Networked Surfaces: A Novel LAN Technology

James William Scott
Churchill College, University of Cambridge

September 25, 2002

This dissertation is submitted for the degree of Doctor of Philosophy

Abstract

Networked Surfaces: A Novel LAN Technology

James William Scott

Networked Surfaces are a novel technology, which allows physical surfaces such as desks to be augmented in order to provide networking and other services to devices placed on top of them. The devices, which are required to be augmented with special hardware, may include notebook PCs, PDAs, peripherals, and other types of device habitually placed on surfaces.

When such a device is placed on a Networked Surface, a handshaking protocol is used to establish a connection between it and the appropriate services. These services may include low-speed and high-speed networking, the provision of power, and also the accurate estimation of the location of the device.

The concept of Networked Surfaces raises many issues in networking, which are explored in this thesis in the context of the OSI networking model. At the physical layer, the hardware required to provide connectivity to services is complex, involving a distributed architecture and use of particular conductive pad layouts on the surface and object. The implementation of a fully functional prototype is described.

At the link layer, methods for connection and disconnection detection are presented and evaluated. The high speed network used in the prototype is discussed, and includes a novel bus arbitration scheme appropriate to the Networked Surfaces environment.

The characteristics of the high speed Networked Surfaces network interface include the possibility of frequent connection and disconnection. This raises issues at the network and transport layers, including those of support for mobility, and of multiple network interfaces. Also discussed are methods of improving the performance of the TCP protocol in these conditions, using a "smart link layer" approach.

Finally, the provision and use of location information is presented. The accuracy of this information is found to be comparable with the best current indoor location technologies, with orientation information also provided with a high degree of accuracy. Integration with other systems in the field of "context-aware" computing is described, as well as some applications that Networked Surfaces can enable in this field.

To my father Tony, my mother Eva, and my brother Mike.

Preface

Except where noted in the text, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration.

I hereby declare that this dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other University. I further state that no part of my dissertation has already been or is being concurrently submitted for any such degree, diploma or other qualification.

This dissertation is comprised of 54,000 words and 63 figures.

James William Scott
September 25, 2002

Acknowledgements

I am indebted to three people for support with this work. Frank Hoffmann, my project partner, taught me much about hardware design, performed a lot of the hardware grudge work in building the Networked Surface prototype, and was a pleasure to work with. Glenford Mapp, my "proxy" supervisor, offered useful advice at every stage, and helped me identify and focus on the most fruitful issues. Andy Hopper, my supervisor, provided me with a clear road towards my PhD, sage advice on research issues, and also helped to locate funding, particularly for my final year. Thank you all.

I would also like to thank many people at the Laboratory for Communications Engineering and at AT&T Laboratories Cambridge for support with various stages of this work. In particular, Mike Hazas provided a sounding board for many ideas, as well as pedantically proofreading and insightfully commenting on various draft publications including this work. Ant Rowstron, formerly of the LCE, contributed valuable ideas in the original stages of the project. Also, Robert Harle, Robert Headon, Eli Katsiri, Diego López de Ipiña and James Weatherall all offered useful comments about interactions between my research and their own work.

Finally, I would like to gratefully acknowledge funding from the Schiff Foundation of Cambridge and from AT&T Laboratories Cambridge.

Publications

This dissertation is based in part on work included in the following publications:

James Scott and Frank Hoffmann. Networked Surfaces. In *Proceedings of Multi-Service Networks 2000*, Abingdon, Oxfordshire, England, July 2000.[1]

James Scott, Frank Hoffmann, Glenford Mapp, Michael D. Addlesee and Andy Hopper. Networked Surfaces: A New Concept in Mobile Networking. In *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, California, USA, December 2000.[2]

Frank Hoffmann, James Scott, Michael D. Addlesee, Glenford Mapp, and Andy Hopper. Data Transport on the Networked Surface. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, Tampa, Florida, USA, November 2001.[2]

James Scott, Frank Hoffmann, Glenford Mapp, Michael D. Addlesee and Andy Hopper. Networked Surfaces: A New Concept in Mobile Networking. To appear in *ACM Mobile Networks and Applications*, 7(5), October 2002.[2]

Frank Hoffmann and James Scott. Location of Mobile Devices Using Networked Surfaces. To appear in *Proceedings of Ubicomp 2002*, Göteborg, Sweden, September 2002.[3]

[1]The first author was awarded the British Computer Society Young Researcher Prize for this presentation.
[2]The first two authors contributed equally to this work.
[3]Both authors contributed equally to this work.

Contents

Figures

Tables

Chapter 1

Introduction

A Networked Surface is a novel concept in local area networking. It may be defined as follows:

> A surface (such as a desk or floor) is described as a *Networked Surface*[1] if a suitably augmented object[2] can acquire connectivity to networking and/or power infrastructure, simply by being in physical contact with that surface.

By "physical contact," it is meant that the object may occupy any position and orientation on the Surface, making the process of connecting to such a Surface transparent to the user. While designs with limitations on the placement of objects are possible, they do not provide a guarantee of transparency to the user.

This chapter will examine the position of Networked Surfaces in the field of networking, and outline their useful properties. The research presented in this thesis will also be introduced.

1.1 Networking Technologies

Current networking technologies can be classified as "wired" or "wireless." The Networked Surface concept aims to incorporate the best qualities of both.

[1] "Networked Surface" is abbreviated to "Surface" occasionally in this thesis. This should not be confused with "surface," which is used to describe a physical surface, including the physical surface portion of a Networked Surface system.

[2] The word "object" is used in this thesis to denote a device which has been augmented with a Networked Surface interface.

1.1.1 Wired Networks

Wired networks require a physical medium for network traffic, which must be attached to each node. Bandwidth may be plentiful, as the entire capacity of the medium can be dedicated to each device using high-speed switches. Overloaded networks can be easily re-provisioned to give more bandwidth by adding more infrastructure.

However, the need to be physically attached to the network implies a lack of mobility; when mobile, the device would not be networked, and the user must manually connect and disconnect wiring, which makes mobility inconvenient. Being wired for networking means that power can be provided with no additional loss of mobility, as the device is already connected to cabling.

1.1.2 Wireless Networks

With wireless networking, bandwidth is inherently shared inside "cells," which are delimited by the range of the transmitted signals. Mobility is "free" within a cell, but there is the problem of handover if the mobile devices cross cell boundaries.

One key drawback with wireless systems is fulfilling their power requirements; batteries must be charged from time to time, causing a loss of mobility during these periods. In addition, the available bandwidth is limited, so populating the network more and more densely will eventually cause the network to be overloaded, without the ability to reprovision at a higher bandwidth.

1.1.3 Networked Surfaces

Networked Surfaces attempt to combine the best qualities of both wired and wireless networking. On one hand mobility is supported, because the use, transport, and connection of wiring is not required. On the other hand, a wired network is used, with the potential for high and dedicated bandwidths, and power may be provided to charge batteries.

However, Networked Surfaces are only useful in cases where objects are placed on physical surfaces, either during use or for storage. For example, a notebook PC is often used on top of surfaces, so networking and power can be provided "on-line," i.e. during use. A PDA, however, may be designed for hand-held use, making a Surface only useful for "off-line" activities such as filesystem synchronisation and battery recharging. Also, Surfaces may only be useful in particular environments, as they require specialised hardware to be located anywhere that use is expected, unlike in a wireless network where a single base station can cater for a large area.

1.2 Properties of Networked Surfaces

This section outlines the properties of Networked Surfaces, which go far beyond networking alone.

1.2.1 Generic Connectivity

Networked Surfaces operate by forming links between the Surface and objects, which are akin to the wires in a cable. These links are very low-level, and are not specific to such details as the network type, or even the physical layer schemes used for networking (e.g. modulation). As such, many types of network may be formed using Surfaces, including computer-peripheral networks such as RS-232 or USB, or computer-computer networks such as Ethernet. Power can also be provided, as discussed below.

This generic connectivity allows many different object types to be supported by a single Surface, ranging from devices only requiring power, to peripherals, to computers such as PDAs or PCs. Some devices, such as notebook PCs, would benefit from the mobility aspects of the Surface. Other devices which are normally immobile, such as peripherals, may simply benefit from the convenience of not requiring wiring.

It is also worth noting that Surfaces can be designed to be upgradeable, to include new types of network supporting a different class of objects, or to provide multiple instances of a particular network in order to increase bandwidth. This property is present in the prototype Surface described later, in that much of the hardware and software is generic and would not need to be replaced if a new service were added.

1.2.2 Convenience

The Networked Surface is a technology of convenience in many ways. Firstly, the act of connecting an object is made very convenient. Secondly, the user of that object does not need to transport wiring. Thirdly, Networked Surface–enabled workspaces will not become cluttered with networking or power cables.

In addition, Surfaces provide transparent connectivity, in that the user does not need to know what type of network is being used. The integration of networking and power connectivity into the environment, so that they are constantly and transparently available, can be regarded as part of the field of “ubiquitous computing.”

1.2.3 Safety

As shall be seen, the prototype Surface uses electrical conduction via conducting pads to provide connectivity. This has safety implications, in that a user may touch those pads, or place conductive objects across several pads, whether deliberately or by mistake. These issues can be resolved by using low-power networks and current limiting hardware.

However, despite raising some safety issues of its own, the Surface can also provide for a safer environment for devices. In particular, since the Surface and object dynamically negotiate the connection provided, there is a low chance of misconnection; with wired networking, a user can accidentally or ignorantly misconnect devices, causing a lack of functionality at best or damage to hardware at worst.

1.2.4 Provision of Location Information

The Networked Surface can also provide location data for objects, including accurate 2D position and orientation details. This can be used for various applications in the field of "context-aware computing," including auto-configuration of peripherals and computers, user input/output, and other services.

1.2.5 Provision of Power

The provision of power is another useful application of Networked Surfaces, which sets it apart from wireless solutions. If mobile devices were habitually stored on a Networked Surface, they would be able to recharge more often, and hence require batteries of lower capacity. This in turn can reduce the weight and/or size of the device.

However, providing power requires several safety considerations, including the current limiting hardware mentioned above. To reduce the risks involved, power can be provided as low DC voltages. This measure would also get rid of the need for AC power transformers on the objects themselves.

1.3 Research Overview

The concept of Networked Surfaces raises a whole spectrum of issues, from the physical implementation of Surfaces through to application layer issues such as the use of location information.

1.3.1 Project History

The Networked Surfaces project was started in October 1998 at the Laboratory for Communications Engineering (LCE)[3], with the author and Frank Hoffmann (both in their first year of PhD research) working full-time on the project. Initial guidance was provided by Ant Rowstron, and by Andy Hopper, the latter being research supervisor for both the author and Frank Hoffmann. In 1999, Ant Rowstron departed the LCE, and Glenford Mapp and Michael Addlesee of AT&T Labs Cambridge joined the project in an advisory role.

The project has focussed on the design and construction of a prototype Networked Surface, following the philosophy that the best way of evaluating a new network technology is to implement it. This has proved fruitful, with many practical issues brought to light, for which novel solutions have been found and realised.

The initial prototype only provided a physical layer, with later prototypes adding specific link layers, and increasing the performance of the system. The final version of the prototype includes support for a megabit-speed network based on a novel link layer, and supports PCI and PCMCIA interfaces.

Most of the work done was to some extent collaborative in nature between the author and Frank Hoffmann. However, the author implemented the software architecture, whereas Frank Hoffmann built the hardware. Firmware was jointly implemented. Collaboration in particular areas is detailed in the relevant chapters.

1.3.2 Thesis Outline

As discussed above, Networked Surfaces are a new type of networking medium, sharing properties from both wired and wireless media. As such, they can be used to provide a physical layer, over which the other networking layers are stacked. The construction of such a physical layer is the subject of Chapter 2, which presents design issues inherent to Networked Surfaces, the problems encountered in implementing a prototype, and the novel solutions found, as well as details of the software and hardware developed.

Chapter 3 looks at this prototype from a networking perspective, providing characterisations of its capabilities in terms of the time required to connect objects, and the bus speeds that the prototype can support. In addition, the design of the link layer protocols used are discussed, with particular attention paid to the use of a novel channel encoding and arbitration scheme.

The discussion then moves into the network layer of the OSI model. Chapter 4 looks

[3]The LCE is part of the Cambridge University Engineering Department.

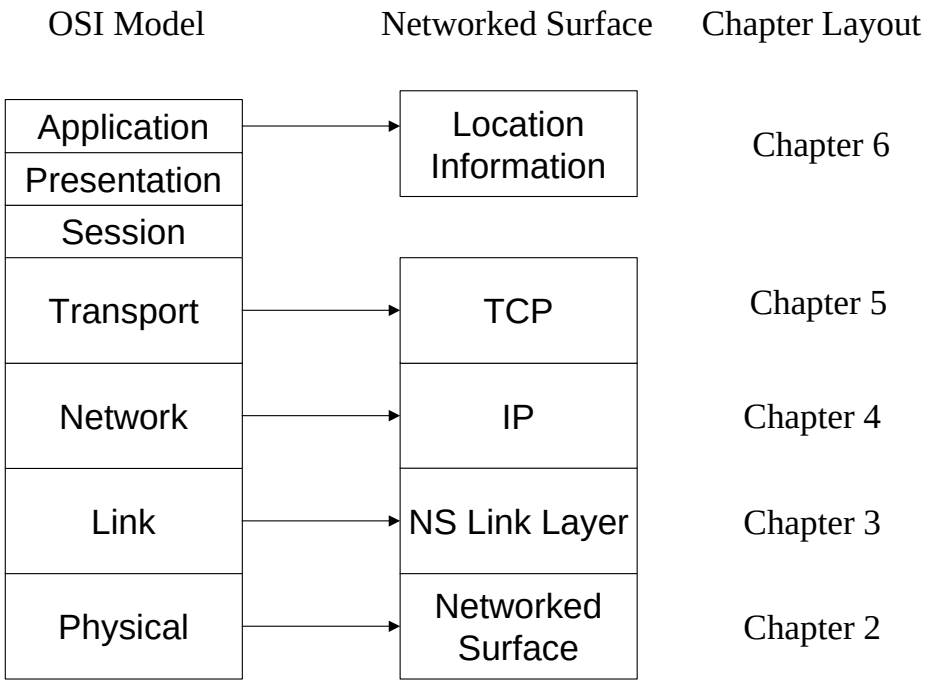

Figure 1.1: Thesis Layout compared to OSI Networking Model

at how Networked Surfaces can interact with other networks, and encompasses the issues of addressing, routing, and the use of multiple network interfaces on an object. A comparison between the Networked Surface prototype and other network types is also presented.

At the transport layer, Networked Surfaces pose an interesting problem in that they are subject to disconnection and reconnection, which may cause transport layer protocols designed for reliable wired networks to perform badly. Chapter 5 discusses these issues in relation to the TCP protocol, and proposes a solution which operates externally to TCP, allowing it to be used with any object type without modifying an object's internal operation.

The final area of research to be explored in this thesis is in the application layer implications of Networked Surfaces. In particular, Chapter 6 explores the calculation of physical location for Networked Surface objects, and the various applications that this gives rise to in the field of "context-aware computing." These applications include some which are Networked Surface-based, such as automatic configuration of computer-peripheral connections on Networked Surfaces.

Finally, the thesis is concluded in Chapter 7, with a summary of important results, and a discussion of future research possibilities. The organisation of this thesis in relation to the OSI networking model is illustrated in Figure 1.1.

Chapter 2

Design and Implementation of a Networked Surface

2.1 Introduction

This chapter will describe the conceptualisation, design and implementation of a Networked Surface. This work, while novel in itself, also forms the basis for the other research presented in this thesis; many of the concepts below are expanded upon in later chapters.

This work, broadly speaking, was jointly carried out between the author and Frank Hoffmann. In more detail, the author was mostly responsible for the software design and implementation, including the device driver and the user-level daemon, as well as the topology simulation. Frank Hoffmann made the greater contribution for the topology implementation, the hardware design and the hardware construction. Work done entirely in collaboration includes the overall architecture design, topology design and protocol design, and the FPGA network interfaces for the I^2C and LVDS networks.

2.1.1 Design Principles for the Networked Surface

Within the scope of the definition of Networked Surfaces set out in the previous chapter, one could design many different Surfaces, depending on the principles one considers important. For the prototype Networked Surface, the following aims were decided upon.

Flexibility is considered important, i.e. the ability to support many different types of device with a single Surface. A more restrictive approach might be to only provide support to a single object type; this may be appropriate for deployment in particular environments. However, in the interests of exploring the full potential of Networked Surface technology, a prototype designed for flexibility is required.

Sparsity of objects is assumed. In other words, Networked Surfaces are assumed to be large compared to size and number of the objects on top of them, therefore making them sparsely covered with objects at any one time. This reflects the arrangement of devices on a typical office desk.

Simplicity is desirable, providing it does not compromise flexibility. Simplicity implies that, out of many equivalent implementations, the one with the least hardware and software overhead is deemed the "better" one. In concert with sparsity of objects, this implies that it is especially important to make the surface-side hardware as simple as possible, since there is much more surface hardware required than object hardware.

2.1.2 Layout of this Chapter

The following section will explore the implications of the principles stated above, and arrive at a number of design challenges inherent in Networked Surfaces as well as an overall architecture for the system. These challenges are then discussed in detail. The prototype implementation of a Networked Surface will then be presented. Finally, some of the principles described above will be examined more closely, and alternatives will be discussed.

2.2 System Design

Networked Surfaces, like all networks, must provide a physical layer for the transfer of data. This entails a choice of a physical medium over which data may be sent. For reasons of simplicity, the chosen medium for the prototype is electrical conduction. The use of other physical media is discussed in the final section of this chapter.

Connectivity is achieved by providing a number of electrical channels between the surface and the object; these channels are analogous to wires in a cable. Based on this requirement, a number of design criteria can be formulated, and a system architecture can be developed. This architecture is illustrated in Figure 2.1, and the rationale behind it is discussed below.

2.2.1 Topology and Handshaking

Data transmission using electrical conduction has a very important implication, arising from the fact that it is difficult to transmit data using one electrical conducting path (such a path on the Networked Surface is termed a *link*). Therefore, the simplest Networked Surface must support at least two links between the surface and each object, namely "data" and "ground."

To meet this requirement, the surface and object base must be split up into many electrically conductive regions (or *pads*), so that when a surface and object meet, enough links

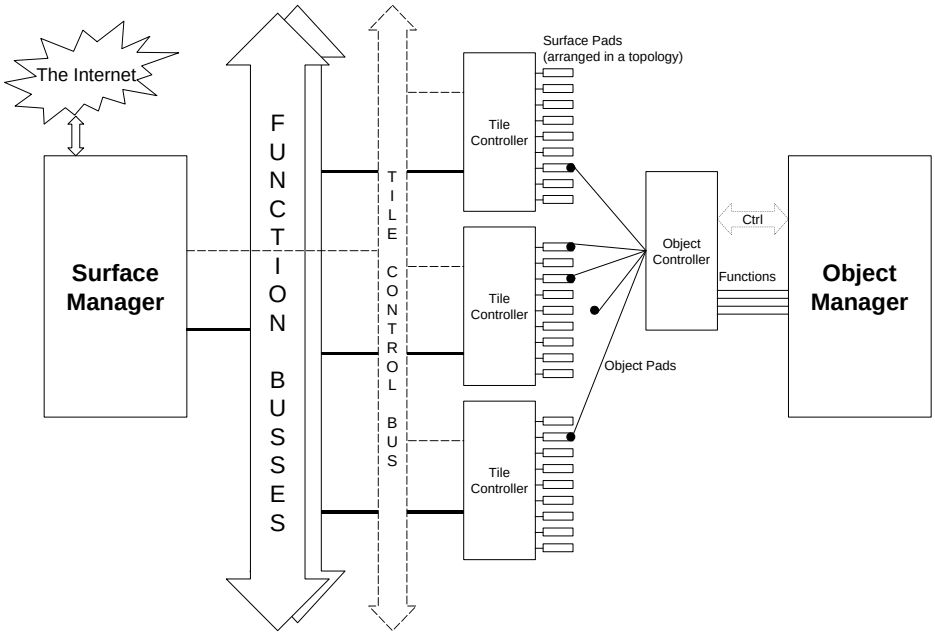

Figure 2.1: Networked Surface Architecture

are created between them to form a *connection* between the object and its desired services (also known as *functions*). The physical layout of surface and object pads that is required to guarantee this important property is termed the *topology*, and will involve the choice of size, shape, and spacing of pads on both the surface and the object base.

The use of many pads has a further implication, namely that some form of *handshaking* mechanism must be in place, so that the surface and object hardware can determine which link is performing which function.

2.2.2 Functions

As discussed, surface and object pads are assigned to functions during handshaking, and when a number of functions have been allocated, a working connection is formed. The set of functions required for a particular connection depends on the choice of network, and on whether power is required.

The choice of network provided is dictated by the type of object that is trying to connect. For example, a notebook computer might require a high-speed network, providing a link layer for IP. A small PDA might require an RS-232 connection in order to synchronise with application software on a PC. A computer peripheral might require a peripheral bus such as the PS/2 standard for pointing devices.

It is of course possible to build Networked Surfaces to support only one of these types of network, and use hardware to support the other types of network by bridging them over the chosen network type; this is one method with which a broad range of devices could be supported. However, this would not give optimum performance for all devices; for example a notebook would not get a high bandwidth from an RS-232 link, and a peripheral would be unable to make substantial use of a network at megabit speeds. Also, a high cost would be incurred in retrofitting existing devices to use Networked Surfaces.

In light of this, Networked Surfaces might be designed to support only one type of object. In this case, any connected object on the Surface would be requiring the same functions as any other object. However, this breaks the principle of flexibility outlined earlier, and does not allow full exploration of the possibilities of Networked Surfaces.

A third option would be to make a Surface that supports multiple networks and therefore multiple types of objects. This option provides the most flexibility, and demonstrates the generic connectivity possible with Networked Surfaces. The rest of this chapter therefore discusses Surfaces of this type.

2.2.3 Distributed Architecture

As stated previously, it is assumed that Networked Surfaces are sparsely populated with objects, i.e. that they are large compared to the objects on them. Since each object must span multiple pads, this means there must be a significant number of pads on each Surface, perhaps hundreds or thousands. At any given time, one or more objects may be trying to connect anywhere on a Surface. Therefore, it is infeasible to have a single entity performing handshaking for the whole Surface; in order to achieve scalability, a distributed approach must be taken.

However, the scalability problem exists only for the connection of objects, in which the whole surface must be concurrently active. In the case of communications with connected objects, a very different scale prevails. It is therefore proposed that, on the surface side, the system criteria suggest an architecture whereby there are multiple entities for handshaking purposes (known as *tile controllers* henceforth), each controlling a small group of pads on the surface (known as a *tile*). There would also be a single entity for communications with connected objects, called the *surface manager*.

This subdivision allows the tile controllers to be ignorant of the communications protocols, and offers a clear control hierarchy since the manager can also perform such regulating functions as may be necessary for the tiles, on a *tile control bus*. Various needs for this bus will become apparent in later sections.

To provide scalability for large surfaces where a single surface manager is unable to cope, two or more separate Networked Surfaces may be used. The only drawback of splitting a Surface into two is that objects may not be able to acquire connectivity if they are placed on the boundary between them.

On the object side, there is no scalability problem since the number of pads is small and bounded. However, one may still refer to the *object controller* and *object manager* entities as logical units, which communicate with their surface counterparts. In reality, these units may be implemented in the same physical device.

2.3 Topology

In order for an object to establish a network connection, a number of independent electrical conducting paths, or *links*, are required. The physical layout of pads on the surface and object required to achieve this is known as the *topology*.

Exactly how many links are required depends on the type of network used, and whether the provision of power is desired. The simplest connection types (based on electrical conduction) require two links, namely ground and either a half-duplex data bus[1] or power. More complicated connections may require more links, e.g. for data buses using multiple links, or when power is required.

2.3.1 Requirements of the Topology

The primary requirement of a topology is that it must guarantee that an object can acquire the appropriate links that it needs to make a connection, regardless of the position and orientation of that object upon the surface.

The secondary requirement of a topology is that it must support objects requiring different numbers of links. This requirement reflects the aim of flexibility outlined earlier, in that the surface under design should cope with objects of varying types.

The tertiary requirement of a topology relates to the principle of simplicity outlined previously. It is that a topology should fulfill the above guarantee with the minimum number of pads (and therefore the minimum data and control circuitry) on the object and the surface. The latter is especially important; due to the principle of sparsity mentioned earlier, a reduction in surface complexity gives greater savings than a reduction in object complexity.

[1] A half-duplex bus is one supporting data transmission in only one direction at a time.

2.3.2 Generic and Non-Generic Pads

In order to fulfill the guarantee of connection, it must first be decided whether surface and object pads are *generic* or *non-generic*, i.e. whether pads are able to switch to any function, or whether they are dedicated to particular functions. It is of course possible to use generic surface pads and non-generic object pads, or vice versa.

Using generic pads requires there to be switching hardware for each pad so that it can independently be connected to any function. With non-generic pads, it is possible to use less switching hardware, though there must still be the capability to switch between handshaking functions and the dedicated function for that pad.

With generic pads on both surface and object, the guarantee required of the topology is greatly reduced; it can simply be that the correct number of links are made for any connection. With non-generic pads on one side, the guarantee is more complicated, requiring that at least one generic pad be in touch with each type of non-generic pad required. For non-generic pads on both sides, the guarantee is even more complex; for each function requires, there has to be a surface pad providing that function in contact with an object pad providing that function.

As discussed in the previous section, the Surface design presented is intended to support multiple network types. While it is possible to imagine topologies that would support dedicated surface-side pads in this system, they do not scale as more networks are added; the topology would have to allow for more and more pads to fit under each object, so that a connection to the particular network that the object wants is guaranteed to be available.

A scalable way of supporting multiple networks would be to have generic surface pads, that can be connected to any of the functions that the Surface provides, i.e. any wire on any network. This is also the most flexible policy, since changing the functions available would not result in any implications for the topology.

With such a Surface, it is also desirable to make the object pads generic. This is because it is uncertain which of the object pads will be in touch with a surface pad at any time. Furthermore, many object pads may be in contact with a single surface pad, so with a generic system each object pad can be used independently to acquire a function, whereas in a dedicated-object-pad system, the entire mapping of surface to object pads would have to be discovered before a suitable allocation of pads to functions could be decided. Finally, unlike on the Surface where there are many possible functions to allocate to each pad, there are only a few functions on the object side to be allocated (corresponding to those functions which the object is configured to request), so the additional switching circuitry needed to make the object pads generic is minimal.

For these reasons, only topologies for Surfaces with generic object and surface pads are considered below. This has implications for the handshaking protocol described in the next section.

2.3.3 A Useful Topology

There are many possible choices for the topology, including hexagonal, grid-like, and brick-like arrangements of surface pads. The advantages and disadvantages of these schemes are not within the scope of this thesis.[2] Instead, one topology which has many useful properties is presented below; it is this topology which is used in the prototype described later in this chapter.

The chosen topology is one where rectangular strips cover the surface, and a small number of circular pads, themselves arranged in a circle, form the object's *footprint*. The gaps, or "margins," between the strips are chosen to be larger than the size of the object pads.

This topology, illustrated in Figure 2.2, has the following properties:

- The small object pad size guarantees that each object pad will never span two strips, and hence ensures that no strip will be short-circuited to strip. Each strip can therefore be connected to a different function.

- By having enough pads in the circle of object pads, guarantees can be made that at least one object pad will be in contact with each strip under the object, within a particular "column" of strips.

- Due to the previous two properties, the footprint required of an object is always bounded, since the object need only span as many strips as the number of functions it requires, and no more.

- The previous properties have not specified a particular number of functions required. Therefore, a single surface topology is able to cope with different object topologies for different numbers of functions.

- Finally, the topology outlined is geometrically simple and therefore easy to manufacture.

[2] See [48] for more details.

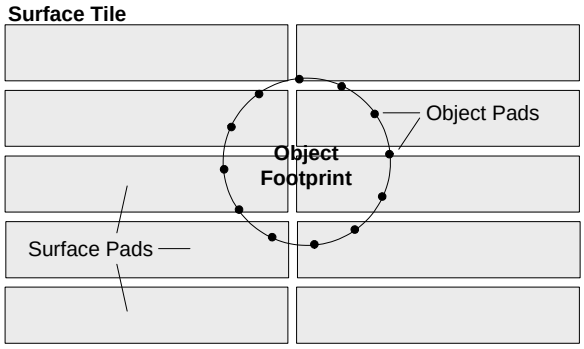

Figure 2.2: Chosen Topology for the Networked Surface Prototype

2.3.4 Dimensions of a Useful Topology

In order to get actual workable dimensions for the chosen topology, a simulation was used to discover values for the size of object footprint and number of object pads on that footprint, for various numbers of functions required. These tests were done for a surface with the characteristics outlined below:

- Width of strip: 2.05cm. This strip size is chosen for ease of manufacture (24 strips per 25cm×25cm tile).
- Length of strip: 12.2cm. This is again chosen for ease of manufacture, and is irrelevant to the results below, so long as an object never touches pads from more than two "columns."
- Size of margin: 0.3cm. This limits the size of the object pads, which must not be able to touch two strips simultaneously.

The specifications above do not cause loss of generality for the results below, as the topology can scale up or down to different surface pad sizes.

As Table 2.1 shows, this topology can support objects with between two and six links. The number of object pads required is approximately three times the number of links, and the diameters are such that they could be accommodated on the base of objects such as PDAs or notebook PCs.

In summary, this is a practical example of a topology that fulfills the requirements of guaranteeing connections, being flexible with regard to object size, and being as simple as possible under those circumstances.

Links Required	Object Pads Required	Diameter of Object (cm)
2	5	2.64
3	9	4.62
4	12	6.78
5	16	8.82
6	19	10.80

Table 2.1: Minimal Object Topologies for Chosen Surface Topology

2.4 Object Connection and Disconnection

This section discusses the formation of connections between surfaces and objects, and the subsequent disconnections. As will be shown, the connection process actually takes place using two separate protocols, a low-level *handshaking* protocol between the tile and object controllers, and a higher-level *registration* protocol between the surface and object managers. Firstly, however, a discussion of the problem of grounding is presented.

2.4.1 Object Grounding

In order to communicate by electrical conduction methods, two parties must share a common ground, otherwise the receiver has no basis against which to compare the transmitted signal.

On the Networked Surface, however, no such ground is intrinsically provided between the surface and a connecting object. Therefore, a means of establishing a common ground for the handshaking process must be found.[3] This must be accomplished by grounding a subset of the surface pads, so that an object attempting to connect would have access to one of these grounded pads.

This has the implication that handshaking will only work if at least two surface pads are spanned by each object footprint, as the object would not be able to communicate with the surface without being in contact with both a grounded surface pad and one used for message transfer. However, this is not a restriction on the system, since no useful network (or power) connection can be made without at least two links, since one of them must be ground.

[3]After handshaking is complete, there is no grounding problem, as one of the established functions must be "ground."

Beacons

The need for selective surface pad grounding has an important implication, namely, in the issue of which party transmits the initial message of any handshaking protocol, known hereafter as a *beacon*. For the reasons explained below, it is prudent for the surface to send the beacons.

If an object were to send beacons, it would first have to discover which of its pads was connected to a grounded surface pad, as well as which of its pads was connected to a listening surface pad, prior to sending the beacon. Detection of both grounded surface pads and listening surface pads would be difficult to achieve, as both operations are passive.

Furthermore, if the object were to send beacons, the tile controllers would have to listen in parallel to many pads in order to try detect incoming beacons. This "listening" operation is more difficult than the task of sending beacons, as it requires many pads to be monitored at one time. As previously established, it is important to keep the tile controllers as simple as possible.

For these reasons, it is prudent to make the surface the beaconing entity and the object the listening entity in the first message of the handshaking process.

Possible Grounding Methods

For handshaking to work, an object placed on the surface must somehow connect its ground to the surface ground. As stated, the surface must have one or more of its pads under the connecting object grounded. This means that one or more object pads is guaranteed to be connected to ground. The difficulty lies in determining which of the object's pads are grounded, and which are going to receive an incoming beacon, at any given time. There also exists the possibility that some object pads are in contact with a surface pad which is being used by another object; those pads will have unknown voltages applied to them; this must not "confuse" the handshaking process.

One method of discovering ground would be to try each of object's pads as a ground connection, and while doing so examine each of the other pads to see if a beacon were detected. Assuming, without loss of generality, that handshaking is accomplished with baseband signalling (i.e. 0V for "low" and 3.3V/5V for "high"), for any detection to take place, a "high" must be sent at the start of the beacon. This may be sent as either the first bit of that beacon, or in a preamble phase before the beacon.

The detection would in this case involve connecting each pad to the ground bus of the object in turn, and reading each of the other pads, looking for a "high" logic level. This, while possible, is complex and does not scale well — for n object pads it takes up to n^2

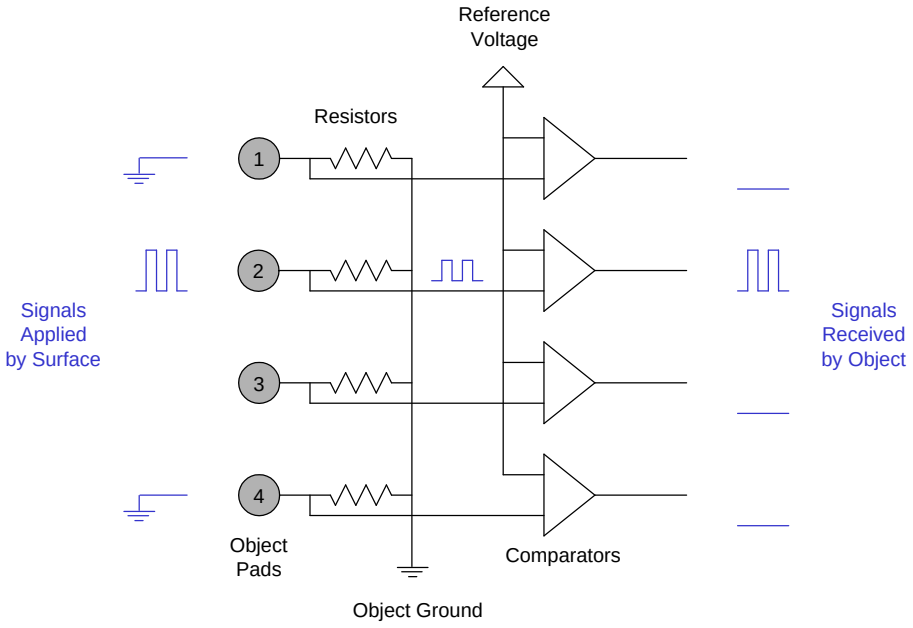

Figure 2.3: Grounding by Consensus

detection attempts (the object must try each possible ground pad and beaconing pad pair). This high detection overhead would either slow down the handshaking protocol, or require specialised detection hardware on the object. Both of these options are undesirable.

Grounding by Consensus

Another grounding solution, with better scalability properties, is to use a resistor network between all (unconnected) object pads, to determine the average voltage level of these pads, and use this as the object ground. Comparators are used to read the voltage level of each object pad, and determine whether the surface is asserting a "high" logic level on that pad. This method is termed *grounding by consensus*, as the logic level present on the majority of the pads most influences the ground level. This technique is illustrated in Figure 2.3, and explained further below.

The diagram shows four object pads, connected to a resistor network and comparators. Object pads 1 and 4 are shown to be touching grounded surface pads, pad 2 is in contact with a beaconing surface pad, and pad 3 is shown to be unconnected (it might be in a margin). The object uses a resistor network to "anchor" its ground against the average voltage of all incoming signals. When the incoming beacon is "high" phase, the object's ground is pulled up slightly, in this case to one third of "high." This causes the beacon signal itself to appear

weaker, when compared with the object ground. This explains why comparators are used; by comparing the weak signal against a low but non-zero "reference voltage," these comparators can recreate the original signal.

The use of this technique has several interesting properties. Firstly, the ground level observed by the object is not static, but changes depending on whether the beaconing pad is currently asserting "high" or "low." Also, the discrepancy between surface ground and object ground, and hence the beacon's apparent voltage level, is determined by the ratio of beaconing pads to grounded pads. If the object has five pads in touch with grounded pads, and three in touch with a beaconing pad, then the beacon will appear, when asserted "high" by the surface, to be five eighths of the voltage level the surface is using for "high."

The grounding by consensus method has better scalability properties than other grounding methods because, for n object pads, it requires at most n detection attempts by the object to hear an incoming tile beacon. This detection would be simply accomplished by cycling through all objects pads looking for a "high" signal.

Grounding By Consensus and Tile Synchronisation

The use of grounding by consensus has one important drawback, namely that it is desirable for only one pad underneath any object footprint to be communicating using the handshaking protocol at any one time. To have more than one surface pad at a level other than ground would make it difficult for the object to interpret the incoming beacon properly, as the ground reference would constantly be changing.

This restriction is simple to implement for a single tile; without loss of generality, it can be assumed that only one pad per tile is beaconing at any one time. However, it is more difficult to ensure that, for objects spanning multiple tiles, the same guarantee is upheld. This is achieved by having a central entity (the surface manager) send synchronising signals to the tiles, and by having the tiles each cycle round their pads in the same order, and only start each cycle when a synchronising message is received. One possible beacon cycling scheme is shown in Figure 2.4. This is the first example of the need for a tile control bus to be present between the tile controllers and the surface manager, other needs for this bus will be identified later.

Consideration must also be paid to the possibility that surface pads which an object is in contact with may already be connected to functions, for another object(s). If this function were ground, there is no problem; however, if the function were a network or power, then this would cause problems with grounding by consensus. This is remedied by temporarily disabling object pads which are receiving signals other than ground or a valid handshaking

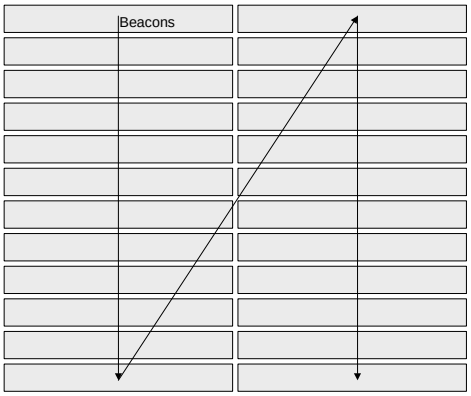

Figure 2.4: A Simple Cycling Scheme for Tile Beacons

signal.

2.4.2 Handshaking and Registration

As noted in the first section, on the surface side a scalable architecture may have many instances of a handshaking entity (or tile controllers) and a single instance of an entity for networking with connected objects and managing controllers (known as the surface manager). The object, with no scalability issues, has one object controller and one object manager, and these entities may be combined in the physical implementation.

The difference between the controllers and the managers is that the controllers are in direct contact with the pads, and therefore have to be simple and numerous (on the surface side). The managers, however, only communicate through the data networks — the very networks that are formed by the handshaking between the controllers. Furthermore, since the network used by the managers may be high-speed and use particular modulation schemes, the controllers may not be able to "understand" those networks. In other words, once handshaking is complete, the controllers are blind to what occurs on the handshaked pads thereafter.

In order for this system to function, it is imperative that connections be made even if objects were to be placed on the boundaries between tiles, or in the worst case at the boundary of four tiles (as shown in Figure 2.5). This is accomplished by making the tile controllers unaware of any relationships between the function buses, and therefore unaware of the particular functions that an object might require to form a connection. As far as a tile is concerned, the state of each of its pads should be separate and unlinked to the state of other pads. It is up to the object controller and the managers to be "aware" that an object

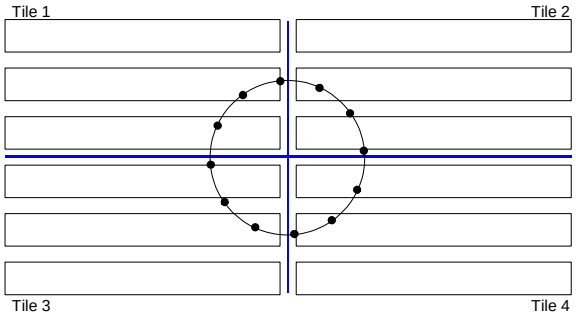

Figure 2.5: Worst Case Object Placement Covering Four Tiles

may be in communication with many tiles at one time.

Unfortunately, this system has an inherent limitation, in that tiles are unable to decide when to disconnect a connected pad, since they are unable to understand the traffic on the buses they provide. Thus, there must be another mechanism to detect disconnection, involving the surface and object managers, which do have the capability of understanding the activity on the buses. This is discussed later.

In order for the surface manager to be able to tell the tile controllers which pads to disconnect when an object leaves the Surface, the surface manager must first obtain this knowledge. Since disconnection results in the loss of networking, this information must be sent on connection when a network is known to be available. The transfer of information about a connection, from the object manager to the surface manager, is known as the "registration" process. The use of a registration protocol also allows both managers to verify that the network established is functioning correctly, before enabling data transfer.

While the system described so far correctly handles situations where connections and made and registered, it breaks down in cases where connections are malformed, for example due to movement of the object before a full connection is formed. Such malformed connections can result in surface pads becoming connected without the surface manager being aware of any object, which means that the misconnected pads are useless for future objects, since they are not subject to the handshaking protocol any longer.

This can be avoided by causing all tiles to automatically disconnect connected pads after a set time has elapsed from their connection. In order to prevent valid connections from being terminated in this fashion, the surface manager must send a message to each tile confirming connected pads, which it can do if and only if the object performs the registration process. In this way, the handshaking system is made robust; pads which are subject to failed connection are automatically re-inserted into the handshaking sequence.

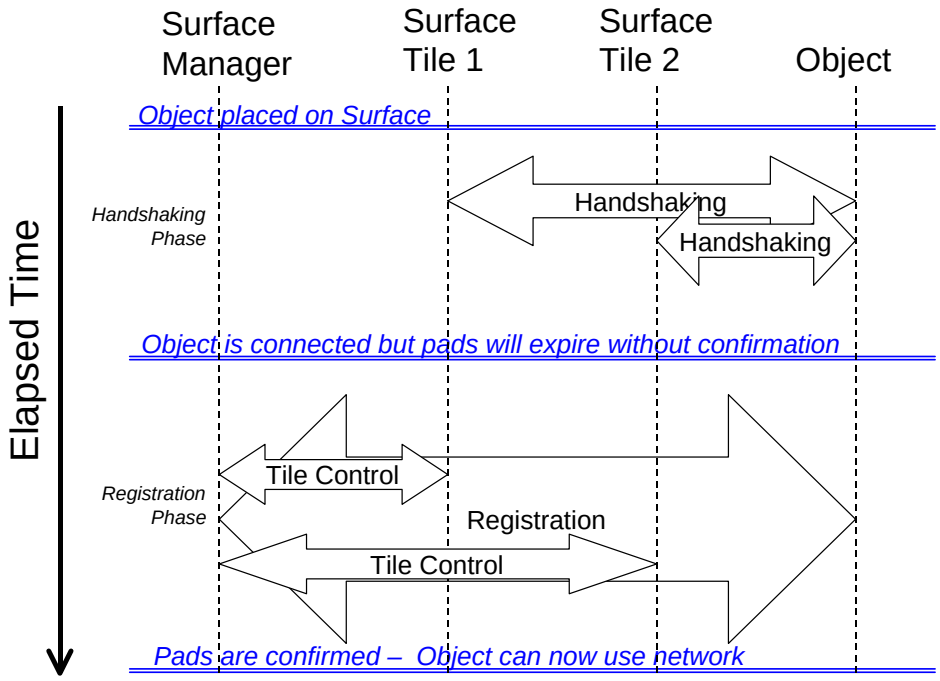

Figure 2.6: A Two-Phase Connection Process

This system is conceptually elegant; the manager "takes responsibility for" particular tile pads on connection, and then transfers responsibility back to the tiles on disconnection. It also has the interesting property that the manager is equipped to disconnect any object at any time, for example in order to enforce Quality of Service (QoS) guarantees. This two-phase connection process is illustrated in Figure 2.6. Both the handshaking and registration protocols are discussed in depth in Chapter 3.

2.4.3 Disconnection Detection

There remains the problem of what constitutes a "disconnection," and how this is detected. Disconnection is defined as one or more of the previously connected object pads losing contact with the surface pad it was connected to. This may occur if the object is either picked up or moved.

