, and assistance offered where necessary, especially for unfamiliar devices.

P1 had difficulty identifying where in the router to change the Wi-Fi configuration. Some participants used the home screen to navigate between sections. One participant stated that the navigation bar at the top of the screen was not clear, and other participants suggested the same thing in that they never used it to navigate between the screens.

Some participants attempted to reconfigure devices by editing them in the Devices screen, rather than on the devices. One possible solution is to use the original network as a mechanism to update devices automatically, or on request using the router UI. One participant explicitly suggested automatic reconfiguration.

Regardless of the technical ability (or lack thereof) to implement automatic reconfiguration, it does appear to be a natural extension of the idea of simplifying the process of updating Wi-Fi configuration. Several participants assumed that the router would do so, suggesting that having to change both the router settings and the devices is not obvious.

It may be the case that using multiple networks contributed to this confusion because devices would still be shown as connected on the Devices screen, which might suggest that it would appear that nothing needs to be done.

7.3 Summary

This chapter was an overview of the results of the usability study for the router described in this thesis. The results of the study suggest that both configuration history and using additional networks for change management have potential to ease management burden and assist with troubleshooting. However, more research is necessary to expand the concepts sufficiently to provide benefits to users outside of a lab environment. Expanding the scope of the research to cover additional use-cases and redesigning the interfaces using the feedback from this study are useful starting points, and would facilitate investigations into the wider potential for configuration tracking as a troubleshooting tool.

The next chapter summarises the thesis and outlines areas where there is a need for future research.

Chapter 8

Conclusion

There are two ways of considering the research presented in this thesis.

First, as a method of improving the management process of home networks after initial installation.

Network configuration items are often isolated, and changes to one item do not (usually) require changes to others. When undoing changes to an individual configuration item, it should not be necessary to undo other (potentially unrelated) changes. It is, therefore, possible to view the undo method as an approach to undo which is optimised for independent changes, and which uses network configuration for evaluation purposes.

As shown in Chapter 2 and Chapter 4, there are several issues faced by householders which could be addressed by providing a history of changes. Also chapter 3 demonstrated several techniques for ‘undoing’ with many designed for use in a specific application.

The requirements of an undo facility and the design of a system to satisfy these requirements are integral to the design and implementation of the router developed for this research. The literature (see chapter 3) heavily influences the design of the ‘undo’ system, particularly concerning how each method could be applied to network configuration.

8.1 Contributions

8.1.1 Multiple Networks

It is the norm to be required to reconnect all devices to a network if settings are changed. Allowing a grace period before disconnecting devices was intended to ensure that all connected devices can continue to function after the change.

The user study did suggest the potential usefulness of multiple networks for reconnecting devices. However, it highlighted usability limitations. Devices can provide a list of networks that are in range, but there is no method that could be used to highlight which SSID is the correct one (which would be worse if the SSID was unchanged).

The study also highlighted the potential for investigating how multiple networks could be used to create new methods for updating Wi-Fi configuration, each of which would also address the difficulties faced, such as using the original network to provide a secure mechanism to send configuration updates to devices. Users would then be able to use the router to update devices, which several participants assumed was the case.

8.1.2 History

While it is fairly common for routers to have the option to view the system log available, even if users knew where to look, logs are not particularly user-friendly because they are intended to be used by experts.

The timeline displays much of the same information as might be included in a system log but organises it around the tasks that were performed, rather than system actions.

However, it is not enough to list the changes. All but one participant failed to immediately identify that the DNS change caused the Internet connection error in task 5. This suggests that people might not be able to link events which did not happen close together. How history is presented will determine how easy it is to identify related events and the usefulness of history. For example, participants did not always identify that the Internet connection error was an error message because it was presented in the same way as other changes. Categorising changes and displaying each category differently may help with this.

8.1.3 Undo

The ability to undo actions is important in many applications, especially if the required action is not clear. Router configuration management can also benefit from an undo function, and such a function is a natural addition to history.

However, it is important that consideration is given to determining *which* changes can be undone. For example, it is important that undoing a change does not itself cause a problem. To do otherwise could cause confusion, instead of helping to reduce confusion.

8.1.4 Revert

There may be times where the exact cause of a problem cannot be determined, so providing the ability to return the state to that before a specific time is a useful addition. Exactly how useful will be determined by how the history is presented, because it may not be clear how long it has been since the perceived problem occurred.

8.1.5 Technical

8.1.5.1 Notifications

Although notifications from a router are not new [116], allowing users to configure notifications to alert the user(s) of individual devices of network changes that may affect them, is.

8.1.5.2 Undo

Network configuration consists of multiple configuration items, each with unique requirements for identifying previous versions. As such, the undo algorithm required a bespoke process which can act on each item independently of others.

Undo commands must check that a previous configuration value is valid before processing the undo. To be valid, a previous version must:

- Be a user-initiated change

  If the previous version was generated by the system, an earlier version of state may be restored which may cause issues. For example, selecting a DHCP lease renewal for a device which has been disconnected for longer than the lease time. In such a scenario, the lease would be recreated again.

  Only selecting user-initiated changes makes it possible to limit the scope of history and undo to be changes that the user made, and not cause confusion by causing the state to change to something the user does not expect.

- Be successful.

  If a configuration change was known to cause an error state, the configuration change should not be a permitted previous version. Otherwise, the undo function could cause more problems than it solves.

- Represent a state which matches user intentions.

  When selecting a previous version, the previous version may result in unexpected behaviour, because it does not match user intent. For example, when undoing the first 'permit' decision for a device, the previous state would indicate that the device was 'pending'. However, users may wish to undo a permit because they accidentally permitted a device. As such, it may be more appropriate to allow the user the option of removing the device instead.

- Exist

  For some configuration items, a valid previous version is one in which the configuration item is removed (such as devices or notifications). For others (such as Wi-Fi) it does not make sense to remove the configuration. In those situations, the original setting should be the last one that the undo system should select.

8.1.5.3 History

To realise the Undo algorithm, it is necessary to have separate histories for each item of configuration, and a global history which consists of references to points in the individual histories. Using this method allows the undo algorithm to operate on separate configuration items independently.

Using multiple histories also makes it possible to keep track of all changes to a configuration, regardless of whether a change would be presented to users. Only storing a single, linear history would add complexity to the task of identifying the correct previous version when performing undo, because important non-undoable changes may have occurred.

8.1.5.4 Revert

The revert algorithm utilises the logic of undo for each configuration type, once the affected configuration items are identified, and what versions must be undone.

The identification process for changes to be reverted takes into account that the result of a revert for any individual configuration item is that all configuration items will be restored to the state they were in before the selected timestamp. Storing all state for a configuration item on every change (instead of just storing changes) means that it is only necessary to undo the oldest change for every configuration item which occurred after the timestamp.

8.2 Research Questions

8.2.1 Technical

1. Is it possible to create a history of configuration changes in a router?

   Yes. Configuration changes can be tracked easily if the datastore has built-in support for historical data, and configuration change is made as a consequence of a change being logged in the datastore.

   Because configuration history was developed as a troubleshooting tool which could be used by users with low networking knowledge, it is only possible to track changes made by users in the UI. It is, however, possible to track changes that were made directly, without using the UI.

2. Can changes be undone and redone?

A result of storing historical changes to configuration items is that it is possible to switch to an earlier version, which has the effect of undoing the change. Redoing changes can be achieved by appending the undo operation to the history, which then makes it undoable.

Changes made to devices by the router during normal operation (such as the DHCP server leasing an IP address) are tracked in the history of the appropriate document. However, allowing users to undo actions triggered by system actions has potential to cause problems. A more appropriate solution would be to consider *what* a user is trying to achieve, or what caused the system to perform an undesirable action, and undo that change.

3. Is it possible to revert to an earlier point in time?

   The undo system is capable of undoing multiple changes at the same time, and each change has a timestamp, which means that it is possible to revert to an earlier point in time. In addition, it is also possible to ensure that only the oldest change to a configuration item is undone, which also reduces the number of undo operations required.

8.2.2 HCI

1. Is a timeline of changes a useful network management/troubleshooting tool?

   The participants in the user study were able to identify issues which need to be addressed. However, including system notification events (such as the Internet connection error from task 5) are not visually different from user changes, so are easy to overlook.

2. Does 'undo' improve users' ability to manage home networks?

   When problems were presented to participants in task 2 and task 5, most participants were able to use the undo option next to the appropriate event in the history to address the problem. For task 2, some participants identified the denied device on the home screen history summary and opted to use the device management screen to fix the issue.

3. Is 'revert' a useful safety net when problems arise?

   During the experiment, several participants relied on revert to complete task 5, caused by a lack of understanding as to what the error meant, and which items would be related. However, allowing users to select an arbitrary point without restriction does have potential to cause confusion, because it does not make sense to revert to a time that would not result in any changes being undone.

4. Do notifications of network changes aid users' ability to understand their network?

   It is not clear from the experiment how important notifications are to aid understanding. Notifications of new devices are one type of notification which could be useful. It is difficult to determine whether notifications of a change in Wi-Fi configuration are useful because all devices were in the same location and used by the same person, so notifications would not have been required.

8.3 Limitations

The work presented in this thesis has some limitations, caused by a range of factors, each of which is described below.

8.3.1 Scope of configuration

The effort required to track changes in a manner compatible with the constraints of the Undo system and with CouchDB is considerable. The best approach is to store the data each component requires in CouchDB and update the database when the components are updated. Placing responsibility for tracking changes on the components of the router means that changes could be made in different ways and still be tracked. However, modifying tools such as Hostapd to support configuration tracking is difficult (because each application works differently), and is impossible for closed-source tools.

Using tools such as Hostapd, therefore, requires that changes be made using an external tool, which can apply the change and track that it took place.

Using a Homework Router as the development platform meant that many components required significant changes to support CouchDB.

The number of possible configuration items is limited, due in part to the architecture of the Homework Router. The Homework Router uses SDN, specifically OpenVSwitch with NOX. Flow data from OpenVSwitch is stored in the HWDB time-series database. There is limited benefit to storing time-series data in CouchDB because there will only ever be new documents created. It is more appropriate to use a database designed for storing the flow data in this case. One side effect of this is that three databases would be required (CouchDB, HWDB and MySQL), and all three would need to be synchronised. Components which use data from HWDB, such as the policy engine, would need to be modified to store data in both HWDB and CouchDB. In the case of the policy engine, the data required in CouchDB would be the creation, editing, and removal of policies.

Using both HWDB and CouchDB would also cause issues because they are both used to trigger processes as data changes. When two processes are triggering changes, there is potential for conflict.

Designing a system which can handle the demands of both database engines was considered unnecessary complexity when building a system to track configuration changes as a proof of concept.

8.3.2 Multiple Networks

The biggest limitation of creating multiple networks is usability. There are no issues when the SSID is one of the settings changed. If the SSID is not changed, but the passphrase is changed, the router would create two networks with identical SSIDs, and there would be no way of distinguishing between the networks on most devices[1] In this case, the usability problems arise because of the Wi-Fi specifications.

Another usability problem may occur if a device regularly connects and disconnects from the network. Any devices that remember network details could attempt to connect to the old network after the new one had been configured.

Another problem with multiple networks is compatibility. At present routers must use Hostapd and they must use a Wi-Fi radio which supports the nl80211 drivers.

[1]It is possible to identify the BSSID (often the MAC address of the Wi-Fi adaptor), which would distinguish the networks. However, requiring users to identify the correct network from the BSSID is counter-productive.

8.3.3 Notifications

Limitations in the implementation of the notification system are inherited from the Homework Router. The limitations considered here are concerned with how the notification system is utilised.

First, a side-effect of the notification mapping system from Homework Routers is that there is limited scope for specifying where notifications should be sent. If a person provides multiple destinations for a service, they will receive the same notification multiple times.

Second, it is not possible to control the events which trigger notifications, or where they are sent, other than by specifying the service to use for each device. It is not possible to specify that a particular service with a particular username should be used for a situation.

Third, notifications are sent to owners of Wi-Fi devices when the configuration is changed, but *only* if the devices are currently connected. Devices which were not connected when the change occurred would not be notified, even if the device was connected to the old network before it was switched off. There is no method to send notifications to devices that connect to the old network that the network will be switched off.

Finally, notifications are not context-aware. In the case of multiple networks, it does not make sense to notify users that they need to update their devices if the old settings are restored within a day of the original change, and the device is still using the old network. In that case, notifications would be sent informing users that they need to change the settings on their device when they do not need to do anything.

8.3.4 Wi-Fi Security

The current implementation only supports WPA, so users do not have the option to change encryption type. By only supporting one security scheme, the devices that could connect is limited. It would not be possible to support older devices which only support WEP. This limitation would only apply if the router were to be used with older devices. If all devices can use WPA or WPA2, this limitation does not apply.

8.3.5 Scope of Undoing and Reverting

The undo system does not support all changes that could be undone. The most significant of these is reverting to a previous time. Reverting multiple changes is not currently possible due to how the revert process works.

Requesting a revert will currently trigger an undo for every command that occurred after the chosen timestamp. Triggering multiple undo commands does not make it possible to identify which commands the revert action undid because there may have been events in between the revert and the time in which it is to be undone. Revert events could keep track of the documents which were changed, which would resolve this issue.

Finding the previous version of a document does not support undoing multiple changes. The current implementation assumes that there will only be one document to request revisions for, and undoing a revert could require requesting the previous revisions for several documents.

Also, each Undo Event that the rollback creates will insert an individual event into the timeline. If the rollback were to know which events were undone, it should present these in a single event, regardless of whether the rollback is undoable or not. In its current form, the presentation of the revert allows the individual Undo events to be undone, but not the revert. If the ability to revert were to be undoable, the individual events would need to be grouped and undone as a group.

8.3.6 System Architecture

CouchDB can replicate data between databases which creates the potential for new features which could use the feature, and not just remote access. One possible enhancement is suggesting improvements to the configuration based on usage, possibly compiled from comparisons from other routers (similar to PeerPressure [168]). Implementing such a feature would be computationally expensive enough to prevent the router software being run on hardware which is more like a traditional router. The current implementation is already unsuitable for a low-power system, mainly due to the requirement for more storage than is typical of routers. Reducing the power requirements of the router are important both to minimise the environmental impact [73] and to enable the network to blend into the background (a larger router may make this impractical).

Using replication requires the use of the _changes feed for applying changes, which does limit the features that can be implemented because all requests must change data in a document. Changing the architecture to use an API for changes, and only using the database to store data would increase the possibility of implementing a wider range of features. Not having to rely on the database would reduce the overhead in providing feedback.

8.3.7 Experiment

Testing the usefulness/usability of a tool for troubleshooting problems requires that there are problems that must be fixed. Field studies might need to cover an extended period of time to gain useful insights. Choosing a lab-based experiment addresses this issue by presenting participants with fixed tasks.

However, artificially creating problems during the experiment (such as task 5) could potentially affect how participants reason about the problem. In the case of task 5, several participants attempted to solve the problem by reverting to the previous day, on the basis that they witnessed the artificially created breakage happen.

8.4 Future Work

The potential future work could be restricted to overcoming the limitations outlined in section 8.3. If the concepts introduced in this thesis were to be utilised in commercial routers, these limitations would need to be solved.

However, there is also scope to consider the usefulness of adding additional features. Details of how each limitation could be addressed and the further study required are outlined below.

8.4.1 Multiple Networks

Of the limitations identified, addressing the usability issues involved with editing the network and creating a new one is the most challenging. The only way to address the problem is to change the Wi-Fi specification to support the easy identification of networks with the same SSID. For example, the problem could be addressed by allowing networks to be broadcast with a label which can be used to label networks as new. Another possible enhancement would be to force devices to prioritise the new network over the old one, so that if the device disconnects and reconnects, it chooses the new settings.

One other possibility is to use the original settings to transmit the new settings to devices, allowing users to make changes entirely in the router management system. Automatic device reconfiguration merits further study to determine if it would be of use to users and if it can be designed in a way which does not cause additional confusion.

Reconfiguring devices automatically would be difficult to implement, and would also require modifications to the way devices handle Wi-Fi (changes to the specification) if the feature were to be implementable in a manner which does not add confusion. Another consideration is that by allowing the router to notify devices of changes, users would not need to interact with network settings on their devices, so this feature represents another solution to the usability issue caused by not changing the SSID.

While allowing the router to send network configuration to devices would address some of the usability issues, there is the potential for such a system to create more usability issues, depending on how it is presented, and if/how users can select those devices that they wish to reconfigure automatically.

8.4.2 Notifications

Providing more flexibility with where, when and why notifications are sent would improve their effectiveness, so they represent a worthwhile enhancement. Some notification types are unreliable. Text messages are the least reliable, mainly due to failed delivery. In all cases, the problems occur when receiving the notifications. The router does not have issues with sending notifications.

Another area for improvement is the wording of notifications. Twitter Direct Messages and text messages have a limit on length. In the case of text messages, this may mean that a message is broken into two or more sections, which may not be received in the correct order. If multiple notifications are sent to the same phone, the parts of the notifications could be further mixed up, as was illustrated many times in the user study.

It may also be worth considering supporting a wider range of notification services, and possibly identifying methods of displaying notifications on the router, such as different coloured lights to indicate that the router has information, or that an error has occurred.

8.4.3 Configuration Management

The router cannot currently be used in real-world situations because it does not provide enough configuration to be useful. It would need to support changing NAT or firewall configuration (some applications require a direct connection to the Internet), or Quality of Service (QoS) rules, possibly implemented using policies.

Supporting an increased number of configuration types also presents challenges for how changes are tracked and undone. For example, if a device were denied because of policy, it would not necessarily make sense to undo the state change if the policy is still active because the device would immediately be denied again. The options here are to allow users to undo the effect (the device is denied), or require that they undo the cause (creating the policy). It may be the case that one may be more natural to people.

8.4.4 Suggestions for Configuration Improvements

As a result of tracking configuration changes, the router will possess details of how the network is configured which could be leveraged to provide users with suggested configuration improvements.

There are several possibilities. As an example, the router could suggest improvements in configuration based on the devices on the network. If the network is configured to use WEP, but all devices support WPA, the network could inform users of this unnecessarily insecure configuration[2]. Capturing network traffic, and increasing the number of configuration items that are tracked make it possible to consider network usage when suggesting improvements.

Another possibility is to use network traffic and configuration history to determine when problems occur and determine the likely cause or suggestions for fixes. If a fix were known to be required, it could be extended to an automatic fix for the error.

8.5 Final Remarks

New technologies mean that people bring an ever-increasing range of devices into their homes, many of which can connect to the Internet. The result is that home networks are more complex, and the increasing complexity means increased difficulty in understanding the network and the changes that occur.

The router is at the centre of the network, and it is aware of the devices which are connected and how it is configured. What is missing, however, is a method to express the state to users and show them the changes that occurred.

Another consequence of an increased number of devices, each with differing form-factors, is that changing the SSID and/or passphrase is a complex task. Depending on the items in use, making such changes could result in essential items being unavailable. For example, if a thermostat were controllable via the Internet, the functionality would be unavailable until the thermostat and the devices used for control purposes were reconfigured.

As with configuration history, there is no reason why the process for changing network settings cannot be improved so that devices are not forcibly disconnected when changes occur.

The router created as part of this research is an attempt to create a solution to both of these challenges.

When network settings are modified, the new settings are defined in a new network, and the old settings remain active for one day. The result is that devices are not disconnected, and can be reconnected at a suitable point before the network is

[2]WEP has a known security flaw [36]

disconnected. The router also notifies the users of each device that they will need to reconfigure the device if it is to remain connected to the network.

When changes are made to any part of the router's configuration, the new version is added to the history, and a timeline is updated to inform users what has changed, and when. It is also possible to undo individual changes or revert to an earlier point.

These features were created with the intention of assisting with management of the network so that any problems can be resolved promptly. The usability study suggested that the router may provide the intended result, but there is room for improvement. However, it has been possible to define specific items which could be addressed in future research.

Appendix A

User Documentation

This purpose of this experiment is to assess the usability and appropriateness of the functionality of a Wi-Fi network. The experiment involves tasks that could be necessary for any network in any home, but which also focus on the features that make the router in this network unique.

To do this, you are asked to perform maintenance tasks on John and Marys home network.

A.1 Router Features

The router used in this network has some features not found on other routers. This section describes these features.

A.1.1 Notifications

The router can send notification messages when important events occur on the network. Notification messages keep you informed of events that occur and any actions that you must perform in response to the events.

The router sends notification messages in response to these events:

- When a new device connects to the network. The person set as the main user (who has overall responsibility for managing the network) receives notifications when new devices connect to the network. In this case, John is the main user, and he receives notifications of this type by text message on his iPhone (John's iPhone).

- When users update network settings, notifications are sent to the owner of every device currently connected using Wi-Fi, to tell them that they need to reconnect.

For this experiment, notification recipients have already been configured. John always receives notifications by text message to his iPhone, and Mary always receives notifications by email on her tablet.

A.1.2 Updating Wi-Fi settings

When updating Wi-Fi settings, the router creates a new network that uses the new configuration. The original network remains functional for one day, allowing time to update all devices to work on the new network while still ensuring that they are functional in the meantime.

The router does not create multiple networks if the only changes are to the channel or mode.

This feature also uses the notification system to alert users of the need to act. Devices remain connected to the old network, so they remain listed as connected after the change. As such, other people may not be aware that the configuration was updated, and that their device must be updated.

A.1.3 Configuration History

The router keeps track of changes made to the configuration of the router. The tracked types of configuration include Wi-Fi settings and device configuration, (such as name, owner and whether or not a device is permitted to connect to the network), and notification settings (such as specifying the phone number for John's iPhone, so that it can receive text messages).

The router presents these changes in a timeline, with more recent changes at the top.

Some changes are for information only, such as the router cannot connect to the Internet.

A.1.4 Undo

You can undo changes you made using the history timeline. Some events on the timeline are for information and are not undoable.

The router deletes all information stored about (and forget the presence of) any device that has the first change to its permission to access the network undone. Forgetting a device cannot be undone.

A.1.5 Revert

An extension of being able to undo is to revert to an earlier point in time. Undoing a change only affects that individual change, but reverting undoes every change from the most recent till the desired point in time.

Reverting to a previous point is useful for situations when you cannot tell what went wrong, but you do know when it was working last.

You have three choices for determining what to revert to:

- Choose a date and time — all events after this are undone
- Choose an event — undoes that event and all subsequent events
- Start again — return to the state at the start of the experiment and delete the history of changes made

A.2 Devices

The networked devices, whom they belong to, the notification service and whether they connect using Wi-Fi or with a cable is outlined in table A.1.

During the experiment, another device will be connected to the network. John's iPhone should receive a text message when this happens. You should be aware

Device Name	Owner	Notification Service	Connection Type
John's Computer	John	Text Message	eth1
John's iPhone	John	Text Message	Wi-Fi
Mary's Phone	Mary	Email	Wi-Fi
Mary's Laptop	Mary	Email	eth2
Mary's Tablet	Mary	Email	Wi-Fi
XBOX One	John	Text Message	Wi-Fi

Table A.1: Experiment Devices

that only devices listed in the table are devices that should be considered part of John and Mary's network. You should deny network access to the device.

A.3 Router Management Interface

The router management system has five sections. Each of these sections is described here, to provide some idea of where you might look to find information when completing tasks.

A.3.1 Home Screen

This screen presents an overview of the state of the router and summarises all other sections.

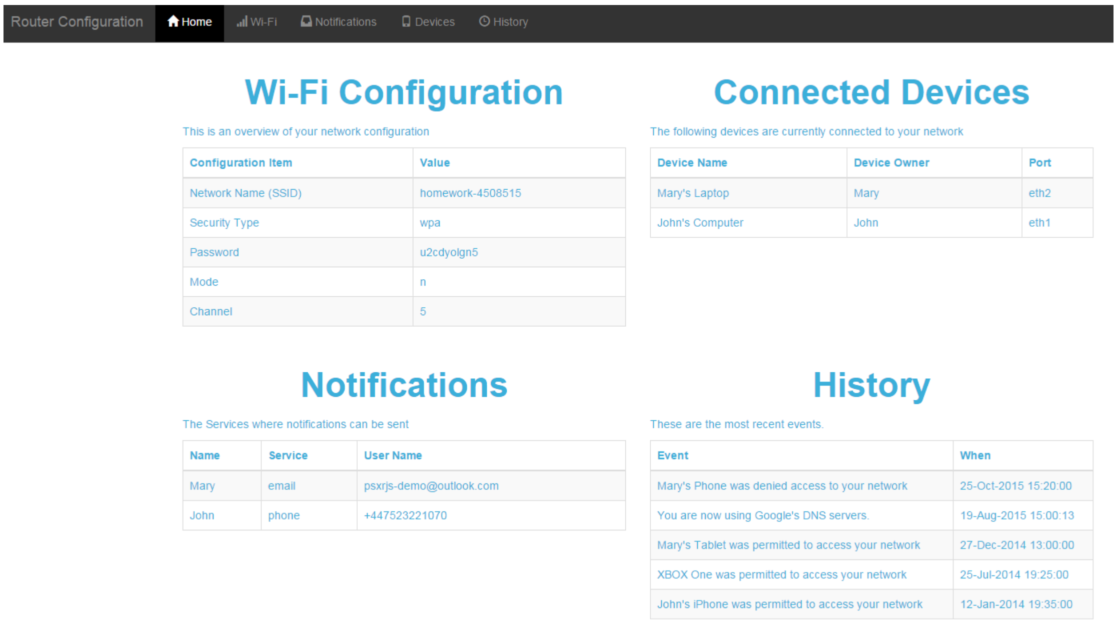
The connected devices panel displays a list of the devices that are connected, and how (with a cable or Wi-Fi). If a device in the list does not have a name, it is a new device.

The Wi-Fi panel shows the active Wi-Fi configuration. The content in the Wi-Fi panel is identical to the information shown on the Wi-Fi screen, but it is not editable.

The notification panel shows the services that each person can use to receive notifications, with usernames.

The history panel shows a text-based list of the five most recent changes.

An example is shown in figure A.1



Wi-Fi Configuration

This is an overview of your network configuration

Configuration Item	Value
Network Name (SSID)	homework-4508515
Security Type	wpa
Password	u2cdyolgn5
Mode	n
Channel	5

Connected Devices

The following devices are currently connected to your network

Device Name	Device Owner	Port
Mary's Laptop	Mary	eth2
John's Computer	John	eth1

Notifications

The Services where notifications can be sent

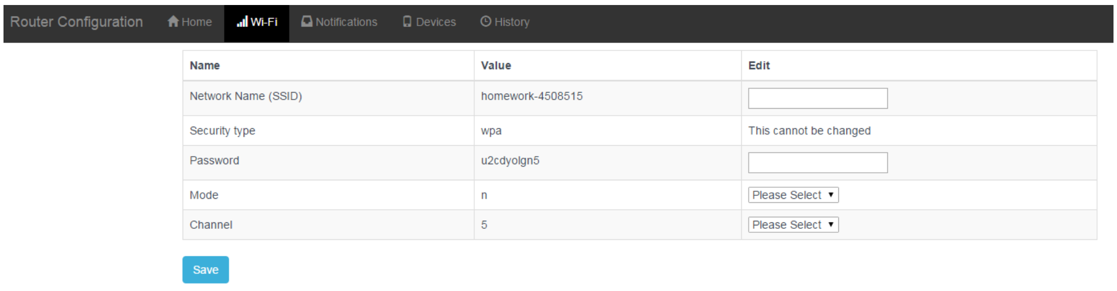
Name	Service	User Name
Mary	email	psxrjs-demo@outlook.com
John	phone	+447523221070

History

These are the most recent events.

Event	When
Mary's Phone was denied access to your network	25-Oct-2015 15:20:00
You are now using Google's DNS servers.	19-Aug-2015 15:00:13
Mary's Tablet was permitted to access your network	27-Dec-2014 13:00:00
XBOX One was permitted to access your network	25-Jul-2014 19:25:00
John's iPhone was permitted to access your network	12-Jan-2014 19:35:00

FIGURE A.1: Home



Name	Value	Edit
Network Name (SSID)	homework-4508515	
Security type	wpa	This cannot be changed
Password	u2cdyolgn5	
Mode	n	Please Select
Channel	5	Please Select

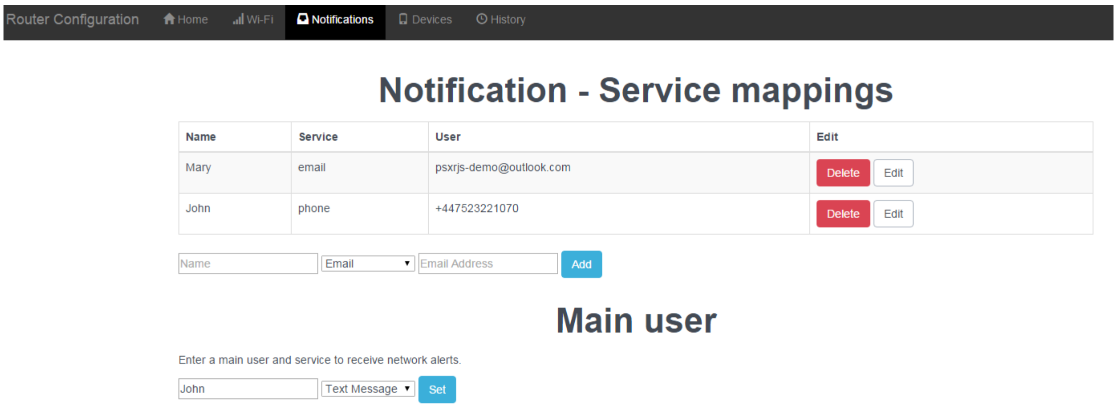
FIGURE A.2: Wi-Fi

A.3.2 Wi-Fi

This screen displays the current Wi-Fi configuration and allows it to be modified. An example is shown in figure A.2

A.3.3 Notifications

The router needs to know where to send notifications for any given service for a particular person. In this case, this means Mary's email address and John's phone number. This screen presents these, and also allows new destinations to be specified, and existing ones edited or removed.



Notification - Service mappings

Name	Service	User	Edit
Mary	email	psxrjs-demo@outlook.com	Delete Edit
John	phone	+447523221070	Delete Edit

Main user

Enter a main user and service to receive network alerts.

John
Text Message

Figure A.3: Notifications

Also on this page, the person who has overall responsibility for the network (the main user) can be specified, together with the service to use to send general network notifications, such as new device connections.

The details on this page are user accounts used on the devices for the experiment and should not need to be modified.

Figure A.3 shows an example of the configuration included on this page.

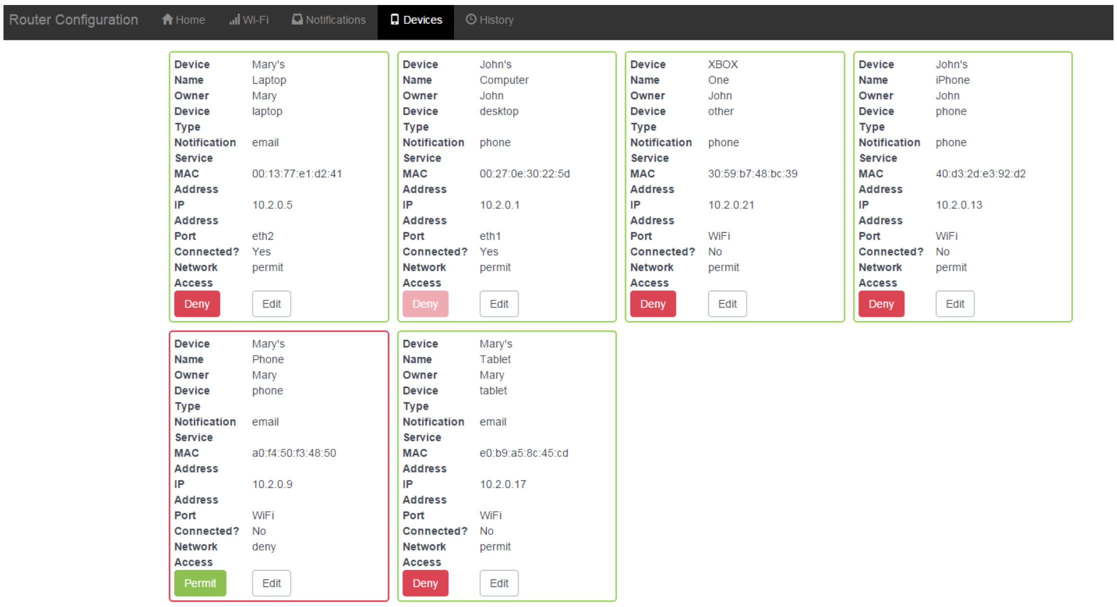
A.3.4 Devices

This screen is where you can manage the devices on your network. Devices must be permitted to connect to the Internet. Devices can also be denied to prevent them from connecting to the network at all. New devices are pending and require a decision as to whether or not to permit them access.

Devices that are pending do not have a name, so are identified by their MAC address, IP address, and hostname. All editable fields are optional, although providing values when permitting a device is useful to help identify devices.