The Importance of Fast Disconnection

Disconnection detection is necessary in order to "recycle" surface and object pads so that they can be used to make new connections. It is important to perform this process quickly, so

that potential new connections are not blocked because the pads required are still connected in the previous configuration.

In particular, disconnection detection becomes especially important when considering an object which has been moved only slightly. This movement may be enough to invalidate existing pad pairs, but not enough to make the object cover a wholly new set of surface pads. In this case, the object may be unable to reconnect until the surface pads it used are re-inserted into the handshaking system. A delay in disconnection detection would result in a delay in reconnection, and therefore a delay for any data packets that may be waiting for transmission. In contrast, erroneous disconnection is not too costly, as the disconnected pads will immediately be made ready for reconnection to take place.

Disconnection detection can take place using one of two methods, using either hardware or software.

Hardware-Based Disconnection

The hardware method is problematic in that the surface buses are, as stated previously, intrinsically shared. While under certain circumstances only one object may be using a bus, this is not true in general. This implies that at times, buses are not driven by an object using them (otherwise other objects would not be able to transmit); in those periods, it is not possible to detect the removal of an object by examining the physical bus alone.

However, the hardware method may be able to give indications of disconnection in particular cases. For example, in power buses, a sudden drop in current may be used as an indicator that an object has disconnected, prompting a test message to be sent to all objects using that power bus. Alternatively, on a data bus hardware methods might show if an object suddenly stops driving the bus during one of its transmissions, which would also prompt a check for disconnection.

Software-Based Disconnection

Software disconnection involves periodic sending of test messages, or "pings," between surface and object. This method can be implemented in more than one place in the protocol stack. In particular, it can be achieved as part of a link layer protocol, or it can be done at a higher layer.

The former method makes Networked Surface–specific requirements of the link layer, and is therefore not suitable for use on buses with a standard link layer. However, using a link layer-based method may result in a faster disconnection detection, since the link layer protocol may be able to continually send "pings" on an otherwise inactive bus, so that

disconnection is detected with as little a delay as possible.

In contrast, a higher layer disconnection can be used with any link layer. However, by using a separate protocol for disconnection, more traffic is imposed on the bus. The "ping" traffic is also sensitive to packet loss, which may cause disconnection to be inferred in error. Some optimisation can be done by using a "promiscuous" mode at the link layer, and only sending "ping" traffic when the bus is otherwise idle. However, this means that all bus data will have to be filtered at the network layer, which puts overhead on the host. Despite this, higher layer disconnection is intrinsically reactive, so a proactive link layer method will inevitably be faster at detecting disconnection. Both of these methods are presented and tested in the following chapter.

2.5 Surface Buses and Object Types

This section describes the types of buses that can be made available on Networked Surfaces, the types of objects that can be supported on Surfaces, and the necessary levels of augmentation for each object type in order to be compatible with Surfaces. In addition, the characteristics required of the tile control bus are discussed.

2.5.1 Networked Surface Bus Requirements

The Networked Surface provides a novel physical layer medium, with characteristics dissimilar to other media. This has implications in the choice of data bus. However, the topic of physical layer characteristics is not within the scope of this thesis, so only a brief overview of the criteria for choosing surface buses is presented.

The first requirement on Networked Surface buses is that they must all be "multi-drop" buses, i.e. capable of supporting many bus participants all using the same physical channel. This is necessary because the architecture presented allows any pad on the surface to be connected to any bus. Therefore, many objects may be connected to each bus. This requirement immediately excludes many common bus types, including switched Ethernet and RS-232.

Next, it is noted that the requirements on the physical topology are directly proportional to the number of links the bus requires. A bus such as IDE, for example, would not be feasible on the topology outlined above, as it requires many tens of links, being a parallel bus. In order to maintain simple topologies and reasonable object sizes, surface buses must be chosen which use a small number of physical links.

Finally, it is noted that if the highest bandwidths the Surface is capable of providing are to be attained, a bus which performs well under the particular physical characteristics of the

Surface is necessary.

2.5.2 Networked Surface Object Requirements

In order to be capable of using the Networked Surfaces architecture presented, an object must fulfill certain criteria. This may be achieved by adding hardware components to the objects, i.e. using "hardware augmentation." Since objects must have physical pads and object controller hardware, some degree of hardware augmentation is always necessary. For other functionality, however, the technique of "software augmentation" may also be used, whereby components can be included internally to the objects themselves, by using software such as device drivers.

All objects must support a *network interface*, as this is required for the registration protocol. This is especially pertinent for objects which may use the Surface for reasons other than networking, i.e. for power or to provide location information. In these cases, hardware augmentation might have to be used, as such objects are likely to have no networking capability. For other object types such as notebook PCs, it may be possible to use software augmentation to provide some part of the networking functionality, thus allowing less hardware to be required.

Finally, all objects must have *object manager* functionality, i.e. an entity on the object must execute the registration protocol and detect disconnection. In network-capable objects, this can be in the form of software augmentation, however in non-networked objects, this must again be achieved using hardware augmentation.

2.5.3 Object Types

Networked Surfaces can potentially support many types of objects. These object types are grouped here in order to identify common levels of augmentation required, and common bus requirements. This will assist in the choice of surface buses for the prototype, and the design of prototype object hardware.

Programmable High Speed Objects

The term "high speed" is used in this context to mean that these object are capable of using the maximum bandwidth that the Surface can provide. These devices include computers such as desktop PCs, notebook PCs, and powerful PDAs. They are assumed capable of running some aspects of the object manager as internal software, and for them the Surface is essentially a type of Network Interface Card (NIC), which may also provide power and location information.

While they may be capable of supporting other bus types already, in the interests of providing the highest bandwidths possible across the Surface, they may benefit from using a physical bus type and link layer chosen to perform well in the Networked Surface environment.

Non-Programmable High Speed Objects

This category carries the largest hardware augmentation requirements, as the hardware must be capable of performing the object controller and object manager functionality, and must support data bridging between the Networked Surface bus and the object bus, both of which operate at relatively high speeds. Devices in this category might include peripherals such as digital video cameras.

Low Speed Objects

"Low speed" is used in this context to describe objects that do not demand maximum bandwidth from the Networked Surface medium, and therefore do not necessarily need to use a physical layer chosen to perform well on Networked Surfaces. This means other bus types may be used so long as they are multi-drop capable, and otherwise suitable for use over Networked Surfaces. In particular, buses which are already supported by certain classes of object may be used; in this case, the augmentation required of those objects is greatly reduced, as the network interface is already provided.

If the low speed object in question does not support a Networked Surface–capable bus, the hardware augmentation may include a data bridge onto a bus supported by the Surface (which may also be low speed). Object manager functionality can either be implemented using software or hardware augmentation, depending on the programmability of the object.

Such objects may include peripherals (e.g. keyboards, modems, etc), or low-powered PDAs.

Non-Networked Objects

Even non-networked objects can make use of the Surface, for location information and power. In these cases the augmentation would take the form of a hardware controller and manager, but without any networking traffic generated other that caused by the registration protocol and disconnection detection. Objects in this category might include devices requiring just power or just location information from the Surface, for example mobile phones or battery chargers.

It is also possible for an object which is not networked to be "monitored," i.e. to have information about it gathered using sensors, and then transmitted over a Surface. The

hardware augmentation would then include some monitoring functions which could send data, either in reaction to a change in the monitored object, or in response to a request for data over the network, or periodically. An example of a suitable object type would be an appliance such as a coffee machine. Networked Surfaces could allow users to remotely observe or control the state of the appliance, e.g. notification when coffee is ready, or automatic switching-off when all occupants of the building have logged out.

2.5.4 Tile Control Bus

In addition to buses for supporting objects, there is a need for a control bus internal to the surface itself. This is a consequence of the distributed architecture used, with many tile controllers being coordinated by a single surface manager.

The main function for the tile control bus is to keep the tiles synchronised as they beacon on their pads. As described earlier, this is necessary due to restrictions imposed by the use of "grounding by consensus."

In addition, there are a number of configuration issues. As shall be seen in the handshaking and registration protocols described in the next chapter, a surface manager must be aware of the tiles it is connected to, and have a means of addressing them individually to give them commands related to the connection and disconnection process. The tile control bus must therefore be an addressed bus, and not a simple broadcast bus.

Also, due to issues yet to be described, it is useful for the tiles to be able to proactively send messages to the manager. This will be useful to indicate when new objects are connecting. For this to be possible, the tile control bus must be multi-master, and also have some method of arbitration, as transmissions on the bus may collide.

2.6 Prototype Design

This section will discuss the design and construction of the prototype Networked Surface.

2.6.1 Rationale for Prototype Construction

The construction of a prototype has proven very valuable in the exploration of the issues brought about by Networked Surface technology. With a prototype, systems integration issues can be tested in a way that is not possible with separate proof-of-concept testbeds. In fact, a major difficulty in constructing prototypes and real systems in general lies in system integration; interfaces between different system components are often candidates for subtle

error. This can highlight problems with the design of the components themselves which may not be apparent when they are only considered individually.

In addition, when trying to characterise and explore the limits of a system, there is no more compelling evidence than having a real system perform to that specification. In the Networked Surface, this is manifested by measurements of characteristics such as bandwidth, connection time, and location accuracy.

Finally, the use of an integrated prototype system means that it is possible to explore different alternatives for a given component, by taking measurements of how the whole system responds with each alternative in place. This provides a more accurate measure of the performance of each alternative choice, as compared to simply characterising each alternative in a stand-alone fashion.

2.6.2 Data Buses

One important design decision for the prototype Networked Surface is the choice of data buses, which depends in turn on the choice of object types to be supported. These decisions are discussed below.

Chosen Object Types

Section 2.5 outlined a number of classes of object type which may be supported on Networked Surfaces. For the prototype Surface, the principles of flexibility and simplicity dictate the choice of which of these object types to support.

Flexibility implies that many object types should be supported, but providing support for all possible types of object is infeasibly complex. The choice was therefore made to provide two different object hardware units. In particular, a "high-speed" object controller, intended to interface with a PCMCIA-compatible device, and using a high-speed bus, was designed. Also, a "low-speed" object controller, using a low speed bus on the Surface, and providing hardware-based surface manager functionality, was decided upon. The latter can support both low-speed object types, and non-networked object types. The high-speed non-programmable object class is not supported, due to reasons of simplicity; high-speed functionality is only demonstrated in object types which are programmable (e.g. notebook computers).

The choice of bus types for these object controllers is presented below.

Categories of Bus Considered

As stipulated previously, buses on the Networked Surface must be multi-drop-capable, i.e. they must support many bus participants on the same physical wire. On the Surface, the physical channel is also likely to be noisy, due to the non-ideal electrical characteristics of the large conducting pads used. This affects the choice of high-speed bus technology; for lower speeds it is less important since such buses have less stringent demands on the channel characteristics.

Another restriction on the choice of bus, for both low and high speed networks, is the number of physical channels the bus requires. Networked Surface connections and object footprints are more complicated as this number grows; in order to keep the prototype as simple as possible, and to lower the time required for connection to occur, physical buses with a low number of channels (one or two) are a better choice.

High-Speed Bus: LVDS

The choice made for the high-speed bus is one based on the Low Voltage Differential Signalling (LVDS) modulation scheme [67][4]. This modulation scheme has many advantages which make it a good choice for Networked Surfaces. Firstly and foremostly, LVDS meets the requirements for a Networked Surface bus, in that it is multi-drop capable, and uses only two physical channels for data (in addition to ground).

Secondly, LVDS performs well under the physical characteristics of the Networked Surface. LVDS uses differential signalling, which involves the use of two physical "wires" for each data channel, on which the data is encoded as being "high" if one wire is at a higher voltage than the other wire, and "low" otherwise. This scheme performs well in the presence of noise, since such noise will likely affect both wires in similar fashion, so the difference will be unaffected. In addition, LVDS also generates less noise, since it uses a low voltage swing, and therefore causes less interference with adjacent channels. More details on physical layer characteristics can be found in [48].

Thirdly, LVDS is flexible, in that it only specifies a physical layer and not a link layer. Experiments can be therefore be made with different coding schemes, bandwidths, arbitration mechanisms, and so on, to find the optimum choices for the Networked Surface. This flexibility is important to this research; Chapter 3 discusses the link layer used over LVDS, which is novel and optimised for the characteristics of Networked Surfaces.

Also, LVDS is a baseband scheme, and is therefore human-readable (given an oscillo-

[4]In particular, the "Bus LVDS" specification is used.

scope). This is useful during the implementation and debugging of a prototype system. Finally, and importantly for prototype construction, there are cheap, readily available, and easy-to-integrate hardware implementations of LVDS transceivers. In the interests of performing experiments involving the use of multiple buses to enhance Networked Surface bandwidth, two LVDS buses are provided in the prototype.

Low-Speed Bus: I^2C

The choice of I^2C [83] as the low-speed bus is also the combination of many factors. I^2C is simple enough to be supported in modest hardware. Despite this, it has many features such as multi-master capability, a built-in arbitration scheme, and addressing. It could therefore be used as a link layer for IP, if necessary. The bus is multi-drop, and offers variable and dynamic bitrates (i.e. bus participants can send at different rates, and can even change bitrate on-the-fly). It uses two physical channels in addition to a common ground.

The I^2C bus has native hardware support in many devices, so some potential Networked Surface objects may already have this bus built-in. For those that do not, the object controller and object manager hardware must bridge between the bus used by the object (this is often RS-232) and I^2C. However, this is made easier by using a microcontroller which supports both I^2C and RS-232 for the object hardware.

Tile Control Bus: I^2C

The tile control bus choice is also I^2C, for many of the same reasons as above, including multi-master and multi-drop capabilities, built-in addressing, and hardware support. The re-use of this bus type keeps the prototype as simple as possible, without compromising functionality.

2.7 Prototype Hardware

The Networked Surface prototype is comprised of hardware and software components. The hardware entities are tile controllers, a surface manager PCI card, one integrated object controller/manager supporting I^2C, and two object controllers supporting LVDS and including PCMCIA interfaces. There are also physical surface tile and object footprints, implementing the topology shown in Figure 2.2. Ribbon cabling is used for power, the function buses, and tile control bus.

Photographs of the prototype hardware are shown in Figures 2.7 to 2.9. The first photo depicts the topology implemented, including a number of object footprints on two tiles. The

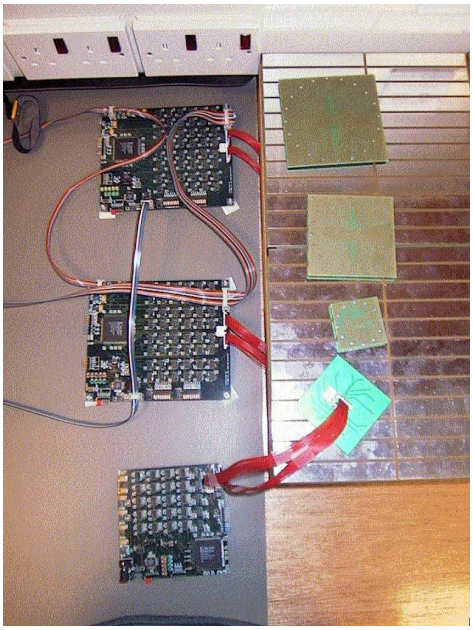

Figure 2.7: Photograph of Prototype Topology and Controllers

second photo shows two tile controllers (top and middle right), which are connected to a surface manager PCI card (left). An I^2C object is shown in the bottom right. The third photo shows a Networked Surface–enabled notebook PC, with the inset showing the LVDS object hardware mounted on the back of the notebook computer.

The hardware entities make extensive use of reprogrammable components, including PIC microprocessors and Field Programmable Gate Array (FPGA) hardware. The former is programmed using a simplified version of the C language, and the latter is configured using the Verilog hardware description language. The use of reprogrammable components has many advantages. Firstly, the protocols used can be modified for experiments or performance optimisation. Secondly, a given piece of hardware can perform a number of functions; for example, an object controller can be reprogrammed to support a different type of object. Finally, development and maintenance of the system is facilitated by the ease of reprogrammability.

This section describes each of the prototype hardware components in detail.

2.7.1 Tile Controllers

The tile controllers are designed to handle 24 pads, arranged as 12 pads in 2 columns on a 25×25cm tile. This tile size was chosen for ease of manufacture. The use of 24 pads per tile is based on the size of object footprint this necessitates, and the need to have large surface

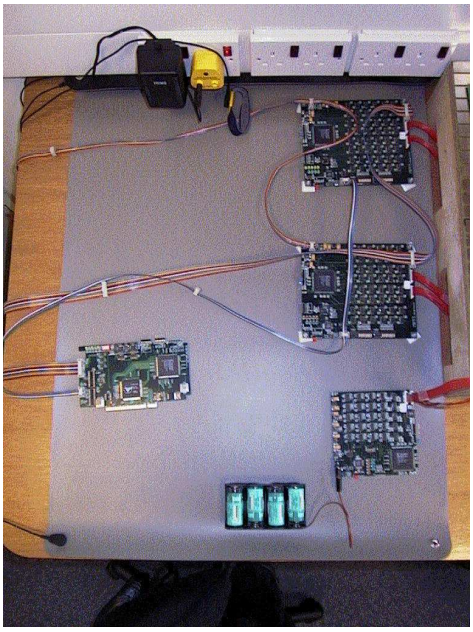

Figure 2.8: Photograph of Surface-Side Hardware

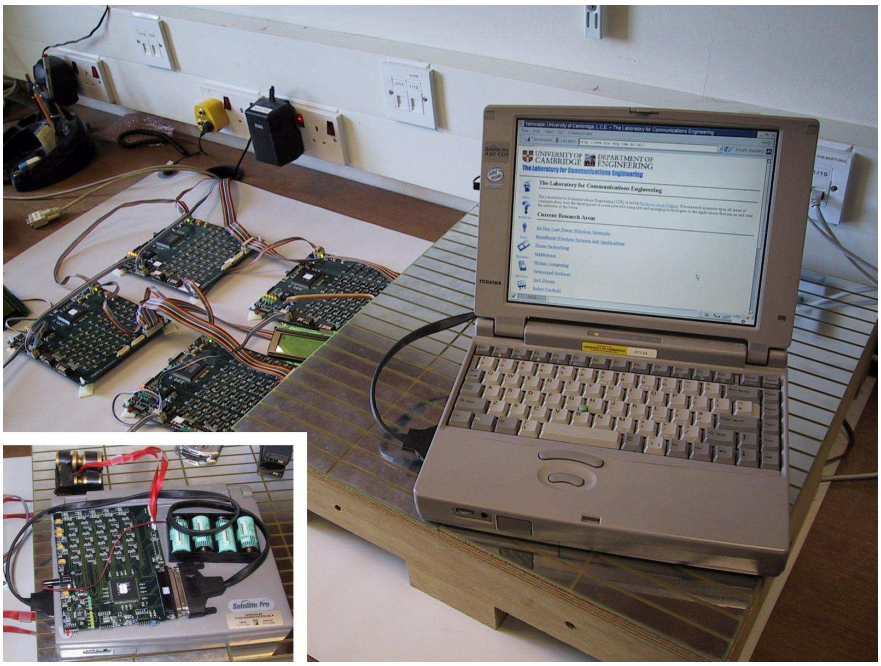

Figure 2.9: Photograph of LVDS Object on Prototype Surface

pads so as to span a large area using as few pads as possible. A four-link object requires a footprint of about 7cm diameter using this topology, which is small enough to fit on the bottom of objects such as PDAs.

The use of two columns of pads has useful properties. Firstly, it makes it possible to have a higher density of objects on a surface; since surface pads in use by an object are unusable by other objects, it is important not to have surface pads that are too wide. An object such as a notebook PC will likely cover the pads it is using in this scheme. This means that it would be possible to fit notebook PCs side-by-side while retaining connectivity for each.

Also, the need for only one pad under each object to be "beaconing" at any one time necessitates that, if a one-column tile were to be used, it must refrain from beaconing half the time. With two columns on each tile, it is possible for each tile to be beaconing at all times, thereby maximising the use of the tile hardware.

The tile controllers are comprised of a FPGA, a PIC microprocessor, and 48 analogue multiplexers ("muxes"), two per surface pad.[5] There are two configuration switches giving a total of 16 configuration bits, set by hand. These configuration switches are useful in two ways, firstly to give each tile a unique identifier for use when communicating with the manager, and secondly to put the tile into various debugging and evaluation modes. Finally, there is support circuitry including a configuration PROM for the FPGA, programming headers, and power circuitry. The architecture of a tile controller is shown in Figure 2.10.

On the tile controller, the hierarchy is simple. The analogue muxes can switch a pad between an open circuit, ground, PIC handshaking lines, 5V power, and any of 10 function buses. They are all controlled by the FPGA, which simply acts as a latch for the mux control lines for the use of the PIC.

The PIC has three interfaces: an interface to the FPGA to set the mux control latches, an I^2C interface to the tile control bus, and an RS-232 interface for handshaking which is routed to each multiplexer. The PIC runs on a 20MHz clock, but only executes one instruction in four clock cycles, giving 5 MIPS. The FPGA also uses the 20MHz clock.

2.7.2 Surface Manager PCI card

The PCI card is designed as a multi-network NIC, and interacts with software running on the manager. It is comprised of two LVDS interface chips, an FPGA, and a PCI bridge chip. The FPGA uses the interface chips to drive two LVDS function buses. It also directly drives two I^2C buses, one for the low-speed data bus and one for the tile control bus. The FPGA is connected to the bridge chip via an i960 bus [53]. This structure is shown in Figure 2.11.

[5] Each pad has two 8-to-1 muxes, which are used to create a single 16-to-1 mux.

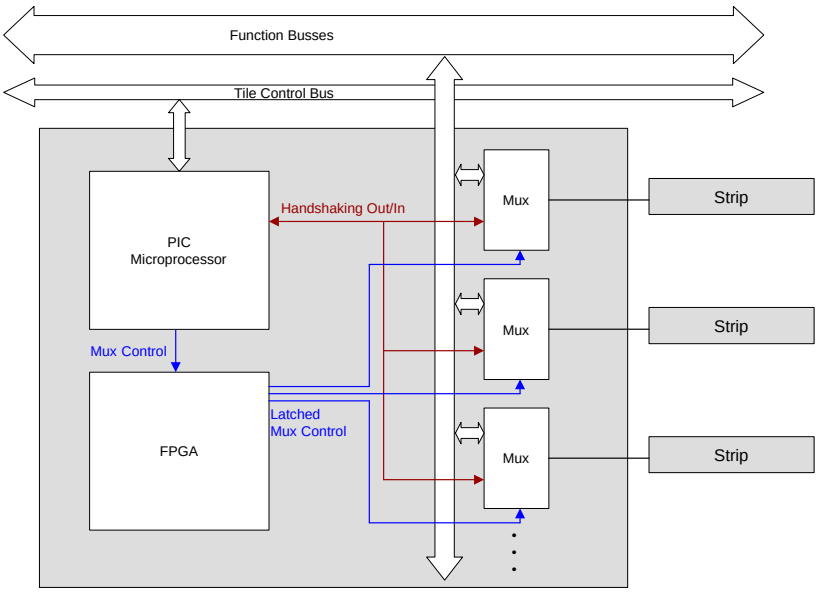

Figure 2.10: Block Diagram of Tile Controller

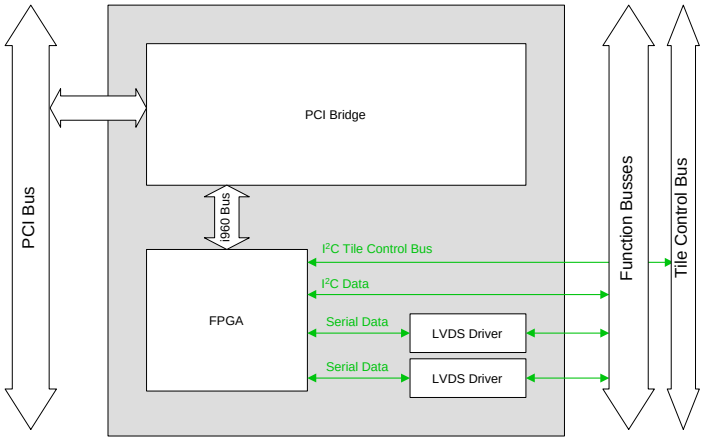

Figure 2.11: Block Diagram of Surface Manager PCI Card

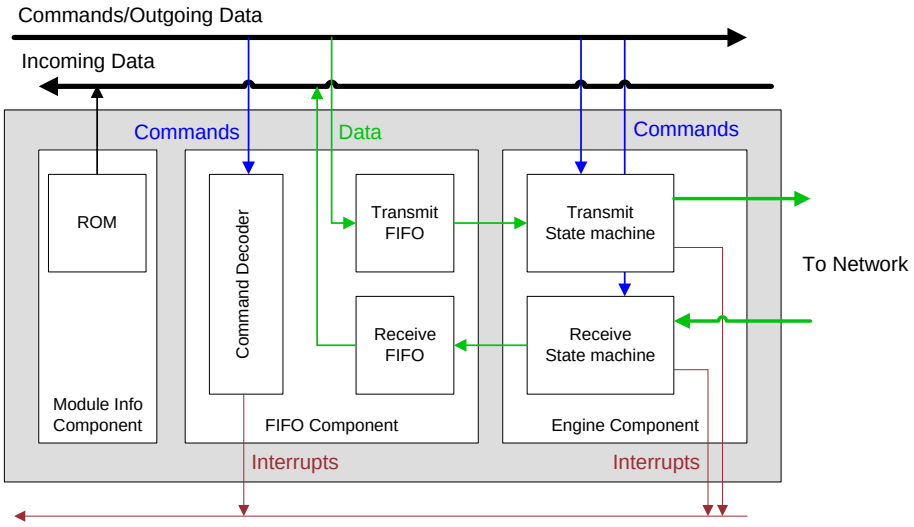

Figure 2.12: Block Diagram of Single FPGA Module

The FPGA is of limited size, and is unable to support the operation of all four buses simultaneously. It can however support any three buses at one time (though the tile control bus must always be one of the three). The reprogrammability of the FPGA is therefore used to change the buses driven, depending on the application of the prototype at any given time. While a larger FPGA (in terms of the number of programmable logic units, or "logic blocks") could have been used, the FPGA chosen is sufficient to test the prototype system, and was cheaper.

Internally to the FPGA, each network interface is assigned a *module*,[6] which performs the functions of a NIC for that bus. The modules are themselves divided into three "components," a "rom" component, a "fifo" component, and an "engine" component. The "rom" component contains a Read-Only Memory (ROM) with static information on the module. The "fifo" component contains First-In-First-Out (FIFO) buffers for outgoing and incoming data for that network. The "engine" component contains logic for obeying the bus discipline for that interface, for transmitting data from the outgoing FIFO, and receiving data into the incoming FIFO. The structure of a module is illustrated in Figure 2.12.

The interface on the PC side takes the form of an address map, which is organised similarly to the above hierarchy, with address lines corresponding to module and component addresses, and other lines for addressing within components. Each engine and fifo component

[6]Not to be confused with the Verilog keyword "module."

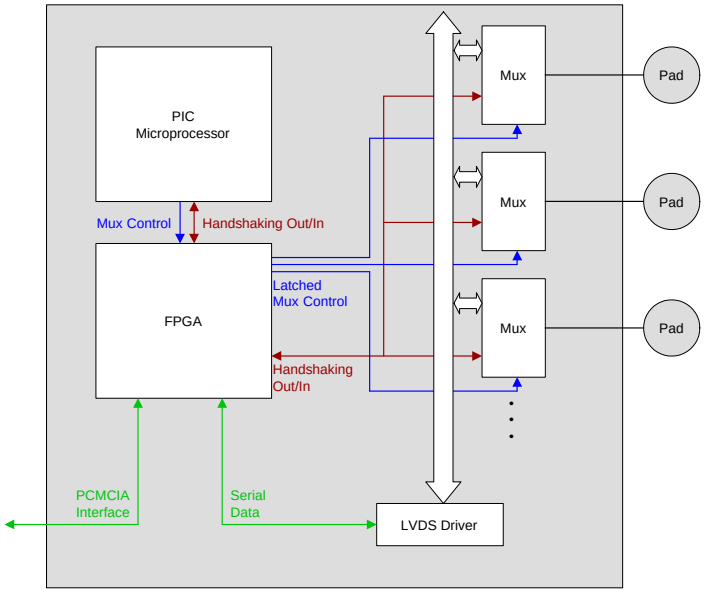

Figure 2.13: Block Diagram of LVDS Object

has interrupt flags, which are raised when events occur warranting attention from software. These are masked with "interrupt enable" registers, and then combined to form a module interrupt line. A further mask is used to hide interrupts from unused modules, before the module interrupts are combined to form the FPGA's global interrupt request line. This is fed through the PCI bridge chip to becomes a PCI interrupt, and allows the FPGA to signal various events to the software driver.

2.7.3 LVDS Object

The LVDS object includes a PIC, an FPGA, an LVDS driver chip, 20 analogue muxes, a grounding-by-consensus resistor network and comparators, and a PCMCIA interface, as well as supporting circuitry. It is designed to provide Networked Surface capability to any PC or PDA supporting PCMCIA cards, in combination with appropriate software. The layout of these components is shown in Figure 2.13.

The LVDS object needs to perform functions similar to both the tile controller and the surface manager PCI card. On one hand, the PIC performs the object side of the handshaking protocol. On the other hand, the FPGA contains NIC functionality, communicating directly with the object's internal PCMCIA hardware, through a cable and a "breakout" PCMCIA card.

The object has 20 muxes, giving the flexibility to support up to six functions, as specified

by the topology presented earlier. The muxes route pads between an open circuit, ground, the LVDS driver chip, directly to the FPGA (for testing), and for handshaking purposes. Circuitry supporting power transfer through the Surface is not included for complexity reasons, as the prototype was designed primarily with the aim of testing the networking capabilities of Surfaces.

Handshaking

Handshaking lines from the muxes are connected to a grounding-by-consensus resistor network and comparators, and then fed in parallel into the FPGA, unlike on the tile controller where those lines feed directly into the PIC microprocessor. This is because objects must be able to listen on each of their pads in parallel for a beacon, and the FPGA is more suited for parallel tasks than the PIC.

In an FPGA, parallel handshaking is simple. Each pad is watched until one is found with a "high" logic level from the grounding-by-consensus comparator. This signal is then routed by the FPGA to the PIC's RS-232 input, with the pad number sent to the PIC simultaneously on other data lines. No action is then taken until the PIC indicates parallel handshaking should be reset, at which point each pad is watched once again. The PIC can choose which pads are candidates for detection in this system by controlling the muxes; like the tile controller, these are controlled via latches in the FPGA.

Note that the FPGA does not judge whether an incoming signal is indeed a beacon or any other type of traffic, it simply stops the detection process when it first sees a "high" signal. During handshaking, the PIC is the "controlling entity" on the LVDS object controller; the FPGA is not programmed with any knowledge of the handshaking protocol.

Connection

When a connection is made, the PIC becomes passive and waits for disconnection, while the FPGA and software driver come into play to register the object, and then provide networking. The FPGA includes a structure similar to the surface manager PCI card, but with only one network, to act as a NIC for the object PC or PDA.

The changeover between handshaking and connected modes is facilitated by an extra FPGA module, called the "object" module. This module does not provide a network, but provides control and status registers for use during connection and disconnection.

On connection, the PIC writes information about the pads it has connected into the FPGA, these are stored in RAM which appears as part of the object module address space, so that the software driver can obtain the pad information it needs to register the object.

The PIC then signals the FPGA that connection is complete, causing an interrupt flag to be raised in the object module. This signals the software driver to start the registration process.

When the software detects disconnection, it writes to a register in the object module, which changes the logic level on a wire in the FPGA to PIC interface, signalling the PIC to disconnect the object's pads and restart the handshaking protocol. Hardware-based disconnection detection is not used in the current prototype; as described previously it is only appropriate in particular cases, for example when power is provided.

2.7.4 I^2C Object

The I^2C object hardware is similar to the LVDS object, but without the PCMCIA interface. The hardware is designed to handle lower speed and non-networked objects, and therefore must perform low-speed bus bridging, and must include object manager functionality.

Unlike the LVDS object, the PIC retains control at all times, executing the registration protocol itself, and then monitoring the network for disconnection by responding to manager "ping" signals, timing out if no "ping" is received in a given time period. The "ping" is not an IP-layer ping; since the I^2C object is not equipped with an IP stack, the "ping" used is built into the registration protocol for this purpose.

This hardware is not designed to connect objects directly to surface buses, since doing so would require the objects support particular bus types such as I^2C. In order to support a broad range of low-speed buses, the PIC and FPGA can be used to bridge between object's native bus format and I^2C. For example, a keyboard's PS/2 interface could be decoded and the data sent as I^2C.

2.8 Prototype Software

Software is used in the prototype system to perform the functions of the surface and object managers (with the exception of the I^2C object). This includes the registration protocol, and the tile control protocol on the surface side. The surface manager must also maintain data on objects on the Surface, in order to perform network layer routing, enable disconnection detection, and provide location and orientation information.

The software is written in the C programming language, and runs on Linux kernel version 2.4. It comprises two components on the surface side, namely a kernel-level device driver, and a user-level daemon. On the object side no daemon is required, as all the work is done in the device driver.

The device driver is responsible for interacting with the hardware, and transferring data

between the FPGA "modules" mentioned above, and suitable "destinations" (e.g. a Linux network device). On the object side, the registration protocol is also executed by the device driver. The surface-side user-level daemon is responsible for the maintaining data on tiles and objects present on the Surface, and executing the tile control and object registration protocols.

Both of these entities are explained in detail below. Firstly, however, a brief introduction to the relevant parts of the Linux operating system is presented.

2.8.1 Linux Operating System Model

Linux, like any operating system, must provide abstractions for the hardware of the computer, so that applications can execute high-level operations without knowledge of the intricacies of each individual device. These applications are said to run at *user-level*, which is a controlled execution environment in which direct hardware access is not permitted.

Abstraction is achieved using *device drivers*, which are bodies of code designed to perform low-level communication with a particular type of device. The device drivers communicate with the operating system core (the *kernel*) to provide abstractions for these devices, such as "character devices" or "network devices."[7] While character devices provide raw data transfer between user-level programs and device drivers, network devices instead feed into the Linux IP stack. User-level programs access this stack by using TCP and UDP *sockets* to communicate with remote hosts.

In addition to sending data to and from the device abstractions, other mechanisms exist for interaction, two of which are relevant to this discussion. *Signals* are a method of asynchronously notifying a user-level program of an event, and may be sent to a program by itself, by another program, by the kernel, or by a device driver. *Ioctls* are a method of transferring out-of-band data between user-level programs and device abstractions (and hence device drivers), for example, the hardware address of a network card could be changed by a user-level program, which would execute an ioctl on the corresponding network device.

To illustrate data transfer using this hierarchy, the process of sending a network packet is used as an example. This packet originates in a user-level program (e.g. `ftp`), which uses a TCP socket to pass the data to the kernel networking stack. This stack adds necessary headers, and the proper network device for routing the packet is chosen. The kernel then passes the data to the device driver responsible for that network device, which sends it on to the NIC using an appropriate device-specific format, along with the commands causing

[7] The word "device" is unfortunately overloaded in the Linux world; it is used in the term "device driver," and also in the abstractions with which devices are represented (e.g. "network device," "character device").

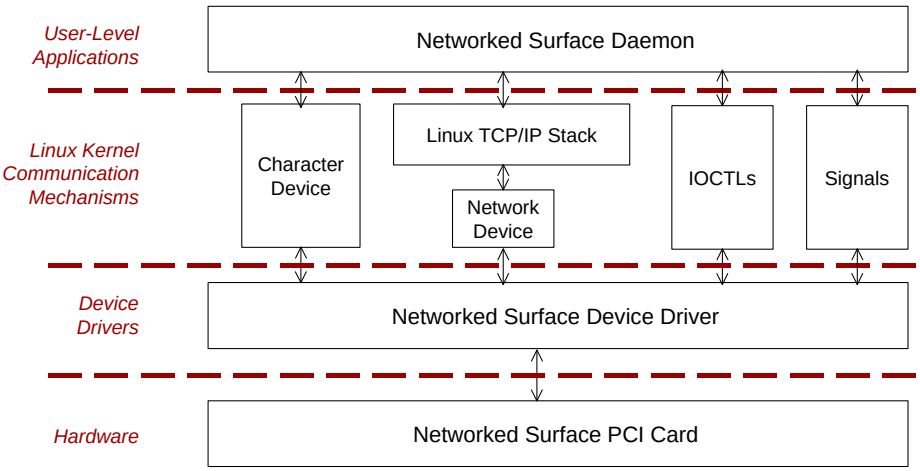

Figure 2.14: Linux and Networked Surface Software

transmission to take place. Received packets would simply follow a reverse path, with the proviso that if the packet received was not destined for the computer in question, the kernel IP implementation would instead re-route the packet back to another network device, for transmission on the next network hop.

The Linux operating system model, along with the location of the various Networked Surface software components, is illustrated in Figure 2.14.

2.8.2 Device Driver

As stated, the device driver must act as an intermediary between the Networked Surface hardware and Linux data channels, such as character devices and network devices. In addition, it must perform control functions as required for the hardware, such as error handling. In cases where the hardware does not perform link layer functions, it must also execute those functions, e.g. for bus arbitration or address filtering.

Because of the complexity of the hardware, the device driver is best designed in layered fashion rather than as a single unit. Furthermore, so as to support different versions of the hardware, it is useful to have a dynamically configurable driver, which scans the hardware in order to determine which resources are present and then sets up the appropriate structures. Using this method, the same compiled driver can cope with a variety of surfaces and objects, without user configuration.

The driver layering is shown in Figure 2.15, and is as follows.

The “media” layer provides the means of communications with the hardware. In the current

prototype there is a PCI version for the surface manager card, and a PCMCIA version for the LVDS objects. They provide abstracted and non-bus-specific communications primitives for the rest of the driver.

The "general" layer only has one implementation, which is present in every driver. It provides an entry point for the kernel to execute startup and shutdown functions, and for interrupt handling. In addition, non-network functions such as global interrupt masking, global status (such as a "this-is-an-object" flag) and object connection and disconnection functions belong in this layer. Interrupts are handled by examining the status flag for each module, and calling the engine interrupt handler for all modules with pending interrupts.

The "engine" layers are the controlling unit for each network, and communicate with their hardware counterparts (the "engine" components of each module). When they receive interrupts from the "general" layer, they are responsible for causing the data layers to transfer data to and from the hardware FIFOs as appropriate, and also causing the destination layer to be notified when there is a completed incoming packet. This layer is also responsible for executing parts of the link layer[8] protocol used, in particular for the bespoke arbitration scheme created for LVDS buses, which is discussed further in Chapter 3.

The "data" layers are identical to one another, and handle the buffering of data in the driver. The use of a separate and globally similar entity for this allows reduction of data copying, and re-use of data buffering code. This is a better solution than explicit data buffering and transfer methods between the engine and destination layers for each network.

The "destination" layers are responsible for interacting with the Linux interface for the appropriate type of device. They are given new outgoing data by the kernel, and new incoming data by the engine layer, and in each case are responsible for causing the data layer to fetch or send data appropriately.

The use of such a generic driver has many advantages. Firstly, as stated, a single driver is capable of coping with many types of Networked Surfaces. Secondly, the driver is easily upgraded to support a new engine or destination. Thirdly, the layering allows the same engine to be used with different destinations, and vice versa. This, for example, is useful for

[8] "Link layer" refers to the OSI network model's layer 2, and not a layer in the driver implementation.

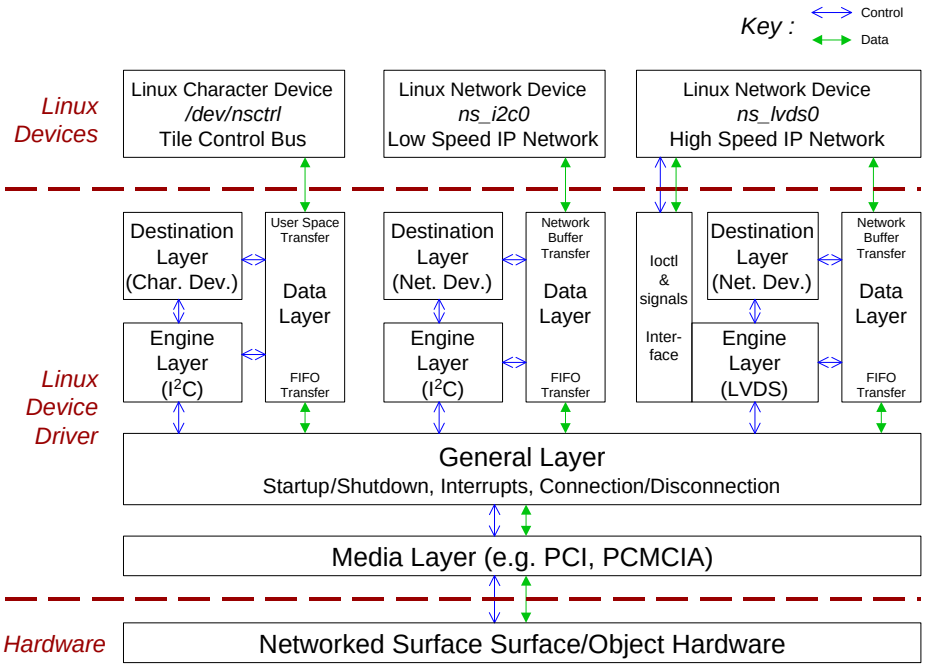

Figure 2.15: Device Driver Layering Model

I^2C, which is provided as a character device in the case of the tile control bus, but may also be presented as a network device if an I^2C bus were used as an IP link layer. This promotes code re-use and maintainability.

Finally, the device driver also executes the object side of the registration protocol; a separate user-level program is not used in the object case for reasons of simplicity. This is accomplished by implementing connection and disconnection functionality in the "general" layer, since such issues are not specific to the particular network used. On connection, the surface and object pad data is obtained from the hardware, and a "new object" message is sent on the (newly available) network to the surface manager. When a reply is received, any necessary configuration of the link layer is completed, and networking is activated.

2.8.3 Driver/Daemon Interfaces

The surface manager's user-level daemon needs interfaces to the driver in order to send and receive object registration and tile control messages. For the latter, the user-level daemon can simply interact with the bus using a Linux character device, with the driver employing the I^2C engine and character device destination layers. However, the object registration messages are intermingled on the LVDS bus with other network traffic, so a simple character

device interface is not possible in this case; a network device is required.

Using a network device, one method of supporting the object registration protocol is to provide support over IP, i.e. using TCP/IP or UDP/IP to a specific port. However, this would exclude non-IP-capable objects from the LVDS bus, and make the registration messages subject to higher latencies as they have to traverse the IP stack in order to reach the daemon.

In the interests of flexibility, an alternative interface bypassing IP was devised. This uses the Linux mechanisms of signals and ioctls. Ioctls are used for message transfer between the daemon and driver, and signals are used to allow the driver to request an ioctl message, since the driver may not initiate ioctls itself.

Finally, the link layer may have Networked Surface–specific functionality which requires co-ordination with the daemon, for example for link layer disconnection detection. This is also achieved using a Linux signals/ioctls interface.

2.8.4 User-Level Daemon

The user-level daemon, named `surfaced`, performs the management functions for the Networked Surface, such as the maintenance of lists of active tiles and objects. The daemon also communicates with tiles and objects via the tile control and object registration protocols, respectively.

While the same functions could have been performed in kernel space, this is undesirable. A kernel space program is harder to maintain, cannot make full use of C libraries, is difficult to test, can crash the machine it is running on, and does not participate fairly in scheduling. For these reasons, a user space solution is preferred. In contrast, the device driver described above must by its nature be in kernel space, to directly access hardware, to receive interrupts, and to provide a low-latency response to interrupts.

The structure of the daemon is illustrated in Figure 2.16, and explained below.

Tile Control

The daemon is responsible for initialising the tile controllers, and then keeping them in "sync." When a tile initialises, the daemon sends it is sent an I^2C address for the tile control bus, and a list of functions that are present on the Surface. This means that tiles do not have to be manually reconfigured if rearranged or used with a different Surface. During normal operation, the daemon sends "sync" tile control messages at strict intervals, to start each cycle of handshaking on the tiles.