The 'notification service' setting is the service used to send Wi-Fi configuration update notifications, and, together with the owner, should be set to a combination listed on the notifications screen, so that the router can send the notifications successfully. The services supported are:

- email
- text message



Device Name	Owner	Device Type	Notification Service	MAC Address	IP Address	Port	Connected?	Network Access
Mary's Laptop	Mary	laptop	email	00:13:77:e1:d2:41	10.2.0.5	eth2	Yes	permit
John's Computer	John	desktop	phone	00:27:0e:30:22:5d	10.2.0.1	eth1	Yes	permit
XBOX One	John	other	phone	30:59:b7:48:bc:39	10.2.0.21	WiFi	No	permit
John's iPhone	John	phone	phone	40:d3:2d:e3:92:d2	10.2.0.13	WiFi	No	permit
Mary's Phone	Mary	phone	email	a0:f4:50:f3:48:50	10.2.0.9	WiFi	No	deny
Mary's Tablet	Mary	tablet	email	e0:b9:a5:8c:45:cd	10.2.0.17	WiFi	No	permit

Figure A.4: Device Management Screen

- growl
- Twitter direct message

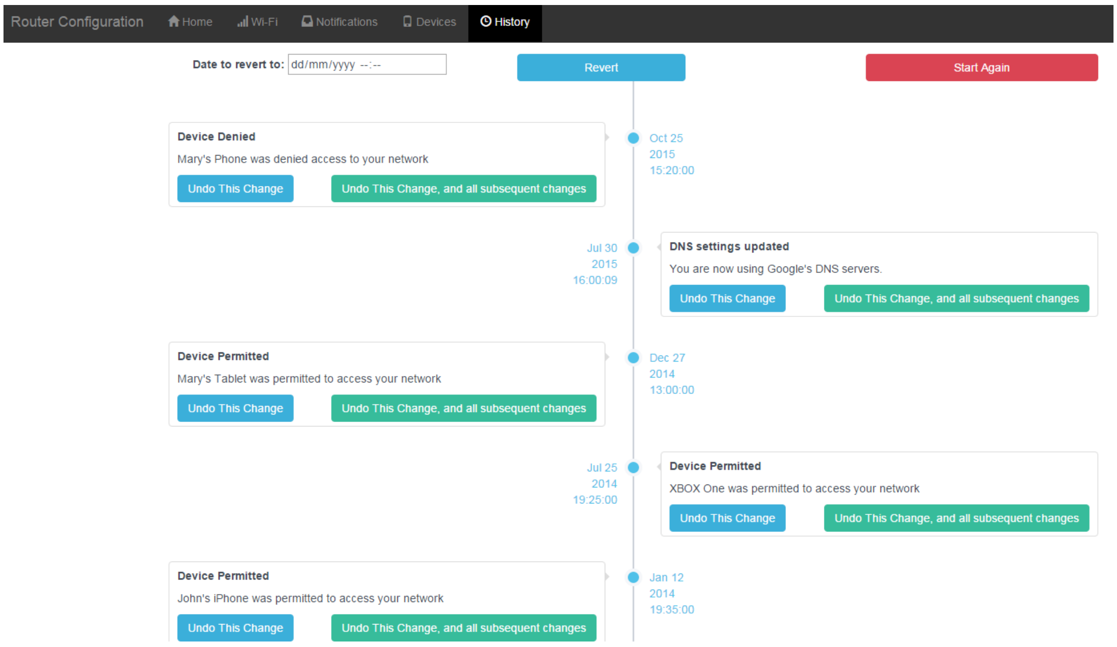
The colour of the border of each device reflects its state. Green means that it is permitted, red means denied, and black indicates that it is new. Changing the status is possible.

An example of the information presented in the devices screen is shown in figure A.4

A.3.5 History

This screen shows a timeline of configuration changes. It is possible to undo changes, either individually, or several at once, to return to an earlier state.

An example timeline is shown in figure A.5.



Device Denied
Mary's Phone was denied access to your network
Oct 25 2015 15:20:00

DNS settings updated
You are now using Google's DNS servers.
Jul 30 2015 16:00:09

Device Permitted
Mary's Tablet was permitted to access your network
Dec 27 2014 13:00:00

Device Permitted
XBOX One was permitted to access your network
Jul 25 2014 19:25:00

Device Permitted
John's iPhone was permitted to access your network
Jan 12 2014 19:35:00

FIGURE A.5: Timeline example

A.4 Tasks

A.4.1 Task 1

The first task is to familiarise yourself with the network. Verify that all the devices listed in the table are connected. There is one device that cannot connect. Can you identify it?

A.4.2 Task 2

Can you identify why the disconnected device identified in task one cannot connect to the network and can you fix it?

A.4.3 Task 3

Using default values for SSID and passphrase is a bad idea, both for security reasons, and because they are harder to both remember and type when connecting new devices.

Can you change the SSID and passphrase of the network to values of your choice, and reconnect the devices listed in the table? You MUST change the SSID. You can use any name you like. Passwords must be 8–63 characters long. Not changing the SSID makes it harder to identify the new network when reconnecting devices. You should find emails on Mary's Tablet and text messages on John's iPhone that cover all devices except John's computer or Mary's laptop. Notifications are not sent for these devices because devices connected with a cable are not affected by Wi-Fi changes.

Changing Wi-Fi settings does not disconnect any devices from the original network.

You need to update each device to connect it to the new network.

A.4.4 Task 4

Can you identify why the Internet is not working, and fix it? The timeline includes an Internet connection failure message. You should identify the associated change that caused the fault and undo this change to fix the problem.

A.4.5 Task 5

Can you return the network to the state it was in before Mary's Tablet was permitted to access the network?

Appendix B

Network Overview

Connection Details

- SSID (Network Name): **homework-4508515**
- Passphrase: **u2cdyolgn5**

Network Notification Messages

John's iPhone receives network notifications as **text messages**, and **Mary's tablet** receives network notifications by **email**.

Devices

The devices used in the experiment are shown in table B.1.

Device Name	Owner	Notification Service	Connection Type
John's Computer	John	Text Message	eth1
John's iPhone	John	Text Message	Wi-Fi
Mary's Phone	Mary	Email	Wi-Fi
Mary's Laptop	Mary	Email	eth2
Mary's Tablet	Mary	Email	Wi-Fi
XBOX One	John	Text Message	Wi-Fi

Table B.1: Device List

Appendix C

User Experience Questionnaire

This questionnaire asks simple questions about the ease with which you could use the router.

Answers have five choices, with one symbolising disagreement to the statement, and five symbolising agreement.

	No				Yes
	1	2	3	4	5
1. I think that I would like to use this system frequently	☐	☐	☐	☐	☐
2. I found the system unnecessarily complex	☐	☐	☐	☐	☐
3. I thought the system was easy to use	☐	☐	☐	☐	☐
4. I think that I would need the support of a technical person to be able to use the system	☐	☐	☐	☐	☐
5. I found the various functions in the system well integrated	☐	☐	☐	☐	☐

	No				Yes
	1	2	3	4	5
6. I thought there was too much inconsistency in the system	☐	☐	☐	☐	☐
7. I would imagine that most people would learn to use this system very quickly	☐	☐	☐	☐	☐
8. I found the system very cumbersome to use	☐	☐	☐	☐	☐
9. I felt very confident using the system	☐	☐	☐	☐	☐
10. I needed to learn a lot of things before I could get going with this system	☐	☐	☐	☐	☐

Appendix D

Network Diagrams

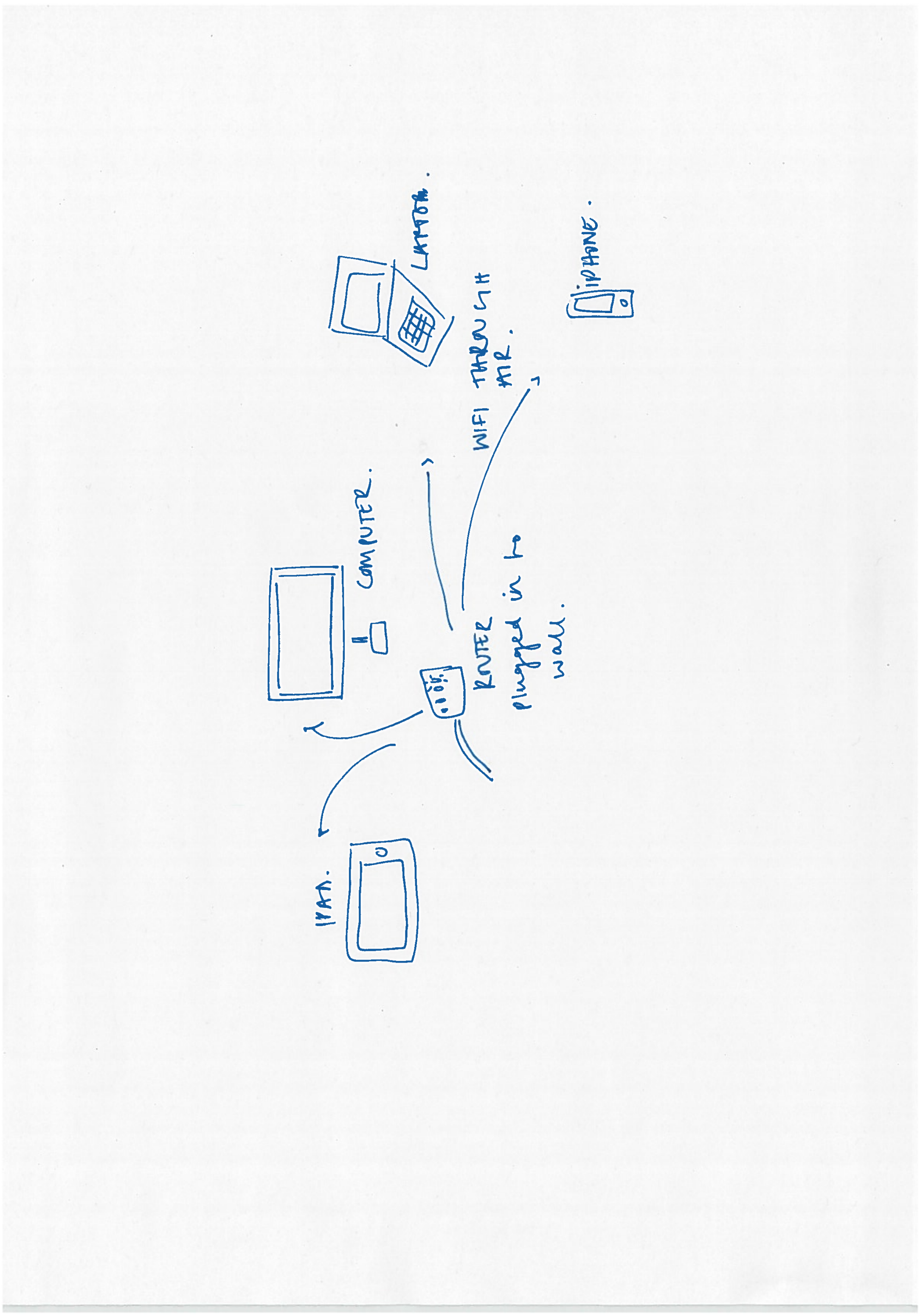


FIGURE D.1: P1's network diagram

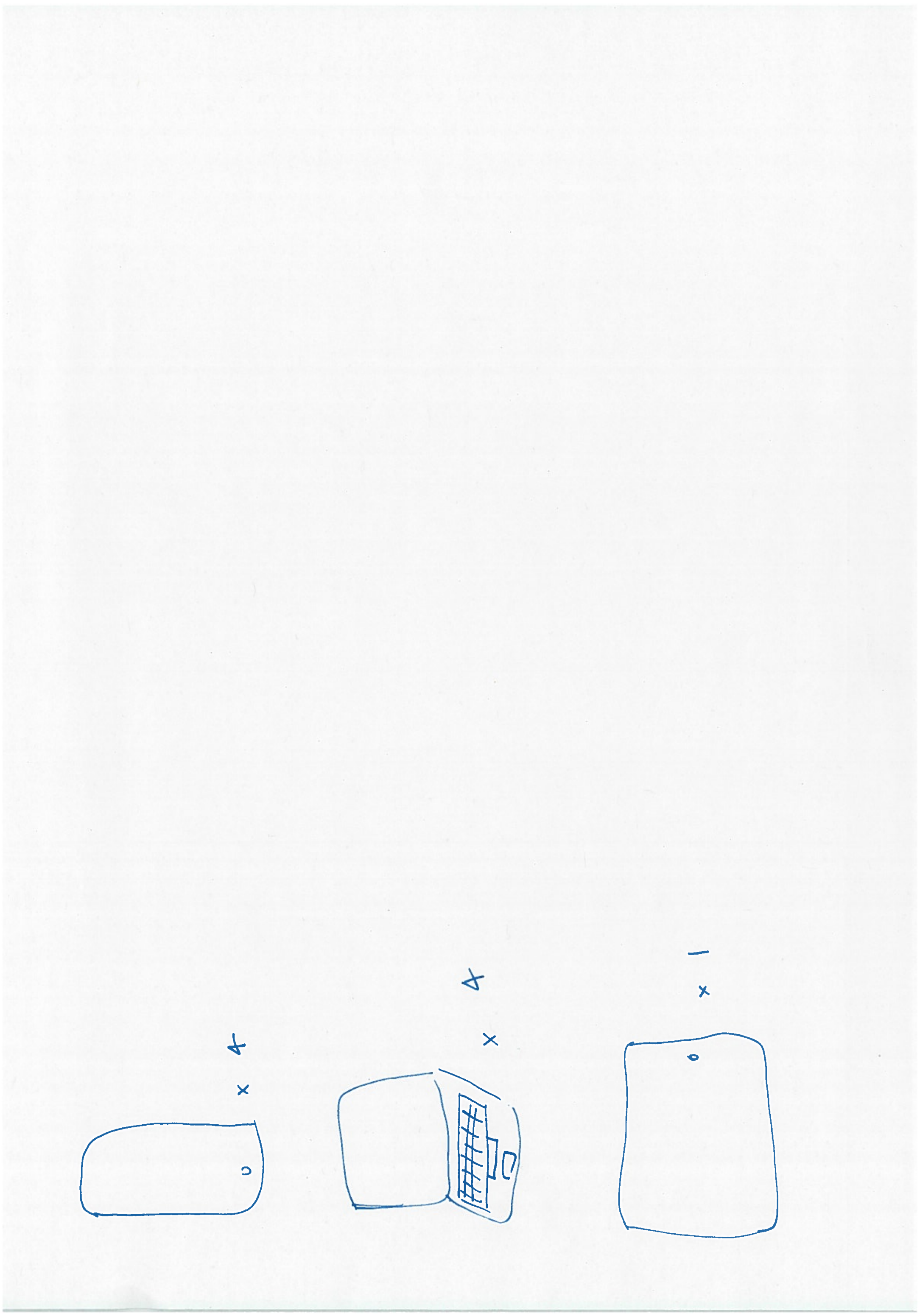


FIGURE D.2: P2's network diagram

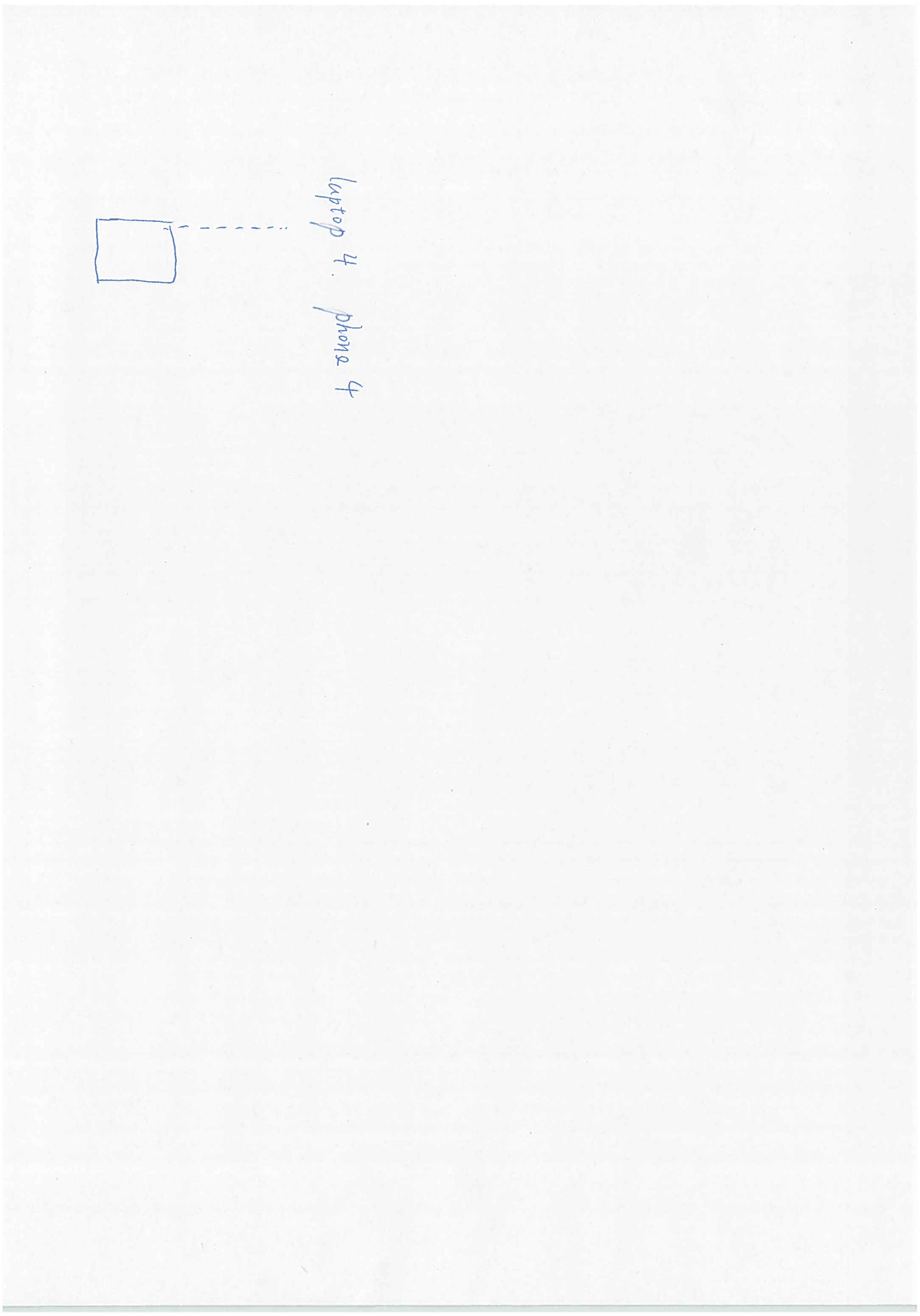


FIGURE D.3: P3's network diagram

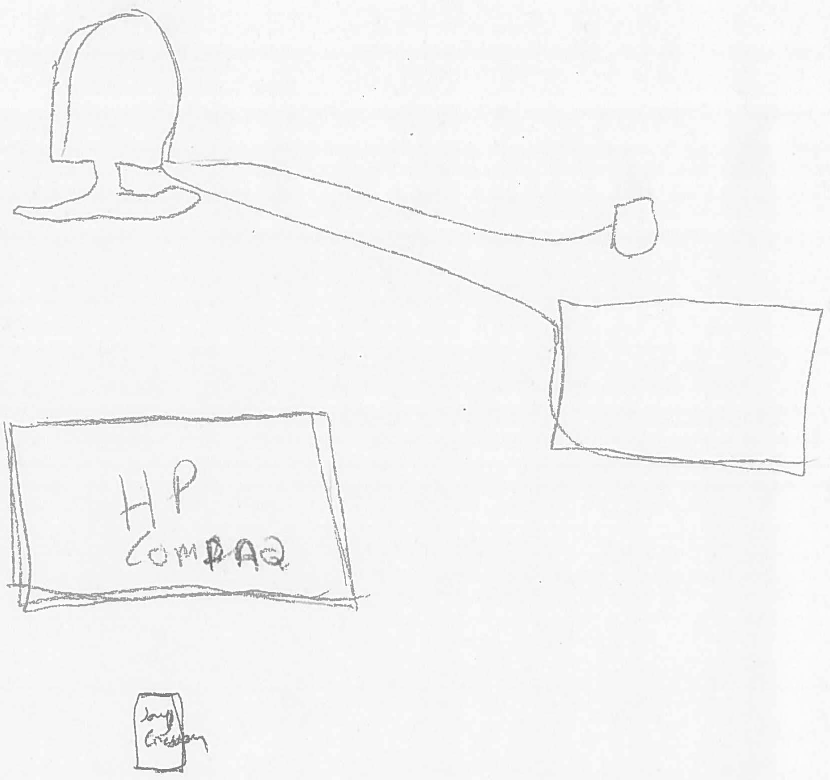


FIGURE D.4: P4's network diagram

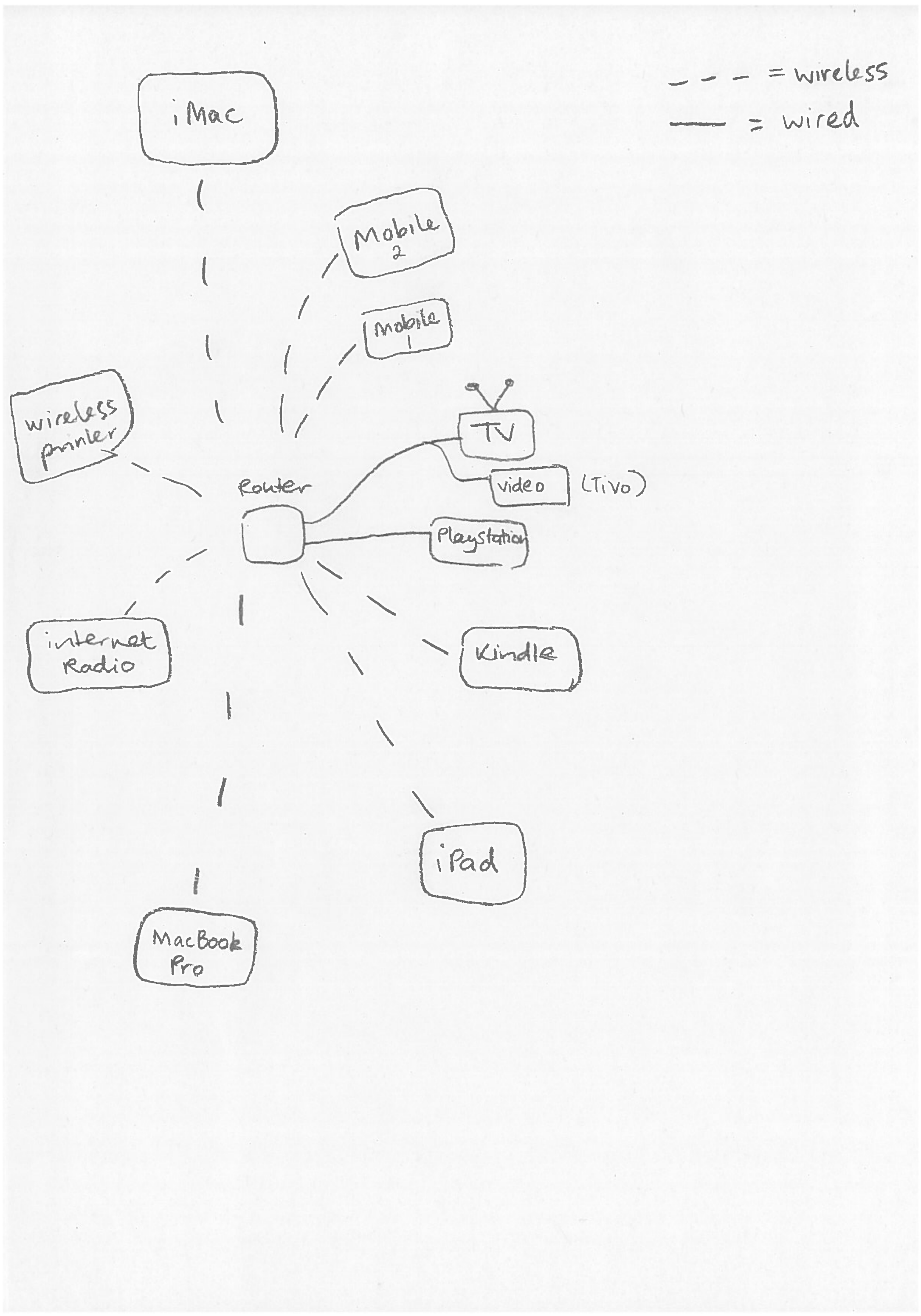


FIGURE D.5: P5's network diagram

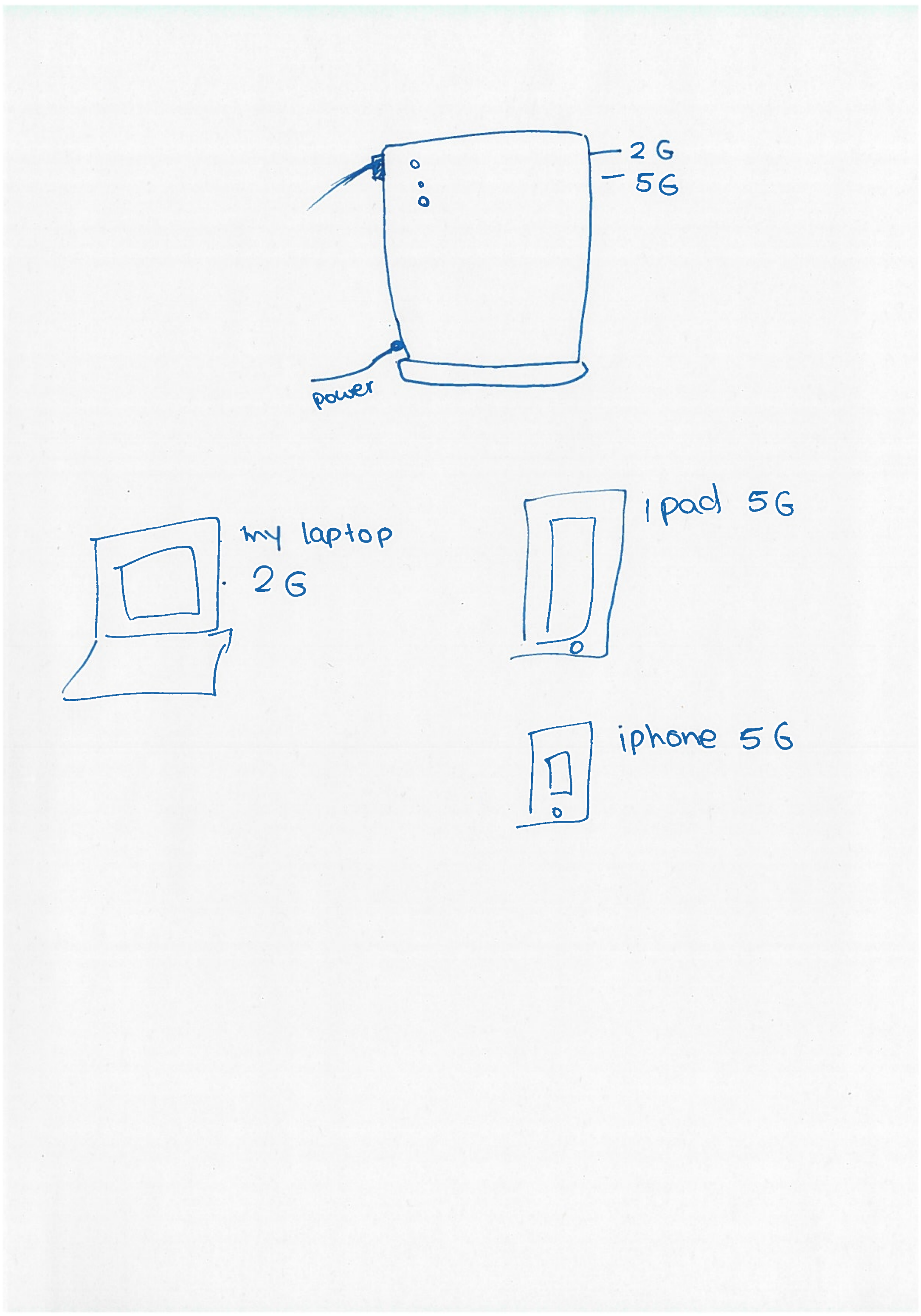


Figure D.6: P6's network diagram

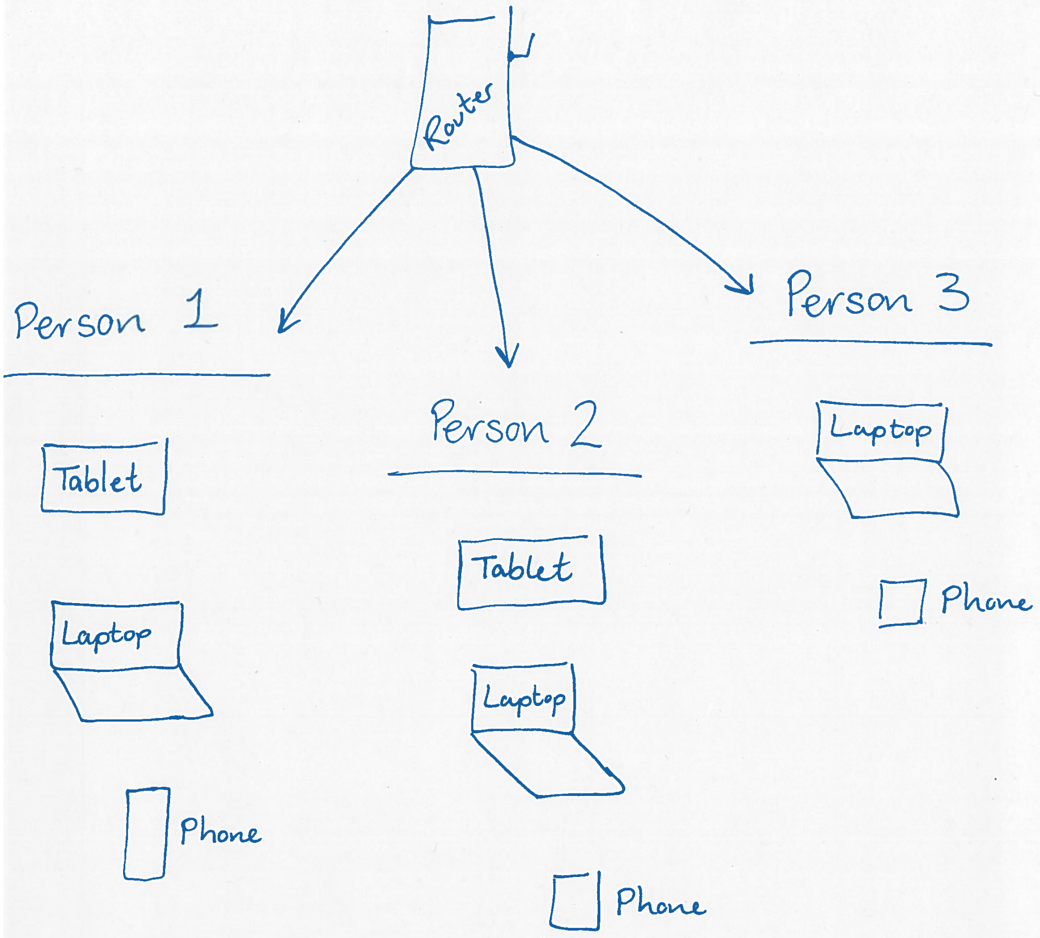


FIGURE D.7: P7's network diagram

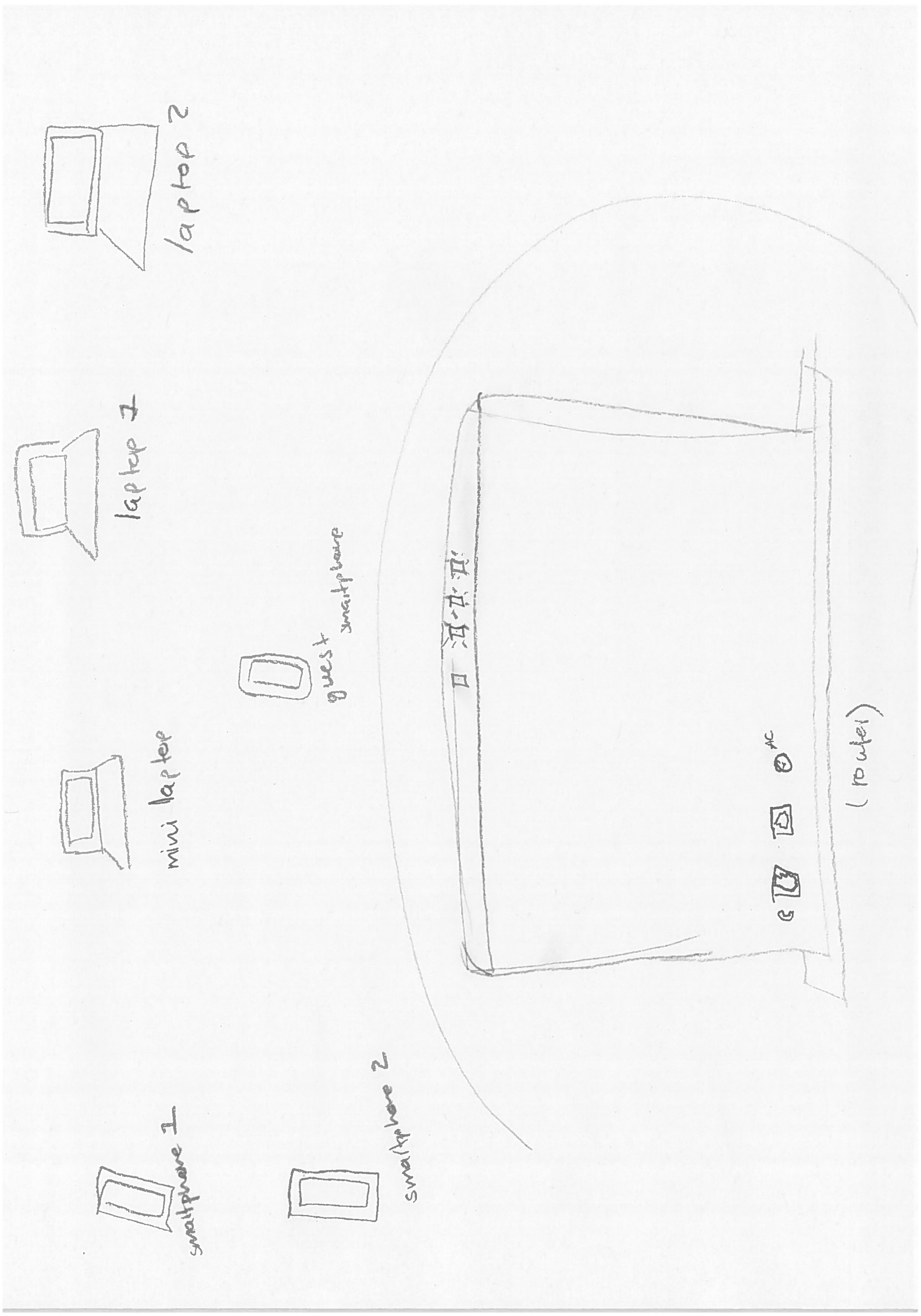