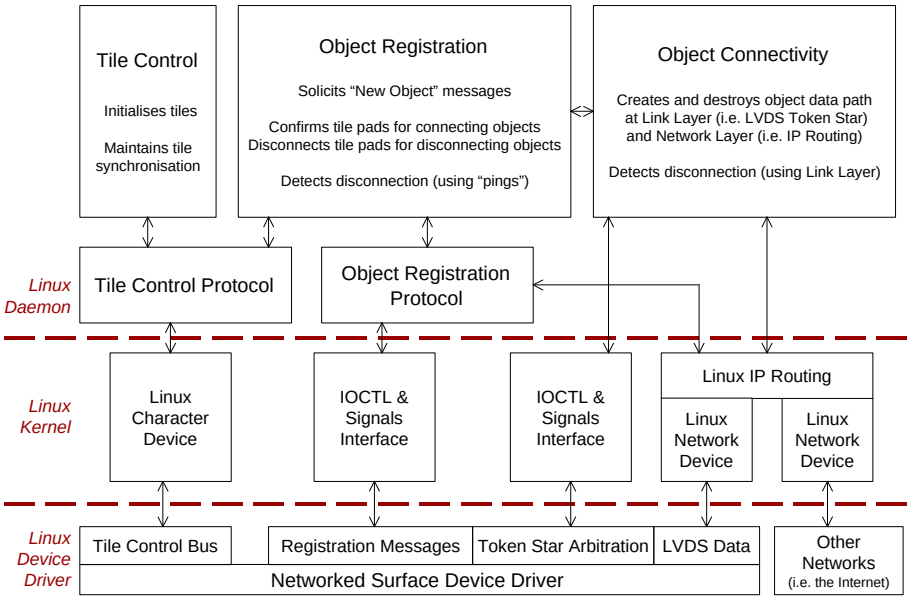

Figure 2.16: Surface Manager Software Daemon Structure

Object Registration

When an object registers, it sends details of the connection it has made to the surface manager. This message includes tile addresses, tile pad numbers, and function numbers, for each link. This message is received by the daemon either via a signals/ioctls interface, or via an IP packet, as described earlier.

When the daemon gets this message, it must construct and send tile control messages to each tile that the object has a pad on (i.e. between one and four tiles), in order to "confirm" the surface pads, and prevent them being timed out by the tiles. After the daemon receives confirmation acknowledgement from all tiles, it must send a message to the object confirming that the connection is valid. Finally, the daemon may be required to perform configuration of link layer and network layer state so that the new object may use the network; this is discussed in later chapters.

Object Disconnection

As previously mentioned, object disconnection can either be detected in a Networked Surface–specific link layer, or it can be done using the object registration protocol by sending periodic "pings."

In the case of the former solution, the daemon must simply wait for the driver to signal

that the object has disconnected. In the latter case, the daemon must periodically send "ping" packets to the object, whilst maintaining some kind of timeout to infer disconnection.

When disconnection is determined (using either method), the daemon must send messages to the tiles with pads connected to that object, to disconnect the pads. It must also reconfigure the link layer and network layer state as appropriate so that the object is no longer expected to be present.

2.9 Alternative Designs

This section will discuss alternative designs for Networked Surfaces, and their strengths and weaknesses as compared to the design chosen for the prototype.

2.9.1 Optimisations to the Existing Prototype

The existing prototype was designed to be as flexible as possible, so that support for many classes of devices could be demonstrated. In retrospect, a couple of optimisations can be identified, which sacrifice some of this flexibility in order to make savings in complexity and in cost.

Analogue Multiplexers

All Networked Surface links are currently made via two analogue multiplexers, one on the surface side and one on the object side. However, there are a number of other means of performing this switching. Note that this discussion does not cover the provision of power, which would require separate switching hardware in any case, as the muxes used in the prototype cannot cope with high enough currents.

One option is to perform multiplexer functions inside the FPGA. This is only possible for certain bus types; an FPGA cannot in general connect two bidirectional lines. However, there are special cases for which such switching can be performed in the FPGA, including "wired-AND" buses, and simplex (i.e. one-way) buses. The former encompasses I^2C, and the latter is used by the (RS-232–based) handshaking. The drawbacks of using the FPGA in this way are that only digital bus types can be supported, and that a propagation delay would be incurred. Also, additional hardware would be required to make grounding by consensus work with this switching system.

A further possibility is to use FPGA switching for the signals which are amenable to such switching, as described above, and to use analogue multiplexers to switch between the other

signals. This would lead to a more complicated control system with two levels of switching, but it would reduce the number of analogue multiplexers required.

Non-Generic Pads

Another possible optimisation would be to limit the number of functions each pad can provide, as this would also reduce the complexity of the switching required. This was previously discussed, but rejected as it imposes restrictions on the object footprint and on the functions that can be guaranteed. However, optimisation is possible if assumptions are made about the type of Surface used.

For example, a Networked Surface can be envisaged on which all objects require one network, one ground line, and one power line, and where each network type uses two physical links. In this case, each object would span four surface pads in a column. The optimisation this leads to is to make each surface pad only provide one of the four "groups" of functions which an object might request. This is illustrated in Table 2.2, for a Surface supporting I^2C and LVDS in this fashion.

Surface pad numbers	Functions provided
0, 4, 8, 12...	Ground
1, 5, 9, 13...	3.3V or 5V or 9V
2, 6, 10, 14...	I^2C "SDA" or LVDS "+"
3, 7, 11, 15...	I^2C "SCL" or LVDS "−"

Table 2.2: Example of Functions Available using Non-Generic Pads Scheme

Hardware Integration

Simplification can also be achieved integrating hardware components so that fewer are required. In the case of the Networked Surface hardware, this could be done by including as much functionality as possible inside the FPGA. One such possibility along these lines was detailed above, namely the inclusion of switching in the FPGA.

Further integration could be achieved by including a microprocessor core inside the FPGA, which would replace the PIC microprocessor. This is not possible with the existing FPGA, which does not have enough logic "blocks" to support this functionality. However, FPGAs are available which are large enough to support such cores, and which also have built-in RAM. Furthermore, there are chips available combining some FPGA-style programmable

logic, a hardware microprocessor, and RAM.

This may be especially useful on the object side, as Networked Surface object hardware should ideally be small enough to conform to a PCMCIA card form factor.

Construction of Topologies

In the prototype, printed circuit boards (PCBs) were used for the fabrication of tiles and object footprints. While these are sufficient for a prototype, they are prone to problems causing bad electrical contact, including warping of the object footprint, and dust or dirt getting between the boards. This does not however indicate any fundamental problems in the use of Surfaces, as there are many other ways to construct physical implementations of the topologies described.

One possibility would be to use spring-mounted pins instead of pads on the object side, so that the object does not have to be perfectly flat. Also, the tile and object footprint may be made of (or lined with) a flexible material, so that the weight of the object causes a perfect contact between the two.

Finally, designs for the object footprint which are easily mounted onto the undersides of potential object types are also of interest. Many notebook PCs have removable sections of casing on their base, used for access to internal components. Custom versions of these removable sections could be constructed with an object footprint included. Also, recent models of PDAs use PCMCIA-compatible "sleeves" to support add-on hardware. Such a sleeve would be an ideal way to enclose both the object hardware and the physical footprint.

2.9.2 Densely Populated Surfaces

The architecture presented above is designed around the assumption that there will be many surface pads unoccupied at any one time. This has led to the requirements for minimisation of complexity on the surface, and to the divide between the tile controller functionality and the surface manager functionality.

However, a Surface could be imagined which is designed to cope with a "dense" placement of objects, i.e. objects on top of most surface pads. In this scenario, the use of shared buses becomes undesirable, as it would result in either many objects sharing each bus, or many buses being routed around the Surface.

One solution would be to restrict the area of buses, so that different buses would span different areas of the Surface. However, if this idea is pursued, it is noted that the surface manager eventually has to handle a bus for each small group of tiles, or at the extreme case, each tile. In this situation, there is no scalability divide between the surface manager and

the tile controller any longer. One can then imagine architectures where each tile controller provides its own surface manager functionality, i.e. each tile independently bridges data between the objects on top of it and the wired network.

This could be done by ensuring that each object only receives networking functions from a single tile, but allowing ground and power functions to be provided by other tiles. In this case, each tile would act as manager for objects for which it provides networking functions. This would involve confirming object pads on itself and on other tiles that may be used for power and ground when an object connects, and causing disconnection of pads on itself and on other tiles when an object disconnects.

The advantage of this scheme is the bandwidth available; such a scheme could provide the maximum bandwidth available in the Surface medium to every object, no matter how densely packed on top of the surface. Another advantage is scalability, in that since each tile manages its own objects, the addition of new tiles would be very simple. The disadvantages are that the tiles must perform the functions of the surface manager themselves, increasing their complexity.

2.9.3 Other Physical Media

The final issue to be examined is the use of electrical conduction as the method for data transfer. While the possibilities outlined below are not within the scope of this thesis, a short summary is presented. More details are given by Hoffmann [48].

Capacitive Networking

The use of capacitive coupling employs an electric field instead of electron transfer for the surface/object interface. The advantages of capacitive coupling are that no ground connection is required, and no possibility of short circuits on the Surface exists. The disadvantages are that the channel is likely to achieve lower bandwidths than conductive channels, is more prone to noise, and is more complicated, requiring specialised modulation and demodulation circuitry for each pad.

One appealing design for a capacitive-based Surface is where only a single network is available across the whole Surface. Since only one function is required for each object, the handshaking protocol is unnecessary, and the network may be provided on all pads regardless of whether an object is present or not. This design is appealing for its simplicity, as it requires only modulation and demodulation hardware for each pad, and no “tile controller” units. However, the bandwidth provided would have to be shared amongst all the objects present. This could be alleviated by restricting the area of each capacitive network, so that each

network is shared between fewer objects.

Inductive Power

Induction can be used for networking also, but another appealing use would be to provide power. Inductive power has the advantage of not requiring pads to operate, but instead uses coils. Furthermore, these coils could be placed behind pads, thereby separating power from networking, and potentially allowing both to be provided using the same physical area. Inductive power is also immune to short circuits, providing guarantees of safety. However, it is not as efficient as electrical conduction, so it is not easy to provide sufficient power using this method. Inductive power has been used in a number of applications, including the recharging of common household electric items such as toothbrushes.

2.10 Summary

This chapter has explored has explored the design and implementation of a prototype Networked Surface. Using the principles of flexibility, sparsity, and simplicity, a distributed architecture was designed. The physical topologies providing guarantees of connection regardless of position and orientation of devices were presented, as well as the various hardware and software components that make up the prototype Surface. Finally, possible improvements to the prototype Surface design were described, as well as alternative designs for Networked Surfaces.

Chapter 3

Networking with Networked Surfaces

3.1 Introduction

This chapter discusses the formation and use of networks on Networked Surfaces, i.e. it examines the various protocols and schemes which govern movement of data between the surface manager and Networked Surface objects.

Included in this discussion are the topics of the handshaking and registration protocols, the bus interface for the high speed LVDS bus, the methods used to transfer data across the software/hardware interface, and the arbitration scheme used on the LVDS bus. These topics fall mainly into the link layer of the OSI networking model, with the exception of the handshaking and registration protocols, which are used to establish the physical layer required for networking to occur.

The work presented below was in part undertaken collaboratively with Frank Hoffmann. Joint work includes the sections on handshaking, registration, and the software/hardware interface. The work on link layer protocols, including the "token star" protocol, is that of the author alone.

3.2 The Handshaking Protocol

The previous chapter presented a distributed architecture for a Networked Surface, and noted that this implies a distributed two-phase connection protocol. The first phase involves "handshaking" between the tile and object controllers, while the second "registration" phase involves surface manager to object manager communication, as well surface manager to tile

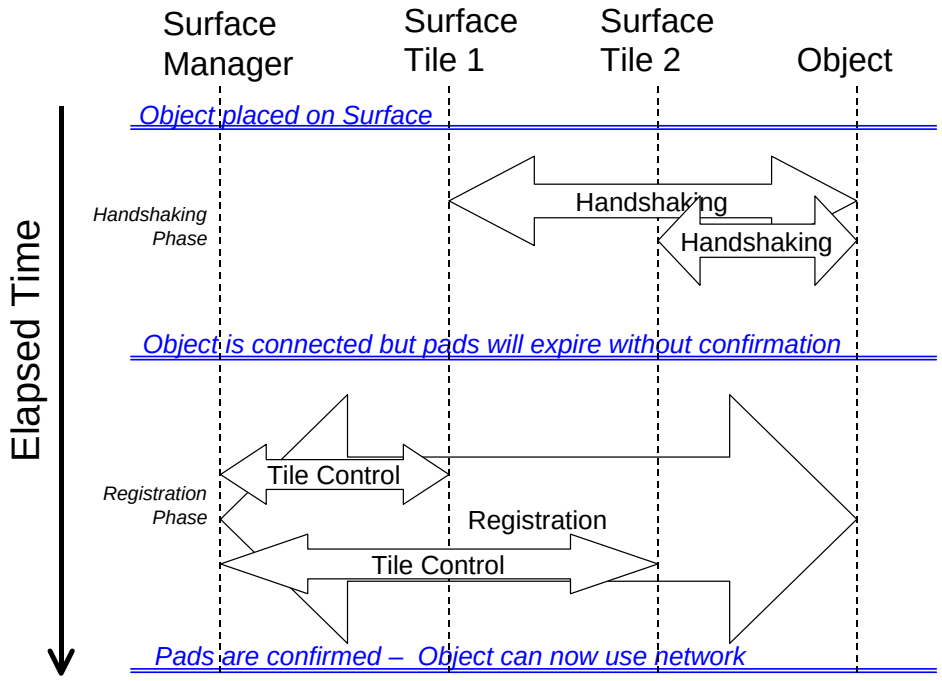

Figure 3.1: A Two-Phase Connection Process

controller communication. This process is illustrated in Figure 3.1.

Also in the previous chapter, the problem of grounding was discussed, as was the "grounding by consensus" solution proposed, whereby a resistor network is used on each object. This network allows the object to use the average voltage level present on its pads as its ground reference. The use of such a network has two implications for handshaking. Firstly, the initial messages in any protocol transaction, the "beacons," are best sent by the tiles and not the objects. Secondly, only one tile pad underneath each object may be used to send protocol messages at any one time.

3.2.1 Protocol States

The handshaking protocol is concerned with the exchange of information between the tile and object controllers, in order to decide upon and synchronise changes in the state of the tile pads and object pads. This is achieved by using a tile pad and object pad pair for half-duplex communication. The overall goal is to move a number of pads from an initial "handshaking" state to a final "connected" state.

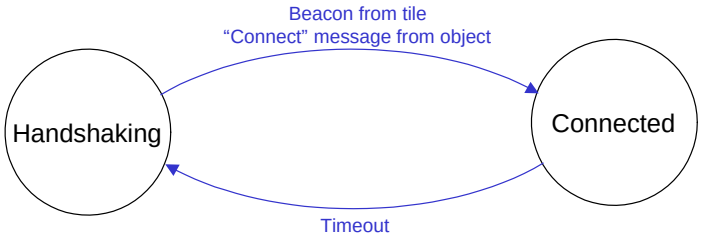

Figure 3.2: Pad States in a Two-State Handshaking Protocol

Pad State Dependencies

The tile controllers must keep independent state for each of the pads they control, since an object may span many tiles, and many objects may be present on a single tile. Each handshaking protocol transaction therefore only affects the tile pad on which it is performed.

For object controllers, the pad states are not independent, since many object pads may rest on the same tile pad. Ideally, the object controller would be able to detect a tile message on many pads simultaneously, and keep track of this many-to-one association. However, meeting this ideal is not necessary for handshaking to function; only one object pad is required to be active for protocol communications between the Surface and object, and a solution using only one pad would need simpler hardware. This approach requires some mechanism of ensuring that the same object pad is chosen for subsequent transactions (if any) with that tile pad, so that the object controller uses the correct state variables. This is achieved in the prototype by ensuring that the lowest-numbered object pad is always chosen, whenever a message is received on many pads simultaneously.

The object controller must also keep “global” state, in addition to state for each pad. This includes the list of functions that are required. When a beaconing tile pad is encountered, this list must be consulted in order to determine which function to request. When the functions in this list are all connected, the object controller must signal the object manager to start the process of registration.

State Transitions

The simplest conceivable handshaking protocol only requires the two pad states outlined above. When an object controller receives a tile beacon, the object controller asks the tile controller to connect it to a function. This would continue until all required functions are connected. The pad state diagram would look like Figure 3.2.

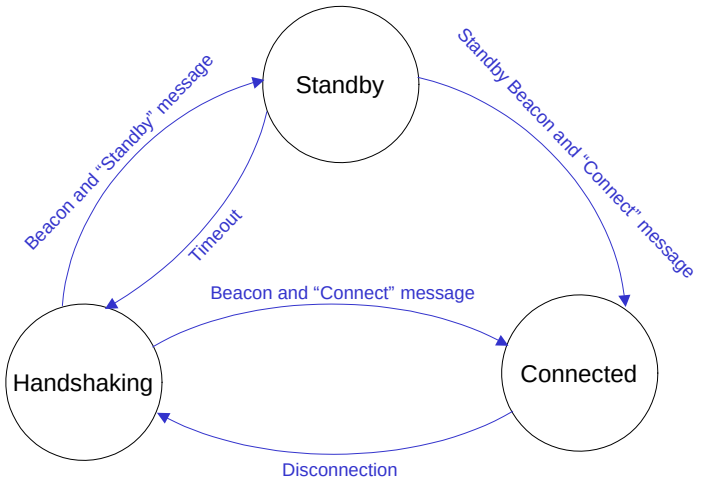

Figure 3.3: Pad States in a Three-State Handshaking Protocol

However, this simple protocol has a number of possible drawbacks, discussed later, which stem from the immediate leap between the "handshaking" and "connected" states. The protocol was therefore modified to include another state, named "standby." In this state, a tile or object pad is reserved for a particular function, but is not yet connected to that function, and is still under the jurisdiction of the handshaking system.

This three-state approach allows an object controller to acquire pads without immediately connecting them. Using this method, an object controller can ensure enough links are available for a connection, and determine which of its pads will be used for that connection, before asking for those pads to be connected to the surface buses.

The three-state system is illustrated in Figure 3.3. Note that the use of this state machine does not preclude the possibility of immediate connection (i.e. not using the "standby" state), the three-state system is therefore a generalisation of the two-state system and can be used to test the merits of both systems.

3.2.2 Protocol Design and Implementation

In order to design the protocol, it is important to identify the information that must be transferred. This obviously includes the transfer from object controller to tile controller of a code representing the function requested. In addition, the use of a two stage connection process (handshaking and registration) requires that the object controller must be informed of the tile address and tile pad number for each tile during handshaking, as this information is needed for registration to occur. Finally, in order to not preclude the use of a two-state

connection process, the information detailed above must be sent in the first transaction on any pad (which will be the only transaction in the two-state case).

Hardware

In the interests of scalability at low cost, the prototype uses low-cost PIC microprocessors, which have the advantage of being reprogrammable, facilitating the development and maintenance of the protocol implementation.

The choice of physical bus used is constrained by a number of factors. Firstly, a single half-duplex channel must be used, since the protocol is executed over a single surface pad–object pad link. Also, the bus used must support the transfer of byte-wide messages, which requires bit framing, byte framing, and the ability to discern a transmission from a bus idle state.

In the prototype, handshaking is performed over a modified 9600bps RS-232 bus using start-bit-stop-bit coding. This was chosen as it fulfills the criteria above, and is internally supported by the PIC microprocessor, making the bus trivial to implement. The modifications are that the physical bus signalling is at 3.3V and 0V, since these logic levels are supported by the FPGA, and also that the bus is inverted as compared to normal PIC operation, so that an idle line is represented as 0V. This is useful so that there is no voltage change when a pad moves from grounded mode to transmitting mode, which makes message decoding simpler for the recipient.

Protocol Definition

The handshaking protocol for a single pad is illustrated in Figure 3.4, and proceeds as follows. The tile controller sends a "beacon" message on that pad, which contains the tile pad number. This message is a single byte, being 5 bits for the pad number (0 to 23 in the prototype), and 3 bits specifying the type of message (i.e. "beacon" in this case).

The object controller replies with either a standby-request, or a connect-request, depending on whether it wishes to use the two-state or three-state handshaking process. This request also includes the pad number, for error-checking purposes. The object controller sends a further byte specifying the function required; this byte has a 7 bit function code and 1 parity bit. The tile controller then replies with its address if it can accept the request, or an invalid address (zero) which acts as a denial code if it does not support that function. If a "standby-request" was used, then the tile and object pads are now put into a "standby" mode. If a "connect-request" was used, then the pads are connected through to the appropriate function.

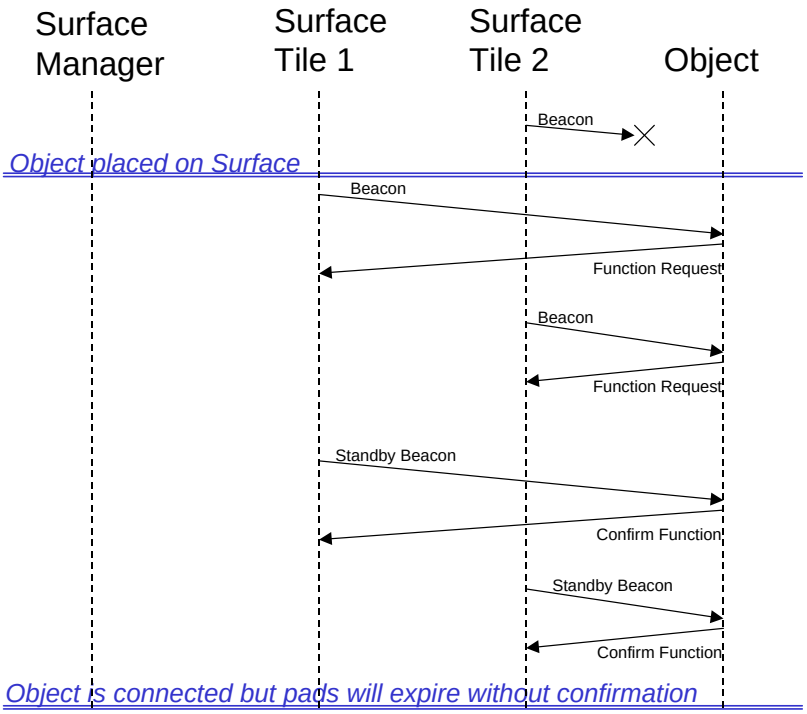

Figure 3.4: Handshaking Protocol (Three-State Variant)

When the current tile pad is in "standby" mode, the tile controller sends a 3 bit "standby beacon" instead of a normal "beacon," as well as the 5-bit pad number. This special beacon is used so that another object controller which may have been put onto the same tile pad does not try to claim that tile pad for itself. The object replies to a "standby" beacon with a single-byte message, again specifying the pad number for error-checking, and using the other 3 bits to specify "no action" or "connect-request." The tile controller replies with an "ack" (again specifying the pad number), and if "connect-request" was specified, it connects the pad through to the appropriate function.

At any stage in this protocol, if either the tile controller or the object controller was to determine an error, either due to failed error checks (e.g. parity checks and pad number comparisons), or improper response, then that party terminates the transaction by simply falling silent. Timers are used so that any such aborted handshakes result in a resetting of the affected pads to their original states, allowing handshaking to start again.

Protocol Timing — Single Transactions

In order to analyse the protocol timing, the timing of a single transaction must first be examined, before looking at timing over whole tiles.

Each handshaking transaction can take a variable amount of time, depending on the type of transaction. A single byte takes approximately 1ms to send, due to 9600bps bus speed and the use of start and stop bits. Between reception of a signal and transmission of its response, the controllers also require some processing time. This time is observed in the prototype to be up to 1ms.

The minimum length transaction occurs if a beacon is sent and not replied to. This takes approximately 3ms (time for the beacon, for processing, and for transmission of any response). The maximum length transaction occurs with a successful beacon and standby or connection request. This takes 4 byte times and 2 processing delays, or 6ms.

Protocol Timing — Tile Synchronisation

The previous chapter noted that for grounding by consensus to function, only one pad may be beaconing to a given object at a time. This maps well onto the use of tiles, with each tile beaconing on only one of its pads at a time. However, the tiles must be kept in synchronisation, otherwise problems may arise in cases where objects straddle two tiles. This is achieved by using "sync" messages from the surface manager, broadcast on the tile control bus to the tiles.

The period of these synchronisation messages (or "cycle time") is key in determining the

time taken for an object to connect. Within one tile cycle, an object is guaranteed to be beaconed on all the surface pads that it spans. Therefore, a successful two-state handshake should take no longer than one cycle.

A naïve approach would be to use the worst case pad time, 6ms, and multiply by the number of pads per tile, 24 in the prototype, resulting in a cycle time of 144ms. However, this worst case only happens once per pad in the lifetime of an object's connection, and the probability that all tile pads are populated with newly placed objects is miniscule. Indeed, such an occurrence may not be physically possible, since a given object may be of much greater size than the "footprint" of pads which is uses.

A more realistic cycle period might be found by looking at the case of two 4-link objects being placed simultaneously. This would require a cycle time of 96ms, or approximately a tenth of a second, and results in connections approximately one third faster than the naïve approach. It is sensible to add a small extra amount to allow for clock drift and worst cases, thus, the cycle time used in the prototype was set at 100ms, i.e. one tenth of a second.

One more parameter must be set — the time spent per pad by each tile. In this case, a naïve approach would simply start processing of a pad immediately after operations on the previous pad had finished. However, this approach would result in tiles with no objects going through their pads in 72ms, and spending over a quarter of their time idle. This discrepancy between rates of cycling for idle and busy tiles may lead to a problem whereby an object straddling two tiles received two beacons at once, due to the loss of synchronisation in the middle of a cycle.

This is corrected in two ways. Firstly, a minimum pad time is imposed, so that the "idle" tiles do not move too quickly ahead of the "busy" tiles. The minimum time chosen is 4ms, this means an "idle" tile completes its cycle in 96ms, leaving 4ms "spare." However, this time is only imposed if the tile has not previously "fallen behind." Tiles which have fallen behind therefore "catch up" to their idle neighbours at a rate of 1ms per idle pad encountered (since they only spend 3ms on idle pads).

Finally, an intelligent choice of tile cycling order is used, shown in Figure 3.5. This has the useful property that objects will never have consecutive interactions on any of their pads.[1] This has two implications, firstly, that a "busy" tile will have more frequent opportunities to catch up with their idle neighbours, and secondly that objects are provided with "free" time between handshaking transactions, in which they may perform other tasks such as the updating of global state.

[1] This statement assumes that objects require five or fewer functions.

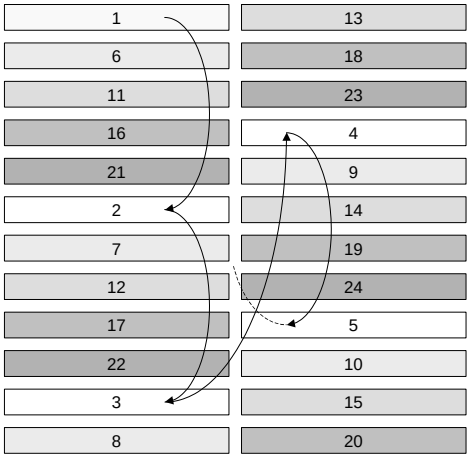

Figure 3.5: Tile Pad Ordering for Optimised Handshaking Timings

3.2.3 Evaluation and Analysis

This section evaluates the two-state and three-state handshaking protocols, presenting timing results for one-object situations, and then looking at two-object situations.

Handshaking with a Single Object

For one-object situations, handshaking times were obtained by using a push button on the object controller to simulate placement of the object on the Surface. This button simultaneously enabled handshaking and started a timer. On connection, this timer was stopped, and the value transferred to the software driver. This timer has a limit of 5 seconds and a granularity of 10µs.

The object was placed at 5 random locations and orientations within a tile boundary, on each of 2 different tiles. 5 locations were also randomly chosen straddling the boundary between the two tiles. 10 readings were taken at each location, with random start times with respect to the position of the tile(s) in the beaconing cycle. Finally, these tests were carried out for both two-state and three-state handshaking modes.

Figure 3.6 shows a comparison of handshaking times between objects on a single tile and objects placed across two tiles. As the graph shows, the use of multiple tiles does not affect either the two-state or three-state versions of the handshaking protocol. This proves that the use of a surface split into tiles for scalability works transparently, and does not impede handshaking.

Figure 3.7 shows a comparison of handshaking times for two-state and three-state hand-

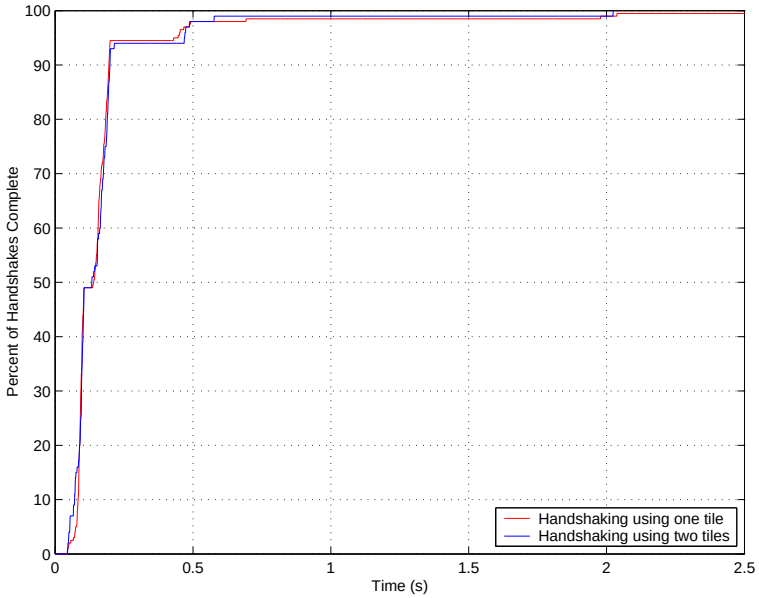

Figure 3.6: Comparison of Handshaking Timings for Objects on One and Two Tiles

shaking. Both protocols perform as expected, connecting the majority of objects in one and two cycles, respectively. Given that there seems to be no penalty in using the faster (two-state) protocol, this would appear to be the obvious choice. However, this result is only for a single object, the multiple objects case will be examined in the next section.

Finally, another important conclusion shown by the results is that the topology set out in the previous chapter works, as connections were made for all the tested random positions.

Handshaking with Multiple Objects

Once an object is connected it may be ignored by the tiles, since the pads it is using are no longer available to the handshaking process. When those pads are due to be beaconed upon, the tile simply does nothing.

However, handshaking objects can affect connected objects, either by being placed on top of the pads they are using, or during the process of connecting pads to functions. Both of these cases and their implications will be discussed below.

The former case may occur when an object is placed on connected pads belonging to another object. If this happens, then the object's use of "grounding by consensus" means that the surface data buses will be pulled towards ground by each object pad on top of it (i.e. if more object pads are in contact, then the pull to ground will be stronger). This effect

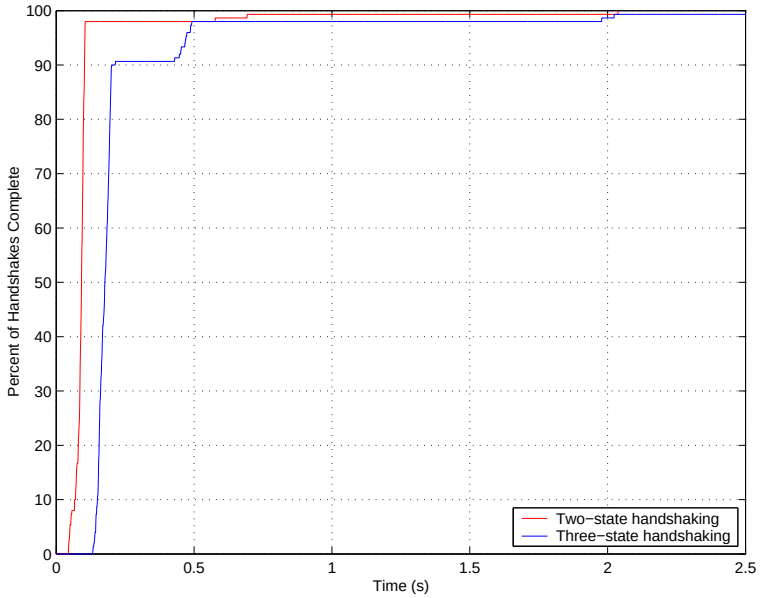

Figure 3.7: Comparison of Handshaking Timings for Two-State and Three-State Protocols

can interfere with buses, especially with buses using open-collector signalling such as I^2C, since they are not always strongly driven, and buses running at high speeds, which may be less tolerant of interference.

This multiple placement can be avoided by guaranteeing that objects have a large enough physical footprint to completely cover the pads that they would use. For the prototype Networked Surface, this "physical guarantee" would be provided by objects with large footprints such as notebook PCs, but not with objects such as PDAs. Different topologies and object sizes could be used to make this guarantee apply to more classes of objects. An alternative to providing this property using physical constraints would be to simply inform users that placing objects close together may cause them to malfunction, so that a user noticing such interference would try moving the objects.

The case of interference between connecting objects and connected objects is now discussed. This interference occurs in the case of two-stage handshaking only. In the two-stage scheme, when an object asks for a function on a pad, the surface and object both connect that pad to the required bus immediately. However, it is possible that more than one object pad is in contact with the surface pad used. Since only one of these object pads is connected to the function, the others are still in handshaking mode and will pull the surface bus to ground, which may disrupt connected objects using that bus.

This effect can be avoided by making the object controller detect cases where multiple

object pads span surface pads, and connecting all the pads through to the function instead of just one. This would also have the good property of "strengthening" the connection, both because resistance would be lowered by using parallel electrical paths, and because if the object were moved slightly so that one object pad moved into a margin, another object pad might still be connected properly. However, this functionality was not implemented in the prototype due to hardware constraints.

Three-state handshaking does not incur these problems, as when an object issues "connect" commands, it already knows about the set of pads which it requires for the connection. The object can therefore disable all pads other than the set required for connection, avoiding interference with surface buses. For these reasons, three-state handshaking is used in the prototype Networked Surface. However, a different implementation of Networked Surfaces, which keeps track of the full mapping between surface and object pads, could use a two-state protocol to achieve faster connection.

Failure Cases of the Handshaking Protocol

A few failure cases of the handshaking protocol will now be discussed. These are the case of pad allocation deadlock, and that of simultaneous messages.

The handshaking protocol in its current form does not handle the case of "pad allocation deadlock," whereby objects placed adjacent to one another are each able to reserve some tile pads, but neither is able to reserve enough pads to form a full connection. Such deadlock is unlikely to happen since the handshaking phase is brief, making it much more likely that the second object placed would find that the first object had already acquired a full connection. It could also be completely prevented by use of physical guarantees, as detailed in the previous section. Another method of breaking this deadlock would be to implement a timeout and random backoff mechanism, so that one of the two objects would back off, allowing the other to form a connection.

A related problem is that of simultaneous transmissions, which might occur when two handshaking objects are on top of the same surface pad, on which a beacon is sent. Both objects may attempt to reply to the beacon, causing a collision. Again, this problem could be solved using physical guarantees, and is rare in any case. However, a solution to this would be to implement handshaking using open-collector signalling, and having an arbitration scheme similar to that in I^2C, whereby objects monitor the bus while transmitting, and back off if the received and sent waveforms differ.

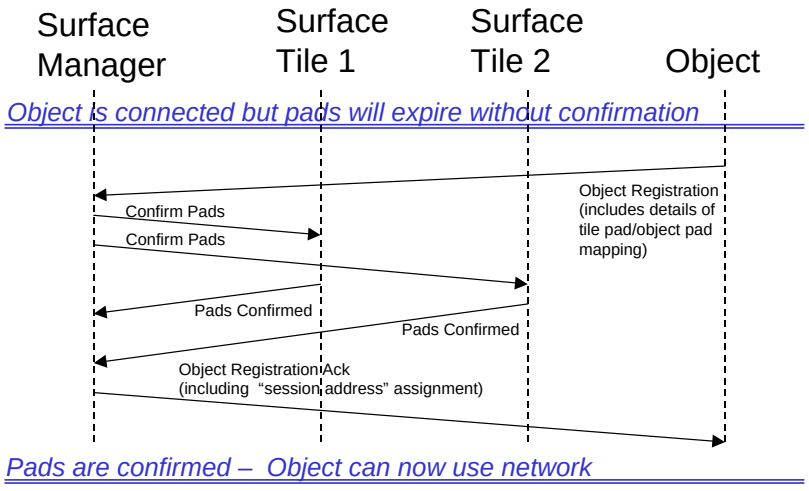

Figure 3.8: Registration Protocol

3.3 The Registration Protocol

This section presents the registration protocol, which is the protocol used between the object manager and surface manager, in order to register the object. First a discussion of solicitation of registration messages is presented, and then the registration protocol is set out in detail. The registration protocol is illustrated in Figure 3.8.

3.3.1 Solicitation of Registration

After handshaking is complete, the object must use its newly available network to register itself with the manager, otherwise the pads it has connected will time out and the object will be disconnected. However, this implies that the chosen network's link layer is using an arbitration scheme that allows an object to initiate a transmission, which may not always be the case. This problem occurs for example in token-based networks. In networks such as Ethernet, where any device may initiate its own transmission, there is no such issue.

Therefore, when an object is connecting its final function, it may use a special "connect and notify" request instead of the normal "connect" request used with the other pads. This causes the tile to send a message on the tile control bus to the manager notifying it that a new object is present. If that manager supports networks with arbitration schemes which do not allow objects to transmit independently, this notification can be used as a trigger for the manager to solicit new bus participants.

3.3.2 The Registration Process

After solicitation (if necessary), the object manager may now send its registration request to the surface manager. This registration message includes details of the tile pads (i.e. tile address, pad number, and function requested) in order for the manager to be able to confirm those functions on those pads. To facilitate the discovery of location information (discussed in Chapter 6), the object manager also sends the object pad numbers corresponding to each tile pad.

The surface manager then uses the tile control bus to communicate with all the tiles on which the object is using pads, and confirms the functions that the object connected during handshaking. It must then wait for replies from the tiles, and may retransmit the requests if no reply arrives. Finally, once all tiles have replied, the surface manager must inform the object of success. The opportunity can be taken at this point to perform any link layer or IP layer configuration required, before enabling the network for normal data transfer. One use of this facility is described in the next section.

3.3.3 Dynamic Addressing

The use of a registration protocol means that every bus participant is known to the surface manager. This property enables a number of possible features, for example, authentication and admission control can be accomplished at registration time. One interesting possibility is the use of non-global addresses at the link layer.

Global link layer addresses are normally used so that no Network Interface Card (NIC) will encounter duplicate address issues when used on the same bus with with any other NIC. However, a global address must be long, for example, Ethernet uses 48-bit global addresses.

With the registration process for Networked Surfaces, it is possible to give each object a "session address" which is only valid for the duration of that connection, on that particular Surface. Because a Surface has a limit on the number of objects that can fit on top of it, the "session address" only has to be long enough to cater for that limit. This facility is shown to be useful in Section 3.5, which discusses the LVDS bus link layer.

3.3.4 Evaluation

Whereas handshaking is inherently time-consuming due to the number of pads involved, registration is comparatively straightforward. It is therefore expected that registration will not contribute significantly to the connection time of an object.

However, evaluating the performance of this protocol is not simple, as the protocol is

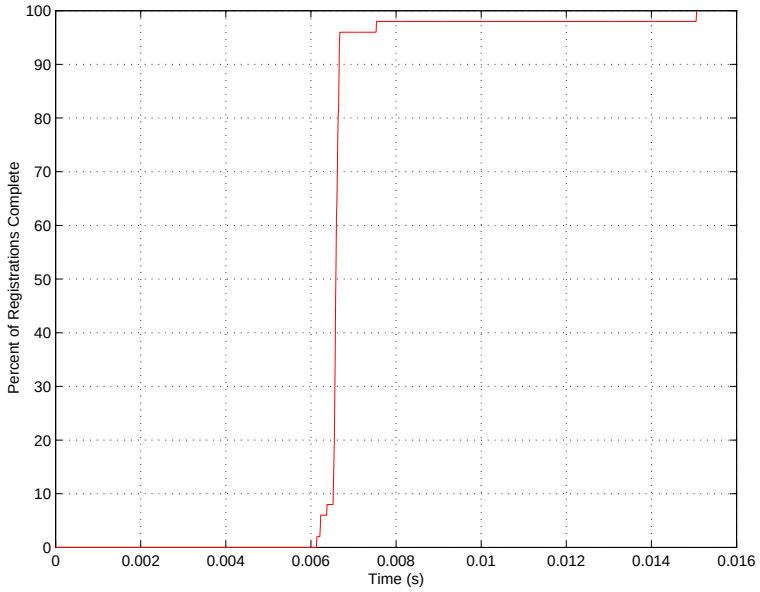

Figure 3.9: Registration Protocol Experimental Timings

meant to be run over any bus that the object establishes, the choice of which would affect the registration time. Tests were conducted using the LVDS bus and the "token star" link layer, both of which are described below. The bus speed was set to 1Mbit/s.

The experimental method consisted of an object being randomly placed on the prototype Surface 50 times, and the time from the end of handshaking to the reception of the final message in the registration protocol was recorded in each case. Results are shown in Figure 3.9. As expected, registration times are insignificant when compared to the handshaking times presented previously.

3.4 High Speed Bus Interface

This section describes the high speed bus interface used in the Networked Surface prototype. Results are presented, including analysis of framing techniques, and of the bus speed capable of being supported by the prototype.

As stated, the high speed bus on the Networked Surface uses the Bus LVDS modulation scheme. However, this does not define any other aspects of the link layer, including aspects of bit, byte and packet framing; these topics are described in this section.

Other aspects of the link layer, i.e. addressing, packet format and arbitration, are described in a later section. The reason for this split is that framing is performed in FPGA

hardware, whereas the other tasks are performed in software. This causes different factors to be important; in FPGAs, there is a relatively small amount of programmable logic, but all computation is performed in parallel. In software, there is a very large memory space available, but processing is done serially. Software is also executed in a multi-tasking environment, so there is the possibility of a delay between a stimulus (such as an interrupt) and its response.

3.4.1 Framing Methods

In order for the receiver to decode the sender's message, it must know when the sender stops transmitting one symbol and starts transmitting another. This can be done by sending an explicit clock, as in done in I^2C. However, this is inefficient on the Networked Surface as it uses more physical wires, which are an "expensive" resource. Another method would be to use an encoding which includes clock signals as well as data signals, such as Manchester encoding. The drawback of this method is that it uses half the bandwidth of the bus for clocking purposes.

A third method is to rely on clock synchronisation for a certain number of bits (such as the eight bits of a byte), and send explicit synchronisation signals at intervals such as bytes. One example of this method is the start-bit-stop-bit scheme used by RS-232.

Yet another method is "bit stuffing," whereby synchronisation is kept by inserting bits whenever the data sequence is such that none are present (i.e. in the case of many zeros, or many ones). The number of identical bits allowed before the "stuffed" bit is set so that clocks can be synchronised often enough to cope with clock drift. The bit stuffing method can be regarded as using a single wire "optimally," in that it only inserts non-data symbols when they are required in order to keep synchronisation. However, in the FPGA environment used in Networked Surfaces, this arbitrary insertion causes the resulting state machine to be large.

8-9 Coding

The chosen framing method is an 8-9 coding, which adds one bit to each byte transmitted. This bit is used to guarantee transitions, and also allows packet framing symbols to be included.

The 8-9 coder works by simply adding a ninth bit to each byte, which is the opposite of the eighth bit; this guarantees a transition at least every nine bits. Any bit sequence in which the ninth bit is equal to the eighth bit is invalid, with the exception of a few special sequences, for "start of packet" and "end of packet." The receiver must use a "bit clock" to time the arrival of bits when transitions are absent; when transitions occur (which is at least

every nine bits), the receiver is expected to correct any drift in this clock.