FIGURE D.8: P8's network diagram

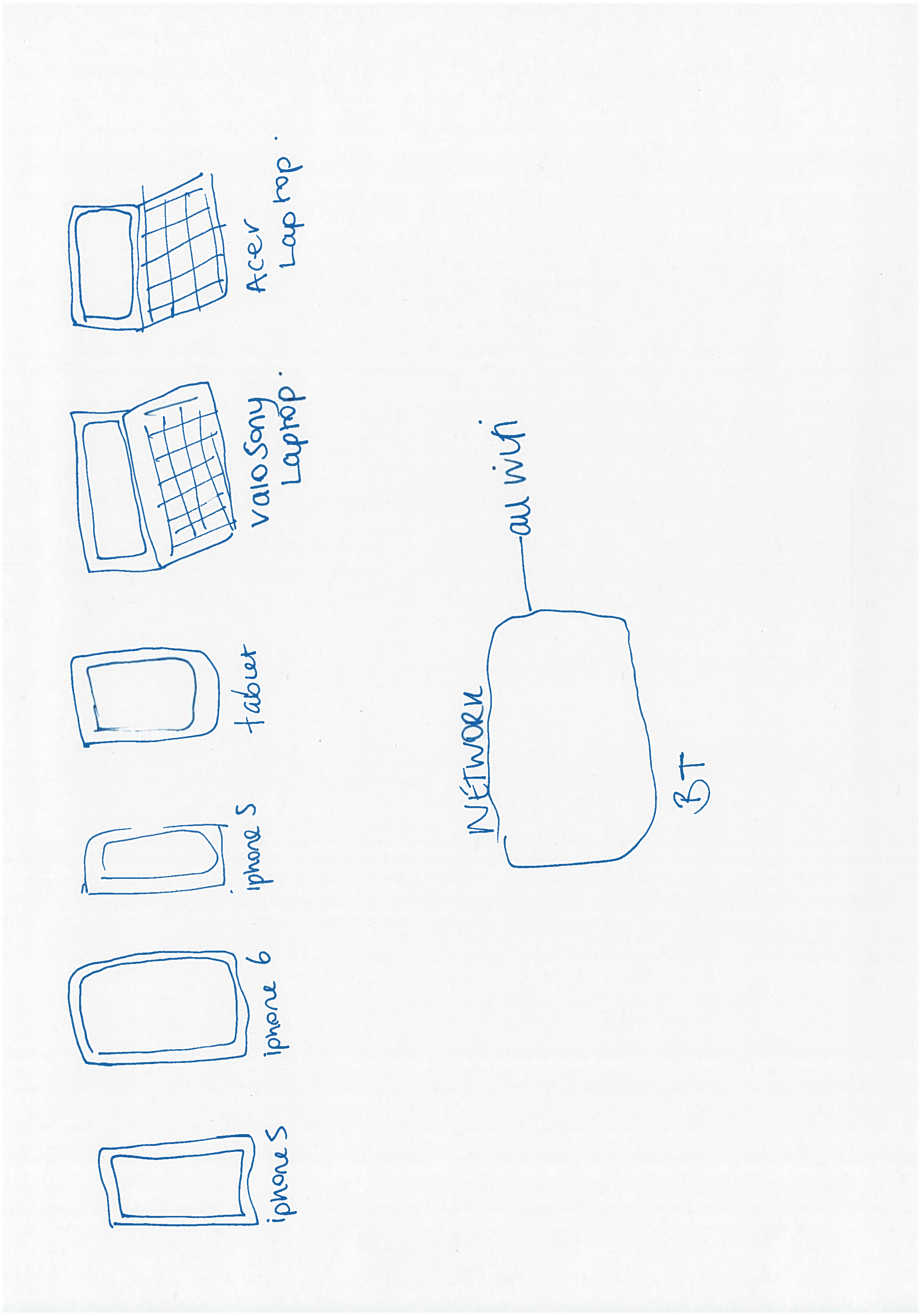


FIGURE D.9: P9's network diagram

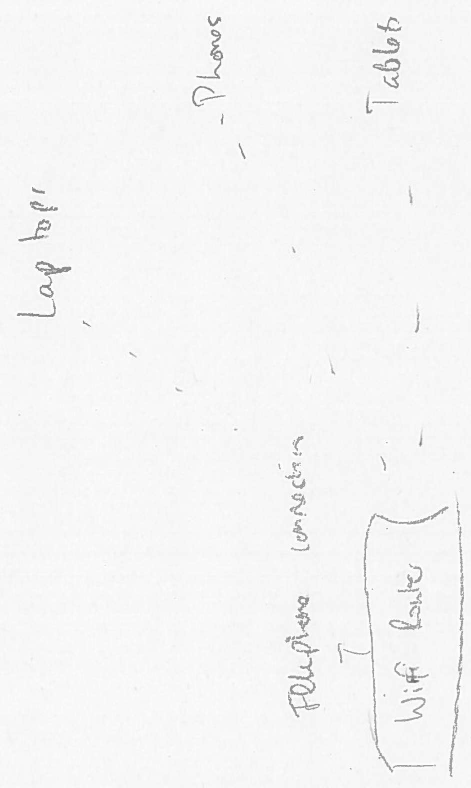


FIGURE D.10: P10's network diagram

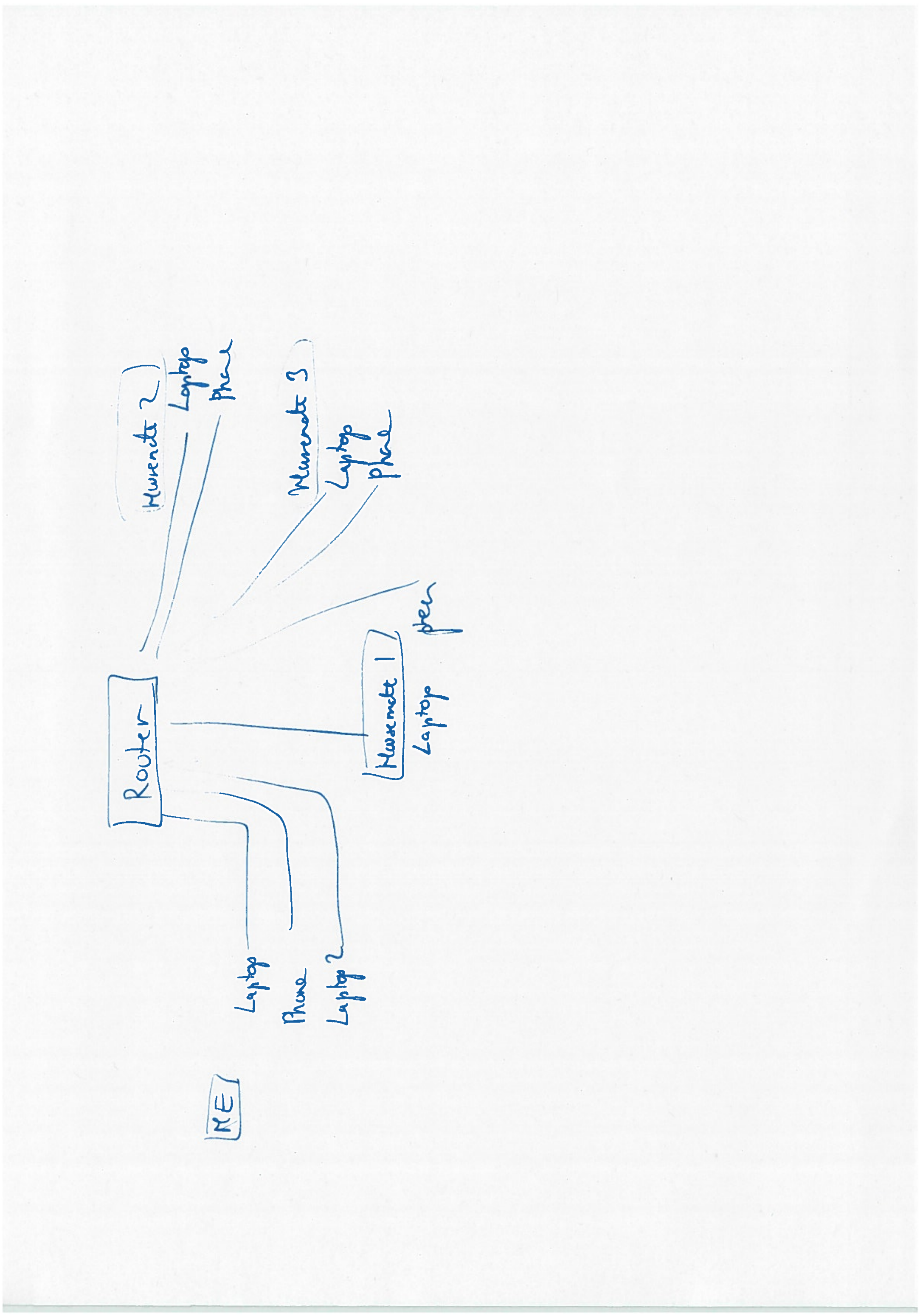


FIGURE D.11: P11's network diagram

Appendix E

SUS Histograms

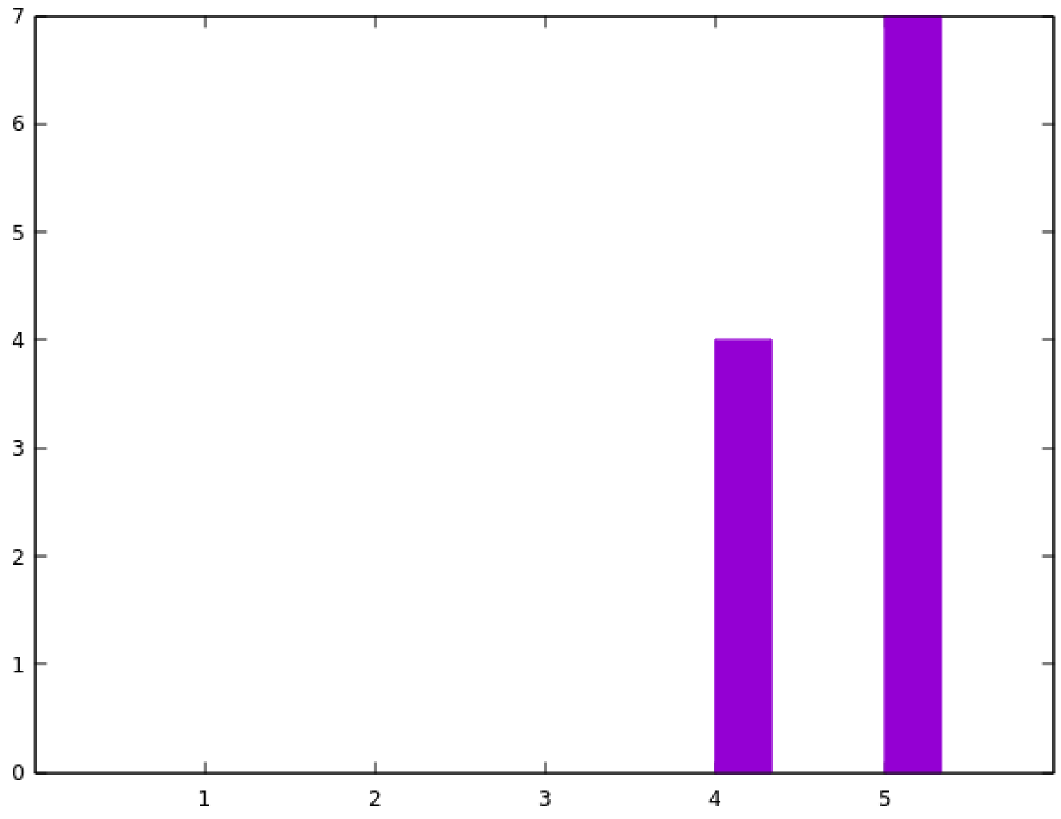


Figure E.1: SUS Question 1 responses

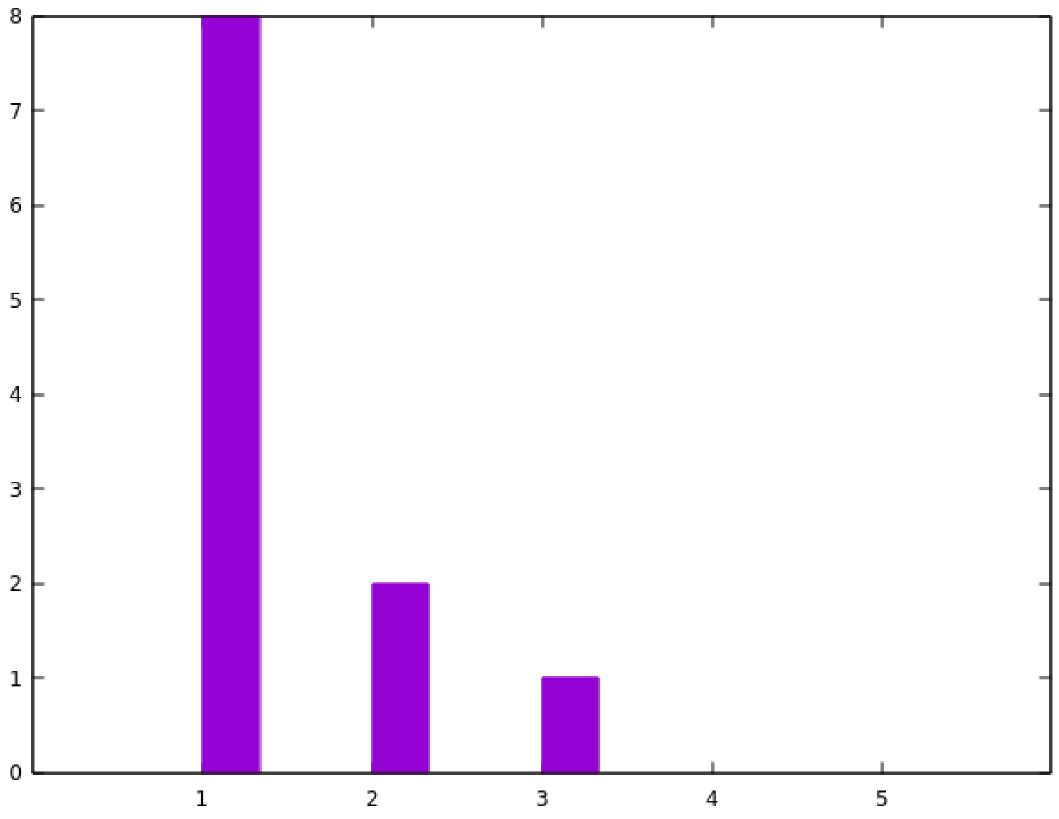


FIGURE E.2: SUS Question 2 responses

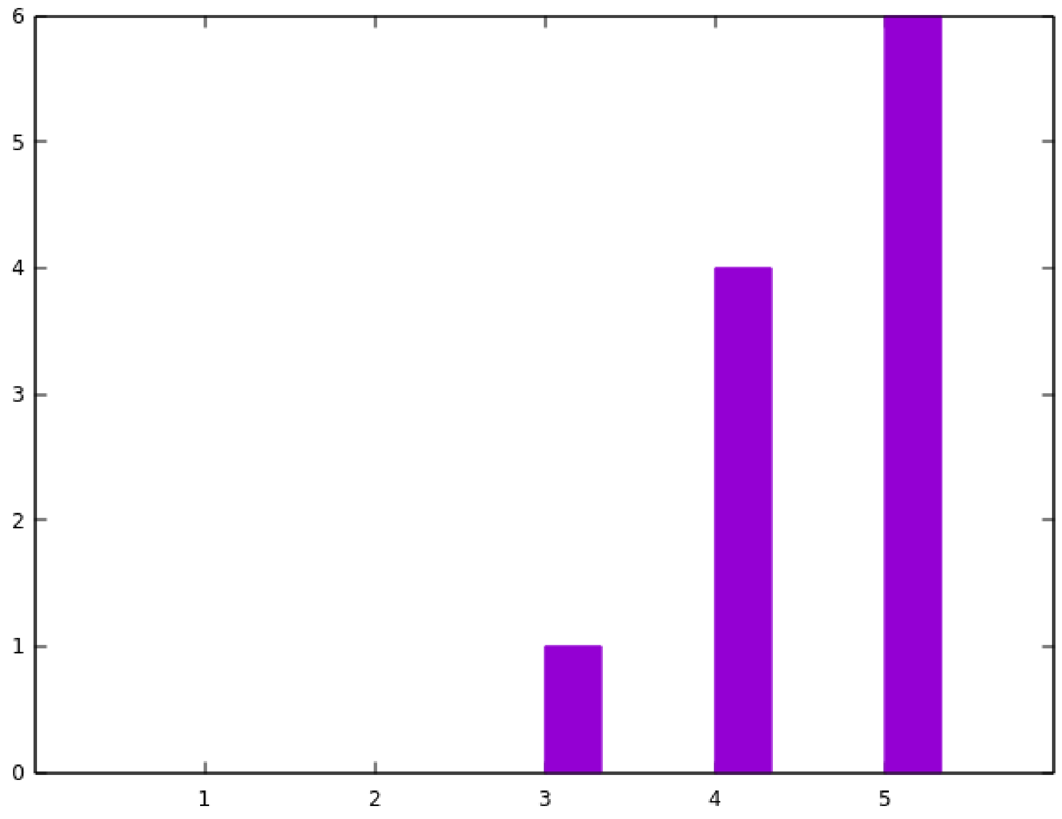


Figure E.3: SUS Question 3 responses

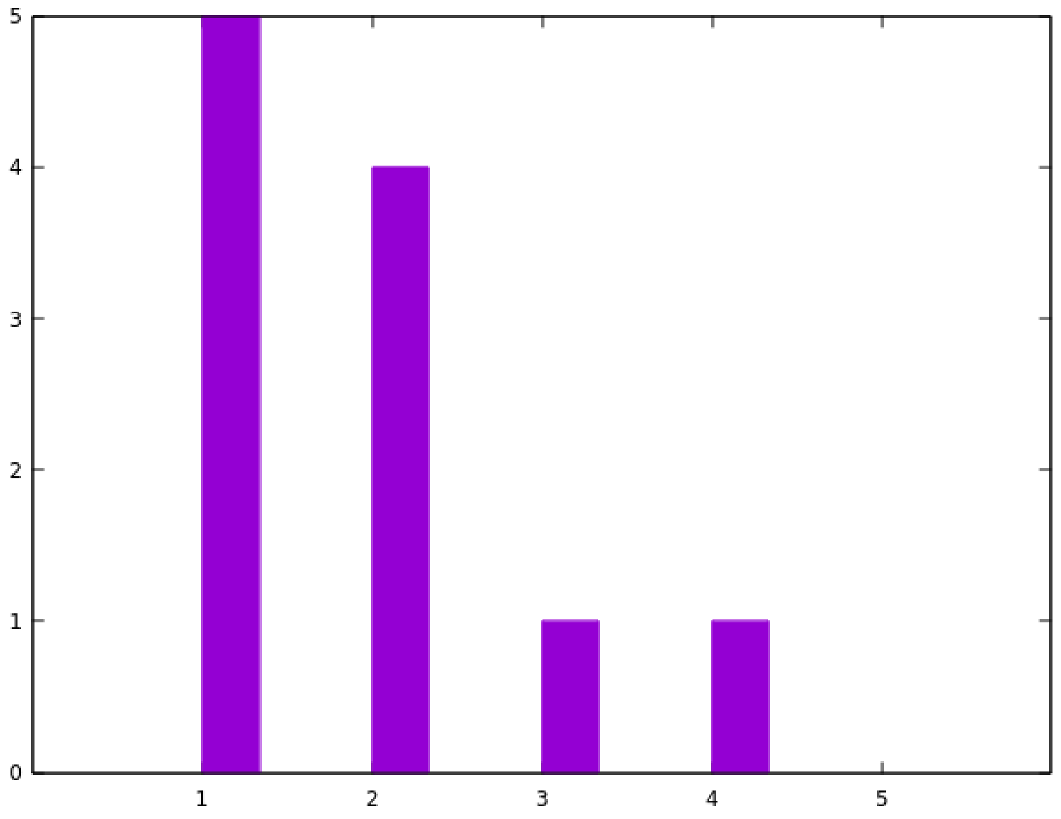


Figure E.4: SUS Question 4 responses