In summary, the 8-9 coder performs bit framing using clock synchronisation, byte framing by having a set number of bits per byte, and packet framing by including special symbols to mark the start and end of packets.

Evaluation of the 8-9 Coding

The 8-9 coding was evaluated against a bit-stuffing coding by implementing both (in Verilog) for the FPGA used in the prototype. The results are shown in Table 3.1 below, which also illustrates the bus overhead for those codings and a start-bit-stop-bit coding.[2]

Coding Scheme	Bus Overhead (control bits per data bit)	FPGA size (%)	FPGA clock cycles required per bit
Start-bit-stop-bit	0.25	not tested	not tested
Bit Stuffing	0.125	12	12
8-9 Coding	0.125	5	5

Table 3.1: Comparison of FPGA Implementations of Link Layer Coding Schemes

As the table shows, the start-bit-stop-bit coding has a higher overhead per byte, using two bits for framing for each byte transferred. This coding was therefore rejected as being wasteful of bandwidth.

Surprisingly, the bit stuffing encoding has the same overhead as the 8-9 coding. This is because the bit stuffing implementation adds two bits whenever it encounters five identical bits. Two additional bits are required so that transitions are guaranteed for long strings of zeroes as well as strings of ones, and also to provide the "start-of-packet" and "end-of-packet" special symbols. While it is possible to reduce the bus overhead by extending the length of identical bits required before stuffing bits need to be inserted, this would result in a state machine requiring more clock cycles, and taking up more space in the FPGA.

The 8-9 coding clearly wins over bit stuffing when considering its implementation in an FPGA. This is because the former encoding is much simpler, in that each bus symbol is precisely 9 bits long, allowing implementation using a single 9-bit shift register. For the bit stuffing encoding, three shift registers are required to decode the incoming data into bytes, and shifting data around those registers is responsible for the higher clock cycle requirements.

As will be seen, the FPGA clock speed is a potential bottleneck for LVDS bus speeds.

[2]Note that the repetition of the numbers "5" and "12" in the table is pure coincidence, and does not imply a direct relation.

It is therefore particularly important to choose an encoding which is fast in terms of clock cycles. The 8-9 coding is capable of running at 8Mbit/s in the prototype, which uses 40MHz oscillators. In comparison, the bit stuffing encoding would only be able to run at 3Mbit/s.

3.4.2 Software/Hardware Interface

Once data has been received by the hardware and framed into packets, it must be transferred to software, which is responsible for arbitration, addressing, and higher layer issues. This transfer also mediates between the immediate response of dedicated hardware, and the higher latency of software-based systems. The software/hardware interface is therefore structured around a data buffer.

One approach, employed by many hardware network interface cards (NICs), is to provide memory in the card for a whole packet. This is not possible using the current Networked Surface prototype as there is no dedicated memory hardware; any buffer space must be provided inside the FPGA, and is costly due to the limited number of logic blocks available. Furthermore, unlike other NICs, the Networked Surface manager hardware is designed to be capable of supporting many networks, so a single packet buffer would be insufficient.

For this reason, FIFO buffers are used, as they have the property that entire packets need not fit inside the hardware. Instead, the FIFO can be incrementally filled and drained during transmission and reception. FIFOs can therefore be smaller than an individual packet; they need only be large enough to cope with software latency in response to interrupts.

FIFO Level Interrupts and Speed Errors

In the prototype system, software/hardware data transfer can only be initiated by the software, although they often take place in response to a hardware interrupt. Interrupts occur when the level of a FIFO crosses a threshold; for example, if the outgoing FIFO is low on data, an interrupt will be generated. Interrupts also occur when a packet is fully transmitted or fully received. The latter is necessary so that the last part of a packet is transferred in timely fashion to the object. All interrupts can be masked; this is useful for example when there is no outgoing packet, as otherwise the empty FIFO would be generating interrupts constantly. Transmission using FIFOs is illustrated by Figure 3.10.

The use of FIFOs which are smaller than packets has an important implication, namely that a bounded interrupt latency is required. Otherwise, a FIFO may overflow, causing an outgoing packet to be terminated prematurely, or an incoming packet to be truncated as new symbols arrive with no free buffer space. This type of error is termed a “speed” error, and can be avoided by having the FIFO sizes large enough and interrupt trigger levels conservative

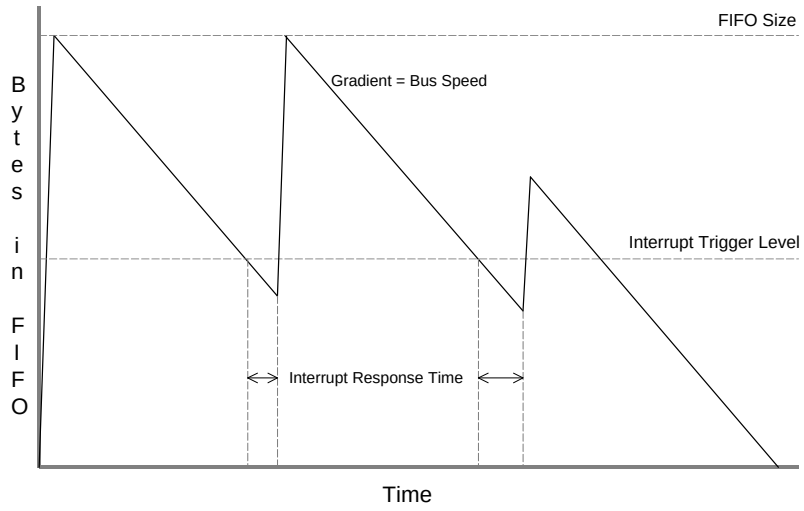

Figure 3.10: Packet Transmission using FIFO for Buffering

enough so that the CPU has adequate time to respond to interrupts. The occurrence of these errors is also affected by the LVDS bus speed, the CPU speed, and the speed of the bus(es) connecting the CPU and the manager hardware.

In the prototype, the trigger levels used are static, and are found by trial and error, i.e. if speed errors are observed during normal operation, the trigger level is made more conservative. This could also be done dynamically, by having the software change the levels in response to the receipt of an error, however, not all speed errors are the result of interrupt latency, as will be seen below.

Packet Delimitation in FIFOs

The use of FIFOs as opposed to packet buffers has a few more important implications. Firstly, since the end of a packet does not necessarily correspond to the boundary of the FIFO, there must be an explicit method of delimiting packets. One such method would be to use out-of-band control data to record the packet boundaries. However, this would require a separate memory area to be used for the delimitation data, and therefore would be costly in terms of FPGA logic.

The chosen method is to use byte stuffing, i.e. to consider a special 8-bit sequence as an "escape character," which is removed wherever it is found when the data is decoded, with the exception of when it is followed by another special 8-bit sequence, the "delimit character," in which case a delimiter is inferred. On encoding, the escape character is inserted twice

whenever it is encountered. Note that the byte stuffing is not used over the physical channel; the data is "stripped" before transmission, and "stuffed" again on reception, and channel bandwidth is not expended by use of byte stuffing internally.

The final implication of the use of FIFOs and packet delimitation is that multiple packets can be present in a single FIFO. This is an advantage over the use of packet buffers in that it is possible to transmit and receive packets back-to-back with minimum latency, and because reception of a packet is not blocked by a previous packet in the buffer.

3.4.3 Bus Speed Analysis

The prototype LVDS bus has a configurable speed, which can be set by a simple software command. Using this, various bus speeds were tested using the `ping` program, which sends ICMP "Echo" [84] messages, and is commonly used to determine network availability. One hundred pings were sent from an object to the manager, for various packet sizes and at various speeds. The results are shown in Table 3.2.

Speed	% of pings OK for packet sizes below			
(Mbit)	38 bytes	100 bytes	500 bytes	2000 bytes
1	100	100	100	100
2	100	100	100	100
3	100	100	100	100
4	100	100	100	100
5	100	100	100	100
6	100	100	99	0
7	100	100	0	0
8	100	100	0	0

Table 3.2: Results of LVDS Bus Speed Tests

The packet sizes shown are inclusive of the ICMP and IP headers, and a 2-byte link layer header. The lowest size (38) is the minimum number of bytes possible for "ping," and the maximum size (2000) is chosen to ensure IP fragmentation is tested. The Maximum Transmission Unit (MTU) of the interface is set at 1500, this is simply a software restriction (dynamically configurable in the prototype) and not intrinsic to Networked Surfaces. The number 1500 was chosen because it is a standard size used in other IP link layers such as Ethernet.

The speeds shown are "on-the-wire" speeds, but due to the 8-9 coding used, only eight ninths of this bandwidth is available for useful data. The upper speed bound of 8Mbit/s is

due to the prototype hardware; since the 8-9 coding requires 5 FPGA clock cycles per bit, and the clock used is 40MHz, 8Mbit/s is the top speed supported.

As the table shows, speeds up to 5Mbit/s are supported by the system without error. From 6Mbit/s to 8Mbit/s, there are problems with larger packet sizes. These can be explained by noting that the FIFO size in the object is 127 bytes, and that the PCMCIA bus implementation used on the object runs at about 5Mbit/s. Therefore, for small packets, the FIFOs are large enough to hold the packet, and the PCMCIA speed is irrelevant. As packets sizes increase, the FIFO requires refilling during transmission, and at higher speeds the PCMCIA implementation used is incapable of doing this quickly enough.

The shortfall of the results listed above is that the physical bus was not tested to its limit; physical issues such as bit errors were not the cause of packet losses in the above test. Unfortunately, the prototype is not capable of testing this limit whilst performing end-to-end networking, due to the 8Mbit/s limitation imposed by the rate the FPGA can be clocked, and by the maximum speed of the 8-9 coder. "Low-level" bus speed measurements carried out to find the on-the-wire bandwidth limit are discussed below.

Finally, it should be noted that these results were obtained after a number of revisions to the system. The first PCMCIA interface built was only capable of 1–2Mbit/s, and the FPGA originally used a 20MHz clock, limiting the bus speed to 4Mbit/s. Modifications were made to alleviate these bottlenecks and provide the results above. Further modifications could also be made, given time, including the use of 16-bit PCMCIA data paths, faster (but more complex) PCMCIA "modes" of operation, faster FPGA clock chips, and optimised FPGA firmware which can make use of the faster clock rate. Nevertheless, the current implementation shows that the Networked Surface is capable of networking at megabit speeds using a baseband unmodulated channel encoding.

3.4.4 Low-Level Bus Speed

Since networking in the prototype Surface is subject to bottlenecks in terms of FPGA clock speed, another method was used to determine the bandwidth available in the prototype LVDS bus. Specifically, a pseudorandom generator was implemented in the surface manager FPGA, and an external signal generator was used to clock the psuedorandom data at a finely-controlled rate. This data was sent across a prototype Surface LVDS bus, reconstructed by a connected object, and then passed back to the surface manager FPGA in a shielded cable. The outgoing and incoming signals were compared, thus allowing the bit error rate to be determined for various speeds.

The results of this test are shown in Figure 3.11. The diagram shows that the LVDS bus

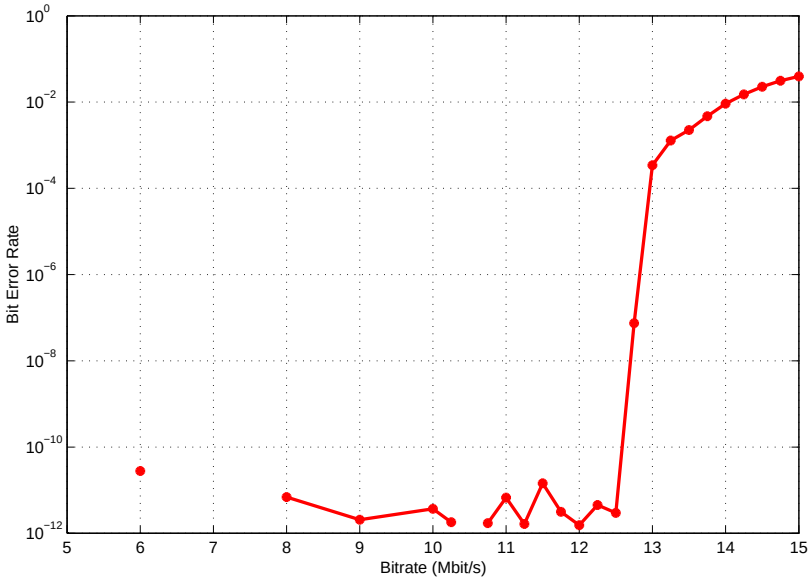

Figure 3.11: Results of Low-Level Bus Speed Experiment

is capable of speeds of up to 12.5Mbit/s with a bit error rate of under 10^{-10}. Full analysis of the causes of bit errors in the prototype's physical layer is outside the scope of this thesis; see [48] for further details.

3.5 Bus Arbitration

Arbitration refers to mechanisms and protocols used to determine which of a number of possible transmitters is allowed to transmit on a shared medium at any given time.

Various arbitration schemes exist, with characteristics suited to different types of network. This section will examine various types of arbitration scheme in the context of Networked Surfaces. A new type of arbitration scheme will be presented, which has advantages making it particularly suitable for the Networked Surface environment.

3.5.1 Existing Arbitration Schemes

The most popular arbitration system in wired local area networks today is Carrier Sense Multiple Access with Collision Detection (CSMA/CD), used for example by Ethernet. However, this requires the ability to detect collisions, which is not possible with the current Networked Surface prototype, due to constraints imposed by the LVDS modulation scheme and the prototype hardware.

Wireless LAN systems suffer from a similar inability to detect collisions, therefore, they use Collision Avoidance (CSMA/CA). This involves the use of control packets on the channel (RTS/CTS), so that nodes which could interfere with a transmission are proactively told not to send for a particular period. The disadvantages of this system include the bandwidth overhead of these messages, the additional latency they cause, and the fact that collisions are not eliminated since the control messages themselves can still collide.

3.5.2 Token Star

The novel properties of the Networked Surface suggest an alternative arbitration scheme. On all Surface buses, a surface manager is always present and acts as the gateway between that bus and other networks. Furthermore, that manager is always informed of new bus participants (via the registration protocol). Therefore, the surface manager is the ideal entity to be in charge of allocating bandwidth on Networked Surface buses.

Basic Design

A scheme is therefore proposed whereby the surface manager sends "grant" messages to each object in turn, to which each object may reply with either one packet of data or a "grant reply" message if no outgoing data is waiting. The manager would then take its turn to transmit one packet to one object, and then continue by sending a grant to the next object, and so on in round robin fashion.

This scheme is called "token star" since it is a token-based arbitration scheme, but with the token being returned to the manager each time, as opposed to being passed around a token ring[3]. The design outlined above is illustrated in Figure 3.12.

One issue important to this arbitration scheme is the size of the grant and grant reply messages. If they are large, the overhead for passing the token around will be high, thereby increasing the amount of time between each object being polled. Using the dynamic link layer addressing scheme mentioned previously, the header length required can be reduced dramatically. In the prototype, a two byte header is used, comprising a single byte address, and a byte representing the packet type (e.g. grant, grant reply, data).

Properties of Token Star

The simple token star scheme described above has some useful properties. Firstly, collisions should not occur, since only one device "has the token" at any one time. This means

[3]Token ring is discussed further in the following chapter.

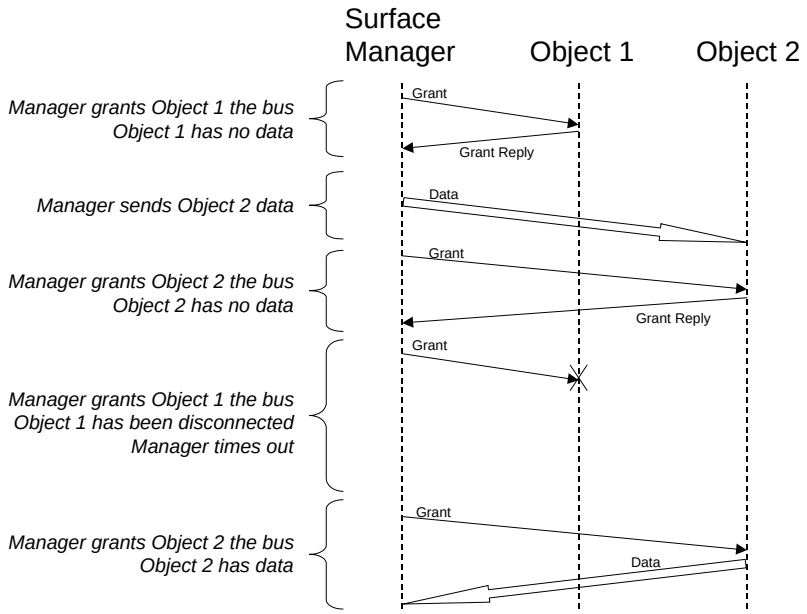

Figure 3.12: Token Star Arbitration

that, at the link layer, it is unnecessary to implement Automatic Repeat reQuest (ARQ) retransmission, with the resulting savings in buffer space and computation time.

Secondly, in the simple scheme outlined above, the bandwidth is automatically split with half going to the surface manager and half going to the objects on the surface. This is appropriate since the surface manager is the gateway between the surface bus and other networks, and therefore has to transmit all the "downstream" packets, which for a symmetric traffic model would be half the traffic. Note that this split is only enforced when the bus is loaded fully; at other times, bandwidth is allocated to any party with data to send.

Thirdly, disconnection detection is performed automatically. This feature is described in more detail later in this section.

However, the scheme inevitably has disadvantages. Chief amongst these is the latency imposed; whenever a packet is ready for transmission at an object, it must wait until it gets the token before it can send the packet. This is true even if the bus would otherwise be free (which ideally would give zero latency before sending starts). This latency is made worse by the use of a software implementation for token star, which means that an interrupt response delay is incurred after every transmission, since the recipient must rely on software to process the message and generate its reply.

Another disadvantage is that the object's processor is loaded with interrupts, even when

no data is being transmitted on the bus. This can be avoided by firstly using a hardware filter to remove packets addressed to a different object, and secondly by moving the token star protocol implementation into the object hardware, so that the host processor would only be interrupted when data packets are being transmitted or received. With the latter optimisation, latency should also be improved, since the operations are being accomplished in hardware, and without incurring the delay inherent in software interrupt handlers.

Design Enhancements for Token Star

In addition to the properties described above, the simple token star protocol could be enhanced to provide other features.

Firstly, the use of a packet limit is not necessarily ideal, as it favours objects generating large packets over those generating small ones. This could therefore be changed to a byte limit, so that an object could send either one MTU-sized packet, or many smaller ones. This would require the token to be explicitly passed back from the object to the manager even in the case of data transmission.

Secondly, the fairness guarantees described previously might be abandoned in favour of prioritising certain objects over others. This can also be applied to "downstream" data, whereby packets for some objects could be sent preferentially over others. The net effect would be to support bandwidth allocation on a per-object basis, which would allow the Networked Surface to support QoS guarantees, such as reservations, at the link layer.

To implement different provisioning for different objects, many methods are possible. The manager could use multiple queues of objects, with some queues being processed more often than others. Alternatively, the manager could provide dynamic byte limits, so that higher priority objects are allowed to send more data at each turn. The latter method has the advantage of reducing the overhead on the bus.

Finally, it is worth noting that the Surface has a unique method of policing objects to ensure the chosen allocations of bandwidth are adhered to. If an object is "behaving badly" by transmitting more than its allotted byte limit, or transmitting out of turn, the surface manager can disconnect that object. Unlike traditional policing using packet dropping, no bandwidth is wasted since no data is discarded. Also, the misbehaving object is prevented from further interference with valid bus participants.

3.5.3 Networking Characteristics of Token Star

To evaluate the networking characteristics of the token star protocol, two tests were carried out comparing the token star protocol to a simple CSMA protocol. For the latter protocol,

packets ready for transmission are immediately sent to the hardware, but the hardware does not start a transmission while a packet is being received. No collision detection or avoidance is used.

Bandwidth

In order to test the usable bandwidth under the token star protocol, the two LVDS objects and the manager were used as three transmitters on the LVDS bus, with the objects sending data to the manager, and the manager sending data to one of the objects. The bus speed chosen for these tests was 1Mbit/s, which was chosen so that the PCMCIA and FPGA clock speed bottlenecks discussed previously could be avoided. The size of packet used was 1000 bytes, which is chosen to be lower than the MTU of the interface, so that IP does not fragment the packets causing the hosts to send more packets than expected.

Each of the three transmitters ran a "slave" test program capable of producing UDP traffic according to a uniform random distribution. This program was used to generate traffic at rates between 100kbit/s and 900kbit/s, in increments of 100kbit/s.[4] A "master" test program was run on a fourth machine, and used a wired Ethernet to communicate with the three slave transmitters. The master directed the sending of data at various bitrates from each transmitter, recording the number of sent and received packets over fifty seconds. Each of the possible combinations of bitrates for which the total bitrate did not exceed 1Mbit/s was tested.

These tests were conducted for both the token star protocol and the CSMA protocol for comparison. The results are shown in Figures 3.13 and 3.14. As the figures show, the token star protocol allows close to 100% utilisation during these tests. This is because the token packets are very small (2 bytes), and consume only a small proportion of the bandwidth. With the CSMA protocol, there is no overhead required for grant packets, and the bandwidth achieved is fully 100%. Scalability of the token star protocol with many objects is discussed later.

The token star protocol experiences very few packet losses, whereas CSMA experiences packet losses at utilisation greater than 40%. The token star packet losses, although few in number, are not due to valid operation of the protocol. It is conjectured (though not proven) that they are caused by high load on the surface manager PC (which is "Transmitter 1" in these tests), causing packets to be dropped internally. Hardware-based implementations of the token star protocol would alleviate this issue, as would the use of larger network buffers. Note that the surface manager performed many roles during this test, firstly as the bus

[4]These transmission rates include the overhead of the 8-9 coding scheme used.

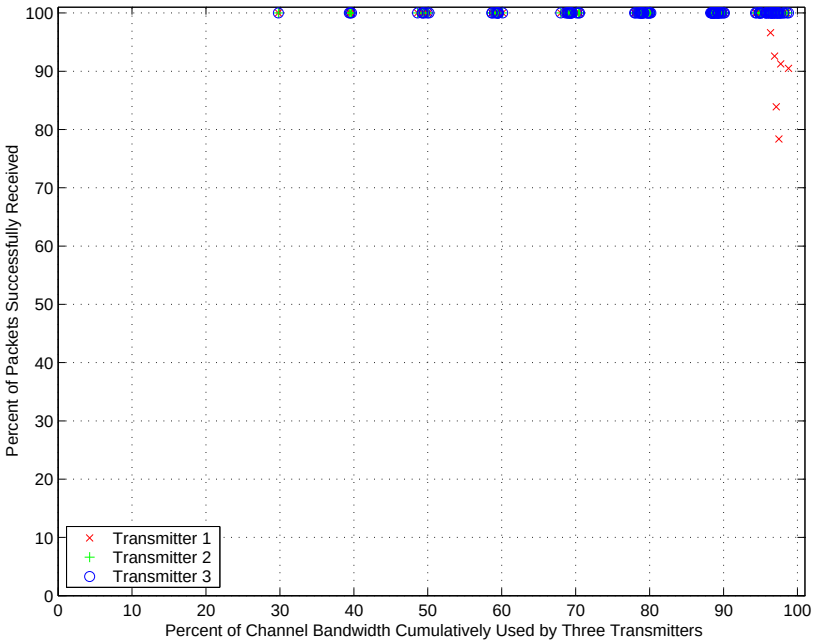

Figure 3.13: Token Star Arbitration Bandwidth Test Results

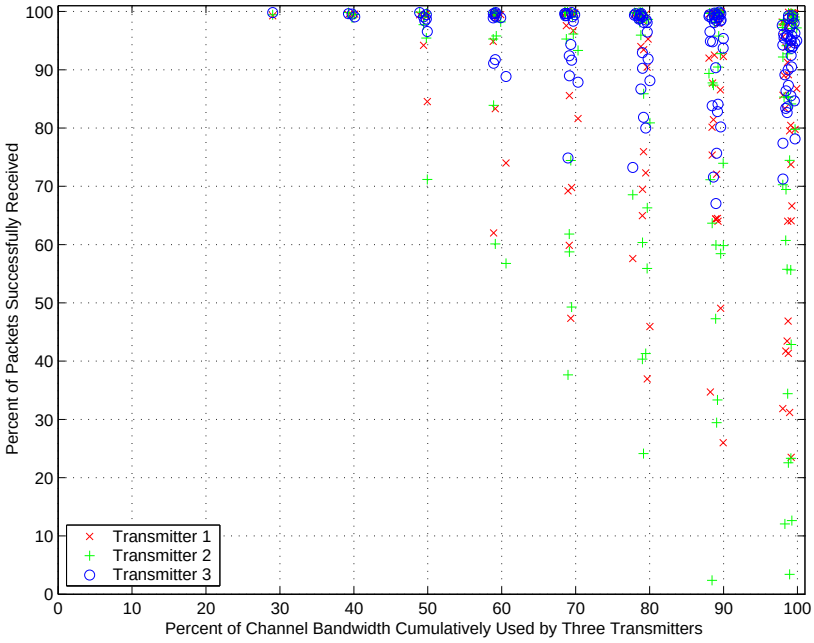

Figure 3.14: CSMA Arbitration Bandwidth Test Results

master, secondly as the generator of traffic, thirdly the execution of the surface daemon for tile control and object registration, and fourthly as recipient for traffic from both of the other senders. In real-life use, the surface manager would not be placed in this situation.

It is also worth noting that disconnection detection had to be disabled while testing the CSMA protocol, as at high bus loads disconnection was erroneously inferred. This is due to the fact that, in this protocol, the disconnection detection messages are simply data packets sent at the application layer, and they receive no special treatment at the link layer. This highlights another advantage of the token star protocol, which performed the tests with disconnection detection enabled (but no disconnections occurring).

In summary, the token star protocol provides a much higher usable bandwidth than CSMA. Even if ARQ was used with CSMA to make the bus reliable, this would not stop packet collisions from occurring and the consequent waste of bandwidth.

Latency

One drawback of the token star protocol is the latency imposed by the need to wait for a grant before sending. This was tested by using the `ping` program to send ICMP "Echo" messages between an object and the manager. The parameters of these tests are detailed below, and the results are shown in Figure 3.15.

- Tests were conducted for both the CSMA and token star arbitration schemes.
- For both arbitration schemes, trials were conducted for the case of one connected object, and for the case of two connected objects (the second object being "idle").
- The minimum length ping message was used. This comprises a 36 byte IP packet, which is transmitted on the bus using 38 bytes, including a two-byte link layer header.
- The LVDS bus was used, with the speed set to 1Mbit/s.
- One thousand pings were sent in each trial, and the mean and standard deviation of the round trip times were recorded.

As the diagram shows, the token star protocol imposes a higher latency on packets than basic CSMA arbitration. This is because data cannot be immediately sent by either the manager or the object, it must be queued until an appropriate time for sending. For the manager, sending can take place after it has received the token back from any object, while an object must wait for a token to be sent to it before it may send.

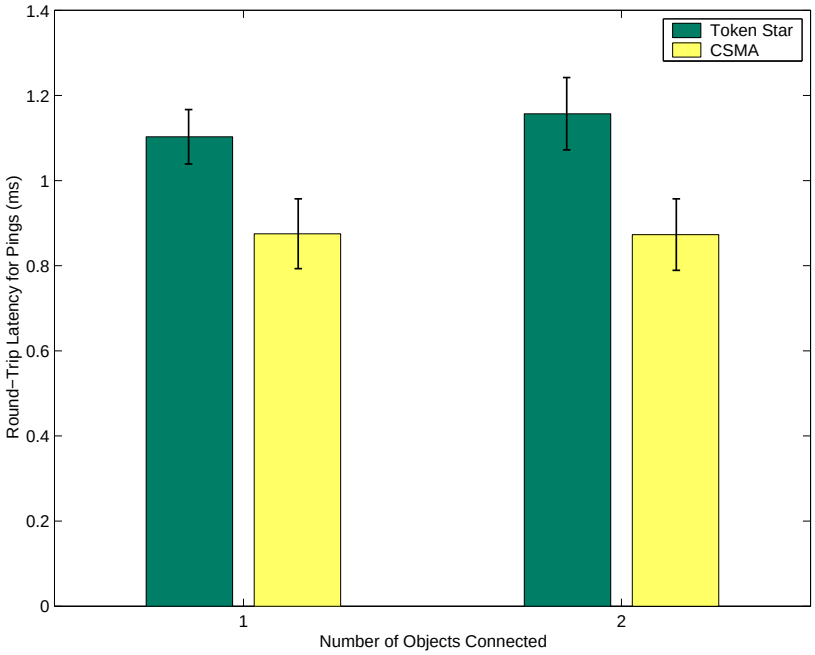

Figure 3.15: Token Star vs. CSMA Latency Test Results

The difference between the two protocols is 0.23ms in the "one object" case, and 0.28ms in the "two object" case. This means that one-way latency is approximately 0.1ms higher in the token star case than the CSMA case. When two objects are present, the latency rises by 0.03ms, and this can be expected to continue as more and more objects are placed on the bus; the issue of scalability is discussed further below.

To put these figures into perspective, the same tests were run on both a fixed 100Mbit/s Ethernet, and a 11Mbit/s WaveLAN. The fixed Ethernet experienced an average latency of 0.22ms, while the WaveLAN exhibited a 2.7ms average latency. The Networked Surface prototype bus latency is therefore comparable to commonly-used networks.

Scalability of Token Star

The issue of the scalability of the token star protocol is now addressed, using a worst-case example of 100 objects present on a single LVDS bus. With 99 "idle" objects and a single "sending" object, the projected latency would rise to 3ms above the "base" latency of 1ms shown by the CSMA protocol. Also, 400 bytes of grants and grant replies would be sent on the bus for every data packet that is sent; this does not take into account the turnaround time of the bus, and assumes each object always replies to every grant (i.e. timeouts do

not occur). Even with these assumptions, if the actual data packet is small, e.g. 40 bytes including headers, then the usable bandwidth would drop from nearly 100% to under 10%, which is unacceptable.

This problem can be remedied in a number of ways. Firstly, the example of 100 objects may be regarded as outside realistic possibility, as each object would receive an average bandwidth of 50kbit/s (assuming a 5Mbit/s total bandwidth); in such a scenario, more than one network bus may be required. The ability of Networked Surfaces to support multiple buses is therefore useful in this case. Furthermore, quantitative limits could be imposed on the number of devices on a single Surface bus at a time, possibly by simply refusing to allow objects to obtain access when the buses are "full."

Another method of providing these limits would be to use physical guarantees of bus availability, by making sure that there are enough buses in each particular area of the Surface so that all devices placed in that area could acquire connectivity, without exceeding the administrative limit. This is helped by the possibility of geographically constraining the span of Surface buses, so that a given bus might only span a small area, making it unlikely (if not impossible) that the device limit would be exceeded.

A final possibility is to employ different polling rates for different devices with token star, this has already been mentioned as being useful for providing QoS guarantees. Idle devices could be placed on a separate polling queue, which is cycled through less often, e.g. once for every ten cycles of the "active" queue. Devices would move between queues on the basis of the quantity of incoming and outgoing traffic related to them. This allows a single token star bus to support more devices without compromising on the proportion of usable bandwidth, and on the latency incurred by active devices. A similar scheme is used in Bluetooth, which is discussed in the following chapter.

Using the methods described above, the bandwidth and latency overhead of the token star protocol can be kept at a low level, regardless of the number of objects placed on a Surface.

3.5.4 Disconnection and Reconnection using Token Star

As mentioned above, one of the advantages of the token star protocol is that it can detect disconnection in the link layer. This allows the surface manager much finer control over when objects are disconnected as compared to an application-level solution; the reasons for this are discussed below.

The speed of disconnection detection also affects the time for *reconnection* to take place. Reconnection time is defined in this context as the time in which an object is unable to

send or receive data, i.e. the time between the physical disconnection of an object and the subsequent reconnection to a network. This is also presented below.

Link Layer Disconnection Detection

On the surface manager, disconnection detection is accomplished by keeping track of how many consecutive grants an object has missed. When this number reaches some limit (which could be just "one"), the object can be considered disconnected, and the surface manager must then instruct the tile(s) used by the object to place the pads concerned back into "handshaking mode."

On the objects, disconnection is detected by maintaining a timer which is reset whenever the object receives a grant or a data packet. The object disconnects if the timer ever reaches a certain pre-defined limit. Since token star generates activity continually on the bus, this limit can be low, allowing disconnection to be detected very quickly. The prototype Surface was configured as follows:

- Objects were allowed 20ms to respond to a grant message from the surface manager, before a timeout occurred.
- Missing three grant messages in a row caused the surface manager to disconnect an object.
- Objects disconnected themselves if they observed 100ms of silence on the token star bus.

This configuration results in very fast disconnection detection. The lowest disconnection time can be found by looking at the case of a single object being connected, and with no data traffic present. In this case, the Surface takes 60ms to disconnect the object, with the object disconnecting itself 40ms later.

When multiple objects and/or data traffic is present, disconnection times are slower, as each object is granted the bus less frequently. A "worst case" estimate might be found by looking at 10 objects using a 5Mbit/s LVDS bus, with each object having data to send and to receive. Each object would take 3ms to send 1500 bytes of data, and a similar time for reception, resulting in 50ms elapsing between grants to each object, so 150ms would elapse before disconnection was detected by the manager. The object would still detect disconnection in 100ms; it is not sensitive to the presence of data traffic.

Comparison with Polled Disconnection Detection

The alternative to link layer disconnection detection is a "polled disconnection" protocol; this is employed when the Surface is configured to use the CSMA link layer instead of token star. The polled disconnection protocol causes the surface manager to send "ping" messages[5] to each object, using the object registration protocol, at intervals of 100ms. If three such pings are sent without reply, then the manager concludes that the object is disconnected. The object disconnects itself if it receives no ping message over a period of 300ms.

Polled disconnection is inferior to token star disconnection in a number of ways. Firstly, polled disconnection messages cause more overhead on the bus, since they must be explicitly sent, whereas token star disconnection detection makes use of messages already being sent in the arbitration protocol. Secondly, the polled disconnection protocol does not take into account the bus load, unlike the token star protocol, in which the bus load influences the time between successive "grant" messages to an object. Thirdly, and most importantly, polled disconnection does not function at high bus loads, as shown by in the previous bandwidth tests, in which the disconnection mechanism had to be disabled. This is a consequence of the CSMA protocol providing no guarantee of packet delivery, for the "ping" packets or otherwise.

The extremely fast disconnection made possible by token star actually proved to be a problem in the prototype, as unnecessary disconnections were occasionally experienced. This was found to be due to the response of the processors in the notebook PCs used as objects. When busy with another task, the objects sometimes failed to process the received grants in timely fashion. One cause of spurious disconnection was recovery from energy-saving modes where the screen and hard disk of an object were in low-power modes.

To combat this effect, a minimum disconnection time was introduced; objects were allowed to miss any number of grant messages so long as they replied to at least one before this time elapsed. This was set to 220ms in the prototype, which still provides very fast disconnection when considered in human terms. Note that even in the "worst-case" situation described previously, disconnection detection would not take longer than this limit. The value of 220ms was chosen by trial and error to reduce the occurrence of spurious disconnections on the prototype to a negligible amount.

It should be noted that this solution is only necessary when the token star protocol is performed in software on the objects, and is therefore prone to latency. If the protocol were instead implemented in hardware, for example in the object FPGA, then there would be a

[5]Not to be confused with the ICMP message "Echo," which is sometimes called "ping."

low and bounded response latency, and the 220ms grace period would not be required.

Reconnection

Reconnection time is defined as the time taken between disconnection and the forming of a new connection. It is a useful metric, as reconnections happen often on a Networked Surface, due to objects being moved.

The case where an object is moved only slightly is of particular interest. This is interesting since the user may not have intended disconnection to occur; the user may have caused the movement due to use of the object (e.g. while typing). In fact, the user may be engaged in network use (e.g. a remote login), so the loss of connectivity may directly inconvenience them.

Therefore, the minimum time for reconnection is of importance when considering the practical use of Surfaces; a user would not wish to wait for a significant period every time they accidentally nudged their notebook PC. Reconnection is complicated in this case by the fact that the object will likely need to use the same pads for the new connection as it used for the old one. Thus, disconnection must take place on both the Surface and object sides, before another connection can be formed. The disconnection times discussed above are therefore a main component of reconnection times.

To test reconnection times, the object manager software was augmented with a "testing mode," which periodically forced a disconnection in the object controller[6]. A duration was recorded between this disconnection and the next object registration "confirmation" message; this represents the "downtime" of the network as experienced by applications and TCP/IP. Fifty of these reconnection times were recorded, at each of ten randomly chosen locations on the prototype Surface. The results are shown in Figure 3.16.

As the figure shows, reconnections occur over a much larger range of times than the handshaking or registration protocol timings shown previously. This is because reconnection is a multi-stage process, involving disconnection detection, tile control (to disconnect pads), handshaking, and registration. Also, if more than one tile is being used by an object, then multiple tiles must be contacted during disconnection and again during the subsequent reconnection. The minimum reconnection time is 0.36s, which is similar to the sum of the minimum disconnection (0.22s) and the minimum handshaking time (0.15s), as expected.

There is a distinct "elbow" in the results above which 10% of the results lie. This portion of the results was found to be due to protocol errors, caused by malfunctions in the I^2C

[6]Forcing disconnection at the object side correctly emulates the disconnection timing; as discussed earlier, the manager takes longer to realise an object has disconnected than the object does to disconnect itself.

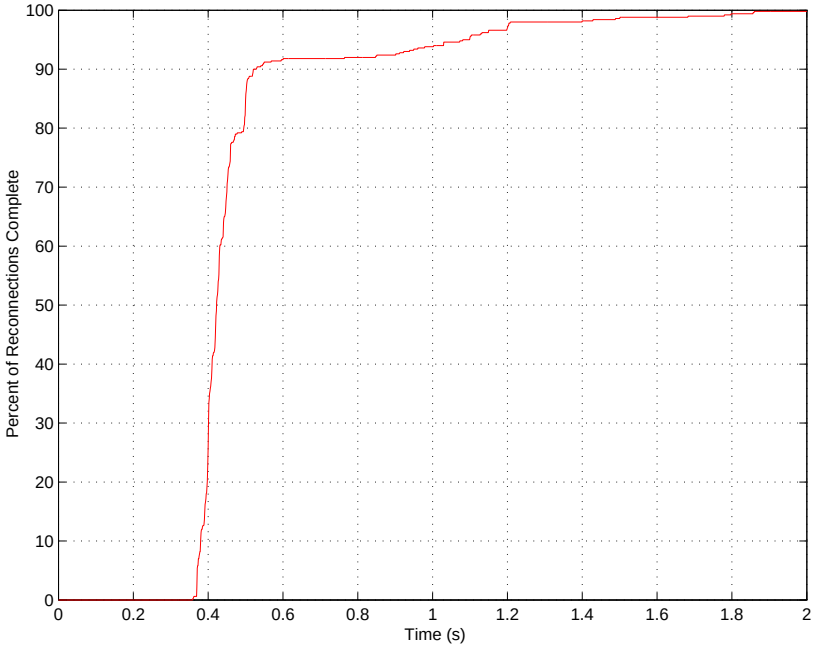

Figure 3.16: Results of Reconnection Timing Tests

functionality offered by the PIC microprocesssors. Although these errors are recovered from, the error-recovery process incurs a time penalty.

The bulk of the reconnections (90%) did not experience such errors, and took under half a second, which in human terms might be regarded as an acceptable reconnection delay. This could be improved by removing the minimum 0.22s disconnection delay, using a hardware implementation of token star as described earlier.

3.6 Summary

This chapter has examined aspects of Networked Surfaces related to providing network connectivity for objects. The Networked Surface prototype has been shown to support connection in under half a second, and bus speeds of up to 5Mbit/s. These facts illustrate that Surfaces can support fast and transparent network setup, and that the prototype achieves network bandwidths comparable to early versions of other LANs such as Ethernet and WaveLAN.

The novel "token star" bus arbitration scheme was also described. The advantages of this scheme are its support for high utilisations of the bus with very few errors, its ability to detect disconnection quickly (around a quarter of a second), and its support for QoS-based

bandwidth allocation.

Chapter 4

Networked Surfaces and Other Networks

4.1 Introduction

This chapter will discuss issues arising when Networked Surfaces are used in conjunction with other networks. The chapter will start with a comparison of Networked Surfaces and other network types. Next, the OSI network layer issues of routing and addressing are discussed in the context of Networked Surface objects. Finally, the more complicated scenario of objects with multiple network interfaces will be examined.

This chapter presents comparatively little in the way of experimental results as compared to surrounding chapters; many of the issues discussed have not yet been implemented fully, or otherwise explored. However, this overview serves to highlight the position of Networked Surfaces in this research space, and to indicate possible future work.

The work presented in this chapter was not undertaken collaboratively.

4.2 Comparison of Networked Surfaces and Other Networks

This section will compare the Networked Surface with other types of Local Area Network (LAN). Comparisons can be made in terms of bandwidth, mobility, error characteristics, arbitration scheme used, and support for Quality of Service (QoS) guarantees.

4.2.1 Wired Networks

Over the history of the networking field, many network types have been presented. It is not within the scope of this thesis to detail them all; the reader is referred to the survey found

in [1]. However, two of the prominent "traditional" wired networking methods are discussed below.

Ethernet

The access network most commonly used today is Ethernet, which is represented by the IEEE 802.3 standard. Ethernet originated in the 1970s, operating at 10Mbit/s, and now operates up to 1Gbit/s, with 10Gbit/s under development. Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is used for arbitration, which offers no guarantees for access latency, and performs badly when the bus is highly loaded.

Although originally designed to be used over a shared medium, modern Ethernets are deployed in "star" format, where each host is connected directly to a hub. "Smart" hubs perform switching, so that each device receives traffic directed to it and it alone, and allowing full-duplex communication with each host. As a wired network, mobility is intrinsically limited.

Token Ring

Token Ring, or IEEE 802.5, is another wired access technology, supporting speeds of 4Mbit/s, 16Mbit/s or 100Mbit/s. The hosts are arranged in a "ring" network (although the physical cabling may be hubbed so that powered-down hosts can be excluded from the ring).

The arbitration used is based on a token passing scheme, whereby a host receiving the token may send a single packet before passing the token on again. In this way, collisions are avoided, even at high bus loads. Support is also provided for some QoS guarantees to be made, with certain hosts designated as higher priority. This system has much in common with the "token star" arbitration scheme discussed for Networked Surfaces, which can also provide guarantees of no collisions and QoS provisions.

4.2.2 Wireless Networks

Wireless LANs are a relatively new addition to the networking world. They intrinsically support mobile operation, providing an advantage that wired networks cannot. However, they suffer from bandwidth constraints, due to the shared medium they use. This shared medium means that, as the number of hosts increases in a particular region, the amount of bandwidth remains fixed, unlike in wired networks where switching may be used to increase the aggregate bandwidth. This effect is tempered by spatial re-use of bandwidth, and by the use of multiple "channels," but as the number of networking-enabled devices grows, wireless bandwidth may become scarce.

In addition, wireless LANs experience greater levels of errors than wired LANs. Bit and burst errors can be caused by interference or channel "fading," which in turn causes packet losses to occur. Two prominent wireless LANs are presented here, which cope with error characteristics in different ways. For a full survey of wireless LAN types, the reader is referred to [34].

WaveLAN

WaveLAN is a wireless local area network, with similarities to Ethernet in the wired world. It can support bitrates between 1Mbit/s and 11Mbit/s. It follows the IEEE 802.11 [51] standard, using Direct Sequence Spread Spectrum (DSSS) for access to the wireless channel. Adaptive bitrates are used to keep bit error rates from being too high; i.e. if the channel is of low quality, then a lower bit rate will be used.

Since collision detection is unreliable in the wireless medium due to the "hidden terminal" problem, CSMA with Collision Avoidance (CSMA/CA) is used. This involves the sending of short messages (RTS/CTS) prior to actual data transmission; any terminal receiving these messages must not send on the channel for a period thereafter, so that the transmission is not subject to collision.

Bluetooth

Bluetooth [19] is another wireless access protocol, using Frequency Hopping Spread Spectrum (FHSS) to provide multiple channels on the wireless medium. Each channel is capable of 1Mbit/s speeds, with up to ten channels. The arbitration used is based on a slotted scheme with one device acting as master in a "piconet" of up to eight devices. QoS support is provided by using slot reservation, e.g. for synchronous data traffic. The bit error rate is non-negligible, but is alleviated using Forward Error Correction (FEC).

4.2.3 Comparison with Networked Surfaces

Table 4.1 shows a comparison of the networking technologies discussed above and Networked Surfaces.

As the table shows, Networked Surfaces provide a middle ground between wired and wireless network types. For bandwidth, it is observed that wired networks perform better by a factor of 10 or 100 than wireless networks. Although Networked Surfaces currently offer only 5Mbit/s, this is true for the first prototype only; the prototype's physical channel has been shown to support bandwidths of up to 12.5Mbit/s, and the use of different switching circuitry or a different modulation scheme may lead to higher bandwidths.

Network Type	Bandwidth (Mbit/s)	Medium Used	Channel Errors	Mobility Support	Arbitration Scheme	QoS
Ethernet	10–1000	Wired (Star)	Low	No	CSMA/CD	No
Token Ring	4–100	Wired (Ring)	Low	No	Token Ring	Yes
WaveLAN	1–11	Wireless (DSSS)	High	Yes	CSMA/CA	No
Bluetooth	1	Wireless (FHSS)	High	Yes	Slotted	Yes
Networked Surface	5	Wired (Bus)	Low	Limited	Token Star	Yes

Table 4.1: Comparison of Local Area Networks

Another important bandwidth characteristic which wired networks offer is the ability to provide switched, dedicated bandwidth to devices, unlike in the wireless medium where bandwidth is shared. Networked Surfaces fall into the former category, since an unlimited number of buses may be used on a Surface. Thirdly, the error characteristics of Surfaces are more like wired LANs than wireless LANs, incurring low bit error rates, as shown in Section 3.4.4. The implications of this are described in the following chapter, which discusses transport layer performance issues.

In terms of mobility support, the Networked Surface is closer to the wireless domain than wired. While Surfaces do not provide the ability to use devices while mobile, they do have significant advantages over wired networks in the areas of convenience and transparency for the user. In a wired network, users must know what type of networking interface is compatible with their device, carry appropriate cabling, and locate an access point to which they can attach their device, in order to achieve connectivity. In wireless networks, users do not need to know what type of network they are using, and the network can be automatically set up whenever the user is "in range." Similarly, the Networked Surface can hide all networking details from the user, and automatically provide connectivity whenever the device is placed on the Surface.

Finally, the arbitration scheme used for Surfaces is capable of providing QoS support, as described in the previous chapter. This property is shared by slotted and token-based arbitration schemes, but not by the CSMA-based schemes used in Ethernet and WaveLAN.

4.3 Addressing and Routing with Networked Surfaces

This section will examine the OSI network layer issues of addressing and routing in the context of Networked Surfaces. This discussion focuses on the IP protocol, being the de-

facto global network-layer protocol implementation, including both the currently widespread IPv4 [85] and its potential successor IPv6 [29].

This topic is presented by firstly outlining issues affecting addressing on Networked Surfaces, and then by examining addressing schemes which have practical use on the Surface, and the contexts they are useful in. Since IP uses addresses in order to perform routing, addressing schemes also determine the routing methods required.

4.3.1 Issues Affecting Addressing on Networked Surfaces

Addressing is at first glance an easy task, with the simple aim of differentiating one device from another. However, this is deceptive, as many issues arise which make addressing complicated. Firstly, addresses have a "scope," which may be global, local, or somewhere in between. Secondly, addresses may be used to aid routing; this is achieved by constraining what addresses may be used, depending on the point of attachment to the network. Thirdly, addresses may be static or dynamic, i.e. they may persistently belong to a single object, or be allocated on a changing basis to many objects over time.[1]

These issues, and their relevance to Networked Surfaces at both the link layer and network layer, are discussed below.

Scope of Addresses

To fulfill its task of identifying devices, an address must be unique within the scope that it is used. For NICs, that scope extends only to the local network, thus allowing addresses to be reused across different LANs. However, since it is not possible to predict which devices may be attached to any given LAN, one simple solution to ensure uniqueness is to provide globally unique addresses, known as "hardware addresses" or "MAC[2] addresses" since they are individual to each item of NIC hardware. While enjoying simplicity due to low configuration requirements, this scheme has the drawback that every frame sent on the network must include a global (and hence long) address. The use of locally-scoped addresses on the Networked Surface link layer, and the bandwidth savings this provides, is described in the previous chapter.

[1] There are still more factors affecting addressing, which are not pertinent to this discussion, including issues of hierarchy in addressing space, the possibility of broadcast, multicast and anycast addresses as well as simple unicast addresses, the use of address translation to allow many objects to share one address, and the use of many addresses by a single object. Some of these issues are discussed in the next section, which deals with objects with multiple network interfaces.

[2] Media Access Control.

At the network layer, global addresses are necessary to facilitate wide area communications. In cases where only local communication is required, schemes such as private subnet ranges (e.g. the IP addresses `192.168.x.x`) [89] are possible, and locally-directed traffic is also supported in other ways in IPv6 [46]. However, these schemes are only useful in special cases; for global networking, globally-scoped addresses are still required.

Routing with Addresses

Addresses can be used in order to facilitate routing. In the IP protocol, an address is split into network and host portions, the former being used to route packets to particular network administration domains, the latter to route packets within a domain to the appropriate device. This technique is only possible when global addresses are used.

For Networked Surfaces, the surface manager must perform routing for data packets between Surface networks and fixed networks connecting the manager to the Internet. For IP, this generates the requirement that devices on a Surface use an address routed via that surface manager. This may be achieved by assigning the surface manager a block of addresses (a "subnet"), and also statically or dynamically allocating each device using that Surface one of these addresses. However, it can also be performed by using Network Address Translation (NAT) at the surface manager. Possible addressing schemes are discussed further below.

Static and Dynamic Addressing

Addresses can be allocated statically to a device using manual configuration, or dynamically using an appropriate protocol. The main advantage of dynamic addressing is that devices may move between networks without breaking the restrictions imposed by the use of addresses for routing, thus providing mobility. Another advantage is the lack of manual configuration. However, dynamic addressing means that devices known by an external correspondent may not be contactable after a change of address. Also, communications occurring as an address change happens may be prematurely terminated.

Schemes such as Mobile IP [82] try to achieve the benefits of both worlds. In Mobile IP, each device has a static ("home") address, as well as a dynamic ("foreign") address when mobile. The static address is used for addressing packets, but routing is performed using the dynamic address. This is achieved through the use of "tunnelling" in Mobile IP for IPv4, and with "binding updates" in IPv6. Dynamic addresses can be assigned using DHCP [32] for IPv4, or using the simpler "router advertisements" [76] offered by IPv6.

For Networked Surfaces, dynamic addressing has already been proposed at the link layer to provide locally-scoped addresses. Depending on the functionality required, use can be

made of either static, dynamic, or "mobile" addressing at the network layer. The possibilities in this area are discussed below.

4.3.2 Addressing and Routing Schemes for Networked Surfaces

Depending on the circumstances in which a Networked Surface is deployed, many addressing schemes are possible. Listed below are four candidate schemes for IPv4 and IPv6 addressing, and the situations in which they may be appropriate.

Pre-Allocated Static Addressing

Pre-allocated addressing is commonly employed when the devices using a particular network are relatively constant, i.e. devices do not frequently migrate between networks. This is appropriate for many wired and wireless networks which are used to network devices normally confined to a particular site. Networked Surfaces are no exception. The strength of this system is that IP addresses are retained for long periods, and so can be used to access services, for example web servers, file servers, mail servers, and so on. The drawbacks are that manual configuration is required, and that enough addresses must be available for all devices.

For IPv6, this mode of addressing is achieved slightly differently; devices construct their own IPv6 address from a host ID and network ID, but DNS must still be configured to associate names (e.g. "www-lce.eng.cam.ac.uk") with the appropriate computers. Normally, the host ID is derived from a globally unique hardware address; while the current Networked Surface prototype requires no such global hardware address, one can of course be provided. The network ID is discovered using "router advertisement" messages, which for Networked Surfaces would be provided by the surface manager.

Dynamic Addressing

Dynamic addressing is achieved in IPv4 through the use of protocols such as DHCP [32], which allow a network gateway controlling a block of addresses to assign them dynamically to devices which become attached to that network. With a Networked Surface, the surface manager could act as DHCP server, and provide IP addresses. This would allow any device compatible with that Surface to acquire an IP address and thereby use that Surface without special configuration.

The main drawback of dynamic addressing is the difficulty of initiating traffic to a device from a remote location, making this scheme unsuitable for servers. However, there are many types of device for which this is not an issue; for example, PDAs normally act as clients in

any communication, whether it be to fetch a web page for the user, or to synchronise the user's data with a server.

When using dynamic addressing over Networked Surfaces in particular, there is a further complication, in that devices may disconnect and reconnect to a Surface often. If a different IP address were allocated each time this occurred, the device's TCP connections would be severed at each disconnection. It is therefore important to use an address allocation policy that re-assigns the same address to devices whenever possible. For IPv6, because addresses are constructed using a known host ID and network ID, the same address is always used for a given device on a given network, thereby avoiding this problem.

NAT-based Addressing

Network Address Translation (NAT) is a means of allowing many devices to use a single IP address. This is useful when addresses are scarce and cannot be allocated to all devices, and also when a device is administratively barred from using more than one address, but needs to support other devices. The former might normally be attributed to the exhaustion of the IPv4 addressing space. The latter might occur when someone with a home network wishes to use a single modem connection to provide all their devices with Internet access.

For Networked Surfaces, either of the above situations may apply, and in either case a Networked Surface could use NAT to make traffic from all objects on it use only a single IP address. In particular, the latter case may be relevant if a small-scale Surface were deployed, for example if a Surface were used primarily to allow peripherals to be connected to a computer. If one of the peripherals (such as a PDA) required an Internet connection, its traffic could be masqueraded as coming from the computer itself.

The use of NAT has the advantage that mobility can be supported without the use of static configuration, and does not even require DHCP to be used. However, the restrictions on communication are even stronger than for DHCP. Even after a device has contacted a remote correspondent, that correspondent cannot initiate a new connection to the device; all connections must originate from the device. This may lead to certain applications malfunctioning (e.g. the `ftp` program) if special precautions are not taken (e.g. the use of a "passive" mode in `ftp`, or the use of pre-configured port forwarding on the surface manager).

Mobile IP

Mobile IP (for IPv4) is a method of allowing device to be mobile, while still retaining TCP connection integrity, and the ability to be contacted through a well-known address (the "home" address). When roaming, a device uses a dynamic "foreign" address, provided by

the "foreign agent." Forwarding of traffic is facilitated through use of a "home agent," with which the mobile node registers its foreign address whenever it moves.

Networked Surfaces may support Mobile IP, thereby allowing objects to be mobile across all Surfaces without breaking TCP connections, and retaining contactability using their normal address. A surface manager can act as both foreign agent for objects visiting that Surface, and as home agent for devices which are normally on that Surface and are currently visiting another Surface. A similar problem arises as for dynamic addressing, in that an object visiting a foreign Surface may disconnect and reconnect. A naïve implementation might assign it a different "foreign address" on each connection, requiring it to re-register with its home agent, and re-inform its correspondents of the change of forwarding address. Care must be taken to re-assign the same foreign address whenever possible, and also to minimise the sending of unnecessary control messages.

With IPv6, the use of mobile objects does not require special provision, as router advertisements from the surface manager allow any object to generate their own "foreign address." Again, care must be taken to not generate unnecessary control messages every time a disconnection and reconnection take place.

4.4 Supporting Multiple Network Interfaces

The previous section established how addressing and routing will be accomplished in the simple case of an object with a Networked Surface interface. The discussion now moves to a situation in which objects may have multiple network interfaces, one of which may be the Networked Surface. This section discusses the rationale behind objects using multiple network interfaces,[3] and the means by which the correct choice of interface might be made. Methods for supporting multiple interfaces at the network layer are then presented, including the issues of addressing and routing, followed finally by a discussion of how Networked Surfaces in particular are affected when multiple network interfaces are used.

4.4.1 Devices Supporting Multiple Networks

In the past, access networks have been arranged in a tree-like fashion, following strict hierarchies. A device would either be a host designed for end users, or a router designed to perform networking tasks but not support users directly. For hosts, this tree-structure was encouraged by the fact that only one network interface would normally be present, thereby only allowing communication to one gateway router in the layer above. Recently, however,

[3] Also known as networking with "heterogeneous networks."

a number of factors have colluded to break down this hierarchy somewhat, resulting in the proliferation of personal computing devices now supporting multiple network types. The reasons behind this are outlined below.

Firstly, many new types of network have come to the fore recently, including wireless LANs, and ad-hoc networks such as Bluetooth. This has been driven by the supporting technologies becoming feasible and inexpensive enough to be commercially viable.

Secondly, devices have been growing "smarter," due to the decreasing cost of embedded computing power. Whereas previous generations of devices such as PDAs and peripherals have not had enough computing power to support networking "stacks" such as TCP/IP, such power is nowadays routinely embedded in these devices.

Thirdly, the proliferation of communications, the Internet and mobile phones being two good examples, have changed the view of networking from being something that is only accessed when required, to something which should be always available. Many applications have migrated into this new service model, e.g. information repositories on the web, and email or SMS instead of letters or faxes.

The first two factors mean that providing multiple network interfaces in devices such as notebook PCs is economically feasible. The demand for networking support in devices has led manufacturers to take advantage of this potential, as providing support for many network types makes a single device suitable for many consumers, who might only use one of the selection. Notebook PCs are available with built-in Ethernet, WaveLAN, Bluetooth, IrDA, modems, USB, Firewire, and so on, and other network types can be supported using modular additions such as PCMCIA cards.

4.4.2 Advantages of Using Multiple Networks

Having established why more and more devices support multiple network types, the discussion now moves to the advantages that the use of multiple networks can provide.

Firstly, different networks have different coverages, for example, a WaveLAN can be used while in a particular 3D volume, while a dial-up network can be accessed when phone service is present. The ability to use both interfaces therefore provides network access at more physical locations.

Next, networks have varying bandwidth, latency, and error characteristics. Having the potential to access a number of networks allows a choice to be made depending on the connection properties desired. Other factors affecting this decision are the cost of access and the support for mobility by a given network; if a user knows they are going to be stationary they may opt for a cheap but non-mobile network rather than more expensive wireless access.

The choice of network also depends on the correspondent desired, or on the information required if that information is accessible at multiple locations. The use of the "best" network may result in higher throughputs, lower latencies, lower costs, and so on. Fourthly, the simultaneous use of multiple network types may be useful, for such reasons as increased aggregate bandwidth, concurrent communication with correspondents best accessed via different networks, or to implement QoS guarantees using reservations of bandwidth on particular networks.

Finally, the distinction noted previously between hosts and routers is no longer as prevalent, as devices with multiple networks can perform routing. The use of such network paths may lead to greater network availability, such as in circumstances where no fixed point of access is reachable, but where a mobile node could forward traffic. Similarities exist between this concept and that of ad-hoc networking, in which objects sharing a limited-range network type can act as routers so that long-range networking can be achieved.

4.4.3 Network-Layer Issues in Using Multiple Networks

The use of multiple networks by objects, while shown to be desirable above, leads to new problems at the OSI network layer, and in particular for IP. In the discussion of these issues below, it is assumed that Mobile IP (or IPv6) is used; if the object were not mobile, then it would not need to use multiple network interfaces. This discussion is purely speculative, and outlines avenues for possible future work.

Mobility and Handover

With multiple networks, the problem of mobility also encompasses the issue of handover.[4] If one network interface becomes disconnected, and another is available, then actions must be taken to re-route connections. The new interface might be available on-demand, or may incur a setup latency (e.g. a modem connection).

Another situation requiring handover to take place occurs when one of an object's interfaces becomes available, while traffic is using another interface. If the newly available interface were to be faster, or in other ways a better choice than the old one, then connections should be re-routed to use the new interface.

One example where both these cases occur would be if a notebook computer had both a WaveLAN interface and a modem interface via a GSM mobile phone. A particular situation might involve the notebook moving out of one WaveLAN base station's range, and after

[4]The term "handover" is chosen to be distinct from "handoff," which describes movement between two base stations of the same network type

some time entering another WaveLAN's range. A real-life scenario for this example might be a person moving between two offices, while using a notebook PC as a terminal logged in to a central computer.

Handover between interfaces can be accomplished in Mobile IP or IPv6, using "binding updates" as previously mentioned. The choice of which network to use at a given time, and the problems of addressing and routing, are discussed below.

Choosing the Most Appropriate Interface

An object must have a method of choosing which interface to use for a given correspondent. This problem may be very simple, for example, if one network were much faster than the other(s), then it might always be chosen when it was available. An example would be a computer with both an Ethernet and modem interface. Handover in this situation could be implemented using detection of link status (e.g. modem carrier), and using static priorities attached to each interface.

There are however many possible scenarios which are too complicated for a simple priority scheme. Examples include situations when comparable speed networks are used, when the faster network is congested, or when a particular correspondent happens to be "closer" if a slower interface were used. One possibility would be for the object to make sure that if the correspondent address indicated that it shared a network, then that network should be used. However, this is naïve, since a multi-hop connection may outperform a single-hop connection when the latter uses a slower network medium.

Another more general solution would be to send "probe" packets to the correspondent using all interfaces to determine which has the best characteristics for that connection. One possible method of "probing" using the TCP protocol[5] would be to send the initial SYN packet using all interfaces, and then only continuing the connection using the interface on which the first SYN ACK was received. This also works if the recipient of the SYN has multiple interfaces, in which case they would send SYN ACKs over every interface. In both situations, there is the need for marking of packets so that a sender using multiple interfaces can eventually find out which interface resulted in the fastest transfer. This "marking" might be accomplished using the TCP "timestamp" option [56], or by another means.

By using the initial packets as probes, this avoids incurring a delay when the connection is started. However, this solution does not work if the first packets are subject to delays in the network, either due to random congestion or due to the fact that the first packet may

[5]TCP is discussed in-depth in the following chapter.

cause the network to perform configuration tasks so that the route is established. This can be remedied by periodic "re-probing."

Addressing and Routing

Addresses in IP are normally assigned per-interface; while this is useful for routing, it confuses the issue of using multiple networks. For example, if one of an object's interfaces was down, then a correspondent trying to reach the object on that interface's IP address (using unmodified IP) would fail, even though valid routes may exist via other interfaces. A number of solutions to this problem are presented below.

DNS. One solution is for DNS [73] to be used to provide multiple IP addresses for an object's hostname. This facility is currently used in load-balancing applications. To be useful, this solution would require IP to attempt to use all of the addresses provided, when trying to contact a correspondent. However, existing IP implementations only try the first address provided.[6] The discussions of other solutions below assume that DNS is configured to return a single "default" interface address for each object.

Multiple IP Addresses. Another solution is for an object to use an IP address per interface. This has the advantage that correspondents which were contacted using a particular interface, would use the same interface by default in future communications. If the object were to want a correspondent to communicate with it on another interface, it would simply send a binding update to that correspondent, directing it to the new interface address.

The disadvantage of this approach lies in the fact that each object needs as many home agents as it has interfaces, and it must keep those home agents up-to-date with the address of the best interface to use, whenever it is not "at home" on that interface.[7]

Note that an object may wish to register as being "away from home" with one of its home agents, even if it was connected to that interface. This would be useful in the situation that the interface had a long latency or low bandwidth, and another faster interface was available. Redirection of new connections to the faster interface would then be accomplished at the home agent, rather than at the object itself, speeding up the move to the faster network.

[6] The Linux 2.4.16 IP implementation was tested, and exhibited this behaviour.

[7] Although the home agents may be implemented on a single computer, and even in a single process, multiple state variables must still be kept up-to-date, requiring many update messages whenever movement occurs.

Single IP Address. A third solution would be for the object to use a single IP address at all times, which would be its address on a designated "best" interface (the "primary interface"). This allows a single home agent to be used. It also lets correspondents track devices which uses many networks over time. This means that application-layer state which is tied to particular IP addresses will remain valid, e.g. authentications and permissions tied to an IP address.

However, a performance penalty is incurred whenever an object wishes to initiate communication with a correspondent on a non-primary interface. In this case, it must always send the correspondent a binding update along with the initial packet of the connection. In the current IPv4 Internet, not all hosts support binding updates, in these cases, triangular routing may have to be used for communications.

IPv6 With IPv6, addresses are 128 bits long, and are composed of a host id and a network id. Although IPv6 can be used with the schemes described above, the presence of long addresses also facilitates another scheme, which enjoys advantages of both the multiple and single IP address schemes. Specifically, the host id portion of a device can be kept constant (they would normally differ for different interfaces), but the network id portion may change on a per-interface basis.

By making correspondents identify hosts based only on the host id portion of the address, the cross-network tracking of devices is enabled. However, because the device is permitted to use multiple source addresses, no binding update is necessary when initiating connections from non-primary interfaces. Also, only one home agent would be required, though this home agent would need to be aware of the multiple network interfaces available to a particular device.

4.4.4 Networked Surfaces and Multiple Network Interfaces

In the case of the Networked Surface as one of multiple network interfaces on an object, the solutions presented above apply. The surface manager is a suitable choice for an object's "home agent"; its ability to detect object connectivity allows it to easily determine when traffic for an object must be re-routed. Using normal Mobile IP, this information would be of no help on its own, as the home agent cannot re-route packets until it is notified of the object's foreign address. However, the addition of a "default" re-routing would enable this information to be used constructively.

For example, consider an object with two interfaces, a Networked Surface interface and a slower Bluetooth interface. The object may experience handover often, as it is moved around

between Surfaces (using Bluetooth in between). Consider also that a scheme was used where the surface manager was provided with a default re-routing for that object to the Bluetooth interface. Then, when the object disconnects from the Surface, the surface manager could immediately and automatically re-route its traffic to the Bluetooth interface. In addition, if it kept data on the TCP connections in use by the object, it could proactively also send binding updates to correspondents, and avoid the need for the object to send them (over the slow and perhaps busy Bluetooth interface).

In addition to the issue of objects with multiple network interfaces, Networked Surfaces have another similar issue intrinsically, as each Surface may support multiple networks (of the same type or different types). The choice of which objects are placed on which bus must be made by the surface manager, according to criteria such as priorities assigned to objects, or the load on each bus. These networks can be addressed using a single subnet, or they may be assigned different subnets. In the former situation, routing is made simpler, but more address space must be allocated to each surface manager.

Interestingly, with the single-subnet addressing scheme, the surface manager has the ability to transparently move objects between equivalent buses, without breaking the object's connection. This is accomplished by simply instructing the tiles to re-route the appropriate pads. Transparent assignment of buses can also be performed during object connection, by "moving" an object during the registration process. In this situation, each network type would have a "default" bus, which the tiles would be configured to place objects on during handshaking; this is necessary to allow registration to be initiated.

4.4.5 Related Work

Finally, some related work in the area of using multiple network interfaces is discussed.

The concept of "wireless overlay networks" [99] offers a hierarchical view of multiple network interfaces. The scenario evaluated in this project included a room-scale infrared network, a building scale WaveLAN, and a wide area wireless network ("Ricochet"). The advantage of the hierarchical approach is that it makes the choice of network interface very simple, in that the available network which is "lowest" in the hierarchy is used at all times. However, the use of a hierarchy embodies assumptions about the networks. Firstly, the smaller-scale networks are always assumed to be the best network. While this may often be true if speed is the only consideration, other factors such as network congestion may go against this assumption. Secondly, the use of only one network type at a time is supported. This has the implication that networks are not chosen on a correspondent-specific basis, which may result in a sub-optimal choice of network for some connections.

A mechanism for using multiple interfaces was proposed in [110]. In this work, the "metric" field in the IP routing table is used to tie traffic for specific correspondents to specific interfaces, thus supporting the use of different interfaces simultaneously. The method of choosing which interface is most suitable for a given correspondent is not addressed, but applications which make such decisions are supported by means of a *bind-to-device* socket option. Work is also presented concerning the updating of home agents and correspondent nodes so that the correct interface is used; this has been superseded by the route optimisation and binding update features of Mobile IP and IPv6.

"Physical media independence" [52] is the concept that heterogeneous networking should be no harder for a user to configure than networking with a single network. Various methods were implemented allowing a device to keep track of its available networks, using factors such as device connectivity and affordability. The choice of default interface is made using predefined priorities, and does not take into account congestion, or the particular correspondent. The aim of simple configuration is however attained, and six methods of sensing device availability are provided.

MSOCKS [69] uses a proxy to avoid the problem of correspondents trying to communicate with a moving host with multiple interfaces. The correspondents instead communicate with a proxy, which is knowledgeable of the multiple network interfaces present on the mobile host, and can route different streams over different interfaces. This solution was built with the assumption that building mobility support into correspondents is prohibitive, but with IPv6 (and to a lesser extent with IPv4's route optimisation), this has been achieved. The use of a proxy also introduces undesirable triangular routing in the case that a mobile host moves a large distance from its proxy. In the limited situation it caters for, however, the MSOCKS system appears to provide transparent use of multiple network interfaces.

Finally, another interesting concept is that of MObile grouPEd Devices (MOPED) [61]. While the proposals discussed in this chapter address the problem of an connectivity to an object with multiple interfaces, the MOPED concept addresses the contactability of a *person*, who may carry multiple computing devices. Parallels exist in the area of addressing and routing when many interfaces may be available; in the MOPED scheme, a NAT approach is followed, and one IP address is used per person.

4.5 Summary

This chapter has discussed the use of Networked Surfaces in conjunction with other network types. First, a comparison was presented of the characteristics of various network types including Surfaces. Next, the problem of connecting Networked Surfaces to other networks

was presented. Finally, the issues brought up by the use of objects with multiple network interfaces were outlined, and possible solutions were presented.

Chapter 5

Reliable Data Transport with Networked Surfaces

5.1 Introduction

This chapter will explore Networked Surfaces in the context of the OSI model transport layer. In the ubiquitous IP world, this is represented by the Transmission Control Protocol (TCP), which is therefore the focus of this research.[1]

Issues handled by TCP include the provision of connection-oriented data streams using numbered packets, reliable packet delivery using acknowledgements ("acks") and retransmissions, checksumming to detect bit errors, receiver windowing to avoid buffer overflows, and congestion control to avoid the overloading of buffer queues in routers.

This section will discuss the networking characteristics of Networked Surfaces, and their effect on the TCP protocol. It will then overview the methods available to improve TCP performance in the Surfaces environment, including the "smart link layer" approach which is the main focus of this chapter.

The work presented in this chapter was not undertaken collaboratively.

5.1.1 Networking Characteristics of Networked Surfaces

In order to identify issues affecting TCP on Networked Surfaces, the networking characteristics of Surfaces must be examined. These include the bit error rate, the packet loss rate, and the connection and disconnection characteristics.

[1] UDP is not discussed, as it provides no reliable delivery guarantees, and is therefore does not "care" about channel characteristics.

The bit error rate of the prototype LVDS bus, at the speeds supported by the prototype hardware (i.e. up to 5Mbit/s), is under 10^{-10}. This means that if 1500 byte packets[2] are sent across this channel, approximately one in each hundred thousand packets will be corrupt. This error rate is comparable to that of wired networks, and is therefore not a characteristic which will cause bad TCP performance.

The packet drop rate is influenced by routers dropping packets (due to buffer overflow), and by physical packet collisions. The former is an indication of congestion, and is dealt with by TCP appropriately. The occurrence of the latter is dependent on the link layer, and in particular the link layer arbitration policy. With the "token star" link layer, there should be no packet collisions. Another approach to ensure negligible packet losses at the link layer would be to use Automatic Repeat reQuest (ARQ), with CSMA for arbitration.

The last characteristic, that of periodic connection and disconnection, is the primary difference between the Networked Surface and other network types. While wired networks may be connected and disconnected, this is not considered part of normal operation. For wireless networks, disconnection may be incurred when devices move out of range of each other. However, this is not a sharply defined event, and involves a degradation of signal strength as distance increases. It is therefore not possible for devices to confidently determine disconnection has occurred, unlike on the Surface, where disconnection detection is explicit and immediate.

Furthermore, with wireless networks it is possible in some cases to perform handoff while two base stations are in range, so that the disconnected period is very small or nonexistent. This is not possible with Networked Surfaces, for which the minimum disconnection time is determined by how long it takes to disconnect and then immediately reconnect. As shown in Section 3.5.4, reconnection generally takes between 200ms and 500ms.

In summary, the key and novel characteristic of Networked Surfaces affecting the transport layer is the presence of periodic explicit disconnections (and subsequent explicit reconnections), with connected periods enjoying low levels of bit error rates and packet loss rates.

5.1.2 The Effect of Disconnection on TCP

Explicit disconnection is now compared with other network events affecting TCP, to see if standard TCP mechanisms can handle this event, or if new methods are required to maintain performance in the face of disconnection.

[2] 1500 bytes is the default Maximum Transmission Unit (MTU) on many network types.

TCP regards all packet losses as indications of congestion. This, while working well for wired infrastructure, has caused many problems when combined with wireless access, in which channel errors causing dropped packets are more common. Distinguishing and coping with such losses, and making TCP fully utilise a lossy channel, is the subject of much research, as discussed later.

Unfortunately, there are many differences between disconnection and lossy channels, which mean that the same solutions may not work well for both cases. Firstly, in the lossy channel case, it is obvious that discovering the loss and retransmitting quickly is desirable. However, with disconnection, retransmissions are not useful until the link is re-established. On the contrary, transmitting and retransmitting packets for a disconnected device is guaranteed to be a waste of link bandwidth.

In order to illustrate the detailed effects of disconnection on a running TCP connection, a simple experiment was conducted in which a file transfer was started over a disconnecting link. The TCP "trace" occurring in this experiment is shown in Figure 5.1.

As the figure shows, the sender does not react to disconnection, and continues sending (pointlessly) until its window is full. It then waits for acks, but times out before any ack arrives and retransmits the first packet. Retransmission occurs two more times, with an increasing timeout period each time — this is because TCP assumes that the lack of response is due to the network being congested, and so it tries to back off to let the network recover.

When reconnection occurs, TCP does not immediately restart, and instead continues to wait until its next timeout. When this happens, the packet gets through, causing an ack to be received and further packet transmission to resume. Note that over 1.5s of connected time was wasted by unmodified TCP in this case.

In order to optimally handle disconnection, this behaviour must be changed in two ways. Firstly, packets should not be transmitted during disconnection, to avoid wasting channel bandwidth. Secondly, there should be no delay between reconnection and the restart of transmission.

5.1.3 Maintaining TCP Performance with Disconnections

In order to maintain TCP throughput despite disconnection, modifications can be made internally to TCP, or externally. The advantages and disadvantages of these classes of solution are discussed below.

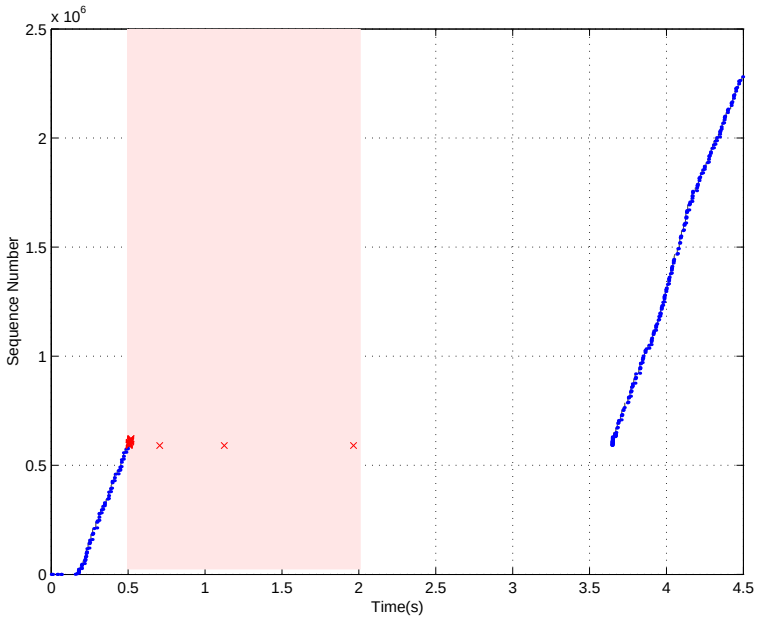

Figure 5.1: TCP File Transfer with Disconnection

TCP traces in this thesis are presented as follows.
The black vertical lines are transmitted segments, the blue dots are acks returning. In zoomed-out plots (such as the one above), these are hard to distinguish individually, and appear as sloped blue line.
The red vertically-shaded portions are periods of disconnection. Segments dropped during these disconnected periods are shown with red crosses.
Segments and acks inserted by the smart link layer (to be described) on reconnection are shown with green circles and green plus marks, respectively.

Modifying TCP to Improve Performance with Disconnections

TCP itself can be modified to include disconnection awareness. This has the advantage of being the most elegant solution, as a globally-deployed disconnection-friendly TCP would achieve performance in the face of disconnection without complicating the networking stack further than it already is. However, the chief disadvantage of this as a workable solution is the difficulty of changing the millions of devices which current support TCP/IP.

External Solutions to Improve TCP Performance with Disconnections

Instead of modifying TCP, a lower protocol layer can be used to "coax" the correct response out of the TCP layer. Such a layer might operate by modifying or retransmitting packets on TCP's behalf. This approach has the advantage of working with any device without requiring modification of its TCP stack, allowing it to be easily retro-fitted, which is useful in the Networked Surfaces context. If the link layer is used for this purpose, then the meta-data that the link layer has access to concerning the state of the network can be of use. This may include data on channel congestion, link disconnection, dynamic error rates, and so on. This data would be easy to incorporate into a link layer optimisation, but hard to integrate into TCP itself, which operates on an end-to-end basis.

Also, one might imagine that with many different types of non-ideal network, the implementation of TCP is being "pulled" in many directions, with a plethora of modifications being proposed to cope with every type of situation, as described later. This does not make for a maintainable codebase, and interactions between the different "solutions" may be subtle and hard to test. External solutions therefore allow the TCP implementation to be kept simple, and therefore maintainable.

Disadvantages of external solutions include the possibility of bad interactions between TCP's retransmissions and retransmissions in other layers [30]. Other issues are that packets and state may be stored in many locations (even within the same host), wasting memory space and processing time. Also, a particular external solution may perform well with some versions of TCP, but with others (including future versions) it may perform badly.

Other disadvantages are that the use of end-to-end encryption [9] may hinder or even disable external schemes, which rely on being able to "sniff" packets. Also, modification may break the end-to-end semantics of TCP, causing incorrect assumptions about the status of packets being transmitted.

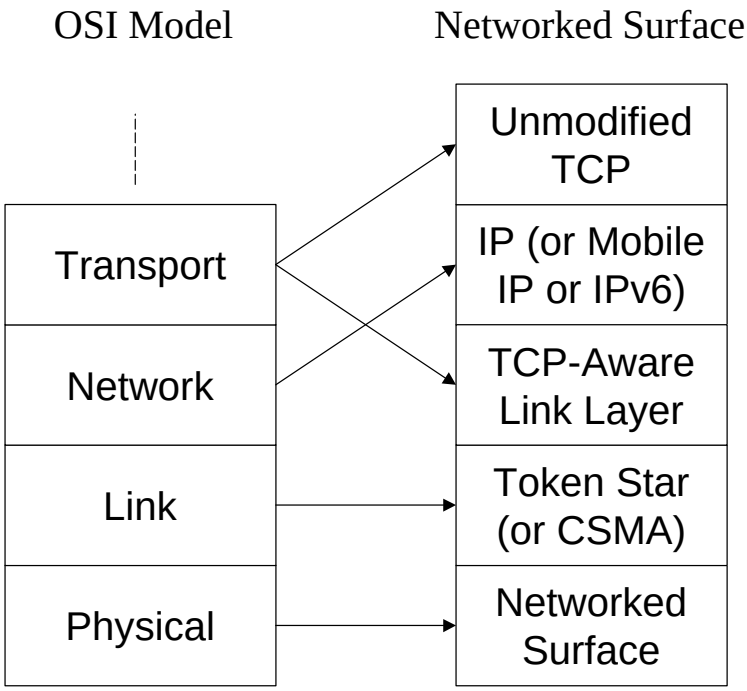

Figure 5.2: Transport Layer Optimisation using Smart Link Layer

Networked Surfaces Viewpoint

From a Networked Surfaces point of view, external solutions allow objects to experience good performance in the face of disconnection, while minimising the augmentation required of an object in order to become Networked Surface–compatible. Integrated solutions, however, requires the use of a modified version of TCP, and would greatly increase the work necessary to make a given object Networked Surface–capable. An external solution, based in the novel link layer of the Surface, is therefore chosen as the most appropriate approach, and is the primary focus for this chapter. This "smart link layer" approach is shown in Figure 5.2.

5.1.4 Overview of Research Presented

This chapter will firstly look at related work involving the TCP protocol, in particular examining research in making TCP compatible with other non-ideal channels, such as wireless links. Next, possible enhancements to TCP using smart link layer techniques will be examined. Various techniques are compared using experiments over a simulated disconnecting channel, and the results are analysed. Finally, the best link layer solution is then implemented and evaluated in a Networked Surfaces context, with experiments in both bulk data

transfer performance and interactive performance.

5.2 Related Work

This section will discuss related research concerning TCP [86], and in particular research concerned with making TCP appropriately handle non-ideal network events such as disconnection.

5.2.1 TCP Congestion Control

The introduction of congestion control was due to the experience of "congestion collapse" [74] in the early Internet. This led to the development of the "slow start" and "congestion avoidance" algorithms [54], to which the "fast recovery" and "fast retransmit" algorithms were added [55].

Various further improvements have since been devised. These include the use of Explicit Congestion Notification (ECN) [37] as a method for signalling congestion without dropping packets (which wastes bandwidth and time). Strategies such as Random Early Detection (RED) were also proposed for gateways, to allow the signalling of congestion before the point of forced loss due to buffer overflow [39].

To improve the speed of congestion recovery, schemes such as selective acknowledgement (SACK) [35], forward acknowledgement (FACK) [70], SMART [58],"limited transmit" [7] and NewReno [38] have been developed. The initial "slow start" period has also been addressed, with optimisations to the slow start scheme [47], and alternatives including "Tri-S" [102] and a packet-pair rate probing approach [57].

New implementations of TCP have been created, including "TCP Vegas" [20] which incorporates a number of optimisations for slow-start and congestion avoidance, "TCP Santa Cruz" [79] which performs well despite path asymmetries, and "TCP Veno" [27] which uses adaptive congestion controls to best utilise bandwidth in wireless situations.

5.2.2 TCP Modifications for Non-Ideal Environments

As previously mentioned, TCP experiences bad performance over non-ideal networks [63, 8, 36], due to TCP's assumptions that link errors are minimal and that packet losses are due to congestion. Using unmodified TCP, such errors are hard to distinguish network errors from congestion [18].

Explicit Notifications

One obvious class of modifications to TCP involves including explicit notifications when packets are dropped, similar to Explicit Congestion Notification (ECN) mentioned above. These include Explicit Loss Notification (ELN) [14], Explicit Bad State Notification (EBSN) [13], Explicit Link Failure Notification (ELFN) [49], Route Failure Notification (RFN) [26], as well as an ICMP-based solution [42].

Another extension along these lines is named TCP Header Checksum (HACK) [16]. HACK postulates that, in many cases, packets experiencing errors may have error-free headers up to and including the TCP header. In these cases, it is possible to use the header to determine the precise sequence number that can be retransmitted, and send a request to the sender for retransmission. This is implemented using two new TCP options, namely a header checksum option and a retransmission request option.

The Split-Connection Approach

Another type of solution for coping with networks with a non-ideal segment was pioneered in Indirect TCP (I-TCP) [12]. In I-TCP, TCP-layer proxying is used at an appropriate gateway; this involves the setting up of two TCP connections at the gateway, with incoming data on one feeding into the outgoing data queue of the other. In this way, the two network types do not share TCP state, hence allowing the "ideal" network segment to proceed as normal. For the "non-ideal" network segment, modifications can be made to TCP to improve performance. Mobility is also handled, transparently to the remote TCP and to the application. The disadvantage is that end-to-end TCP semantics are not upheld, in that acks are sent for data which has not actually reached the final recipient.

A split-connection approach has also been followed by M-TCP [21], with some differences. Firstly, the two TCP connections are much more interdependent, with the proxy only sending acks for data once it has been ack'd by the final recipient. This preserves end-to-end semantics in TCP. Secondly, M-TCP does not normally acknowledge the most recently received byte. On disconnection, M-TCP uses this unacknowledged byte to send a new acknowledgement, which advertises a zero-length window and puts the sender TCP into "persist" mode. In this mode, the TCP state is frozen, with no congestion window shrinkage or retransmission timeout occurring. While useful in this case, especially for handling periods of disconnection, the "window shrinking" practice is strongly discouraged by the original TCP specification [86], although it is recommended that TCP implementations react properly to such shrinking.

Other Approaches

Caceres and Iftode [24] were one of the first to examine the problem of disconnected periods affecting TCP, which they found to occur during mobile handoff. They proposed a system augmenting TCP so that, on reconnection, a mobile host would retransmit a number of duplicate acks for the packet it has most recently received. This should force its TCP correspondent into entering fast retransmit mode. This work requires only small modifications to the mobile device's TCP implementation, and is similar to one of the "smart link layer" approaches evaluated for Networked Surfaces, as discussed later.

WTCP [97] completely rewrites TCP's congestion and flow control dynamics, to optimise for Wireless WAN scenarios, including packet-pair rate measurement on startup, receiver-controlled rate adjustment, and SACK without retransmission timers for loss recovery.

5.2.3 External Methods for Improving TCP in Non-Ideal Environments

Instead of modifying TCP to improve performance in the face of non-ideal networks, methods external to TCP can be used to achieve a similar goal.

Snoop [15] is one of the earliest implementations of a solution in this area, and tackles the case of a lossy wireless link on the periphery of a network. The problems addressed were firstly the interpretation of wireless link errors as congestion, and secondly the TCP reaction to the latency of mobile handoff. The solutions proposed were to include local retransmission and duplicate ack suppression at the wireless base station, and to have a period before handoff where the new base station is buffering packets as well as the old base station. Snoop maintains the end-to-end semantics of TCP, and results in performance improvements in the areas it tackles. It does however require base stations to have higher amounts of memory and processing power.

The "Delayed Dupacks" scheme [101] offers a two-tiered solution to the same problem. Firstly, the TCP receiver is slightly modified, to delay duplicate acks by a set amount, and not send them at all if new data arrives prior to the timeout. Secondly, the base station and mobile host use a reliable link layer protocol, which uses acknowledgement for each packet and fast retransmission. The main difference between Delayed Dupacks and Snoop is that the Delayed Dupacks link layer, while being "smart," is not TCP-aware, therefore avoiding problems when encryption is used. Despite this, performance is comparable to Snoop with an appropriately chosen timeout value.

TULIP (Transport Unaware Link Improvement Protocol) [80] is a link layer protocol attempting to solve the wireless loss problem without being aware of transport-layer state. It does this by classifying packets by the type of service they require, with TCP data packets

demanding "reliable" service, and other types of packets (such as UDP) requiring "unreliable" service. For services requiring reliable delivery, the TULIP protocol uses acknowledgement (with a SACK bitmap[3]) on a per-packet basis, and delays out-of-order packets to guarantee in-order delivery to higher layers.

The aim of TULIP is to recover from many transmission losses before TCP's coarse-grained timeouts take place, thereby emulating a channel with a lower loss rate, and avoiding invocation of TCP congestion control. The low-level solution is shown to have good performance, especially with regard to burst losses which confuse the Snoop scheme. In certain situations, problems may occur with differentiating the service level required, particularly when encryption and tunnelling are used (for example, a secure connection to a Mobile IP host).

AIRMAIL (Asymmetric Reliable Mobile Access In Link layer) [10] is another reliable link layer protocol. It is designed to place minimum load on the mobile units, instead shifting work to the base station. While no performance results examining TCP behaviour are given, this is another example of a reliable wireless protocol which has the potential to improve TCP performance over lossy radio links.