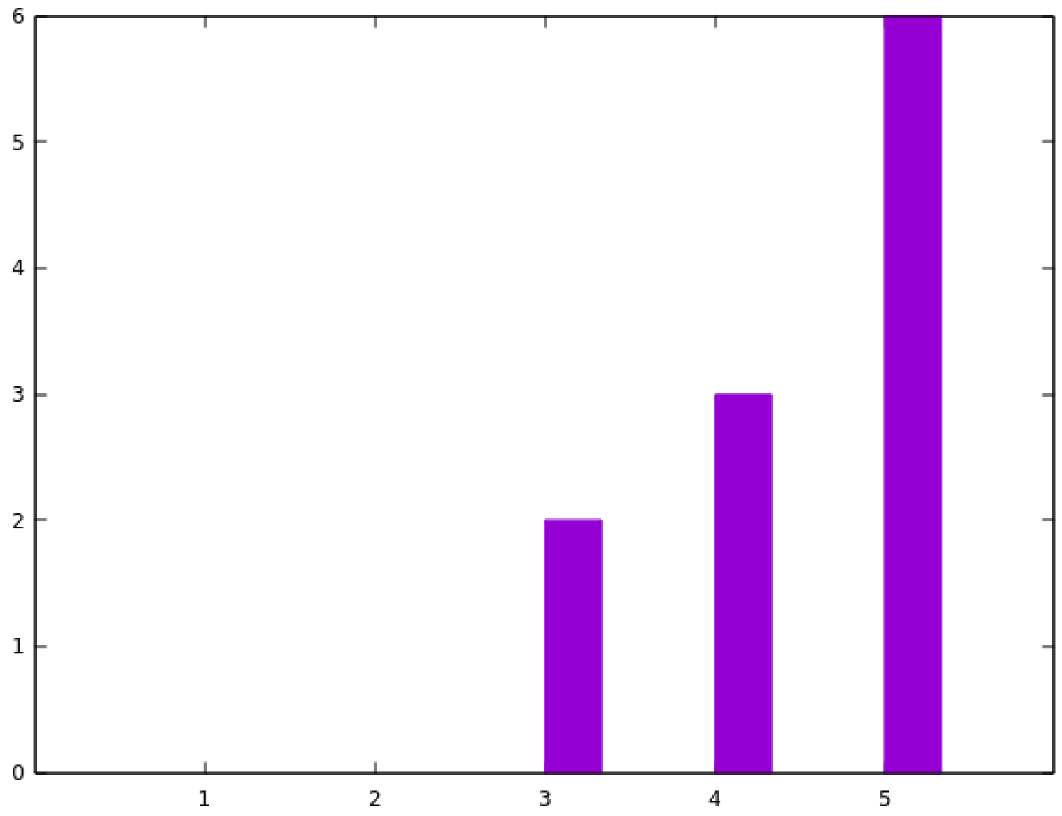


FIGURE E.5: SUS Question 5 responses

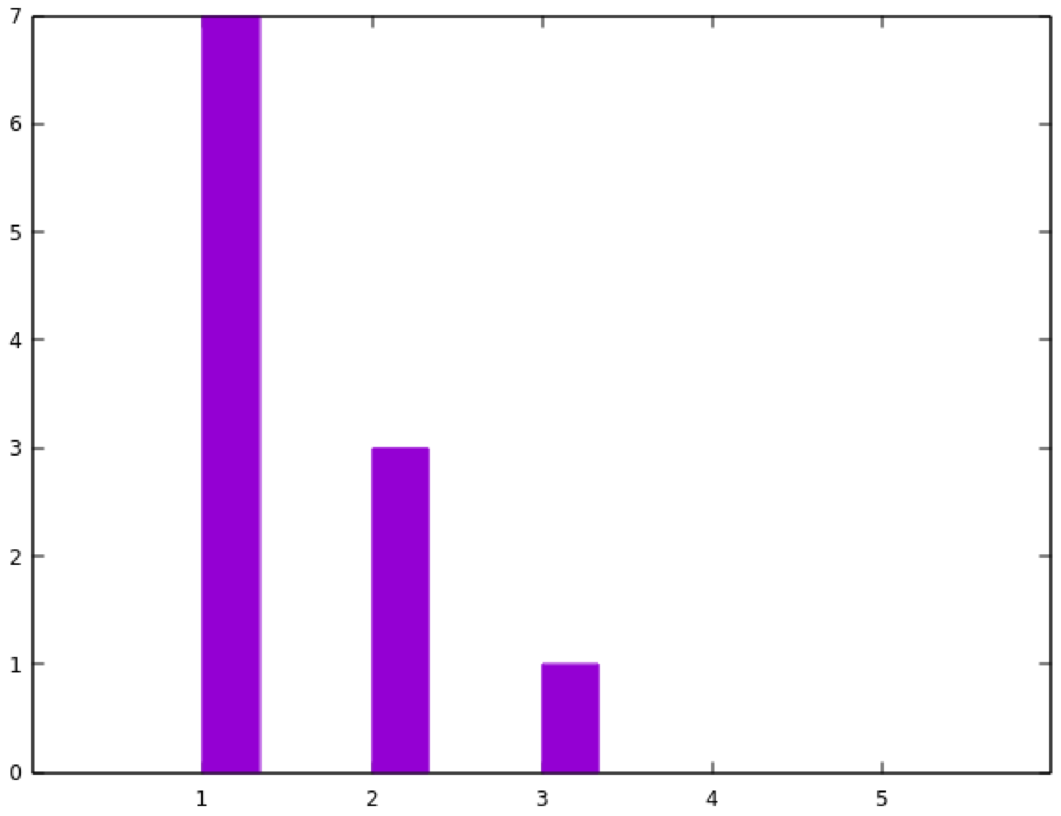


FIGURE E.6: SUS Question 6 responses

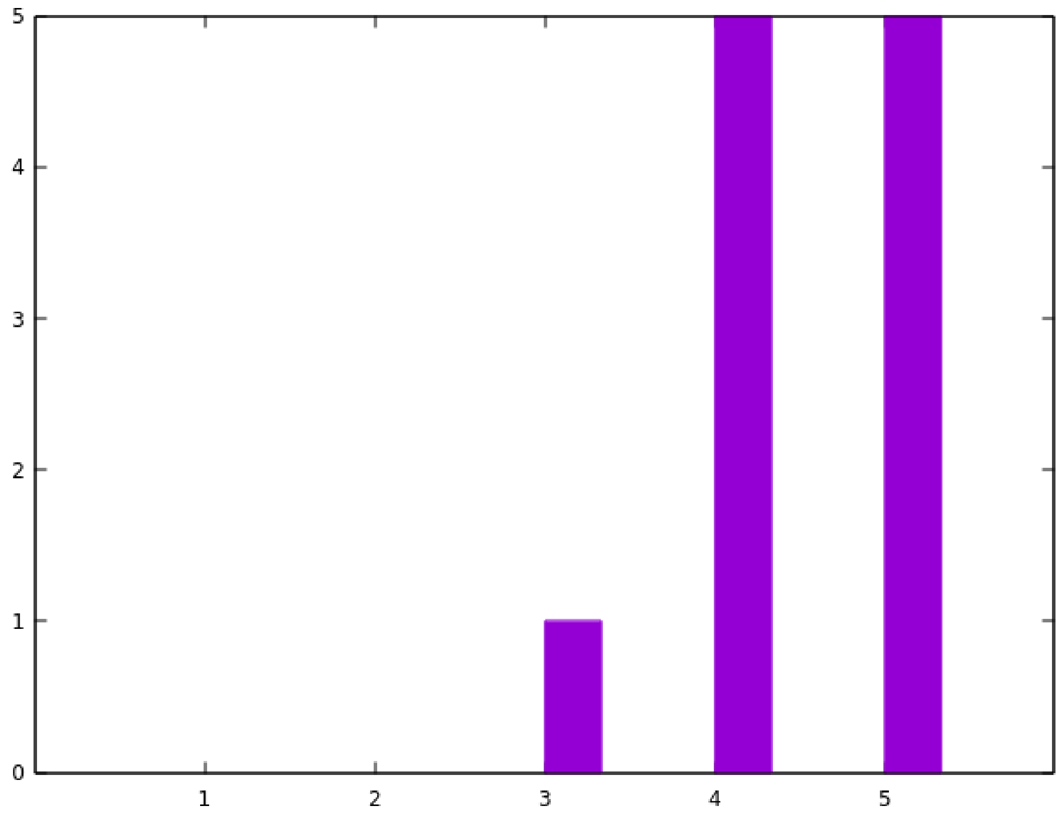


FIGURE E.7: SUS Question 7 responses

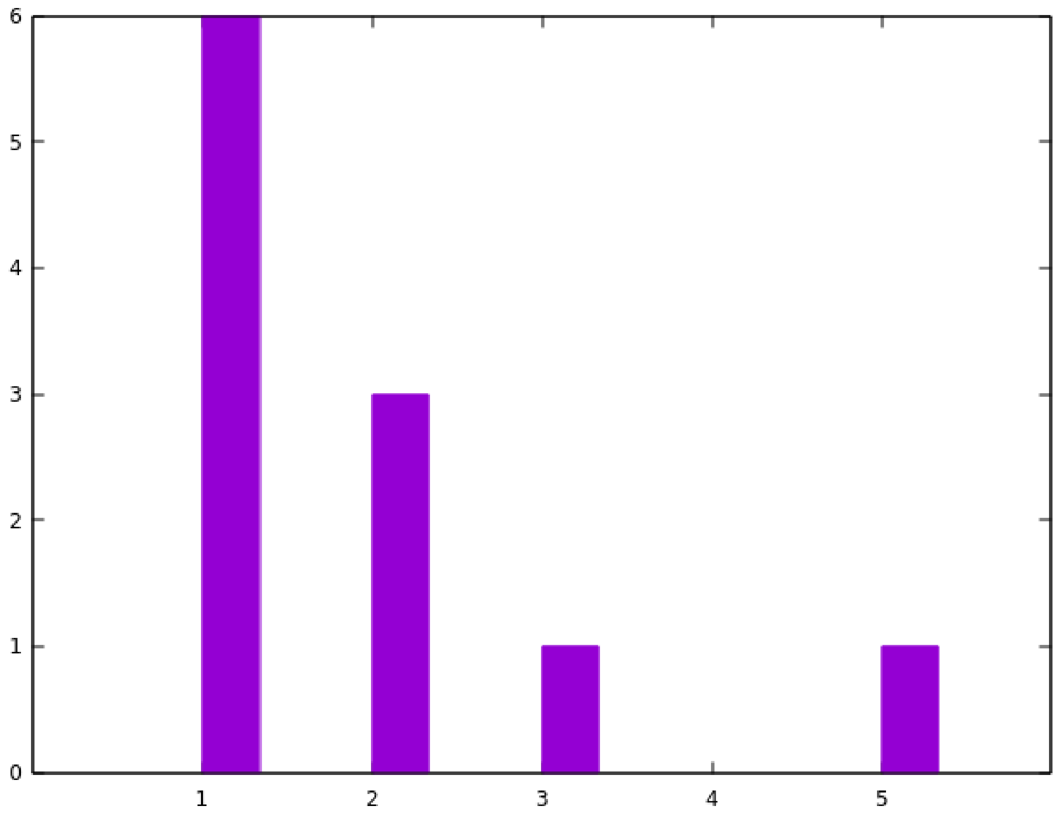


FIGURE E.8: SUS Question 8 responses

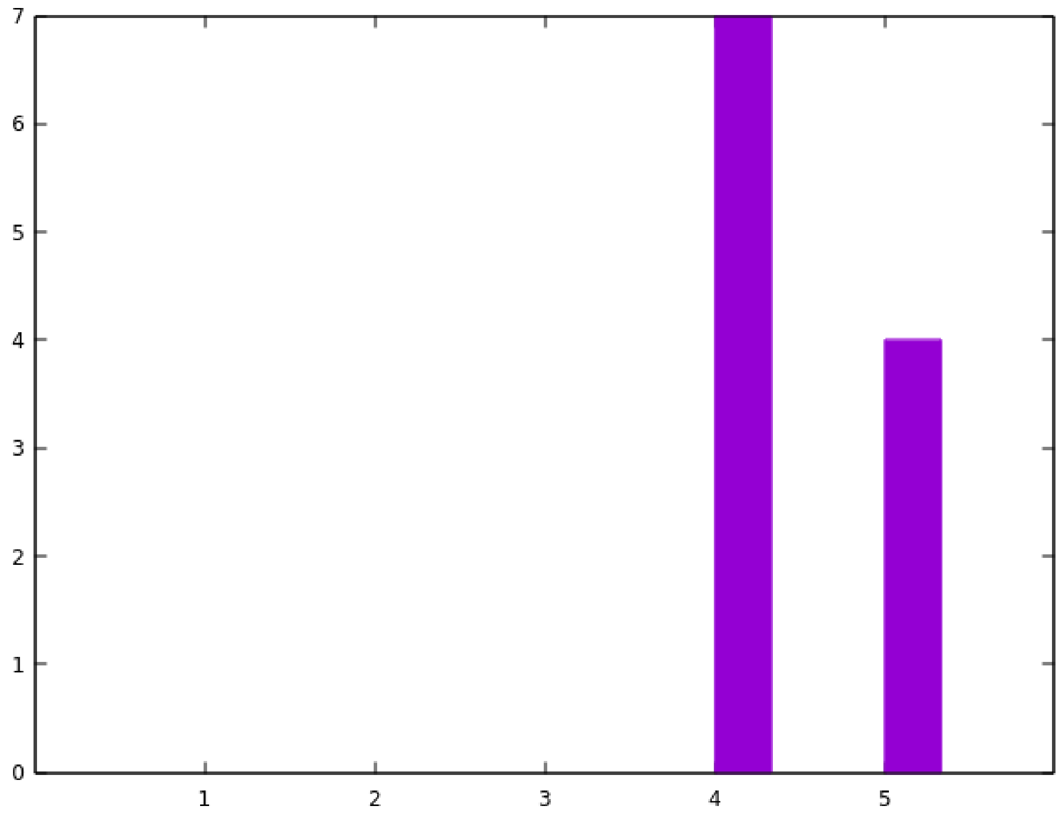


FIGURE E.9: SUS Question 9 responses

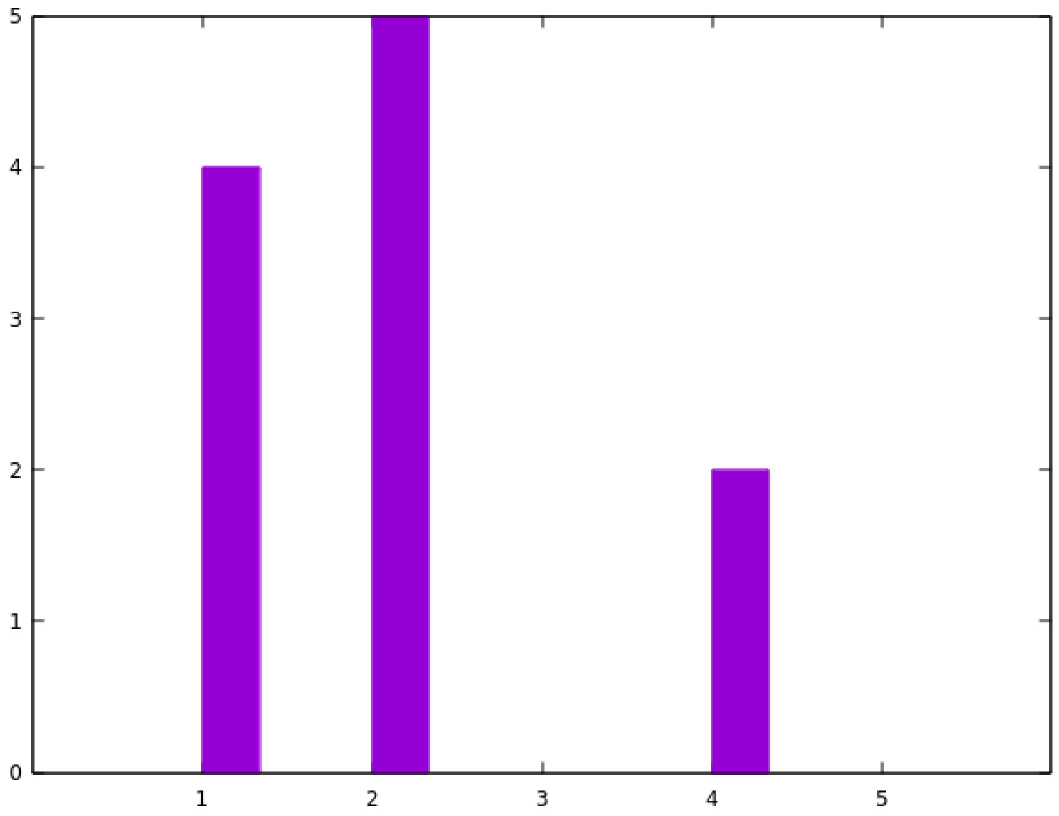


Figure E.10: SUS Question 10 responses

Bibliography

[1] Couchdb wiki. https://cwiki.apache.org/confluence/display/COUCHDB/Apache+CouchDB+Wiki.