ATCP [65] proposes another external solution to TCP, which is implemented as a layer just beneath TCP. This layer implements ECN, and reacts to losses due to errors by putting TCP into a "persist" state and retransmitting on TCP's behalf. It also reacts to route changes (which it detects on reception of the ICMP "destination unreachable" message) by putting the sender into "persist" mode until the route is re-established, and resetting the congestion window. Finally, it offers re-ordering of packets so that TCP does not generate duplicate acks and cause fast retransmit to be invoked unnecessarily.

5.3 A Link Layer Solution for Disconnection-Friendly TCP

This section will discuss how TCP performance can be improved using a smart link layer. This approach was chosen as it does not require TCP to be directly modified, thereby reducing the amount of augmentation that a device requires in order to use the Surface. The use of this solution on the Networked Surface is discussed later.

5.3.1 Requirements of a Link Layer Solution

A method of ensuring that TCP connections make efficient use of available bandwidth is required, in the face of periodic disconnections. For example, with a 1Mbit/s channel which

[3] a set of bits specifying the delivery status of packets in the current window

is connected 80% of the time, a 1Mbyte file transfer should be accomplished in about 10s.

Also considered important is a minimal use of resources such as processing and memory. This follows from the need for a non-object-specific solution. In the worst case, the solution will have to be implemented in hardware externally to the object. In a better case, the object itself may run a device driver for the Networked Surface interface which could handle these issues, but that driver should not make assumptions about the design of the object's TCP/IP stack, and should be able to fit into a small amount of memory.

What is not required is a method for reducing packet errors; packets are either delivered by the Networked Surface with the negligible loss characteristics of a wired network, or not at all.

5.3.2 Design of Link Layer Solution

At the link layer, packets are visible as they enter and leave each hop of the network, allowing them to be recorded and/or modified, though the latter is more computationally expensive as checksums must be recalculated. Packets may also be inserted or dropped, in either the receive or send queues. Finally, disconnection and reconnections are assumed to be asynchronously detected (this is true of Networked Surfaces, and any other network type for which the link state can be determined).

As shown previously, TCP wastes bandwidth by not reacting to reconnection. In order to force a reaction, the obvious solution is to insert packets into the network at reconnection time. This can be done in one of two ways, either by inserting packets into the incoming queue, or inserting packets into the outgoing queue. These methods are illustrated in Figure 5.3, and are discussed in further detail below. Firstly, however, some other parameters for this solution are discussed.

Parameters for Packet Insertion

The decision above to insert packets on reconnection, in either the incoming or outgoing queues, still leaves a number of important questions. In particular, there is the issue of how the inserted packets are constructed, and the issue of how many different packets are inserted.

The former issue can be broken down into two possibilities; either the packets are copies of packets TCP has already sent, or they are constructed afresh by the smart link layer, by using information received from other packets. While the latter approach gives the maximum flexibility to the smart link layer, allowing it to choose precisely the content of its inserted packets to provide the quickest recovery, this approach is resource-intensive, in that the

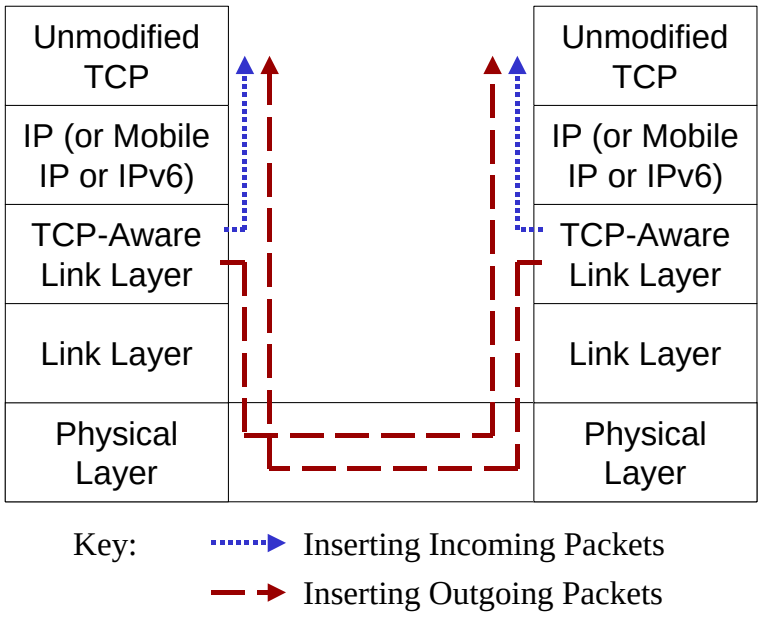

Figure 5.3: Types of Smart Link Layer

inserted packets must be constructed and checksummed by the link layer itself.

In contrast, although the use of pre-transmitted packets provides less flexibility, it also requires fewer resources, and assumes less about the particular TCP implementation being used, in that any unknown options or parameters present in packets are simply passed on without modification. This tactic is pursued in this research.

The next consideration is the number of packets that should be repeated on reconnection. While it is possible to pursue policies such as link layer retransmission of all unacknowledged data on reconnection, this would require a large amount of buffer space, may result in a large wasted bandwidth, and may interfere with TCP's own retransmissions. For these reasons, it was decided that only one packet per TCP connection would be buffered.

This policy still leaves the opportunity for repeats of the buffered packet to be sent on reconnection. Repeating packets may be useful as they can cause the receiving TCP state machine to be forced into a fast retransmit mode, since the packets will cause duplicate acks to be received. The benefit of repeating packets is examined experimentally below.

Inserting Incoming Packets

Inserting incoming packets (known hereafter as "re-receiving" packets) means that the packets are directly inserted into the receive queue of hosts at either end of the disconnecting

link. This means that the inserted packets do not traverse the disconnecting link, though they may be transmitted across other parts of the end-to-end network.

The advantage of this approach is that, in the likely case that the disconnecting link is at the periphery of the network, one of the TCP correspondents receives its "kick-start" very quickly, as no network latency will be incurred for re-reception on this side.

However, re-reception of packets is by definition never going to provide the receiving TCP stacks with any data they have not seen before. These packets are therefore confined to repeating old data, old acks, and/or old window advertisements.

The choice of data to re-receive is determined by that data which would cause TCP to immediately send out more data, and initiate recovery mechanisms bringing the connection back up to full speed as quickly as possible. Since these mechanisms are governed largely by the reception of acks, the best information to re-receive is obviously the highest ack already received.

Finally, in order to ensure that idle connections are not needlessly "kick-started," it is sensible to only re-receive on reconnection if there was a send attempt on that connection during the disconnected period. (Other methods of monitoring connection activity, such as idle timers, could also be used.)

Inserting Outgoing Packets

When reconnecting, packets can also be inserted into the disconnecting link's outgoing queue (known hereafter as "re-sending" packets[4]). This has the disadvantage that some network latency is incurred before either TCP stack receives its "kick-start." However, the advantage is that the re-sent packets may include new data, new acks, and/or new window updates.

This new information can be found in two ways. It can be obtained from packets that TCP sends around the point of disconnection, but before the lack of incoming acknowledgements forces TCP to stop. It may also be obtained from packets retransmitted during the period of disconnection. The latter, although providing no new data, are important for a number of reasons. Firstly, during the disconnected period, packets that were sitting in the receive queue may be processed, hence generating new acks. Secondly, for the same reason, the receiver's advertised window may increase. Thirdly, and importantly for interactive TCP sessions, retransmitted data segments may combine multiple segments which fall under the path MTU, hence providing a single packet containing much or all of the outstanding unacknowledged data.

[4]The term "re-sending" is chosen to be distinguishable from "retransmission." The former is caused by the smart link layer, the latter by TCP itself.

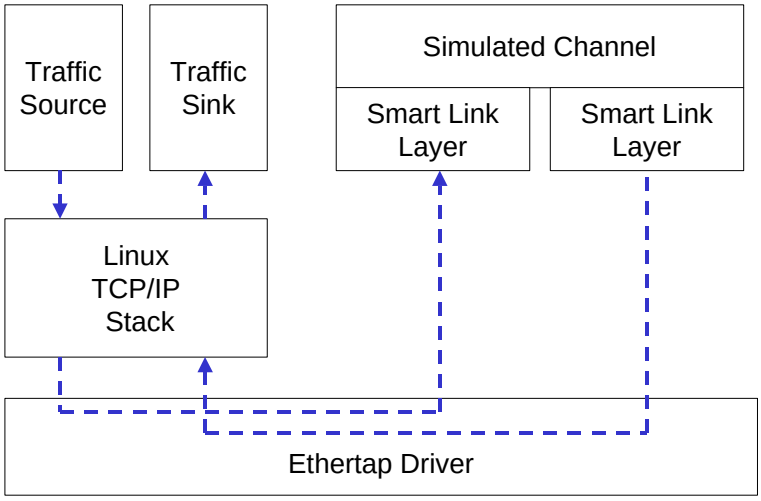

Figure 5.4: Ethertap Experiment Setup

In order to avoid disturbing idle connections with this policy, packets should only be re-sent if there is unacknowledged data, i.e. acknowledged data should never be re-sent.

5.3.3 Experimental Analysis

To analyse the effect of the various link layer methods described above, an experimental setup using a simulated network channel was constructed. This allowed tests to be run using real traffic, but with precise control over the network connectivity, and also provided a testbed for implementation and bug-fixing of the smart link layer algorithms.

Experimental Setup

To simulate a disconnecting channel, the Linux "ethertap" driver was used. This allows network packets to be routed to a user-level program, which simulates the lossy channel, and implements the send and receive components of the smart link layers. This setup is illustrated in Figure 5.4.

To implement the simulated channel, a two-state Markov model was used, with one state having 100% reliability and the other state having 0% reliability. The mean time spent in each state was configurable to allow different channel characteristics to be simulated. This is illustrated in Figure 5.5. The period spent in each state was modelled by uniform random distributions, between half and one-and-a-half of the desired means; this ensured that the results were not subject to interaction between TCP timers and the channel timing.

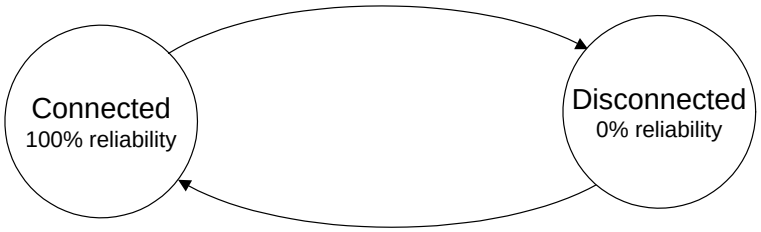

Figure 5.5: Simulated Channel Markov Model

This simulation program was also made to reverse the IP addresses and TCP port numbers of all packets, thereby "mirroring" packets so that the local TCP/IP stack thinks the packets are incoming rather than outgoing. This change allows networking tests to be performed on a single machine.

Using this arrangement, networks with various connection and disconnection patterns can be simulated, and various smart link layer algorithms can be tested.

Optimisations Tested

Out of the possible optimisations outlined previously, five were evaluated experimentally. These "smartlvls" are outlined below.

Smartlvl 0 This is the control case, and represents unmodified TCP.

Smartlvl 1 The most recently received packet for each TCP connection is always buffered. If a send is attempted during the disconnected period, this packet is re-received when the channel is reconnected.

Smartlvl 2 As for Smartlvl 1, but the packet is re-received 5 times on reconnection.

Smartlvl 3 A single buffer contains the "most useful" outgoing packet per TCP connection, which is re-sent whenever channel reconnection occurs. The "most useful" packet is the transmitted packet obeying the following criterion (highest priority first): highest ack, lowest unacknowledged sequence number, longest length of data, and longest window advertisement.

Smartlvl 4 As for Smartlvl 3, but re-sending 5 times on reconnection.

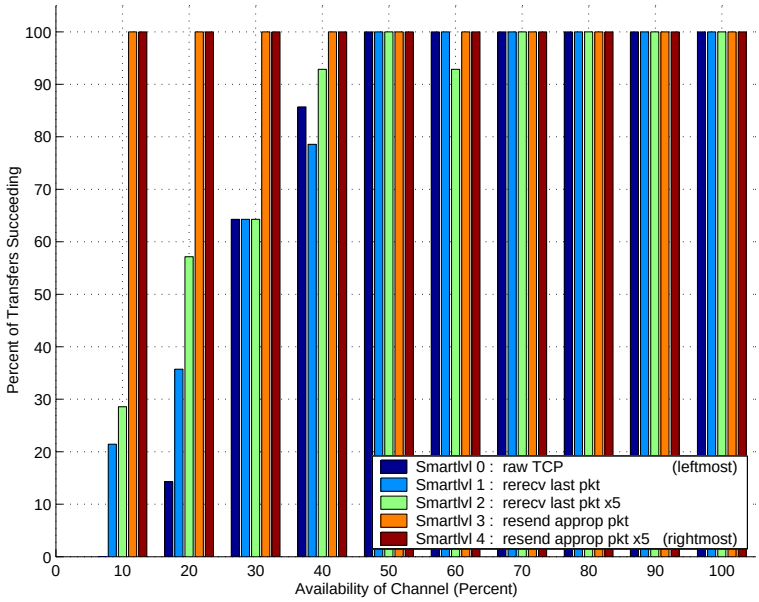

Figure 5.6: TCP File Transfer Tests over Simulated Disconnecting Channel

Results

In order to determine the relative performance the various optimisations outlined above, timed bulk file transfers were sent using the disconnecting channel described above. These were conducted with the following parameters.

- File size used was 50Mbyte.
- Mean "uptime" was set to 0.5s, while mean "downtime" was varied from to 0.0s to 4.5s to simulate channel availabilities from 10% to 100%.
- 14 trials were conducted at each of 10 availability levels.
- The success or failure of each trial, and the time it took if successful, was noted.

As the Figure 5.6 shows, the trials all succeed when availability is high. For availabilities from 40% and below, smartlvls 0, 1 and 2 begin to fail. Smartlvl 0 (raw TCP) fails most quickly, and at 10% availability experiences no successful transfers. Smartlvls 1 and 2, which use re-reception of packets, show some improvement, but smartlvls 3 and 4 are the definite "winners," with 100% of transfers at these smartlvls completed successfully, even when the channel is only available 10% of the time.

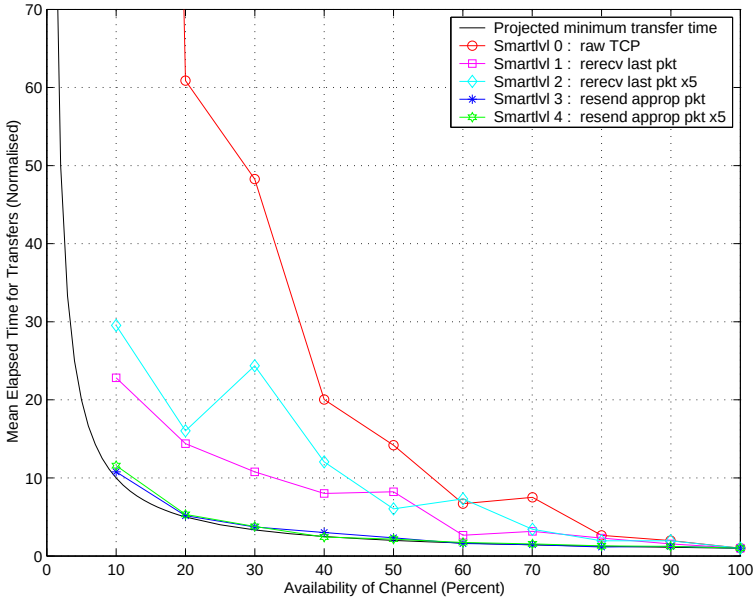

Figure 5.7: Mean Duration of Successful TCP File Transfer Tests over Simulated Disconnecting Channel

Figure 5.7 shows the mean transfer time for the successful transfers, with an unmarked black line indicating the transfer time if the channel were used optimally. As expected, unmodified TCP degrades most quickly, and smartlvls 1 and 2 exhibit some improvement. Smartlvls 3 and 4, however, stay very close to the optimal line, providing good performance even on a channel with 10% availability. This is further illustrated in Figure 5.8, which also shows the low standard deviation of the trials, as compared to those of unmodified TCP. This plot also allows the observation that, even at 80% or 90% availability, unmodified TCP has already diverged from optimal performance, with degradations of about 100% and 50% respectively from the optimal case.

Analysis using TCP Traces

To explain the behaviour above, traces of file transfers were taken with various smartlvls. These are shown in Figures 5.9 to 5.12. Figure 5.1 above shows a trace for unmodified TCP, and includes a key useful for interpreting the traces.

The traces for smartlvls 1 and 2[5] show that TCP does not respond immediately when

[5]The five individual plus marks for the five re-receptions used in smartlvl 2 are not distinguishable at this scale, and look like a single plus mark.

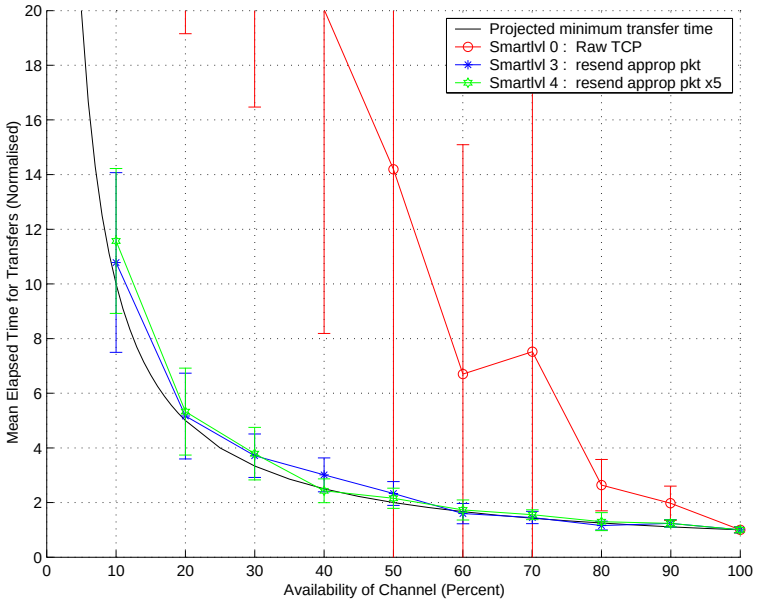

Figure 5.8: Detail of Figure 5.7 for Selected Smartlvls, with Standard Deviations

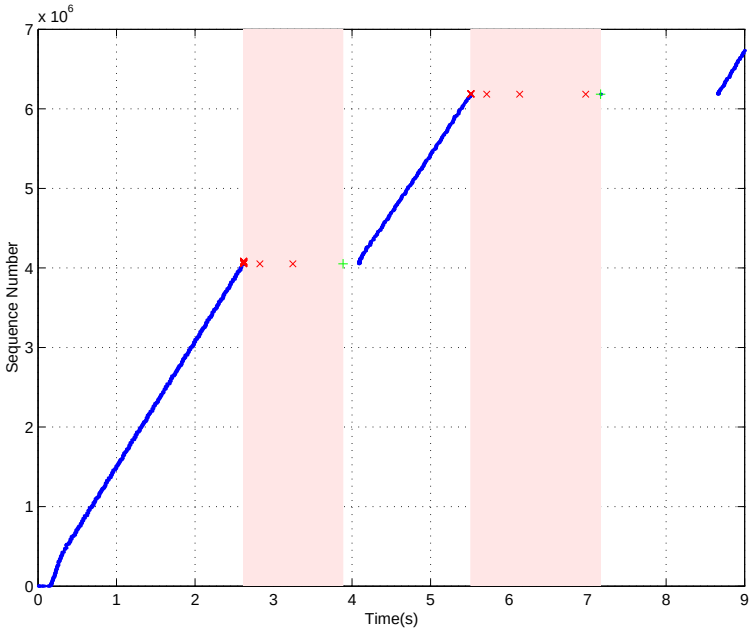

Figure 5.9: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 1)

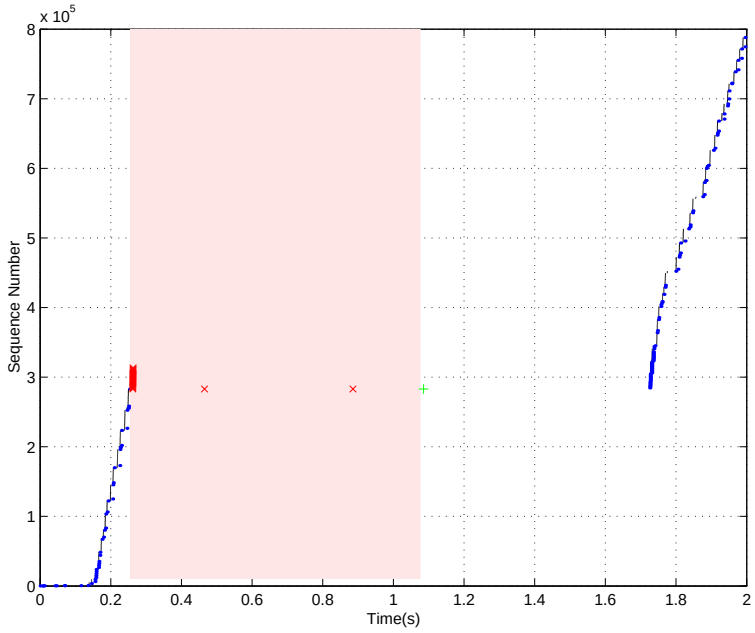

Figure 5.10: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 2)

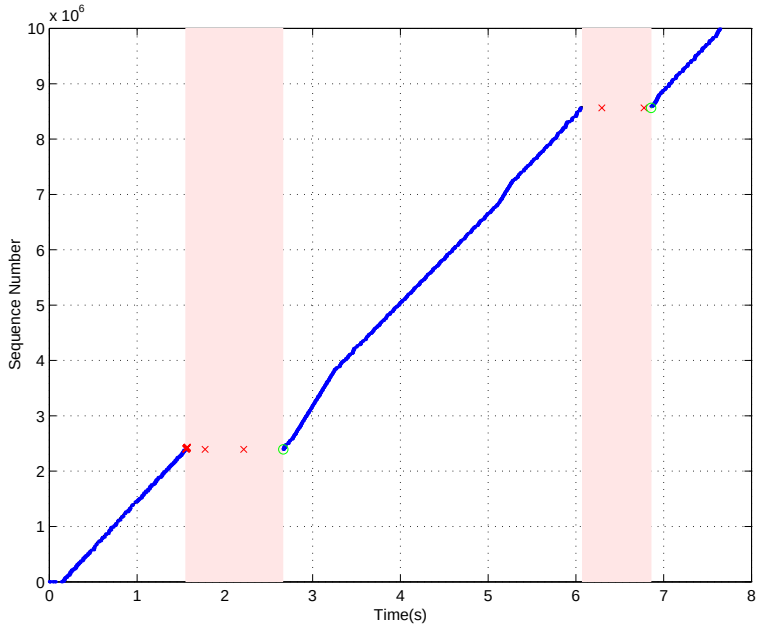

Figure 5.11: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 3)

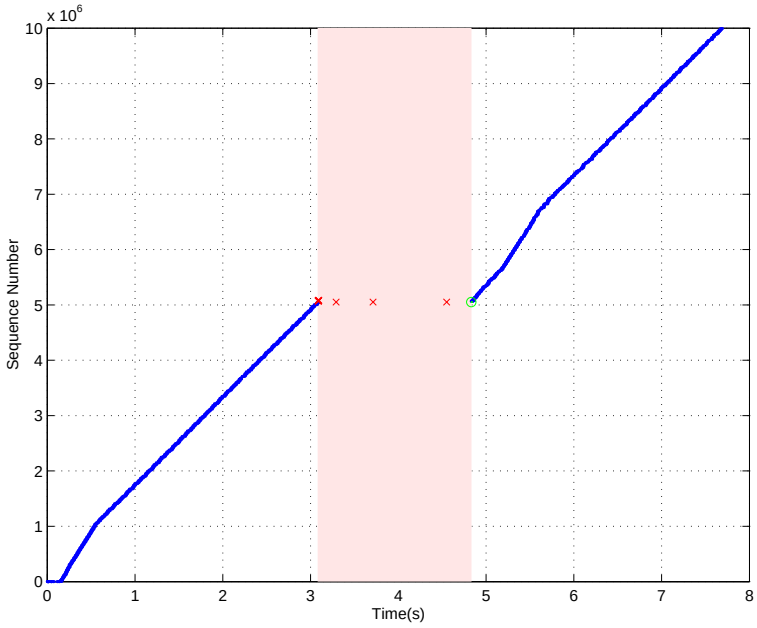

Figure 5.12: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 4)

receiving repeated acks. Although repeated acks are used as a signal to TCP to start fast-retransmit of a packet, this is not successfully invoked in either case. This can be explained by noting that the fast-retransmit mechanism is designed to be used *before* retransmission takes place due to timeout. If such a timeout has already occurred, then the congestion window has been reset to one packet. This explains why re-receptions of an old ack do not cause any response, as the window is not advanced by this event.

The traces for smartlvls 3 and 4[6], on the other hand, show that TCP is immediately restarted after reconnection. This is because the packet re-sent on reconnection is chosen so that it sends unacknowledged data. When this packet is acknowledged, the congestion window is opened and slow-start proceeds as normal. Re-sending five packets at a time with smartlvl 4 does not seem to have any added effect; although they may cause multiple acks of that packet, the fast-retransmit mechanism is not useful in this case, due to the congestion window being reset as described previously.

The conclusion of these experiments is that a "smart link layer" employing re-sending of a well-chosen packet on reconnection can improve the performance of TCP on disconnecting channels. Whereas unmodified TCP experiences bad performance even when the channel is

[6]The five individual circles for the five re-sent packets used in smartlvl 4 are not distinguishable at this scale, and look like a single circle

90% available, the use of the smart link layer gives near-ideal performance at availabilities down to 10%.

5.3.4 Comparison with Existing Solutions

The smart link layer solution presented above is now compared with other solutions to TCP performance problems, which were discussed in Section 5.2.

Firstly, the link layer solution is independent of the TCP implementation used, and can therefore be added to Networked Surface devices without requiring internal modification. This is not the case with the majority of the related work discussed, including all solutions in the category of TCP modifications, and also some in the category of link layer modifications. In the latter case, ATCP, AIRMAIL and the "Delayed Dupacks" scheme require internal modification of devices, and are therefore less appropriate for Networked Surface use.

Next, the smart link layer has very low memory requirements, as it stores only one buffered packet per connection. Many other link layer solutions are heavyweight in comparison, requiring the buffering of all unacknowledged packets, including TULIP and Snoop. This allows the smart link layer to be used in more circumstances. It may be feasible to support this solution in hardware used to augment objects, so that no internal device resources are required. Also, if used in software, the overhead imposed is low, and would not burden objects with low memory, e.g. PDAs.

The solution presented does *not* attempt to bypass the slow-start procedure of TCP, nor does it try to induce a raising of the congestion window. This policy is in contrast to many of the TCP solutions presented, which attempt to keep the congestion window wide despite bad channel characteristics. This difference is largely due to the timescales for which the solutions are designed; single packet losses happen on a microsecond scale, while disconnections may last a number of seconds. Also, during disconnection, objects may be moved to a different Surface, or a different bus on the same Surface. This may cause the congestion characteristics to change, so a slow-start is appropriate to discover the correct new value of the congestion window.

Finally, the smart link layer solution does not break TCP semantics, and does not require more than one retransmission of a packet, thereby ensuring that it is not prone to bad interactions with TCP retransmissions.

5.3.5 Further Work

While the experiments conducted above have demonstrated the usefulness of link layer optimisations, their scope is limited in a number of ways. Firstly, the channel model does not

include a channel latency; all "transmissions" occur with zero latency (other than the latency imposed by processing time). Secondly, the tests are only executed over a single hop. One interesting possibility with multiple-hop networks is the use of a smart link layer for only particular hops of the network. For example, the effects of smart link layers on non-ideal channels at the periphery of a reliable wired network could be tested.

To fully explore these issues, there are a number of possible approaches. One method would be to use a multiple hop test network, in which each link may be subject to a disconnection profile, and each node's "smartlvl" may be individually controlled. Another method would be to implement this generic network in a network simulator. However, for the purposes of this thesis, the interesting scenario for deployment of the smart link layer is to make TCP perform well on Networked Surfaces. Implementation and evaluation of the link layer solution on Networked Surfaces was therefore undertaken, and is presented in the following section.

5.4 Evaluation of Link Layer Solution on Networked Surfaces

This section will describe the effects of the link layer solution described above, when deployed in the context of the prototype Networked Surface. Two types of test were conducted. The first is similar to the experiments described above, and evaluates the bulk transfer performance of TCP. The second aims to characterise the interactive response of TCP over a disconnecting Surface network, by using the Virtual Network Computing (VNC) remote desktop tool.

5.4.1 Performance for Bulk Transfers

In order to examine TCP performance for bulk transfers over a disconnecting Networked Surface, some method of applying controlled disconnections to the Surface must be used. This was achieved by augmenting the Networked Surface device driver to allow an object to be configured to disconnect at random intervals, with the mean interval specified by the user. The "uptime" and "downtime" of the network was recorded, so that its availability could be calculated. Using this method, file transfer times were used to measure TCP performance, in similar fashion to the tests in the previous section.

Results of Bulk Transfer Tests

The following parameters were used in the bulk transfer experiments:

- Networked Surface LVDS network was used, at 1Mbit/s.

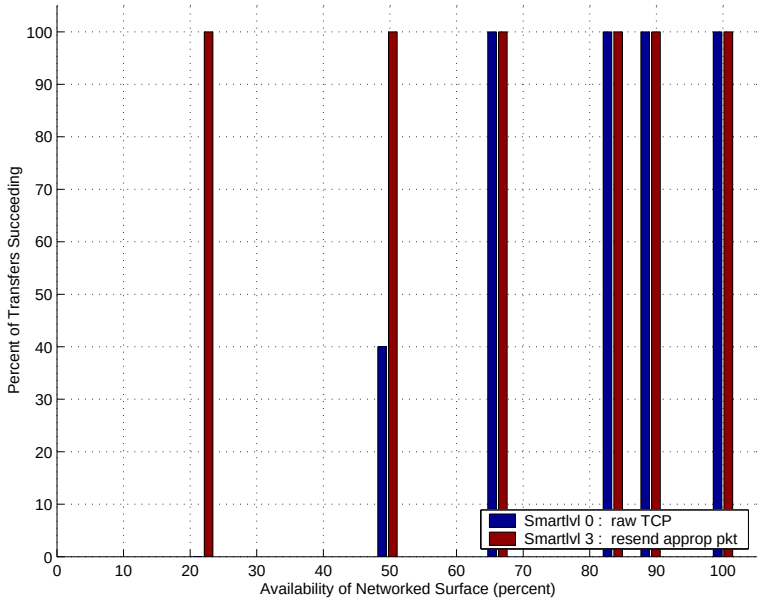

Figure 5.13: TCP File Transfer Tests over Disconnecting Networked Surface

- Disconnections were simulated at various rates, producing various availabilities.
- Smartlvls 0 and 3 were used; smartlvl 0 is the unmodified TCP case, and smartlvl 3 is the best-performing smart link layer optimisation, as shown previously.
- 10 transfers of 5Mbyte were conducted at each availability and smartlvl, at randomly chosen locations on the Surface, and times were recorded.

As Figure 5.13 shows, the smart link layer completed 100% of transfers, down to an availability of 23%, while unmodified TCP did not reliably transfer the file at a 50% availability or less. Figure 5.14 shows that the smart link layer stays relatively close to the "ideal" transfer time, even down to 20% availability. Unmodified TCP has twice the overhead of the smart link layer at around 65% availability, and very bad performance at lower availabilities.

5.4.2 Interactive Performance

While bulk transfer performance is important for some applications, the user of a Networked Surface object may also wish to communicate interactively. Examples of interactive applications are remote terminal programs, web interfaces, and real-time multimedia applications such as streaming audio or video.

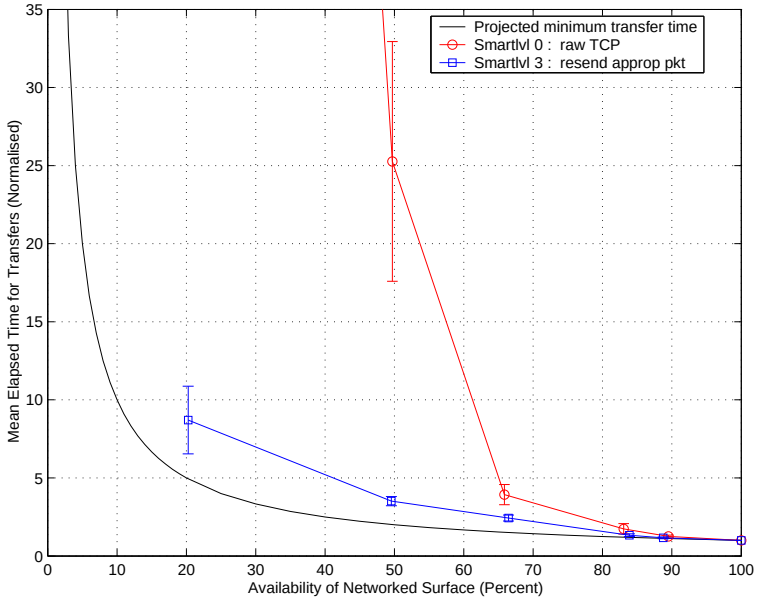

Figure 5.14: Mean Duration of Successful TCP File Transfer Tests over Disconnecting Networked Surface

Such applications may not stress the bandwidth of the network available, so the tests above are not necessarily applicable in this case. What is applicable, however, is *synchronisation*, i.e. the need for a local interface and the remote application to be representing the same state as much as possible. An example of bad synchronisation is when a user clicks on a webpage link, but only after a number of seconds does the page change to reflect this action. Another example might be a remote desktop mouse icon not following the local mouse icon closely while it is being moved.

In order to test the smart link layer's benefits for interactive applications, a quantitative metric must be found for synchronisation performance. As described below, the "frame rate" of a remote desktop application is one such metric.

Testing Method

The VNC [93] remote desktop application allows a user of one computer to interact with a remote computer, by "forwarding" the remote display across a network, and similarly relaying keyboard and mouse input. The VNC protocol operates as follows. When the client connects to the server, it issues an "update request" to that server. The server responds by waiting until its display differs from its record of the client's display (which in the first instance is immediately), and then sending a "framebuffer update" containing changes to

the client's display. On receiving this update, the client applies it and immediately sends another "update request."

Since the server only sends updates in response to requests from the client, this protocol is self-clocking. The requests and updates are sent over TCP, which retransmits the data if it arrives corrupted or is lost in transit, providing the guarantee that all messages eventually get delivered correctly (if channel conditions permit).

Due to the protocol outlined above, only one update is sent at any time. This implies that, for small updates, the frame rate achieved is determined by the latency of the TCP connection used, and not by its bandwidth. The frame rate is therefore a good measure of interactive performance.

Results

In order to gather frame rate data, tests were conducted using to the following parameters.

- A very small VNC desktop session (200×200 pixels) was set up on a machine on the wired network.
- A program was run on the remote desktop, which made a small dot appear and disappear at regular intervals. The effect of this program was to cause the display to require an update every 200ms.
- Using the same disconnecting "test mode" as for the bulk transfer tests, the Networked Surface was configured to have various availabilities.
- As for the previous tests, smartlvls 0 and 3 were used.
- For each test, the VNC viewer program was run for 100s, and record was made of the availability of the channel and the number of updates received over this time period.
- 10 tests were conducted at randomly chosen locations on the prototype Surface, for each smartlvl and availability.

Figure 5.15 shows the results of these tests. The smart link layer performs well, providing 80% of the frame updates even when the channel availability is halved, as opposed to 40% for the unmodified TCP case. These results indicate that interactive performance of TCP over the Networked Surface channel is significantly improved when a smart link layer is used.

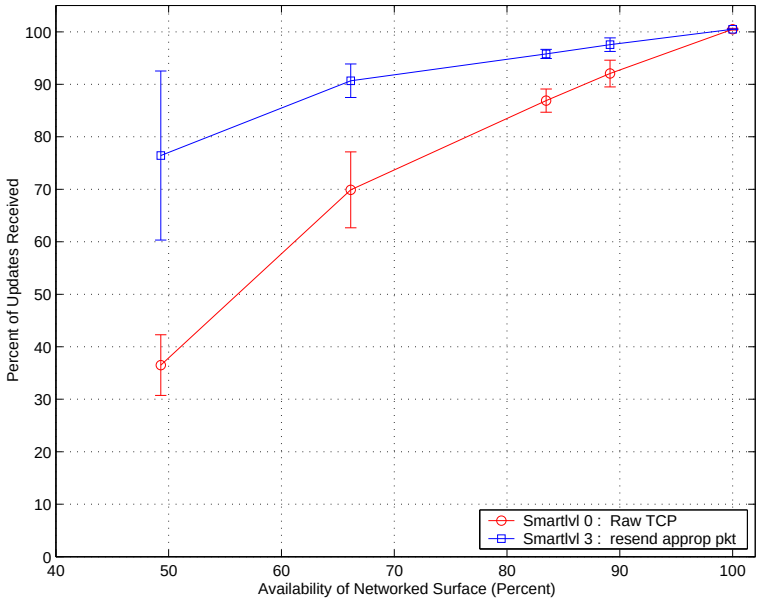

Figure 5.15: VNC Interactivity Tests over Disconnecting Networked Surface

5.5 Summary

This chapter has described a smart link layer solution to TCP performance problems over disconnection-prone networks. It has enumerated a number of possible smart link layer strategies, and used tests over a simulated disconnecting channel to find the most effective one.

The chosen solution involves the buffering of a single packet per TCP connection on both sides of the disconnecting channel, and therefore imposes low additional overhead, rendering hardware implementations of the smart link layer feasible. This is useful in the context of making devices Networked Surface–compatible without having to conduct internal modifications for each object. Furthermore, this solution could be used to improve TCP performance on any network type where periods of disconnection are expected, such as wireless networks which are prone to long periods of bad signal quality.

The smart link layer was evaluated on the Networked Surface prototype, and shown to greatly improve performance over unmodified TCP for both bulk transfer and interactive applications.

Chapter 6

Location Information from Networked Surfaces

6.1 Introduction

Whereas previous chapters have discussed networking issues from the physical layer to the transport layer, this chapter moves away from networking and into the application layer. As mentioned earlier, Networked Surfaces are capable of providing information on the location of devices on top of them. The word "location" is used in the context of this thesis to include two quantities, namely position and orientation.

This chapter will start by discussing the design and implementation of an algorithm for locating objects. The accuracy achieved by this algorithm is then evaluated, using both simulation and experimentation. Methods for improving this accuracy are then examined, along with simulation results incorporating these improvements.

Next, the properties of other types of location systems are examined, and their merits compared with location provided by the Surface. Further background information is then provided on the applications that have been developed to make use of location information, which are part of the field of "context-aware" computing. The possible integration between Surface-based location and existing context-aware systems is then discussed. Finally, some context-aware applications with particular relevance to Networked Surfaces are presented.

The algorithm design presented in this chapter is the result of joint work between the author and Frank Hoffmann. The algorithm and testing software were implemented by the author.

6.2 Obtaining Location Information

This section presents the algorithm that the surface manager uses for deducing location information, given details of the mapping from tile pad to object pad. First, the information available to determine location is examined. Next, the results expected from the algorithm are discussed. Finally, the chosen algorithm is presented.

6.2.1 Information Used to Deduce Location

The surface manager must determine position and orientation information using data on the tile layout, the object pad layout, and the mapping between the surface and object pads.

During initialisation, the tile controllers contact the surface manager, which uses their addresses and static configuration to determine their locations. Similarly, the object managers are aware of their pad layout, though in the current prototype this data is hardcoded.

When an object connects, the object manager is made aware of a one-to-one mapping between the object and surface pads used to make the connection (including information on the tile on which each surface pad lies). The number of mappings available is equal to the number of links that object is using. The object manager transfers this information to the surface manager during registration, allowing the surface manager to perform the location and orientation calculations. This is illustrated in the first two panels of Figure 6.1.

6.2.2 Position and Orientation Results

A device can be located in three dimensions using three position variables and three orientation variables. However, when on a Surface, one position variable and two orientation variables are already decided; the height is that of the Surface, and the object is oriented parallel to the plane of the Surface.

This leaves three variables to be computed by the location algorithm, which shall be called x, y and θ. The former two are the 2D position of the centre of the object on the surface, and the latter is the angle between a vector on the surface and another on the object. This is illustrated in the final panel of Figure 6.1.

For the surface, the vector used to determine θ is parallel to the y dimension, for the object, the line from the origin to the first pad is used. This has the implication that it is important to mount the pad circle on the object device in a specific fashion, perhaps with the first object pad pointing towards the "front" of the object. Furthermore, to determine the positioning of the object device itself, for example for a visualisation application, the position of the pad circle on the object base must also be known, as well as the dimensions

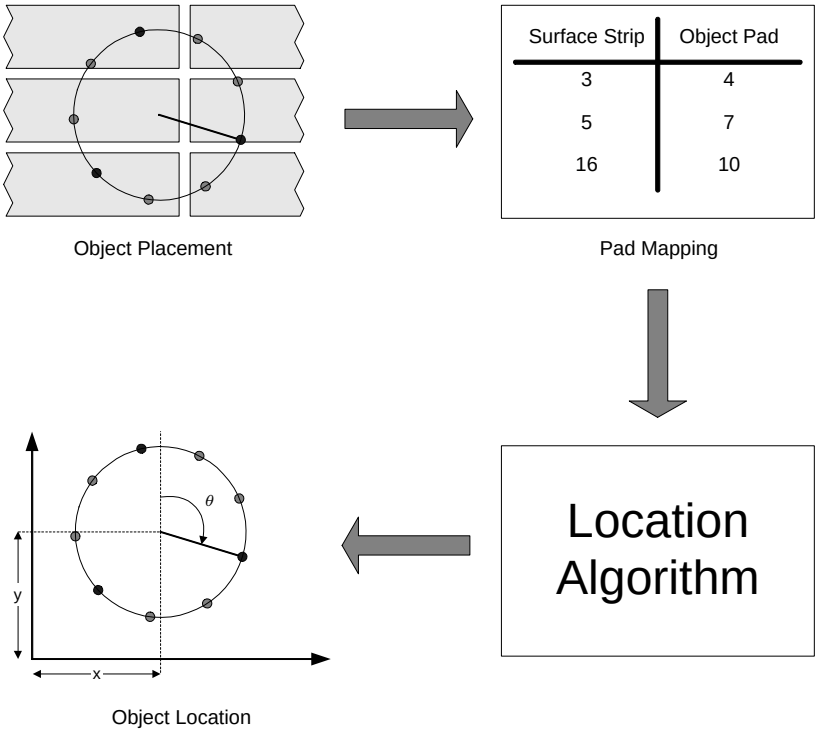

Figure 6.1: Object Location Process

of the device.

One inherent aspect of the location problem is that there is no right answer — for a given mapping of surface to object pads, there will be many possible values of x, y, and θ which result in valid placements. The location information can not therefore simply be provided as a single exact figure, and various means of reporting locations with known inaccuracies must be examined. One option is to simply report the range of the x, y and θ variables, and/or the midpoint of these ranges. Another option would be to use the median or mode of each variable. Yet another is to use the "inertial" centre for x and y, i.e. minimising the distance-squared between that centre and each other possible point.

Interestingly, the accuracy expected of each variable differs, both intrinsically, and based on the position of the object. For x and y, this is due to the rectangular surface pad layout (as presented in Chapter 2), with pads which are long in the x direction and short in the y direction. Therefore, a particular object pad to surface pad mapping gives a larger x range than y range. One example of the accuracy depending on position is found by looking at objects spanning one and two columns; in the latter case, the x dimension would be known more accurately.

6.2.3 Algorithm Design

In order to determine position and orientation from this data, many approaches can be taken. The three dimensions of x, y, and angle θ can be searched exhaustively in six different ways. Heuristics[1] can be used to narrow this search, but in general exhaustive searches will take $O(g^3p)$ time, where g is the grain of search used and p is the number of pad mappings provided.

Heuristics can also be used by themselves, to find locations in $O(p)$ time, where p is the number of pad mappings provided, for example by determining the bounds of the surface pads spanned by the object for position, and using the knowledge of which object pads have highest and lowest y values for orientation.

However, such heuristics do not make full use of the information provided, in the above example ignoring all pads which are not at x or y extremes, and therefore do not provide the most accurate information.

The chosen algorithm is designed to give the benefits of an exhaustive search, combined with optimisations reducing the computation required. For all θ (using some grain g), and for

[1]Heuristics are algorithms which try to estimate some quantity from a data set, without undertaking a full analysis of that data set. For example, a heuristic for the mean of a sorted set could be to use the middle item. They may be described as "shortcuts" or "rules-of-thumb."

each pad mapping p, ranges consisting of the minimum and maximum x and y are determined for the centre of the object, such that the object pad would be placed inside the appropriate tile pad. These ranges can then be combined to give, for that θ, an x range and y range for which all the object pads are placed on the appropriate surface pads.

By performing the range combination iteratively (i.e. updating the range after each pad mapping), an optimisation can be introduced whereby many θ are quickly dismissed; this is achieved by checking that the combined range is non-zero after each iteration. Pseudocode for this algorithm is shown in Figure 6.2, which allows location to be computed in $O(gp)$ time, as compared to $O(g^3p)$ for naïve exhaustive search.

For each θ which works (i.e. resulting in a valid x and y range for which all pads are placed correctly), statistics must be recorded so that the final x, y, and θ estimate can be worked out. The method of estimation used is that of a weighted mean for each variable. In addition, the maximum and minimum values for x, y, and θ are also recorded, to give an indication of the accuracy of the estimation.

When calculating the x weighted mean, each x is weighted by the x, y, and θ range for which a valid location is achieved, with y and θ weighted means calculated similarly. The computation of this statistic can be achieved in $O(1)$ time for each valid θ, thus maintaining the speed of the algorithm. This statistic minimises the average error in the location reading; to minimise the maximum error experienced, the midpoint of the valid range should be used instead (and since the ranges are also provided, this is easily computed if necessary).

It is important to note that the algorithm presented locates objects to the maximum accuracy possible given the pad mappings provided, and is not simply an model-based estimate. This is because the algorithm simply applies trigonometrical formulae to a known situation, thus making it as accurate as an algorithm which, for example, calculated the sum of two numbers. It is therefore not possible to improve this algorithm in terms of accuracy; no other procedure could output smaller ranges, or location estimates which are more precise in the average case. The only improvement possible, apart from speed optimisations, is to increase the grain g used to search θ. The value used for g in the prototype is 1000; this means that each iteration spans one third of a degree, which is negligible in human terms.

6.3 Evaluation and Improvement of Surface-Based Location

This section will evaluate the accuracy of the location algorithm presented above, using both experimentation and simulation. It will then explore ways in which the location accuracy can be increased, and show simulation results incorporating these improvements.

```
for(theta from 0 to 2*PI with granularity 'g') {
   minx = miny = 0;
   maxx = maxy = BIG_NUM;

   for(each object pad 'opad') {
      // determine vector from object centre to opad, for this theta
      padx = <distance in x dimension from object origin to opad>;
      pady = <distance in y dimension from object origin to opad>;

      // calculate x and y ranges for the object origin
      // such that the object pad is inside the appropriate tilepad
      padmaxx = tilepad[opad].maxx - padx;
      padminx = tilepad[opad].minx - padx;
      padmaxy = tilepad[opad].maxy - pady;
      padminy = tilepad[opad].miny - pady;

      // update possible range for object centre
      maxx = min(maxx, padmaxx);
      minx = max(minx, padminx);
      maxy = min(maxy, padmaxy);
      miny = max(miny, padminy);

      // eliminate this theta if the object centre is not locateable
      if(maxy < miny || maxx < minx) goto nexttheta;
   }

   // at this point, theta is known to work from minx:maxx, miny:maxy
   <record statistics>;

nexttheta:
}
```

Figure 6.2: Location Algorithm Psuedocode

6.3.1 Evaluation of the Location Accuracy using Experimentation

The most obvious means of evaluating the algorithm above is by simply measuring object locations, and comparing these measurements with the output of the algorithm. However, this approach incurs some disadvantages, in that it is time consuming, and difficult to measure the physical location with enough precision (i.e. down to the millimetre scale).

Experimentation is therefore used in a limited fashion, to verify that the location algorithm works in real-life cases. This was achieved in two ways, firstly by using a trial set of measurements with a particular object size, and secondly using a visualisation tool. In both cases, only the four-link object was tested, as this is the object used in the prototype. Location accuracies for other object sizes are explored later, using simulation.

Verification using Measurement

The task of measuring the location of an object on the Surface is not easy; the centre of the object footprint is obviously covered by that footprint, which has weights and wiring on top of it. However, since footprints are mounted on square circuit boards, easy-to-measure points can be found in the corners of those circuit boards. The tiles are also mounted on square circuit boards, so again the easiest points to measure are the corners.

A test program was therefore implemented, into which distances measured between surface corners and object corners may be input. Four such measurements are required; from each of two opposite object corners, two adjacent surface corners must be measured. This is illustrated in Figure 6.3.

Using this tool, tests were conducted according to the following criteria:

- The distance between the measured object corner locations was checked to be accurate to within 5mm, if not, the reading was discarded (as the measurement was inaccurate).
- 50 measurements (accurate to within 5mm as above) were taken.
- The ranges returned by the location algorithm were checked against the measured position.
- The differences between the measured and calculated values of x, y, and θ were recorded.

The recorded results are shown in Table 6.1, and indicate that the algorithm is accurate in terms of position to under 2cm in most cases. In addition, the measured location never fell outside the ranges specified by the location algorithm. These results will be compared later with simulation results.

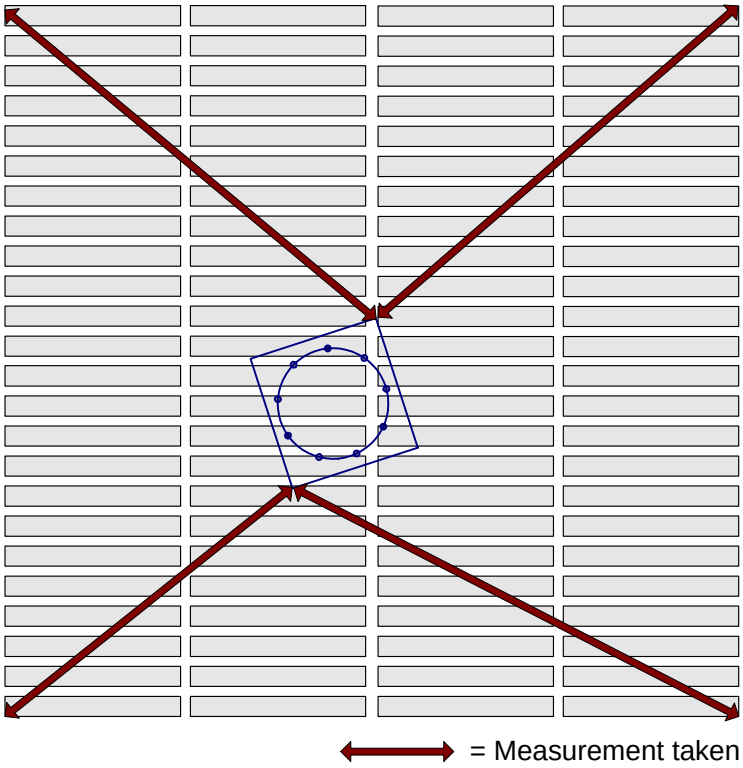

Figure 6.3: Method for Measuring Object Locations

Quantity Measured	Mean Error Found	Maximum Error Found
x	13mm	53mm
y	3.0mm	8.7mm
(x, y) vector	14mm	53mm
θ	6.3°	17°

Table 6.1: Results of Location Accuracy Measurements

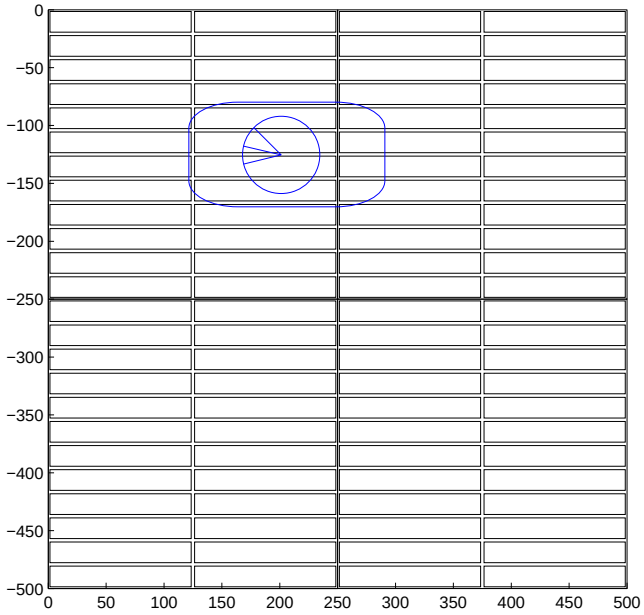

Figure 6.4: Graphical Location Depiction for a Location with Large Error Ranges

Verification using Visualisation

In addition to measurements, visualisation of object locations can be used to verify the correctness of the location algorithm. A visualisation tool for object location was therefore implemented, and proved useful in a number of ways.

The tool facilitates quick visual confirmation of the accuracy of the location algorithm. Moreover, by displaying the error ranges it also allows a user to easily notice that the ranges returned by this algorithm differ greatly from depending on where the object is placed. Example visualisations with large and small error ranges are shown in Figures 6.4 and 6.5 respectively.

These visualisation diagrams are organised as follows. The circle shows the estimated object location, and the rounded rectangle shows the bounds of that object (i.e. one can imagine the circle being moved around, so long as its edges do not cross the rectangle). The three lines show the estimated θ value and the range of possible θ values. It should be noted that the circle is not necessarily centred within the rectangle, and the middle line does not necessarily bisect the other two; this is due to the choice of weighted means as the "best estimate" location. A "skewed" visualisation is shown in Figure 6.6.

High error ranges were generally noticed when the object was situated on one column of surface pads, due to the much bigger range of x movement possible in this situation whilst

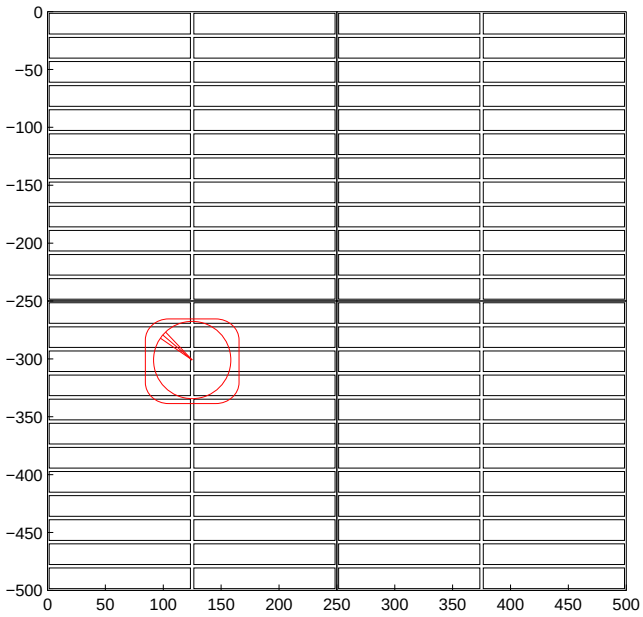

Figure 6.5: Graphical Location Depiction for a Location with Small Error Ranges

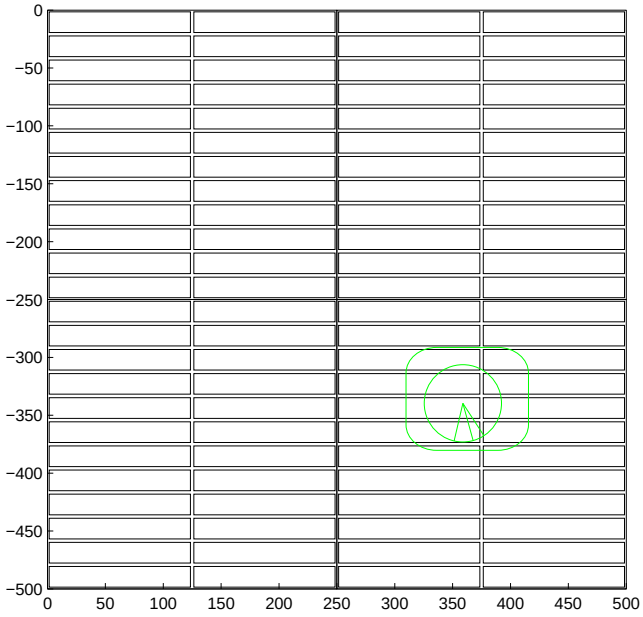

Figure 6.6: Graphical Location Depiction for a Location with Non-Centred Error Ranges

maintaining contact with the same set of pads. When two columns are spanned, the lowest error ranges are seen, due to the fact that stricter x limits are imposed due to the need to be in contact with pads in both columns.

Finally, the visualisation tool was used to check the reliability of the location algorithm in the real-life case. This was achieved by placing a four-link object randomly 100 times on the Surface, and conducting a visual verification that the object was placed as depicted using the tool. In no case was the actual location of the object outside the error ranges depicted by the visualisation.

6.3.2 Evaluation of Location Accuracy using Simulation

As pointed out previously, finding a precise measure of the location accuracy by experimentation is time consuming. Another method of evaluating this accuracy is to simulate object placement randomly across a Surface, and compare the results of the location algorithm against the simulated location. Using many such simulated trials, a mean error for position and location can be calculated.

The use of simulations has a number of advantages. Firstly, they are very fast, allowing many more trials to be conducted in a given period of time. Secondly, they do not incur the human measurement errors that experimental trials are subject to. Thirdly, since fast simulations can quickly perform many trials across various object locations, bugs in the location algorithm which only occur in "awkward" positions of the object may be found more readily. Fourthly, the simulation tool proved invaluable for testing and bug-fixing during the location algorithm's development. Finally, simulation is also useful to explore possible means of improving location accuracy, without having to implement and test changes in the prototype Surface itself.

Simulation Harness

"Simulation harness" is a term used to describe the software which simulates object placements in order to test the location algorithm. This harness must randomly choose an object location for each trial, and then construct a mapping between object pads and tile pads (including detection of when object pads fall into margins). To do this, a simulated surface of nine tiles in a square arrangement is set up, and the harness chooses random locations at any position on the centre tile. The surrounding tiles are required because although the object centre is inside the centre tile, its pads may fall outside the bounds of that tile.

Since an object footprint may cover more tile pads than its minimum guarantee (due to spanning two columns), a random sample of tile pads must then be made, e.g. if a four-link

object footprint spans six tile pads, a random sample of four of those tile pads must be made. For each of these tile pads, the object pad with the lowest pad number[2] is chosen; this mimics the behaviour of the prototype object hardware.

The result is a list of l tile pads and l object pads, where l is the number of links used by that object. This is then passed to the location algorithm described above, which returns estimates for x, y and θ, and also worst-case ranges for those quantities.

The simulation harness then performs three tasks. Firstly, it ensures that the random position is within the ranges specified by the location algorithm; this tests the algorithm implementation's correctness. Secondly, it records the error made by the algorithm in each of the three quantities provided, thereby allowing calculation of a mean error. Finally, the simulation harness keeps track of the maximum error experienced (in x, y, the (x, y) vector, and θ). This statistic is not normally used in characterisation of location systems; measures such as the 90% confidence level are more usually quoted. However, with the Networked Surface location algorithm, there is a maximum error that can occur for a given topology, in contrast to other types of location system where the maximum error is unbounded.

Interestingly, the maximum error is *not* minimised by this location algorithm, which instead minimises the mean error. The maximum error could be minimised by taking x, y, and θ values at the middle of their ranges, however, this approach would result in higher errors in the average case.

Simulation Results

The results of the simulation described above, using one million trials and for object footprints for two to six links, are shown in Figures 6.7 and 6.8.

As shown, the mean accuracy available depends highly on the number of links used by an object (and hence the number of pad mappings provided). This is especially true of the angular error, whose mean varies from over 35° to under 5° for two to six link objects respectively.

As predicted, the x error forms the majority of the mean positional error, being about five times the y error; this is also true for the maximum positional error. However, the maximum errors do not decrease as fast as the average errors as the number of available pad mappings grows. Indeed, the maximum y error is actually at its minimum in the two-link case. Note that, as previously mentioned, the location estimate returned is not designed to minimise the maximum error.

[2]Object pads are numbered anticlockwise from an arbitrary "zeroth" pad.

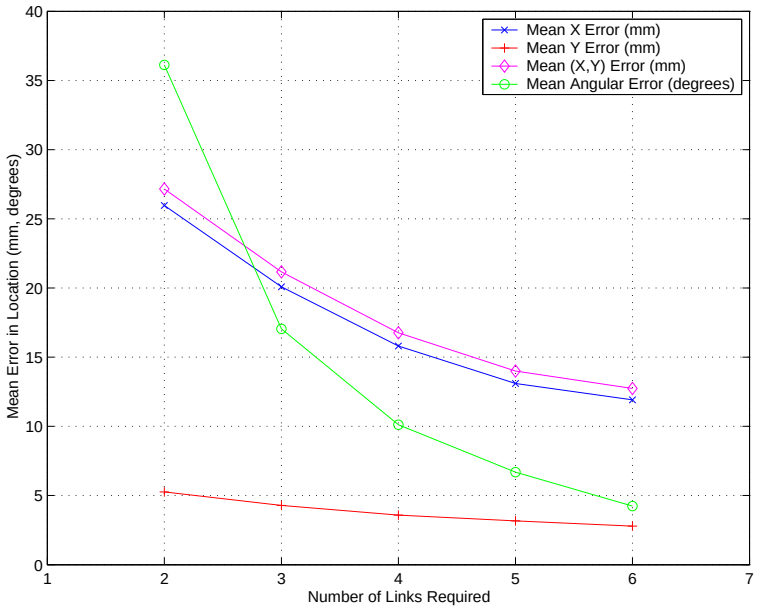

Figure 6.7: Mean Errors in Location Simulations

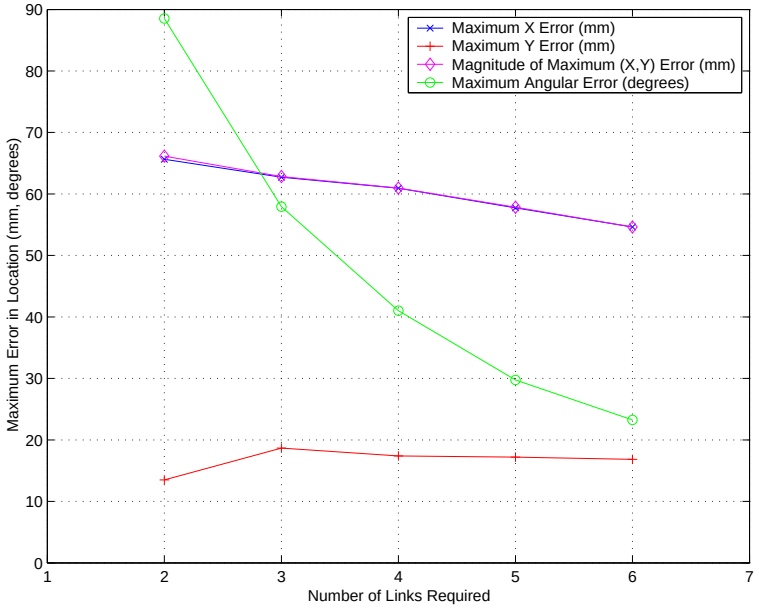

Figure 6.8: Maximum Errors in Location Simulations

Another result of the simulations was, for every one of the five million trials, the location algorithm included the real location in its returned range of possible locations. This demonstrates the correctness of the algorithm, and indicates its reliability.

The overall conclusions for location accuracy are that, for a typical four-link object such as those used for the networking tests outlined in previous chapters, the current Surface prototype will locate the object with an average error of 1.6cm, and at worst an error of 6.1cm. For orientation, the average error is 10°, and the worst case error is 41°.

Comparison of Simulated and Experimental Results

Table 6.2 shows a comparison between simulated and experimental results, for the case of a four-link object.

Quantity Measured	Mean Simulated Error	Mean Experimental Error	Difference
x	15mm	13mm	2mm
y	3.6mm	3.0mm	0.6mm
(x, y) vector	16mm	14mm	2mm
θ	7.8°	6.3°	1.5°

Table 6.2: Comparison of Simulated and Experimental Results

For all measurements, the experimental results provided slightly more accurate locations than the simulated results. This could be regarded as negligible when compared to the measurement errors of up to 5mm, and when considering that only 50 trials were conducted in the experimental tests, due to time limitations.

Another explanation for this difference can be found in an assumption made by the simulation harness. Specifically, when more tile pads are available for an object than required, the simulation makes a random selection. In reality, the prototype is more likely to pick some configurations than others, depending on the tile beaconing order. Due to the particular beaconing order used,[3] whenever an object spans more than one column of tile pads, it is very likely that pads from both columns will be chosen. As the visualisation experiments showed, location estimates made using tile pads from two columns are more accurate than when only one column is used.

Irrespective of which of the explanation(s) above is true, the main conclusion to be drawn from this comparison is that the experimental results broadly agree with the simulation

[3]See Figure 3.5.

results. This provides an assurance that the simulation is realistic.

6.3.3 Improving the Location Accuracy

As noted previously, the location algorithm used should give the most accurate location that is possible given the pad information provided, because of the inherent limit on accuracy imposed by the topology used. The implementation of the algorithm can introduce inaccuracies, namely in the grain of θ increment used and in the numeric precision, but these are made negligible by using a fine θ grain and high-precision floating point representations.

The limits of precision found above therefore are limits due to two factors, the pad information provided and the topology used. This section will discuss the former of these improvements only, the latter is considered outside the scope of this thesis. Two possible methods of increasing the pad information are discussed below, and the results of simulations of these improvements are shown.

Improving Accuracy using Further Input Data

The current prototype restricts input data to the location algorithm in two ways. Firstly, only data on the tile pads necessary to make a connection is revealed. Secondly, only data on a single object pad for each of these tile pads is provided.

In the first case, it is possible, even likely, that many object pads span each surface pad that is in use. The current prototype always chooses the lowest-numbered object pad, and does not record information on the other pads; this behaviour is emulated by the simulation harness described above. The controller firmware responsible for this behaviour could be modified to note when multiple object pads span a single surface pad, by monitoring the pads during handshaking to see which sets of pads exhibit identical binary signals. The use of this data is known henceforth as the "duplicate pads" optimisation.

For the second case above, it is noted that in many cases object footprints make contact with more tile pads than they are designed to guarantee. For example, a four-link object placed across two tile pad columns may in fact make contact with eight tile pads. Information on pad mappings involving the spare tile pads could be gathered to further enhance location accuracy. In order to get this data, the controller firmware could be modified to note the tile pad information in the beacons it receives, even when it already has enough links reserved. This will only work if another object is not using the "spare" pads. This technique is known henceforth as the "all links" optimisation.

Both of these optimisations are feasible, and only require modification of the object firmware. However, it is simpler just to simulate this modification, by modifying the simu-

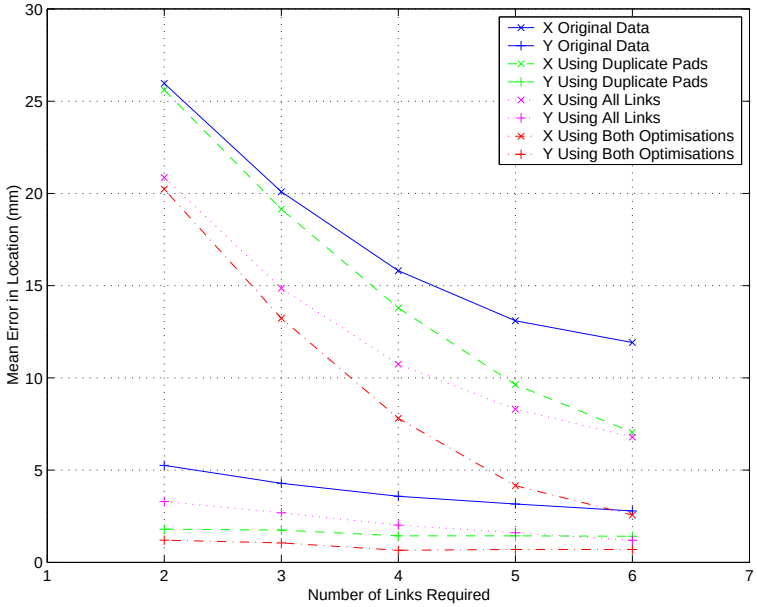

Figure 6.9: Mean X and Y Positional Errors in Location Simulations using Improved Data

lation harness to mimic either or both of these cases.

Simulated Results Incorporating Further Input Data

The results from simulations using improved input data are shown in Figures 6.9 to 6.14. An analysis of the effectiveness of the optimisations shown on these graphs is now presented.

In Figure 6.9, the behaviour of the optimisations for x can be explained by noting that as the number of links required grows, the object footprint grows. For low footprint size, the all links optimisation provides good x accuracy when two columns are spanned, as the small footprint is localised over the column boundary. As the footprint size approaches the tile pad width, the x accuracy is helped more by knowledge of the duplicate object pads on a single tile pad (which span nearly the full width of that pad) rather than knowledge of more tile pads.

In the case of the y variable, the opposite is true. With low object size, the chance of spanning two columns, and therefore of gaining more data with the all links optimisation, is low, so the duplicate pads provide, in general, more data localising the y coordinate. As the objects get bigger, the chance of spanning two columns is increased, so there is a higher likelihood of gaining more data due to the all links optimisation.

Figure 6.11 shows the duplicate pads optimisation to be more effective for θ; this can be

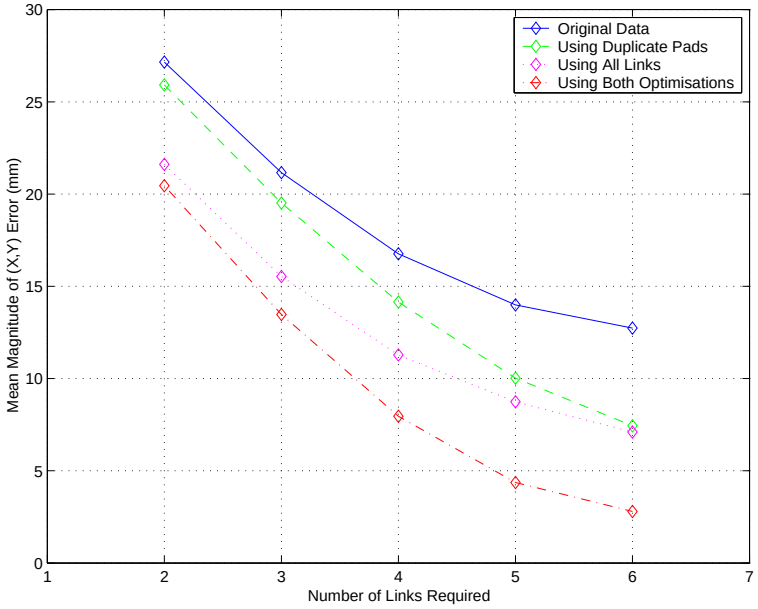

Figure 6.10: Mean Magnitude of Positional Error in Location Simulations using Improved Data

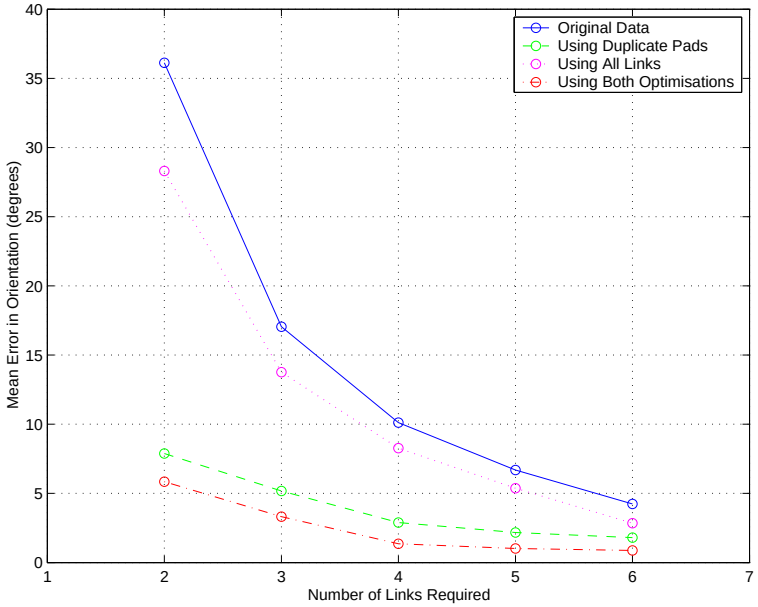

Figure 6.11: Mean Orientation Error in Location Simulations using Improved Data

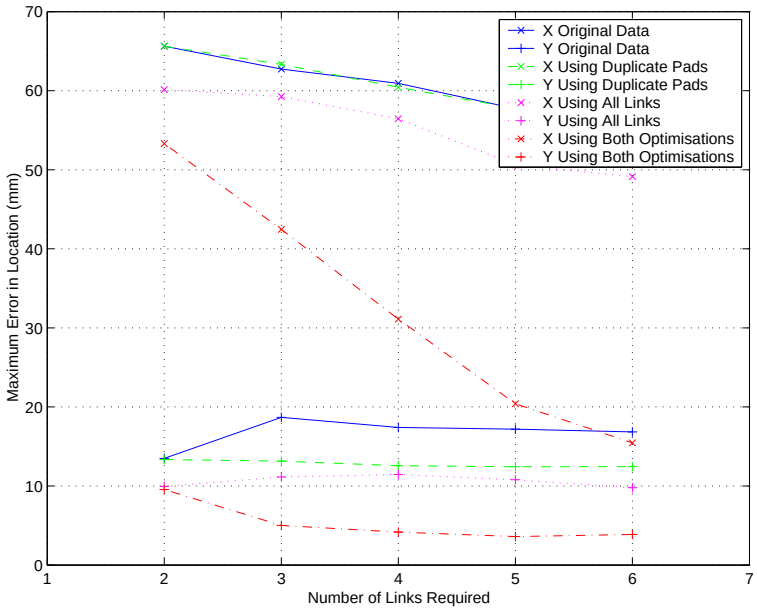

Figure 6.12: Maximum X and Y Positional Errors in Location Simulations using Improved Data

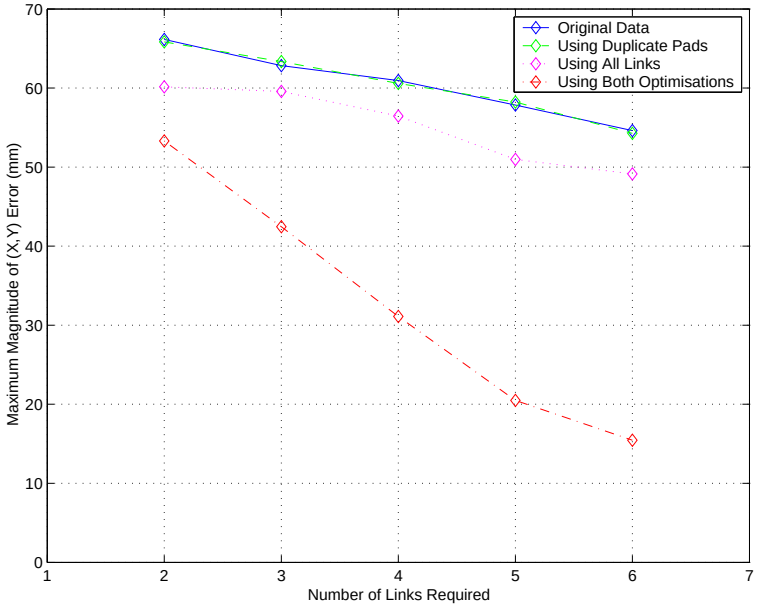

Figure 6.13: Maximum Magnitude of Positional Error in Location Simulations using Improved Data

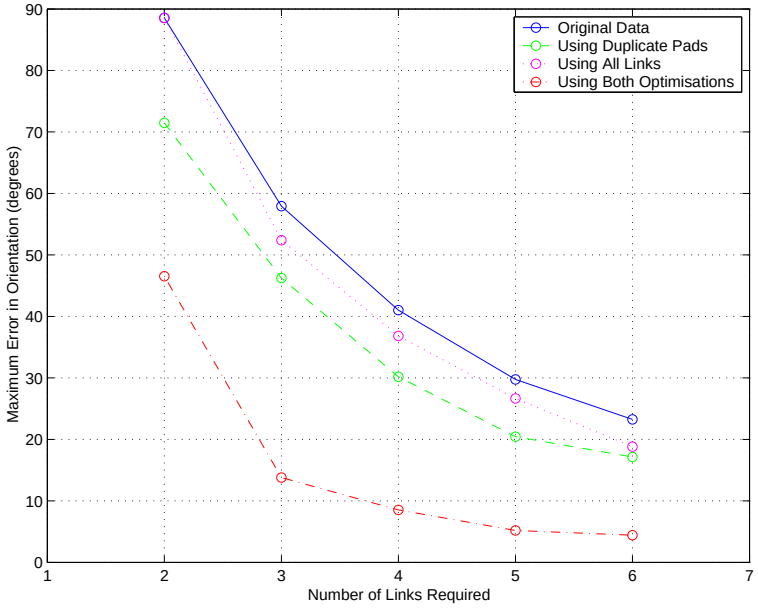

Figure 6.14: Maximum Orientation Error in Location Simulations using Improved Data

explained by simply noting that more additional pad mappings are provided in the duplicate pads optimisation. This is particularly true of cases where no extra links can be found, and for these cases the duplicate pads data is likely to be particularly plentiful (as all the object's pads are divided amongst a smaller number of tile pads).

Importantly, it can be seen that both optimisations independently improve mean accuracy in all four variables, and when using both optimisations the improvements made are approximately additive. This is because the optimisations are highly independent, as one only operates on tile pads already in use, whereas the other refers to unused tile pads only.

For the maximum positional errors shown in Figures 6.12 and 6.13, the duplicate pads optimisation is seen to give no improvement. This shows that the worst cases for the unmodified algorithm and for the duplicate pads optimisation coincide. It is conjectured (though not proven) that these worst case errors happen when an object spans two columns, but only pads in one of the columns are used; in this case, the location estimate would have a large x error.

Interestingly, for maximum errors in all but the y variable case, the use of both optimisations greatly (i.e. more than additively) increases accuracy. This can be explained by noting again the independent nature of the optimisations, which therefore have different worst-case scenarios. Each optimisation's worst case is therefore made better when the other

optimisation is brought into play, hence greatly reducing worst-case errors when both are used.

Finally, the example of the four link object used above is revisited. When using both optimisations, the average precision of the location data is improved from 1.6cm to 0.8cm, with the worst case error reduced from 6.1cm to 3.1cm. For orientation, the average error is improved from 10° to 1.4°, and the worst case from 41° to 8.5°. This level of accuracy makes the Surface suitable for providing precise position and orientation data to many applications, as described later in the chapter.

6.3.4 Algorithm Speed

To conclude the discussion of the location algorithm, some experiments were undertaken to show that the implementation constructed is capable of performing in real-time. The parameters of the experiments are shown below, and the results are shown in Table 6.3.

- A 1.4GHz Athlon PC was used in this experiment.
- Trials were conducted with the θ grain g set to 1000 (the default), and 10000.
- The simulation harness described was used to test 1000 random locations. This means that the results below are conservative, as they include the time taken by the harness.
- Times were recorded for the cases of the lowest number of pad mappings (a two-link object, with no optimisations used), and the highest number of pad mappings (a six-link object, with both optimisations used).

Number of θ grains g	Average Location Time (Few Pad Mappings)	Average Location Time (Many Pad Mappings)
1000	0.91ms	2.0ms
10000	9.2ms	20ms

Table 6.3: Speed of Location Algorithm

As the results show, the location algorithm is fast enough to be useful in providing real-time location information on Networked Surfaces, even if the grain g is increased tenfold from the current value used. The increase in processing time appears to be linear in g, which agrees with the prediction of $O(gp)$ made for the algorithm's performance.

6.4 Location Systems Survey

This section will discuss other systems measuring position and orientation, with precisions ranging from metres down do millimetres. The systems will be grouped according to the medium they use to locate devices, and only systems useful in an indoor environment will be discussed. The section concludes with a comparison of these systems and the Networked Surface.

6.4.1 Infrared-Based Systems

The first computer-based location systems were achieved using infrared. These systems have the advantage of being simple, but they are inherently limited to room-based accuracies.

The Active Badge system [104] involves tagging people and equipment with battery-powered transmitters, which periodically (every fifteen seconds for people) transmit short infrared beacons. Receivers are placed at the top of walls, typically using two in opposite corners of a small room. The system was the first of its kind.

The Locust Swarm [98] system developed at MIT is another infrared-based system, but with an alternative emphasis; in this system the beaconing transmitters are located in the environment, and the receivers are located on the tagged objects. This configuration allows for privacy guarantees to be made; objects can locate themselves without any central body knowing their location. However, this makes it more difficult for centralised services to be constructed.

In both of the systems above, there is also the possibility of reverse information flow so that data other than beacons can be exchanged, such as location-dependent messages, or to provide a very-low-rate network. Both systems were designed primarily for location, however, and as such only support the bandwidth necessary for that purpose.

The Smart Badge system [17], takes the concept of "badges" further. Firstly, encryption is added to ensure privacy, while making location data available to authorised central servers. Secondly, a number of sensors are included in each badge, so that other types of context information can be provided in addition to location. These sensors detect levels of light, temperature, humidity, tilt, and sound. Finally, "intelligent" badges are also presented, which are further enhanced to include graphical user interfaces, facilitating applications which are not possible with simpler badges.

The ParcTab system [95] moves away from a "badge" paradigm, by looking at how PDAs can be enhanced to include location-aware applications. The location technology used is again infrared-based, and therefore room-scale. The networking possibilities of infrared

are emphasised, with a bandwidth of 20kbit/s provided.

6.4.2 Ultrasound-Based Systems

Ultrasound-based systems can achieve a higher grain of accuracy than infrared systems, and therefore have been used in more recent location-discovery research efforts.

The Active Bat system [105] provides location information on tagged objects to a resolution of 3cm, using multilateration of four or more distance measurements. Like the Active Badge, transmitters are placed on objects and receivers are placed on the ceiling, the latter in a grid with approximately one metre spacing. The system is baseband, and each transmitter is triggered one-at-a-time by a radio signal. In addition, the system is capable of detecting the orientation of the object, either by relying on the object itself to obscure the signal so that only particular receivers can pick it up, or by placing multiple transmitters on an object.

The Dolphin system [45] is another ultrasonic location system, using broadband transducers. A prototype of this system displays an accuracy of around 2cm. The use of broadband ultrasound allows multiple transmitters to send simultaneously, and provides resilience against ultrasonic noise.

The Cricket system [87] uses ultrasound with radio triggering, but like the Locust Swarm, the transmitters are ceiling-based, and mobile objects use receivers to locate themselves. The system is much less accurate than the Bat system, aiming only to locate devices within about 2m, but it does enjoy the properties of privacy and decentralisation. The system has been since been enhanced [88] to calculate orientation using multiple receivers, with an accuracy of 3°.

Finally, the commercially developed Constellation [40] system uses a hybrid of ultrasound and inertial sensors to achieve a location accuracy of under 1cm. Ultrasonic transmitters are placed in the ceiling, and users wear helmets equipped with three receivers. Inertial measurements allow location information to be kept up-to-date in between ultrasonic readings.

6.4.3 Physical Sensor Systems

The Active Floor [5] explores the use of physical force sensors to enable computers to monitor an environment. A surprisingly small number of sensors in a floor is shown to give the ability to identify a user by their gait pattern, using hidden Markov modelling. While this research did not focus on locating the user, the system is inherently capable of such location with resolution of at most one floor tile, which in this system is a 50cm square. The downfall of the system is that it requires an object to be perambulating in order to recognise it (i.e.

making repeated movements of its weight on the floor). While suitable for people, this would not be applicable to equipment.

The Smart Floor [78] is also based on ground reaction force measurements, with up to 93% identification accuracy achieved. Unfortunately no position accuracies are discussed, however, the tiles are also of size 50cm square. Both of these systems have the advantage of not requiring any form of tagging, although they do have to be trained to recognise users.

6.4.4 Camera-Based Systems

The TRIP System [66] uses cameras to recognise circular barcodes, identifying both the pattern on the barcode and the location and orientation with respect to the camera. It has the advantage of using very cheap tags (printed paper), but tags are also easily obscured. The system is fairly accurate, with errors of about 3cm and 1° at a range of 150cm.

The Matrix object tagging method [90] is based on 2D grid barcodes, providing 2^{16} possible identifiers. Unfortunately, accuracy is not discussed, but the applications of the system (discussed later) and its similarities to the TRIP system above indicate that accuracies similar to TRIP are attained.

The Pfinder system [108] uses cameras to directly track users. The system tries to locate users' hands and arms; this information is then used in applications such as gesture recognition. It detects a new user by assuming that movement in the camera's view is due to a person, and creating a "blob" model of them. It does not try to identify the user, and cannot keep track of more than one person. The accuracy is dependent on the distance of the person from the camera. At 150cm range, the system has accuracy of around 1cm for position, and 4° for orientation.

The Easyliving project's person tracking system [62] detects location by using stereo cameras to track individuals as they walk around a room. The system is not intended to identify users (though identification is provided for in other ways), but colour histograms are used to keep track of multiple users separately. Multiple cameras are used to make all portions of a room visible. The authors claim 10cm accuracy along the 2D plane of the floor.

6.4.5 Radio-Based Systems

Radio-based location detection is intrinsically hard, as it is not easy to detect distance from a radio signal. This is in contrast to sound (which travels slowly), and infrared (which does not pass through walls).

The 3D-iD system [107] uses active RFID tags. Antennas throughout the environment beacon periodically, causing the tags to generate a response including their ID code. The

system then calculates times-of-flight between at least three antennas and each tag, and thereby estimates the 3D location of the tag, with an accuracy of 1m.

Another solution, adopted by Want et al. [103], uses an infrared-based system to give room-level location, and a very-low-range RFID system to let devices know what is close to them within that room. This results in a mix of very-local information as well as coarse-grain information, but does not specify the position within the room of the locally-detected devices, or their relative orientations. The accuracy is therefore hard to characterise; for instance, object A might know that it was in the same room as objects B and C, and that it was near B but not near C.

A completely different approach to radio-based location is to use the signal strength from multiple fixed transceivers to locate a device. The RADAR system [11] does this using signal strength data provided by a WaveLAN network with multiple base stations, achieving a grain of about 3m. However, this is highly dependent on the setup of the network and of the device, requiring considerable configuration, and the error level of 3m is not small enough to guarantee that the device will be located in the correct room. One key advantage of this system is that, given that there is a wireless network already installed, there is very low additional overhead. This is analogous to the use of the Networked Surface for location information.

The Nibble system [25] also uses WaveLAN signal strength, but with the prototype system having far more wireless access points (10–14 in one experiment). They were therefore able to achieve high accuracies for room-grain location, with the correct room identified up to 97% of the time.

Another method of using wireless networks for location is simply to note which base station the mobile host is using. This provides a coarse grain of location, since the device may be anywhere in the range of the base station. However, location systems built in this way are very simple, requiring very little additional work once the wireless network is deployed.

One example of this technique is found in the GUIDE project [28], in which tourists visiting the city of Lancaster are provided with location-specific information, e.g. data concerning a tourist attraction. The range of a WaveLAN is quoted as 200m in free space, but lower when used in indoor environments.

Another example of a base station–based location system is described in [60]. In this project a grid of Bluetooth transceiver nodes was installed in a trade show hall, in order to provide networking capability throughout the building. The system was also used to track the position of devices, and thereby to provide location dependant services, such as an active map. The location accuracy achieved in this experiment was 10–20m.

6.4.6 Comparison of Location Systems

Table 6.4 summarises the different location technologies discussed above, and compares them against the properties of Networked Surface–based location.