[2] Nox classic. https://github.com/noxrepo/nox-classic/wiki.

[3] Pox wiki. https://openflow.stanford.edu/display/ONL/POX+Wiki.

[4] upstart - event-based init daemon. http://upstart.ubuntu.com/.

[5] Upstart intro, cookbook and best practises. http://upstart.ubuntu.com/cookbook/.

[6] Abowd, G. D., and Dix, A. J. Giving undo attention. *Interacting with Computers 4*, 3 (1992), 317–342.

[7] Abowd, G. D., and Mynatt, E. D. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI) 7*, 1 (2000), 29–58.

[8] Agarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V. N., and Voelker, G. M. Netprints: Diagnosing home network misconfigurations using shared knowledge. In *NSDI* (2009), vol. 9, pp. 349–364.

[9] Akella, A., Judd, G., Seshan, S., and Steenkiste, P. Self-management in chaotic wireless deployments. *Wireless Networks 13*, 6 (2007), 737–755.

[10] Alliance, W.-F. Wi-fi certified wi-fi protected setup, 2010.

[11] Archer Jr, J. E., Conway, R., and Schneider, F. B. User recovery and reversal in interactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS) 6*, 1 (1984), 1–19.

[12] Archer Jr, J. E., and Conway, R. W. Cope: A cooperative programming environment. Tech. rep., Cornell University, 1981.

[13] Balfanz, D., Durfee, G., Grinter, R. E., Smetters, D. K., and Stewart, P. Network-in-a-box: How to set up a secure wireless network in under a minute. In *USENIX Security Symposium* (2004), vol. 207, p. 222.

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

[15] Beaudouin-Lafon, M., and Karsenty, A. Transparency and awareness in a real-time groupware system. In *Proc. ACM Symposium on User Interface Software and Technology, UIST'92, Monterey (USA)* (nov 1992), ACM, pp. 171–180. Également Rapport de Recherche LRI 704, octobre 1991.

[16] Bell, G., Blythe, M., and Sengers, P. Making by making strange: Defamiliarization and the design of domestic technologies. *ACM Transactions on Computer-Human Interaction (TOCHI) 12*, 2 (2005), 149–173.

[17] Bellotti, V., and Edwards, K. Intelligibility and accountability: human considerations in context-aware systems. *Human–Computer Interaction 16*, 2-4 (2001), 193–212.

[18] Berlage, T. *Object-oriented application frameworks for graphical user interfaces: the GINA perspective.* Oldenbourg, 1993.

[19] Berlage, T. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Trans. Comput.-Hum. Interact. 1*, 3 (Sept. 1994), 269–294.

[20] Berlage, T., and Genau, A. A framework for shared applications with a replicated architecture. In *Proceedings of the 6th annual ACM symposium on User interface software and technology* (1993), ACM, pp. 249–257.

[21] Berlage, T., and Genau, A. From undo to multi-user applications. *Human Computer Interaction* (1993), 213–224.

[22] Blumenthal, M. S., and Clark, D. D. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology (TOIT) 1*, 1 (2001), 70–109.

[23] Bly, S., Schilit, B., McDonald, D. W., Rosario, B., and Saint-Hilaire, Y. Broken expectations in the digital home. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems* (New York, NY, USA, 2006), CHI EA '06, ACM, pp. 568–573.

[24] Blythe, M., and Monk, A. Notes towards an ethnography of domestic technology. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques* (2002), ACM, pp. 277–281.

[25] Boros, S. Policy-based network management with snmp.

[26] Brand, S. *How buildings learn: What happens after they're built.* Penguin, 1995.

[27] Brooke, J. Sus-a quick and dirty usability scale. *Usability evaluation in industry 189*, 194 (1996), 4–7.

[28] Brown, A., Mortier, R., and Rodden, T. Multinet: usable and secure wifi device association. *ACM SIGCOMM Computer Communication Review 42*, 4 (2012), 275–276.

[29] Brown, A., Mortier, R., and Rodden, T. Multinet: Reducing interaction overhead in domestic wireless networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 1569–1578.

[30] Brown, A., Mortier, R., and Rodden, T. An exploration of user recognition on domestic networks using netflow records. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication* (New York, NY, USA, 2014), UbiComp '14 Adjunct, ACM, pp. 903–910.

[31] Brown, A. B., and Patterson, D. A. Undo for operators: building an undoable e-mail store. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (2003), USENIX Association, pp. 1–1.

[32] Brundell, P., Crabtree, A., Mortier, R., Rodden, T., Tennent, P., and Tolmie, P. The network from above and below. In *Proceedings of the First ACM SIGCOMM Workshop on Measurements Up the Stack* (New York, NY, USA, 2011), W-MUST '11, ACM, pp. 1–6.

[33] Brush, A. It@ home: Often best left to professionals. In *Position paper for the CHI 2006 Workshop on IT@ Home* (2006).

[34] Calvert, K. L., Edwards, W. K., Feamster, N., Grinter, R. E., Deng, Y., and Zhou, X. Instrumenting home networks. *SIGCOMM Comput. Commun. Rev. 41*, 1 (Jan. 2011), 84–89.

[35] Calvert, K. L., Edwards, W. K., and Grinter, R. E. Moving toward the middle: The case against the end-to-end argument in . . . *IN HOME NETWORKING. SIXTH WORKSHOP ON HOT TOPICS IN NETWORKS* (2007).

[36] Cam-Winget, N., Housley, R., Wagner, D., and Walker, J. Security flaws in 802.11 data link protocols. *Communications of the ACM 46*, 5 (2003), 35–39.

[37] Carter, W. Fault detection and recovery algorithms for fault-tolerant systems. In *Proceedings of the EURO IFIP'79 conference* (1979).

[38] Cass, A. G., and Fern, C. S. Modeling dependencies for cascading selective undo. In *IFIP INTERACT 2005 Workshop on Integrating Software Engineering and Usability Engineering* (2005).

[39] Cass, A. G., and Fernandes, C. S. Using task models for cascading selective undo. In *Task Models and Diagrams for Users Interface Design.* Springer, 2007, pp. 186–201.

[40] Cass, A. G., Fernandes, C. S. T., and Polidore, A. An empirical evaluation of undo mechanisms. In *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles* (New York, NY, USA, 2006), NordiCHI '06, ACM, pp. 19–27.

[41] Chandra, R., and Bahl, P. Multinet: Connecting to multiple ieee 802.11 networks using a single wireless card. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* (2004), vol. 2, IEEE, pp. 882–893.

[42] Chandra, R., Padmanabhan, V. N., and Zhang, M. Wifiprofiler: cooperative diagnosis in wireless LANs. In *Proceedings of the 4th international conference on Mobile systems, applications and services* (New York, NY, USA, 2006), MobiSys '06, ACM, pp. 205–219.

[43] Chen, D., and Sun, C. Undoing any operation in collaborative graphics editing systems. In *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work* (2001), ACM, pp. 197–206.

[44] Chesneau, B. Couchdbkit. http://couchdbkit.org/docs/api/.

[45] Chetty, M., Banks, R., Harper, R., Regan, T., Sellen, A., Gkantsidis, C., Karagiannis, T., and Key, P. Who's hogging the bandwidth: the consequences of revealing the invisible in the home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 659–668.

[46] Chetty, M., and Feamster, N. Refactoring network infrastructure to improve manageability: a case study of home networking. *ACM SIGCOMM Computer Communication Review 42*, 3 (2012), 54–61.

[47] Chetty, M., Haslem, D., Baird, A., Ofoha, U., Sumner, B., and Grinter, R. Why is my internet slow?: making network speeds visible. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011), ACM, pp. 1889–1898.

[48] Chetty, M., Kim, H., Sundaresan, S., Burnett, S., Feamster, N., and Edwards, W. K. ucap: An internet data management tool for the home. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015), ACM, pp. 3093–3102.

[49] Chetty, M., Sung, J.-Y., and Grinter, R. E. *How smart homes learn: The evolution of the networked home and household*. Springer, 2007.

[50] Choudhary, R., and Dewan, P. 'multi-user undo/redo. *Technical Report 125* (1992).

[51] Cole, I., Lansdale, M., and Christie, B. Dialogue design guidelines. In *Human factors of information technology in the office*. Wiley New York, 1985, pp. 212–241.

[52] Cowan, R. S. *More work for mother: The ironies of household technology from the open hearth to the microwave*, vol. 5131. Basic Books, 1983.

[53] Crabtree, A., Mortier, R., Rodden, T., and Tolmie, P. Unremarkable networking: The home network as a part of everyday life. In *Proceedings of the Designing Interactive Systems Conference* (New York, NY, USA, 2012), DIS '12, ACM, pp. 554–563.

[54] Crabtree, A., and Rodden, T. Domestic routines and design for the home. *Computer Supported Cooperative Work (CSCW) 13*, 2 (2004), 191–220.

[55] Crabtree, A., Rodden, T., Tolmie, P., Mortier, R., Lodge, T., Brundell, P., and Pantidi, N. House rules: the collaborative nature of policy in domestic networks. *Personal and Ubiquitous Computing 19*, 1 (2015), 203–215.

[56] Dean, J., and Ghemawat, S. Mapreduce: a flexible data processing tool. *Communications of the ACM 53*, 1 (2010), 72–77.

[57] Dewan, P. A tour of Suite user interface software. In *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology* (1990), ACM, pp. 57–65.

[58] Dewan, P., and Choudhary, R. Primitives for programming multi-user interfaces. In *Proceedings of the 4th annual ACM symposium on User interface software and technology* (1991), ACM, pp. 69–78.

[59] Dix, A., and Harrison, M. Principles and interaction models for window managers. In *Proceedings of the Second Conference of the British Computer Society, human computer interaction specialist group on People and computers: designing for usability* (1986), Cambridge University Press, pp. 352–366.

[60] Dix, A. J. *Formal Methods and Interactive Systems: Principles and Practice.* PhD thesis, The University of York (United Kingdom), 1987. AAIDX82063.

[61] Dix, A. J. *Formal methods for interactive systems*, vol. 16. Academic Press London, 1991.

[62] Edwards, W. K., and Grinter, R. E. At home with ubiquitous computing: Seven challenges. In *Proceedings of the 3rd International Conference on Ubiquitous Computing* (London, UK, UK, 2001), UbiComp '01, Springer-Verlag, pp. 256–272.

[63] Edwards, W. K., Grinter, R. E., Mahajan, R., and Wetherall, D. Advancing the state of home networking. *Commun. ACM 54* (June 2011), 62–71.

[64] Edwards, W. K., Igarashi, T., LaMarca, A., and Mynatt, E. D. A temporal model for multi-level undo and redo. In *Proceedings of the 13th annual ACM symposium on User interface software and technology* (2000), ACM, pp. 31–40.

[65] Edwards, W. K., and Mynatt, E. D. Timewarp: techniques for autonomous collaboration. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (1997), ACM, pp. 218–225.

[66] Edwards, W. K., Newman, M. W., and Poole, E. S. The infrastructure problem in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 423–432.

[67] Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems. In *Acm Sigmod Record* (1989), vol. 18, ACM, pp. 399–407.

[68] Elmore, B., Hamilton, S., and Ivaturi, S. Designing software for consumers to easily set up a secure home network. In *CHI'07 Extended Abstracts on Human Factors in Computing Systems* (2007), ACM, pp. 1735–1740.

[69] Engelbart, D. C. Nls teleconferencing features: The journal, and shared-screen telephoning. *Fall COMPCON 75 Digest of Papers* (1975), 173–177.

[70] Ensor, J. R., Ahuja, S., Horn, N., and Lucco, S. The rapport multimedia conferencing system-a software overview. In *Computer Workstations, 1988., Proceedings of the 2nd IEEE Conference on* (1988), IEEE, pp. 52–58.

[71] Forlizzi, J., and DiSalvo, C. Service robots in the domestic environment: A study of the roomba vacuum in the home. In *Proceedings*

of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction (New York, NY, USA, 2006), HRI '06, ACM, pp. 258–265.

[72] Gancarz, M. *Linux and the Unix philosophy.* Digital Press, 2003.

[73] Gibbens, M., Gniady, C., and Zhang, B. Towards eco-friendly home networking. In *2014 International Green Computing Conference (IGCC)* (2014), IEEE, pp. 1–10.

[74] Goldberg, D., and Richardson, C. Touch-typing with a stylus. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems* (1993), ACM, pp. 80–87.

[75] Gordon, R. F., Leeman, G. B., and Lewis, C. H. *Concepts and implications of interactive recovery.* IBM Thomas J. Watson Research Center, 1984.

[76] Gordon, R. F., Leeman Jr, G. B., and Lewis, C. H. Concepts and implications of undo for interactive recovery. In *Proceedings of the 1985 ACM annual conference on The range of computing: mid-80's perspective: mid-80's perspective* (1985), ACM, pp. 150–157.

[77] Gray, J. N., Lorie, R. A., Putzolu, G. R., and Traiger, I. L. Granularity of locks and degrees of consistency in a shared data base. In *IFIP Working Conference on Modelling in Data Base Management Systems* (1976), pp. 365–394.

[78] Grinter, R. E., Edwards, W. K., Chetty, M., Poole, E. S., Sung, J.-Y., Yang, J., Crabtree, A., Tolmie, P., Rodden, T., Greenhalgh, C., and Benford, S. The ins and outs of home networking: The case for useful and usable domestic networking. *ACM Trans. Comput.-Hum. Interact. 16*, 2 (June 2009), 8:1–8:28.

[79] Grinter, R. E., Edwards, W. K., Newman, M. W., and Ducheneaut, N. The work to make a home network work. In *ECSCW 2005* (2005), Springer, pp. 469–488.

[80] Grover, S., Park, M. S., Sundaresan, S., Burnett, S., Kim, H., Ravi, B., and Feamster, N. Peeking behind the nat: An empirical study of home networks. In *Proceedings of the 2013 Conference on Internet*

Measurement Conference (New York, NY, USA, 2013), IMC '13, ACM, pp. 377–390.

[81] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review 38*, 3 (2008), 105–110.

[82] Habibi Gharakheili, H., Exton, L., Sivaraman, V., Matthews, J., and Russell, C. Third-party customization of residential internet sharing using sdn. In *Telecommunication Networks and Applications Conference (ITNAC), 2015 International* (2015), IEEE, pp. 214–219.

[83] Handley, M. Why the internet only just works. *BT Technology Journal 24*, 3 (2006), 119–129.

[84] Hughes, J., O'Brien, J., and Rodden, T. Understanding technology in domestic environments: Lessons for cooperative buildings. In *Cooperative Buildings: Integrating Information, Organization, and Architecture*. Springer, 1998, pp. 248–261.

[85] ICANN. Available pool of unallocated ipv4 internet addresses now completely emptied. https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf, February 2011.

[86] IEEE. Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 2: Fast basic service set (bss) transition. *IEEE Std 802.11r-2008 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008)* (15 2008), 1 –126.

[87] IEEE. Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE P802.11-REVmb/D12, November 2011 (Revision of IEEE Std 802.11-2007, as amended by IEEEs 802.11k-2008, 802.11r-2008, 802.11y-2008,*

802.11w-2009, 802.11n-2009, 802.11p-2010, 802.11z-2010, 802.11v-2011, 802.11u-2011, and 802.11s-2011) (29 2012), 1 –2910.

[88] IEEE 802.11. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[89] Igarashi, T., Edwards, W. K., LaMarca, A., and Mynatt, E. D. An architecture for pen-based interaction on electronic whiteboards. In *Proceedings of the working conference on Advanced visual interfaces* (2000), ACM, pp. 68–75.

[90] J. Chris Anderson, Lehnardt, J., and Slater, N. Couchdb definitive guide. http://guide.couchdb.org/draft/index.html.

[91] J. Chris Anderson, Lehnardt, J., and Slater, N. Couchdb definitive guide - standalone applications. http://guide.couchdb.org/draft/standalone.html.

[92] J. Chris Anderson, Lehnardt, J., and Slater, N. Couchdb definitive guide - views. http://guide.couchdb.org/draft/views.html.

[93] Karagiannis, T., Athanasopoulos, E., Gkantsidis, C., and Key, P. Homemaestro: Order from chaos in home networks, 2008.

[94] Kim, W.-Y., Whang, K.-Y., Lee, Y.-K., and Kim, S.-W. A recovery method supporting user-interactive undo in database management systems. *Information Sciences 114*, 1–4 (1999), 237 – 253.

[95] Kosinski, P. Pedit, a programmable editor. *IBM TJ Watson Research Center internal document* (1980).

[96] Kowtha, S., and Jiang, X. An n-state driven policy-based network management to control end-end network behaviors. In *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on* (2006), IEEE, pp. 5–pp.

[97] Kruskal, V. Managing multi-version programs with an editor. *IBM Journal of Research and Development 28*, 1 (1984), 74–81.

[98] Kuarsingh, V., Brzozowski, J., McQueen, J., and Grundemann, C. An incremental solution to advanced home networking.

[99] Kurlander, D., and Feiner, S. Editable graphical histories. In *IEEE Workshop on Visual Languages* (1988), Citeseer, pp. 127–134.

[100] Lampson, B. W. Bravo manual. *Alto User's Handbook* (1976), 31–62.

[101] Lantz, K. A. An experiment in integrated multimedia conferencing. In *Proceedings of the 1986 ACM conference on Computer-supported cooperative work* (1986), ACM, pp. 267–275.

[102] Lauwers, J. C., and Lantz, K. A. Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1990), ACM, pp. 303–311.

[103] Lee, M., Kim, Y., and Lee, Y. A home cloud-based home network auto-configuration using sdn. In *Networking, Sensing and Control (ICNSC), 2015 IEEE 12th International Conference on* (2015), IEEE, pp. 444–449.

[104] Leeman, Jr., G. B. A formal approach to undo operations in programming languages. *ACM Trans. Program. Lang. Syst. 8*, 1 (Jan. 1986), 50–87.

[105] Linksys. http://www.linksys.com/gb/wireless-routers/c/smart-wi-fi-wireless-routers/?sort=all&q=%3AsortByProductRank, January 2016.

[106] MacIntyre, B., and Feiner, S. Future multimedia user interfaces. *Multimedia systems 4*, 5 (1996), 250–268.

[107] MacIntyre, B., and Mynatt, E. D. Augmenting intelligent environments: Augmented reality as an interface to intelligent environments. In *Intelligent Environments Symposium* (1998), pp. 23–25.

[108] Mancini, R., Dix, A., and Levialdi, S. Reflections on undo. *University of Rome* (1996).

[109] Mancini, R., Dix, A. J., and Levialdi, S. Dealing with undo. In *Human-Computer Interaction INTERACT'97* (1997), Springer, pp. 703–705.

[110] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[111] Meng, C., Yasue, M., Imamiya, A., and Mao, X. Visualizing histories for selective undo and redo. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific* (1998), IEEE, pp. 459–464.

[112] Meyer, B. *Object-oriented software construction*, vol. 2. Prentice hall New York, 1988.

[113] Miniwatts-Marketing. Internet world stats. http://internetworldstats.com/stats.htm, June 2015.

[114] Mortier, R., Bedwell, B., Glover, K., Lodge, T., Rodden, T., Rotsos, C., Moore, A. W., Koliousis, A., and Sventek, J. Supporting novel home network management interfaces with openflow and nox. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2011), 464–465.

[115] Mortier, R., Rodden, T., Lodge, T., McAuley, D., Rotsos, C., Moore, A., Koliousis, A., and Sventek, J. Control and understanding: Owning your home network. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on* (Jan 2012), pp. 1–10.

[116] Mortier, R., Rodden, T., Tolmie, P., Lodge, T., Spencer, R., crabtree, A., Sventek, J., and Koliousis, A. Homework: Putting interaction into the infrastructure. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2012), UIST '12, ACM, pp. 197–206.

[117] Myers, B. A., Borison, E., Ferrency, A., McDaniel, R., Miller, R. C., Faulring, A., Kyle, B. D., Doane, P., Mickish, A., and Klimovitski, A. The amulet v3. 0 reference manual, 1997.

[118] Myers, B. A., and Kosbie, D. S. Reusable hierarchical command objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1996), ACM, pp. 260–267.

[119] Mynatt, E. D., Igarashi, T., Edwards, W. K., and LaMarca, A. Flatland: new dimensions in office whiteboards. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (1999), ACM, pp. 346–353.

[120] Norman, D. A. *The psychology of everyday things.* Basic books, 1988.

[121] Norman, D. A. *The design of everyday things: Revised and expanded edition.* Basic books, 2013.

[122] O'Brien, J., and Rodden, T. Interactive systems in domestic environments. *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '97* (1997), 247–259.

[123] O'Brien, J., and Shapiro, M. Undo for anyone, anywhere, anytime. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop* (2004), ACM, p. 31.

[124] Pantzar, M. Domestication of everyday life technology: Dynamic views on the social histories of artifacts. *Design Issues 13*, 3 (1997), pp. 52–65.

[125] Papagiannaki, K., Yarvis, M. D., and Conner, W. S. Experimental characterization of home wireless networks and design implications. In *INFOCOM* (2006).

[126] Patterson, J. F. Comparing the programming demands of single-user and multi-user applications. In *Proceedings of the 4th annual ACM symposium on User interface software and technology* (1991), ACM, pp. 87–94.

[127] Pediaditakis, D., Mostarda, L., Dong, C., and Dulay, N. Policies for self tuning home networks. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on* (2009), IEEE, pp. 29–32.

[128] Pefkianakis, I., Lundgren, H., Soule, A., Chandrashekar, J., Le Guyadec, P., Diot, C., May, M., Van Doorselaer, K., and Van Oost, K. Characterizing home wireless performance: The gateway view. In *Computer Communications (INFOCOM), 2015 IEEE Conference on* (2015), IEEE, pp. 2713–2731.

[129] Petersen, M. G. Remarkable computing: the challenge of designing for the home. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI'04* (2004), 1445–1448.

[130] Point-Topic. World broadband statistics - q1 2013. http://point-topic.com/services/global-broadband-statistics.

[131] Poole, E. S., Chetty, M., Grinter, R. E., and Edwards, W. K. More than meets the eye: transforming the user experience of home network management. In *Proceedings of the 7th ACM conference on Designing interactive systems* (New York, NY, USA, 2008), DIS '08, ACM, pp. 455–464.

[132] Poole, E. S., Chetty, M., Morgan, T., Grinter, R. E., and Edwards, W. K. Computer help at home: methods and motivations for informal technical support. In *Proc of CHI* (2009).

[133] Poole, E. S., Edwards, W. K., and Jarvis, L. The home network as a socio-technical system: Understanding the challenges of remote home network problem diagnosis. *Comput. Supported Coop. Work 18*, 2-3 (June 2009), 277–299.

[134] Prager, J., and Borkin, S. *POLITE project progress report.* IBM, 1979.

[135] Prakash, A., and Knister, M. J. Undoing actions in collaborative work. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1992), CSCW '92, ACM, pp. 273–280.

[136] Prakash, A., and Knister, M. J. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction (TOCHI) 1*, 4 (1994), 295–330.

[137] Rana, A. I., and O Foghlu, M. Policy-based network management in home area networks: interim test results. In *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on* (2009), IEEE, pp. 1–3.

[138] Rekimoto, J. Time-machine computing: a time-centric approach for the information environment. In *Proceedings of the 12th annual ACM*

symposium on User interface software and technology (New York, NY, USA, 1999), UIST '99, ACM, pp. 45–54.

[139] Ressel, M., and Gunzenhäuser, R. Reducing the problems of group undo. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work* (1999), ACM, pp. 131–139.

[140] Rhyne, J. R., and Wolf, C. G. Tools for supporting the collaborative process. In *Proceedings of the 5th annual ACM symposium on User interface software and technology* (1992), ACM, pp. 161–170.

[141] Rodden, T., and Benford, S. The evolution of buildings and implications for the design of ubiquitous domestic environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2003), ACM, pp. 9–16.

[142] Rose, B. Home networks: a standards perspective. *Communications Magazine, IEEE 39*, 12 (2001), 78–85.

[143] Salmon, B., Hady, F., and Melican, J. Learning to share: a study of sharing among home storage devices. *CMU Parallel Data Laboratory Technical Report* (2007).

[144] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS) 2*, 4 (1984), 277–288.

[145] Scherf, K. Consumer electronics association (cea) conference. *Parks Associates Panel on Home Networking* (2002).

[146] Seifried, T., Rendl, C., Haller, M., and Scott, S. Regional undo/redo techniques for large interactive surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2012), CHI '12, ACM, pp. 2855–2864.

[147] Shehan, E., and Edwards, W. Pinning the tail on the networked donkey: Why it@ home needs network visualization. In *Position paper for the CHI 2006 Workshop on IT@ Home* (2006), Citeseer.

[148] Shehan, E., and Edwards, W. K. Home networking and hci: what hath god wrought? In *Proceedings of the SIGCHI conference on Human*

factors in computing systems (New York, NY, USA, 2007), CHI '07, ACM, pp. 547–556.

[149] Smith, S. L., and Mosier, J. N. *Guidelines for designing user interface software.* Mitre Corporation Bedford, MA, 1986.

[150] Spenke, M., and Beilken, C. An overview of gina—the generic interactive application. In *User Interface Management and Design.* Springer, 1991, pp. 273–293.

[151] STAR, S. L. The ethnography of infrastructure. *AMERICAN BEHAVIORAL SCIENTIST 43*, 3 (1999), 377–391.

[152] Starner, T., Mann, S., Rhodes, B., Levine, J., Healey, J., Kirsch, D., Picard, R. W., and Pentland, A. Augmented reality through wearable computing. *Presence: Teleoperators and Virtual Environments 6*, 4 (1997), 386–398.

[153] Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., and Tatar, D. Wysiwis revised: early experiences with multiuser interfaces. *ACM Transactions on Information Systems (TOIS) 5*, 2 (1987), 147–167.

[154] Sun, C. Undo any operation at any time in group editors. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work* (2000), ACM, pp. 191–200.

[155] Sventek, J., Koliousis, A., Sharma, O., Dulay, N., Pediaditakis, D., Sloman, M., Rodden, T., Lodge, T., Bedwell, B., Glover, K., and Mortier, R. An information plane architecture supporting home network management. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on* (May 2011), pp. 1–8.

[156] Teitelman, W., Goodwin, J., and Bobrow, D. G. *Interlisp reference manual.* Xerox Palo Alto Research Centers, 1978.

[157] Tolmie, P., Crabtree, A., Egglestone, S., Humble, J., Greenhalgh, C., and Rodden, T. Digital plumbing: the mundane work of deploying ubicomp in the home. *Personal and Ubiquitous Computing 14*, 3 (2010), 181–196.

[158] Tolmie, P., Crabtree, A., Rodden, T., Greenhalgh, C., and Benford, S. Making the home network at home: Digital housekeeping. In *ECSCW 2007*. Springer, 2007, pp. 331–350.

[159] Tolmie, P., Pycock, J., Diggins, T., MacLean, A., and Karsenty, A. Unremarkable computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2002), CHI '02, ACM, pp. 399–406.

[160] TP-Link. 2011 connectivity report. https://thenextweb.com/uk/2011/12/09/uk-households-have-an-average-of-4-6-devices-connected-to-wifi, 2011.

[161] Twidle, K., Dulay, N., Lupu, E., and Sloman, M. Ponder2: A policy system for autonomous pervasive environments. In *Autonomic and Autonomous Systems, 2009. ICAS'09. Fifth International Conference on* (2009), IEEE, pp. 330–335.

[162] Verhofstad, J. S. M. Recovery techniques for database systems. *ACM Comput. Surv. 10*, 2 (June 1978), 167–195.

[163] Viehböck, S. Vulnerability note vu#723755 - wifi protected setup (wps) pin brute force vulnerability. https://www.kb.cert.org/vuls/id/723755, 12 2011.

[164] Vitter, J. S. Us&amp;r: A new framework for redoing (extended abstract). *SIGSOFT Softw. Eng. Notes 9*, 3 (Apr. 1984), 168–176.

[165] Vlissides, J. M. *Generalized graphical object editing*. PhD thesis, stanford university, 1990.

[166] Vlissides, J. M., and Linton, M. A. Unidraw: A framework for building domain-specific graphical editors. *ACM Transactions on Information Systems (TOIS) 8*, 3 (1990), 237–268.

[167] Wang, H., and Green, M. An event-object recovery model for object-oriented user interfaces. In *Proceedings of the 4th annual ACM symposium on User interface software and technology* (1991), ACM, pp. 107–115.

[168] Wang, H. J., Platt, J. C., Chen, Y., Zhang, R., and Wang, Y.-M. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI* (2004), vol. 4, pp. 245–257.

[169] Wang, Y.-M., Russell, W., and Arora, A. A toolkit for building dependable and extensible home networking applications. In *Proceedings of the 4th conference on USENIX Windows Systems Symposium-Volume 4* (2000), USENIX Association, pp. 10–10.

[170] Wang, Y.-M., Russell, W., Arora, A., Xu, J., and Jagannatthan, R. Towards dependable home networking: An experience report. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on* (2000), IEEE, pp. 43–48.

[171] Wang, Y.-M., Verbowski, C., Dunagan, J., Chen, Y., Wang, H. J., Yuan, C., and Zhang, Z. Strider: A black-box, state-based approach to change and configuration management and support. *Science of Computer Programming 53*, 2 (2004), 143–164.

[172] Want, R., Schilit, B., Adams, N., Gold, R., Petersen, K., Goldberg, D., Ellis, J., and Weiser, M. The parctab ubiquitous computing experiment. Tech. rep., Citeseer, 1995.

[173] Weiser, M. The computer for the 21st century. *Scientific american 265*, 3 (1991), 94–104.

[174] Wise, A. Little-jil 1.0 language report. Tech. rep., Technical Report 98-24, University of Massachusetts at Amherst, 1998.

[175] Wise, A., Cass, A. G., Lerner, B. S., McCall, E. K., Osterweil, L. J., and Sutton Jr, S. M. Using little-jil to coordinate agents in software engineering. In *Engineering of Software*. Springer, 2011, pp. 383–397.

[176] Wright, P., Monk, A., and Harrison, M. State, display and undo: a study of consistency in display based interaction. *University of York* (1992).

[177] Yang, J., and Edwards, W. K. Icebox: toward easy-to-use home networking. In *IFIP Conference on Human-Computer Interaction* (2007), Springer, pp. 197–210.

[178] Yang, J., and Edwards, W. K. A study on network management tools of householders. In *Proceedings of the 2010 ACM SIGCOMM Workshop on Home Networks* (New York, NY, USA, 2010), HomeNets '10, ACM, pp. 1–6.

[179] Yang, J., Edwards, W. K., and Haslem, D. Eden: supporting home network management through interactive visual tools. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology* (New York, NY, USA, 2010), UIST '10, ACM, pp. 109–118.

[180] Yang, Y. A new conceptual model for interactive user recovery and command reuse facilities. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1988), ACM, pp. 165–170.

[181] Yang, Y. Undo support models. *International Journal of Man-Machine Studies 28*, 5 (1988), 457 – 481.

[182] Yiakoumis, Y., Katti, S., Huang, T.-Y., McKeown, N., Yap, K.-K., and Johari, R. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (2012), ACM, pp. 1114–1119.

[183] Yiakoumis, Y., Yap, K.-K., Katti, S., Parulkar, G., and McKeown, N. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks* (2011), ACM, pp. 1–6.

[184] Young, R. M., and Whittington, J. Using a knowledge analysis to predict conceptual errors in text-editor usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1990), ACM, pp. 91–98.

[185] Zehl, S., Zubow, A., Wolisz, A., and Doering, M. Resfi: A secure framework for self organized radio resource management in residential wifi networks. *arXiv preprint arXiv:1601.00517* (2016).

[186] Zhou, C., and Imamiya, A. Object-based nonlinear undo model. In *Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International* (Aug 1997), pp. 50–55.

[187] Zhou, X., and Calvert, K. Lightweight privacy-preserving passive measurement for home networks. In *Communications (ICC), 2015 IEEE International Conference on* (June 2015), pp. 1019–1024.

[188] Zisiadis, D., Kopsidas, S., Varalis, A., and Tassiulas, L. Enhancing wps security. In *Wireless Days (WD), 2012 IFIP* (2012), IEEE, pp. 1–3.

[189] Zubow, A., Zehl, S., and Wolisz, A. Bigap–seamless handover in high performance enterprise ieee 802.11 networks. Tech. rep., Technical Report TKN-15-004, Telecommunication Networks Group, Technische Universität Berlin, 2015.