System Name	Medium Used	Typical Accuracy	Orientation Provided	Networking Provided
Active Badge	Infrared	Room-Grain	No	Limited
Locust Swarm	Infrared	Room-Grain	No	Limited
Smart Badge	Infrared	Room-Grain	"Tilt" only	No
ParcTab	Infrared	Room-Grain	No	20kbit/s
Active Bat	Ultrasonic	3cm	Yes	No
Dolphin	Ultrasonic	2cm	Yes	No
Cricket	Ultrasonic	2m	Yes	No
Constellation	Ultrasonic/Inertial	1cm	Yes	No
Active Floor	Force Sensor	50cm	No	No
Smart Floor	Force Sensor	50cm	No	No
TRIP	Vision	3cm	Yes	No
Matrix	Vision	Unknown	Yes	No
Pfinder	Vision	4cm	Yes	No
EasyLiving	Vision	10cm	Yes	No
3D-iD	Active RFID Tagging	1m	No	No
[Want99]	Infrared/Low-Range Radio Hybrid	Room-Grain & 10cm	No	No
RADAR	WaveLAN Signal Strength	3m	No	2Mbit/s
Nibble	WaveLAN Signal Strength	Room-Grain	No	11Mbit/s
GUIDE	Nearest WaveLAN Base Station	up to 200m	No	11Mbit/s
Bluetooth-Based	Nearest Bluetooth Base Station	10–20m	No	1Mbit/s
Networked Surface	Electronic	1cm	Yes	5Mbit/s

Table 6.4: Comparison of Indoor Location Technologies

As the table shows, the Networked Surface provides one of the most accurate location services available, and is the only system to provide both orientation and networking as well. The disadvantages of Networked Surface–based location lie in extensive hardware requirements, and the fact that only objects augmented with Networked Surface hardware can be located. This latter constraint can be avoided by use of another location technology in tandem with Surfaces; this possibility is discussed in a later section.

6.5 Context-Aware Computing

The computer-based location systems discussed in the previous section form a branch of a broader field, namely "context-aware computing"[4], which looks at the use of context information, such as location information, in applications. This field is closely tied to "ubiquitous computing" [3, 106].

This section describes a subset of context-aware computing applications, specifically those making use of location information. Later sections discuss how the Networked Surface can contribute to this field.

6.5.1 Context-Aware Middleware

While some projects focus on implementing applications for specific location systems, it also makes sense to look at location systems as application-independent, and vice versa. This can be accomplished using "context-aware middleware."

Such middleware can perform the functions of interpreting and collating context information, possibly from many context-providing systems, and can also distribute this information to applications. For Networked Surfaces, only middleware which is capable of dealing with fine-grain location is of interest, and so only these systems will be discussed.

One such framework, the SPIRIT project [6], collates context information from Active Badges and Active Bats (see above) into a single resource. There is a basic notion of conflict-resolution (Bat readings always overrule Badge readings). Applications developed have included detailed visualisations of workspaces, desktop "teleporting," nearest-phone call redirection, and augmented reality applications.

Another such framework is included in the QoS DREAM project [75], which looks at the provision of QoS in reconfigurable multimedia applications. The relevance of location information in this case is to allow redirection of multimedia streams according to a user's location. This system currently uses the Active Badge as a location technology.

One feature of the QoS DREAM system is its event-driven architecture, which has been used to implement event filtering. This allows applications to register for notification when specific conditions are met, thereby reducing the amount of context data that must be sent to applications, and making context-aware applications simpler to implement.

Leonhardt [64] proposes a system for multi-sensor location tracking, which tackles the issue of combining inputs from many types of location sensors into a single coherent location

[4] "Context-aware computing" is also known as "sentient computing" [50]

database, including methods for resolving inconsistencies. Location information is provided by Active Badges, GPS, and workstation logins.

Maass [68] presents an architecture for supporting location-aware applications using the LDAP directory protocol. Active Badge data is used for location information in the prototype. Event-based querying is also supported.

Other contributions to the development of context-aware applications have included software toolkits facilitating their implementation [94, 31].

6.5.2 Directory and Visualisation Applications

The most obvious application of location information is simply displaying it to interested users, as a sort of real-time directory [43]. This was extended to the use of graphical visualisations of office spaces [72, 96].

Another application developed with the Active Badge system is the tagging of equipment as well as people. This allows equipment movements to be tracked, so misplaced or borrowed equipment can quickly be found when required. Finally, with the advent of fine-grain location systems, more detailed visualisation applications are possible, including full 3D models [4].

6.5.3 Follow-Me Applications

A large category of location-aware applications can be classified as "follow-me" applications, in which user mobility is made more convenient by the automatic movement of applications and forwarding of services as a user moves from place to place.

One such application was developed with the Active Badge system, and is known as "teleporting" [92], whereby a user's computer desktop(s) can be displayed on the closest computer to them at all times, activated using buttons on their badge. This was later extended to multiple platforms with the use of Virtual Network Computing (VNC) [93].

With the fine-grain location afforded by the Active Bat, this system was further upgraded so that information on the precise position and orientation of users and equipment could be used [44]. With such information, it is possible to infer the need for teleporting by noting when users stand in front of and face suitable equipment.

Finally, the Active Bat system has been used to implement a functioning nearest-phone call forwarding system. When an incoming call is received and the user is not in their normal office, the user's Bat is made to beep. They can then press the Bat button to indicate acceptance of the call, which will be routed to the closest phone. The notion of "closest" takes into account not just distance, but also obstacles and orientation; the nearest phone

may not always be the most easily accessible, as it may be on the other side of a partition or desk.

6.5.4 Location-Dependent Information

The Stick-E Note project [22, 81] proposed location-dependent information be distributed by use of the metaphor of virtual "Post-It" notes, attached in the memory of computers to specific locations. The application discussed was an information guide for use in theme parks such as Disney World. In addition, notes could be attached not only to locations but to other contextual cues such as the users present, the temperature, and so on.

The Cyberguide project [2] has developed a mobile context-aware tour guide, which uses GPS outdoors and a custom infrared-based system indoors for positioning. A PDA is used to present a tour guide application, and displays information based on the location detected. Specific applications developed include a tour guide around a building of the Georgia Institute of Technology, and a bar/restaurant guide for Atlanta.

The GUIDE project [28] has also implemented a handheld context-aware guide, aimed specifically at tourists. Coarse location information is provided by simply detecting which WaveLAN base station is closest; the WaveLAN is also used to transfer location-dependent information, which may be dynamically updated.

Sumi and Mase [100] have trialled a context-aware digital assistant in the role of supporting participants of a conference. Location was provided both by infrared, and by RFID tagging. Applications included an "site map," including timely information on ongoing presentations, and the people attending them.

Gellersen [41] generalises these concepts as "environment mediated" computing, and proposes software frameworks whereby interactions between humans via computers are mediated by some entity representing the environment in which the interactions take place. This can be used to accomplish location-based communications (e.g. "This copier is broken"), or object-based communications, (e.g. virtual annotations to books).

6.5.5 Intelligent Environments

One of the original works on intelligent environments is the Digital Desk project [77], in which cameras and projectors are used to make a physical desk act in many ways like an active computer terminal. An early application was a simple calculator, into which numbers could be input from paper documents, using OCR techniques for their extraction. This work is one of the first attempts to make surfaces respond intelligently.

The Augmented Surfaces project [91] expands on this idea, with both tables and walls used as active workspaces. The Matrix 2D barcode system described previously is used for object recognition and location. Location awareness allows users to indicate particular devices, facilitating convenient transfer of documents between these devices.

Other early work by Elrod [33] looks into climate controlled environments using location data to automatically turn off air conditioners and lights when people are absent. This work precedes a whole range of work into intelligent and responsive environments.

Another project, MUSICFX [71], examines the use of stored preferences of a group of people to decide on the music played in a fitness centre environment. The location technology used is very simple; a swipe card reader is used to determine which users are present.

The Aware Home project [59] intends to explore ubiquitous computing and context-aware issues using a custom-built house. For person location sensing, the Smart Floor is used in ten places to track users movements. Also, RFID tags are attached to some objects, in particular frequently lost objects, thus aiding in finding those objects easily.

The EasyLiving project [23] also looks into a home environment, using cameras to track people. Applications implemented include automatic redirection of audio, aggregation of I/O devices to form a single coherent experience, automatic environmental control, and teleporting of user interfaces as the user moves.

Yan and Selker [109] have presented a context-aware "office assistant," which acts as an automatic receptionist, using voice synthesis and recognition to communicate with visitors to an office. The occupancy level of the office is tracked using pressure-sensitive mats at the office door, detecting ingress and egress.

Finally, the concept of real-world interfaces is taken a step further when fine-grain location systems are used. With the Active Bat system described earlier, a number of applications are outlined involving the designation of particular regions of the environment as interfaces. [4] Examples of this are the use of a Bat as a 3D mouse when near a large LCD display, and also the use of "smart posters" in the environment with "virtual buttons" on them; by moving a Bat next to the virtual button and pressing a (real) button on the Bat, actions can be initiated.

6.6 Networked Surfaces and Existing Context-Aware Systems

This section will look at possible integration between the Networked Surface and other context-aware computing systems. First, the complementary use of Surfaces and other location technologies is discussed, followed by the integration of Surfaces as a location technology in two location-aware middleware platforms. Context-aware computing applications using

Networked Surfaces are discussed in the next section.

6.6.1 Other Location Systems

The Networked Surface provides location data for very specific object types, namely, electronic devices which are commonly placed on Surfaces. Many classes of device are not located, either because they are unsuitable for Networked Surfaces, or because they are at that moment not placed on a Surface.

It is therefore possible to contemplate use of other location systems in tandem with Networked Surfaces, either for identifying devices which are not surface-based, or for keeping track of Networked Surface objects even when they are not on a Surface.

TRIP

One particularly suitable system might be the TRIP system described above, which is a camera-based location system using 2D barcodes for recognition. This system can provide position with errors of around 3cm and 1°, which is comparable to the Networked Surface. However, for an object to be located, very different criteria apply; the object must be within range, roughly facing the camera, and tagged. Another difference is the cost — TRIP tags are simply paper barcodes, and hence very cheaply produced.

The TRIP and Networked Surface systems can therefore be considered complementary. On one hand, the TRIP system can locate non-electronic objects and objects which do not sit on Surfaces, and hence provide a Surface implementation with a means of cheaply gathering location information about a wider range of objects. On the other hand, a Networked Surface can aid a TRIP system in a number of ways. Firstly, the TRIP system requires networked cameras; these cameras could use the Networked Surface as a wire-free network which could also provide power. Secondly, the TRIP system requires the location of its cameras to be statically determined. Using Surfaces, cameras could be located automatically, and furthermore could be moved around without the need to re-configure the system, allowing a set of cameras to cover different regions at different times.

Systems Tracking Users

Many location systems are inherently designed for the tracking of users (for example the Active Badge and Active Bat systems), with the tracking of equipment being regarded as a useful side-effect. The Networked Surface, on the other hand, cannot track users except indirectly through methods such as analysis of connections and disconnections, and moni-

toring of a user's "virtual" presence (i.e. where they are logged in, or which of their devices is receiving user input).

This difference means that it may be sensible to use both the Surface to locate electronic objects, and a human-centric tracking system for users. One advantage of using both systems would be that electronic devices do not need location-finding hardware, as their networking hardware automatically provides location. Another advantage would be that both users and equipment can be tracked, and the availability of both types of location may enable more applications.

Looking more closely at one example, the Active Bat, one can find further advantages. Firstly, for devices which are commonly left on desks, a 3D location sensor can be considered "overkill," as the height is already known. Also, the Bat system is limited in the number of locations it can acquire over a period of time; if equipment could use other location methods such as the Surface, then the location of users could be refreshed more often.

6.6.2 Context-Aware Middleware

As described above, location providers and location-aware applications are often connected using middleware. It is therefore useful to consider how to implement data transfer between the Networked Surface and some context-aware middleware systems. Actual implementation of these interconnections is considered outside the scope of this thesis.

Discussed below are possible interconnections between the Surface and two location-aware middleware systems, namely the SPIRIT and the QoS DREAM projects mentioned previously. These were chosen due to accessibility; both projects have a presence at the LCE, making such interconnections feasible in the future.

Integration with SPIRIT

The SPIRIT system is built using CORBA interfaces. Linking the Surface with this system would therefore be achieved by using CORBA to access the necessary interfaces.

One difference between the Surface and many other location systems is that the Surface intrinsically provides events, namely the connection and disconnection of objects. Other systems (such as the Badge and Bat systems already being used in SPIRIT) provide periodic messages concerning object location, since these systems must continually poll objects to keep location information up-to-date.[5] Some "glue" software may therefore be required to

[5]With the aid of inertial sensors, the Bat system has been optimised to not poll stationary objects.

support the use of event-style notification in SPIRIT, in which a "connect" event provides a location that is guaranteed to be accurate until (just before) a "disconnect" event.

One advantage of integration with the SPIRIT system is that many applications have already been developed using this platform, as described earlier.

Integration with QoS DREAM

The QoS DREAM project is built using events as the basic methodology. This makes integration with the Networked Surface simple, in that Networked Surface connection and disconnections map easily onto QoS DREAM events.

6.7 Networked Surfaces and Context-Aware Applications

The provision of location information alone does not fully cover the potential contribution of Networked Surfaces to context-aware computing. This section will discuss context-aware computing applications which not only rely on Surfaces for location information, but also use Surfaces in other ways. Two such classes of application are discussed, namely, auto-configuration of Surface-based devices, and ubiquitous interfaces using Surfaces.

6.7.1 Auto-Configuration of Networked Surface Devices

Using location information, Networked Surfaces can automatically configure peripherals and computers. This may be achieved by connecting them to particular networks, or by notifying devices of other devices that are present. This allows users to indicate associations between devices by simply placing them in a particular fashion, e.g. in close proximity, or at particular orientations with respect to one another.

The use of physical placement to indicate associations is intuitive, and (as explained below) from the user's point of view it would appear that devices have acquired extra functionality. This is consistent with the aims of context-aware computing, and provides devices with context-aware functionality without requiring internal modification of those devices.

Non-Networked Peripherals

The simplest auto-configuration application of Surfaces involves non-networked peripheral devices. Devices in this category, such as keyboards or joysticks, expect to be connected directly to a computer.

The Networked Surface could accomplish this in a number of ways. Firstly, a peripheral bus such as USB could be implemented on the Surface, to which a computer and its associated

peripherals could be connected. By implementing a number of peripheral buses, the Surface could support multiple computers, each using their own sets of peripherals.

Another method would be to have the surface manager act as a proxy master for the slave peripheral devices. The computers and their peripherals would then be connected using tunneling of data over an inter-computer network, which may itself be a bus on the Surface. The peripherals would be represented on their respective computers as "virtual devices," which would be implemented as small "skeleton" programs that redirect reads and writes via the surface manager. The advantage of this solution is that only one peripheral bus needs to be implemented on the Surface; for low-bandwidth peripherals such as keyboards, this would be possible without running out of bus bandwidth.

In either case, position and orientation data concerning the computers and peripherals can be used to automatically decide which peripherals should be slaved to which computers. In the former case, the surface must decide which peripheral bus a given device should be placed on. In the latter case, the decision concerns the choice of computer on which the "virtual" device should be created. Note that, for this application, orientation data may be of particular value, as it is possible for an object to be placed so that it is roughly equidistant from two computers, but not possible that it can "face" both computers simultaneously.

Networked Peripherals

Networked peripherals such as networked printers can also make use of auto-configuration. The use of Surfaces could greatly simplify the installation process for such devices. For example, the addition of a new printer to an an office environment would normally require the setting up of networking and power cabling, and then require various computers close to that printer to be manually configured to use that printer.

With a Networked Surface, the printer could automatically be networked and powered, and print jobs might automatically be routed to it from nearby workstations. If a printer were to be offline for some reason, automatic re-routing of jobs might occur to the next nearest printer. The users of such a system would not have to be aware of such things as the printer's network identifier, making the configuration process transparent from their point of view.

Active Configuration

Instead of just passively connecting devices, the notion of auto-configuration can be expanded to include the active execution of particular tasks when devices are connected. This execution would be conditional on the relative location of these devices, perhaps using a stricter set of

criteria than that for "passive" configuration.

For example, many classes of portable device have some means of locally storing data, which is expected to be downloaded or synchronised with a computer at intervals. Examples include digital cameras, PDAs, multimedia devices such as MP3 players, and so on.

When such a device is placed on a Surface, it could (as mentioned above) be automatically associated with the closest computer. The choice can then be made, depending on the relative locations, to actively prompt that computer to execute an appropriate application, such as a data transfer or synchronisation program. Location sensitivity might include only activating the program if the two devices were placed very closely, with a "passive" connection established for devices not in immediate proximity.

Interface Mobility

The final class of self-configuration applications discussed is interface mobility. When a device with limited I/O capability is placed next to another with greater I/O capability, the former device might be made to export its interface to the latter. Networked Surfaces would again be useful to provide the locations of the devices and equip them with networking.

One application in this domain would be a "virtual docking station" for notebook PCs and PDAs. When placed next to a full-size keyboard and/or monitor, they could be made to export their displays, and receive remote inputs.

Another application might be in a "universal controller," i.e. a portable device which is used by a person to control many other types of device, depending on which direction it is "pointing." Such a controller may even be used to control non–Networked Surface devices, if the locations of those devices were known through another means.

6.7.2 Ubiquitous Interfaces

In the examples above, applications are using location information as an input; this information in turn originates from the placement of objects by users. In other words, the Networked Surface is being used as an input device. The concept of using Surfaces to facilitate human-computer interaction is now explored further.

Similarities exist between the proposals below and applications described in the literature presented in the previous section. The contribution of Networked Surface to this field is a novel means of human-computer interaction.

User Input

The use of object positioning for auto-configuration has already been introduced. However, there are a number of further possibilities for acquiring human input from Surfaces.

To start with, in addition to simply being placed on a Surface, objects can also be moved from place to place on Surfaces. This movement generates various data which may be regarded as useful input for applications. The most obvious data generated is the two locations; these can be combined to give a vector, which may be of interest. Also, this vector might be pointing at a device, the identification of which could be used as input data. Finally, the velocity of the object's movement can be estimated, yielding another form of input.

It must be noted that the usability of this data relies strongly on the relative accuracy of Surface-based location, both for position and orientation. Other location systems may provide data too coarse for use in computing such data as the vector describing the movement of an object.

This concept facilitates the use of any type of Surface-enabled device as a "real-world pointing device." Devices would be moved in the fashion of a computer mouse, but with the realm of interest being the real world as opposed to a computer screen. The drawback of this idea is that many types of device are not practical to use as "real-world mice," for example a notebook computer may be too large and cumbersome to accurately "point" with.

One may instead consider the possibility of using a dedicated device as a real-world pointer; such a device may take a number of forms. A standard computer mouse could be augmented to allow pointing in the real world as well as the virtual world. Alternatively, a new type of device could be constructed, perhaps incorporating features particularly relevant to real-world pointing (e.g. the ability to point in the vertical dimension).

Finally, there is the possibility of augmenting Networked Surfaces to include pressure sensors, thus allowing users to input location information using their own hands, and removing the need for a physical pointing device. If a large number of sensors was installed, the Surface could be used for input in the fashion of a touch screen display. If fewer sensors were installed, broad gestures (such as running a finger from one object to another) might still be discernable.

Output to the User

Outputs to the user may also take a number of forms. Firstly, outputs can be accomplished using normal Networked Surface devices' output facilities, including speakers, monitors, LCD displays, etc. An example application would be directing messages for a user (such as email)

to the closest suitable and available device.

While users cannot be located directly using Surfaces, they can be indirectly located by monitoring activity on devices, and monitoring the user's presence in the virtual domain. For example, if activity were detected from a particular keyboard, and the console attached to that keyboard belonged to a logged-in user, that user might be inferred to be in front of the keyboard. Other location systems can also be used to detect user location, for example by using the TRIP system in conjunction with Networked Surfaces as outlined earlier.

In addition to output using existing devices, it is also possible to consider augmenting Surfaces to include dedicated output methods. One might imagine a Surface with an array of built in LEDs; allowing it to directly perform output, and making Surfaces a true interface in their own right.

This possible application space using direct output is very large. One example might be to circle connected devices on the Surfaces with LEDs, and then use lines between these circles to indicate associations. Also, areas could be outlined to indicate special uses, for example an area near a printer could be used for automatic printing of the current document on any device placed inside it. A similar application is described in the Augmented Surfaces project [91] using cameras, projectors, and wired networking and power. Networked Surfaces can provide a single enabling technology to achieve these goals, which could be transparently integrated into the environment.

Finally, the Surface could use the outputs and inputs to facilitate dialogue with users. Again referring to the example of auto-configuration, in many cases it might be sensible to ask for some sort of confirmation from the user, for example before starting an automatic transfer. With I/O functionality implemented in the Surface itself, this dialogue could be performed directly. Output could be in the form of simple icons such as ticks and crosses, using LEDs. Input, as discussed previously, could use either movement of the devices themselves, the use of a special "real-world pointer," or the users own hands, if pressure sensors were used.

6.8 Summary

This chapter has shown how the Networked Surface can acquire location information about connected objects, with a typical object located to an accuracy of one centimetre and a few degrees. This compares favourably against the field of existing location systems. The chapter has also shown how Networked Surfaces fit into the field of context-aware computing. Differences, parallels and synergies with existing context-aware systems were discussed, as were some location-based applications which Networked Surfaces can enable.

In this chapter, the concept of a Networked Surface was expanded beyond its original

goals of providing networking and power, firstly by providing support for location-aware applications, and thence to supporting direct interaction with users. The Networked Surface concept is therefore shown to offer a unified paradigm for a whole range of tasks currently performed by many different methods.

Chapter 7

Conclusions

This chapter concludes the thesis, firstly by reiterating important results and findings, and then by summarising the areas of future work identified.

7.1 Summary of Important Results

This thesis has shown that Networked Surfaces are a viable new method for mobile networking. In addition, Surfaces have been shown to contribute to the fields of ubiquitous and context-aware computing. The key properties of Networked Surfaces are summarised below.

7.1.1 Physical Characteristics and Connections

The prototype "topology" allows many "footprints" to be used, each providing a different number of links. The object footprints described are small enough to fit on the underside of devices such as PDAs. The requirement of guaranteed connections irrespective of position and orientation was upheld.

To form connections, the problem of grounding was overcome using the "grounding by consensus" method. The distributed solution of using many tiles was shown to have no adverse impact on connection performance. The use of a two-phase connection process (i.e. handshaking and registration) was demonstrated to work, with 90% of connections taking under a quarter of a second. Disconnection detection was also presented, with the token star link layer protocol performing this in under a quarter of a second for the prototype. Reconnection is performed in under half a second in 90% of cases.

7.1.2 Networking Performance

The bandwidth supported by the prototype LVDS bus was measured to be up to 5Mbit/s. Low-level speeds of up to 12.5Mbit/s were also shown, but are not feasible in the prototype due to hardware bottlenecks.

At the link layer, the token star protocol was shown to be suitable for the Networked Surface, due to its fast disconnection detection, and its ability to perform well at high network loads. Bus utilisations of nearly 100% were demonstrated, and the latency of token star buses in the prototype was measured to be around 1ms, which is comparable with existing network technologies.

Network layer issues such as addressing and routing were discussed. In particular, the case of objects with multiple network interfaces was presented, and solutions were proposed for the issues raised.

At the transport layer, the connectivity characteristics of Networked Surfaces were demonstrated to cause bad performance in TCP. The "smart link layer" solution presented does not require modification of TCP. This solution is therefore usable without modifying the internal networking stacks of Networked Surface objects. The "smart link layer" was demonstrated to greatly improve TCP performance in the presence of periodic disconnection, both using a simulation of a disconnecting channel and using measurements on the Networked Surface itself.

7.1.3 Location Information

The Surface was shown to provide accurate location information, with the prototype capable of providing a four-link object with mean errors of 8mm for position and 1.4° for orientation. This makes Surfaces the most accurate indoor positioning system known to the author. The Surface can reliably provide location information for any connected object, and furthermore can guarantee maximum errors of 31mm and 8.5° (for four-link objects).

The use of Surfaces in conjunction with other location technologies was then explored. This includes the use of "context-aware middleware" to facilitate the aggregation of location data from many sources (including Surfaces). These proposals allow Networked Surfaces to be used in the support of other location technologies and applications.

Finally, Networked Surfaces were presented as an enabling technology for context-aware computing applications. Examples were discussed in the areas of auto-configuration and ubiquitous interfaces. For the latter, which involves adding human input/output facilities to Surfaces, the Networked Surface concept was presented as a unifying paradigm for applications in the fields of mobile networking, context-aware computing, and ubiquitous computing.

7.2 Future Research

This section will discuss future research opportunities presented by Networked Surfaces.

7.2.1 Evolution of Networked Surfaces

Section 2.9 presented a number of ways in which the Networked Surfaces physical layer could be implemented differently, including optimisations to the current prototype, and alternative approaches to Networked Surface implementation.

The optimisations discussed were in the areas of hardware integration, and the use of non-generic pads to reduce the switching hardware required. The physical construction of topologies was also examined, with methods for integrating object footprints into devices such as notebook PCs and PDAs.

Alternative approaches to Networked Surfaces include the use of other physical media, such as capacitive coupling for networking and inductive power transmission. Both of these approaches have the advantage over conductive methods of having fewer safety considerations, as no exposed electrical contacts would be used. The provision of power is also particularly interesting, as it offers a decisive advantage of Surfaces over wireless networking technologies.

7.2.2 Link Layer Improvements

Chapter 3 presented the connection, disconnection, and reconnection performance of the Networked Surfaces prototype. It noted that these could be improved in a number of ways. For connection, a two-state handshaking protocol could be used if certain hardware changes were to be made. For disconnection, the use of a hardware-based link layer implementation would allow faster disconnection detection. Reconnection is affected by both of these factors.

Also discussed in Chapter 3 were the bus speeds available in the Networked Surface prototype, and a number of methods were identified for improving this performance. Firstly, the "raw" bandwidth limit of the current prototype LVDS bus could be realised by removing hardware bottlenecks, as discussed in Section 3.4.4. Secondly, the switching technology used in the prototype (i.e. the analogue muxes) could be replaced to increase bandwidth. Finally, another physical bus standard may prove to outperform the LVDS standard in the Networked Surface environment.

The use of a hardware-based link layer also has implications for the token star link layer protocol, as presented in Section 3.5.3. A hardware implementation of the token star protocol would have the effect of reducing latency. Also, the amount of software augmentation required

of an object would be greatly reduced, allowing Networked Surface hardware to be easily ported to a number of object types.

7.2.3 Networking Issues

Quality of Service (QoS) guarantees can be supported in the prototype Surface in a number of ways, as outlined in Chapter 3. Firstly, the use of multiple buses on the Surface allows QoS-based partitioning of objects. Also, the token star protocol supports bandwidth allocation on a per-object basis. The Networked Surface therefore provides an interesting testbed for implementing QoS-aware networking policies.

Another topic for future exploration is the use of multiple network interfaces, which leads to new problems for IP, as described in Chapter 4. These include the choice of address allocation policy, mechanisms for routing data connections over the "best" interface for that connection, and methods of coping with mobility.

At the transport layer, further exploration is possible into the "smart link layer" solution presented in Chapter 5. This includes their performance when used in only part of a multiple-hop network connection, and their performance when used in tandem with other methods such as proxying.

7.2.4 Ubiquitous and Context-Aware Computing

Chapter 6 described a number of uses of location information provided by Networked Surfaces which have not been implemented; these are therefore suitable for future exploration.

Links with "context-aware middleware" were proposed in Section 6.6.2. The implementation of such links would allow existing applications to use location information from Networked Surfaces, which may lead to some useful extensions to these applications. Another interesting topic for future exploration is the use of other location systems (such as TRIP) in tandem with Networked Surfaces.

A number of context-aware computing applications particularly suited to the Surface environment were discussed in Section 6.7, but implementation of such applications was left for future work. These include the use of auto-configuration for Surface-based devices, and the use of Surfaces as ubiquitous user interfaces. In addition, many of the existing context-aware applications described in Section 6.5 may be ported to a Networked Surface environment.

Appendix A

Progress of this Thesis

The progress of this thesis is illustrated in Figure A.1.[1]

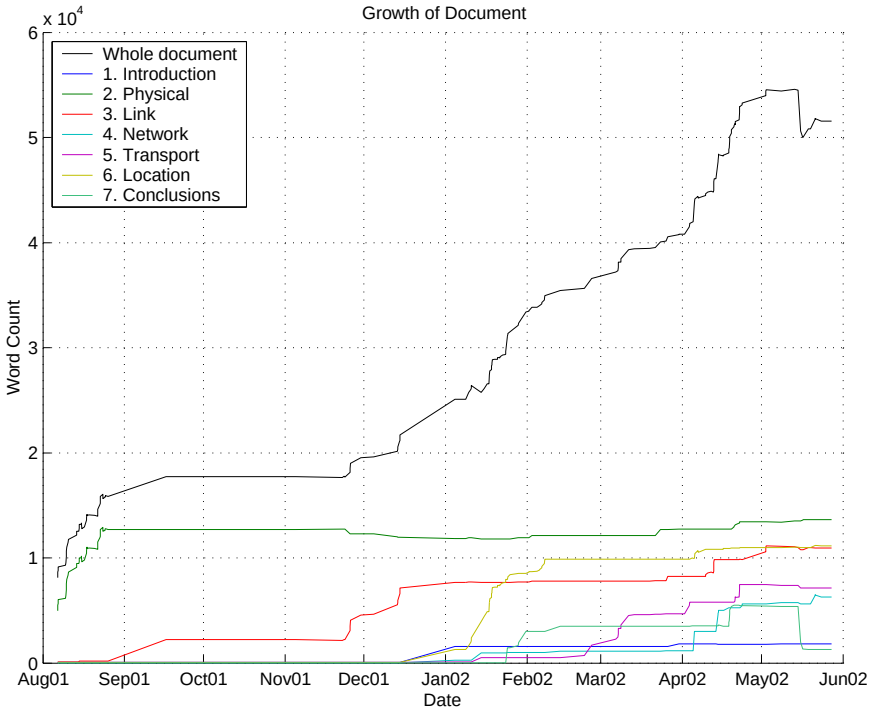

Figure A.1: Length of Thesis over Time

[1]This excludes the list of references.

Glossary

ARP Address Resolution Protocol — a method of finding devices' hardware addresses if their IP addresses are known.

ARQ Automatic Repeat reQuest – a method of providing reliable link layer packet delivery, using per-packet acknowledgement and retransmissions.

Connection In a Networked Surfaces context, describes an end-to-end connection between an object and a Surface, which may include functions such as networking and power. Requires a number of "links" to be formed.

CSMA Carrier Sense Multiple Access — an OSI link layer arbitration method, whereby potential senders do not initiate transmission if the bus is in use.

CSMA/CA CSMA with Collision Avoidance — as for CSMA but with the senders and receivers transmitting small control packets before transmission commences, so that other senders do not transmit simultaneously, thus avoiding collisions.

CSMA/CD CSMA with Collision Detection — as for CSMA but with the senders additionally performing collision detection, and backing off if collisions are detected.

DNS Domain Name Service — a hierarchical system used in the Internet to associate IP addresses with hostnames (e.g. www.cam.ac.uk).

FIFO First-In-First-Out memory — a buffer operating as a queue for data.

FPGA Field Programmable Gate Array — programmable hardware consisting of many "logic blocks."

Function In a Networked Surface context, describes a use for a single "link," to provide a service such as networking or power.

Function Buses The buses running through a Networked Surface, used to provide services such as networking and power to objects.

I^2C A simple networking specification for integrated circuits. See [83].

ICMP Internet Control Message Protocol — a "helper" protocol used with IP for sending simple administration messages, e.g. "destination unreachable."

IP or IPv4 Internet Protocol, version 4 — an OSI network layer protocol in widespread use.

IPv6 Internet Protocol, version 6 — an OSI network layer protocol, the proposed successor to IPv4.

LAN Local Area Network — a network designed for local area communications, e.g. Ethernet.

Link In a Networked Surfaces context, describes a single physical channel between the Surface and an object. Multiple links are used by an object to form a "connection."

LVDS Low Voltage Differential Signalling — a physical layer standard for high-speed networking buses. See [67].

NAT Network Address Translation — a method for masquerading many IP devices as using a single IP address.

Networked Surface A novel networking technology using physical surfaces to provide connectivity.

NIC Network Interface Card — an item of hardware used to interface a computer to a network.

Object A device augmented with a Networked Surface interface.

Object Controller Part of a Networked Surface object interface, responsible for handshaking with tile controllers.

Object Manager Part of a Networked Surface object interface, responsible for management of the object controller, and providing NIC functionality for the device itself.

OSI Open Systems Interconnect — a set of ISO communications standards. The abbreviation "OSI" is also commonly used to refer to the OSI Reference Model, a seven layer model for networking.

Pads Physical areas on a Networked Surface and objects, which are used to form connections between the two. In the prototype Surface, electrically conductive pads are used.

PDA Personal Digital Assistant — a small, mobile computer typically used by a single person for applications such as calendars, address books, etc.

PIC A brand of microcontroller useful in embedded applications.

PROM Programmable Read-Only Memory.

QoS Quality of Service — the ability to guarantee service levels to users of a particular system.

ROM Read-Only Memory.

RS-232 A serial bus discipline for sending byte-wide messages.

Surface Abbreviation for Networked Surface.

Surface Manager Part of a Networked Surface, on the surface-side, performing management functions, and data bridging between surface buses and other networks.

TCP Transmission Control Protocol — an OSI transport layer protocol, in widespread use in the Internet, providing reliable end-to-end data transmission.

Tile Collection of pads forming part of the physical surface in a Networked Surface.

Tile Control Bus Part of a Networked Surface, used by the surface manager to manage tile controllers.

Tile Controller Part of a Networked Surface, responsible for handshaking with objects placed on a tile.

UDP User Datagram Protocol — an OSI transport layer protocol, in widespread use in the Internet, providing unreliable end-to-end data transmission.

VNC Virtual Network Computing — a platform-independent system for remotely interacting with computer desktops.

References

[1] Bandula W. Abeysundara and Ahmed E. Kamal. High-speed local area networks and their performance: a survey. *ACM Computing Surveys*, 23(2), June 1991.

[2] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, 3(5):421–433, 1997.

[3] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, March 2000.

[4] Michael D. Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, August 2001.

[5] Michael D. Addlesee, Alan Jones, Finnbar Livesey, and Ferdinando Samaria. The ORL Active Floor. *IEEE Personal Communications*, 4(5):35–41, October 1997.

[6] Noha Adly, Pete Steggles, and Andy Harter. SPIRIT: a resource database for mobile users. In *Proceedings of CHI '97*. ACM, March 1997.

[7] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's loss recovery using limited transmit. Internet-Draft `draft-ietfsvwg-limited-xmit-00.txt`, August 2000.

[8] Farooq Anjum and Leandros Tassiulas. On the behavior of different TCP algorithms over a wireless channel with correlated packet losses. In *Proceedings of ACM SIGMETRICS '99*. ACM, 1999.

[9] R. Atkinson. IP encapsulating security payload (ESP). RFC 1827, August 1995.

[10] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin. AIRMAIL: A link-layer protocol for wireless networks. *ACM Wireless Networks*, 1(1):47–60, 1995.

[11] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings of INFOCOM 2000*, pages 775–784, 2000.

[12] Ajay V. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 1995 International Conference on Distributed Computing Systems*, pages 136–143, 1995.

[13] Bikram S. Bakshi, P. Krishna, Nitin H. Vaidya, and Dhiraj K. Pradhan. Improving performance of TCP over wireless networks. In *Proceedings of the 1997 International Conference on Distributed Computing Systems*, 1997.

[14] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[15] Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4):469–481, 1995.

[16] Rajesh Krishna Balan, Lee Boon Peng, K. Renjish Kumar, Lillykutty Jacob, Winston K. G. Seah, and A. L. Ananda. TCP HACK: TCP header checksum option to improve performance over lossy links. In *Proceedings of INFOCOM 2001*, April 2001.

[17] H. W. Peter Beadle, G. Q. Maguire, Jr., and M. T. Smith. Location-based personal mobile computing and communication. In *Proceedings of the Ninth IEEE Workshop on Local and Metropolitan Area Networks*, pages 23–24, Banff, Alberta, Canada, May 1998.

[18] Saad Biaz and Nitin H. Viadya. Distinguishing congestion losses from wireless transmission losses: A negative result. In *Proceedings of the International Conference on Computer Communications and Networks*. IEEE, 1998.

[19] Chatschik Bisdikian. An overview of the Bluetooth wireless technology. *IEEE Communications Magazine*, 39(12), December 2001.

[20] Lawrence S. Brakmo and Larry L. Peterson. TCP vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[21] Kevin Brown and Suresh Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communication Review*, 27(5):19–43, 1997.

[22] Peter J. Brown, John D. Bovey, and Xian Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications*, pages 58–64, October 1997.

[23] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. EasyLiving: Technologies for intelligent environments. In *Proceedings of the International Conference on Handheld and Ubiquitous Computing*, pages 12–29, September 2000.

[24] Ramon Caceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal of Selected Areas in Communications*, 13(5):850–857, 1995.

[25] Paul Castro, Patrick Chiu, Ted Kremenek, and Richard Muntz. A probabilistic location service for wireless network environments. In *Proceedings of Ubicomp 2001*, September 2001.

[26] Kartik Chandran, Sudarshan Raghunathan, S. Venkatesan, and Ravi Prakash. A feedback-based scheme for improving TCP performance in ad-hoc wireless networks. In *Proceedings of the 1997 International Conference on Distributed Computing Systems*, pages 472–479, 1997.

[27] Fu Chengpeng. *TCP Veno: End-To-End Congestion Control Over Heterogeneous Networks*. PhD thesis, Chinese University of Hong Kong, 2001.

[28] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of developing and deploying a context-aware tourist guide: The GUIDE project. In *Proceedings of the Sixth International Conference on Mobile Computing and Networking*, Boston, Massachusetts, USA, August 2000.

[29] S. Deering and R. Hinden. Internet Protocol, version 6 (IPv6) specification. RFC 2460, December 1998.

[30] A. DeSimone, M. Chuah, and O. Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proceedings of GLOBECOM '93*. IEEE, 1993.

[31] Anind K. Dey, Gregory D. Abowd, and Andrew Wood. CyberDesk: A framework for providing self-integrating context-aware services. In *Proceedings of the 1998 International Conference on Intelligent User Interfaces*. ACM, 1998.

[32] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[33] Scott Elrod, Gene Hall, Rick Costanza, Michael Dixon, and Jim des Rivieres. Responsive office environments. *Communications of the ACM*, 36(7):84–85, July 1993.

[34] Kathy Fall, Prathima Agrawal, and K. M. Sivalingam. Survey of wireless network interfaces for mobile computing devices. In *Proceedings of the 1997 IEEE International Conference on Personal Wireless Communications*, December 1997.

[35] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[36] Anne Fladenmuller and Ranil De Silva. The effect of Mobile IP handoffs on the performance of TCP. *ACM Mobile Networks and Applications*, 4(2):131–135, 1999.

[37] Sally Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[38] Sally Floyd and Tom Henderson. The NewReno modification to TCP's fast recovery algorithm. RFC 2582, April 1999.

[39] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[40] Eric Foxlin, Michael Harrington, and George Pfeifer. Constellation: A wide-range wireless motion-tracking system for augmented reality and virtual set applications. In *Proceedings of ACM SIGGRAPH '98*. ACM, July 1998.

[41] Hans-Werner Gellersen, Michael Beigl, and Albrecht Schmidt. Environment-mediated mobile computing. In *Proceedings of SAC '99*. ACM, 1999.

[42] Samir Goel and Dheeraj Sanghi. Improving TCP performance over wireless links. In *Proceedings of TENCON '98*, pages 332–335. IEEE, December 1998.

[43] Andy Harter and Andy Hopper. A distributed location system for the active office. *IEEE Network*, 8(1), January 1994.

[44] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *Proceedings of the Fifth International Conference on Mobile Computing and Networking*, Seattle, Washington, USA, August 1999. ACM/IEEE.

[45] Michael D. Hazas. *Indoor Ultrasonic Location Systems*. PhD thesis, University of Cambridge, 2002 (to be submitted).

[46] R. Hinden and S. Deering. IP version 6 addressing architecture. RFC 2373, July 1998.

[47] Janey C. Hoe. Improving the start-up behavior of a congestion control sheme for TCP. In *Proceedings of ACM SIGCOMM '96*, pages 270–280. ACM, August 1996.

[48] Frank Hoffmann. *Networked Surfaces — Design, Implementation and Characterisation of the Physical Layer*. PhD thesis, University of Cambridge, 2002 (to be submitted).

[49] Gavin Holland and Nitin H. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Proceedings of the Fifth International Conference on Mobile Computing and Networking*, pages 219–230. ACM/IEEE, August 1999.

[50] Andy Hopper. The Clifford Paterson Lecture, 1999. Sentient Computing. *Philosophical Transactions of The Royal Society of London*, 358(1773):2349–2358, August 2000.

[51] IEEE. *ANSI/IEEE Standard 802.11*. IEEE, December 1999.

[52] Jon Inouye, Jim Binkley, and Jonathan Walpole. Dynamic network reconfiguration support for mobile computers. In *Proceedings of the Third International Conference on Mobile Computing and Networking*. ACM/IEEE, September 1997.

[53] Intel. *i960 Hx Microprocessor Developer's Manual*, chapter 15. Intel, 1998.

[54] Van Jacobsen. Congestion avoidance and control. In *Proceedings of SIGCOMM '98*. ACM, 1988.

[55] Van Jacobsen. Modified TCP congestion avoidance algorithm. Email on `end2end-interest` mailing list (`ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt`), April 1990.

[56] Van Jacobsen, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

[57] Srinivasan Keshav. A control-theoretic approach to flow control. In *Proceedings of SIGCOMM '91*, pages 3–15, Zurich, Switzerland, September 1991. ACM.

[58] Srinivasan Keshav and Samuel P. Morgan. SMART retransmission: Performance with overload and random losses. In *Proceedings of INFOCOM '97*, pages 1131–1138. IEEE, 1997.

[59] Cory D. Kidd, Robert J. Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth Mynatt, Thad E. Starner, and Wendy Newstetter. The Aware Home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings*, 1999.

[60] Rolf Kraemer. Bluetooth based wireless internet applications for indoor hotspots: Experience of a successful experiment during CeBIT 2001. In *Proceedings of 26th Conference on Local Computer Networks*, pages 518–524, Tampa, FL, USA, November 2001. IEEE.

[61] Robin Kravets, Casey Carter, and Luiz Magalh aes. A cooperative approach to user mobility. *ACM Computer Communications Review*, 31(5), October 2001.

[62] John Krumm, Steve Harris, Brian Meyers, Barry Brumitt, Michael Hale, and Steve Shafer. Multi-camera multi-person tracking for EasyLiving. In *Proceedings of the Third IEEE International Workshop on Visual Surveillance*, 2000.

[63] Anurag Kumar. Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM Transactions on Networking*, 6(4):485–498, 1998.

[64] Ulf Leonhardt and Jeff Magee. Multi-sensor location tracking. In *Proceedings of the Fourth International Conference on Mobile Computing and Networking*. ACM/IEEE, 1998.

[65] Jian Liu and Suresh Singh. ATCP: TCP for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 19(7):1300–1315, July 2001.

[66] Diego López de Ipiña. Video-based sensing for wide deployment of sentient spaces. In *Proceedings of the Second PACT 2001 Workshop on Ubiquitous Computing and Communications*, Barcelona, Spain, September 2001. ACM, IEEE.

[67] LVDS. A high speed multi-drop bus signalling architecture. `http://lvds.national.com/`.

[68] Henning Maass. Location-aware mobile applications based on directory services. *ACM Mobile Networks and Applications*, 3(2):157–173, 1998.

[69] David A. Maltz and Pravin Bhagwat. MSOCKS: an architecture for transport layer mobility. In *Proceedings of INFOCOM '98*, pages 1037–1045. IEEE, 1998.

[70] Matthew Mathis and Jamshid Mahdavi. Forward acknowledgement: Refining TCP congestion control. In *Proceedings of ACM SIGCOMM '96*, pages 281–291, 1996.

[71] Joseph F. McCarthy and Theodore D. Anagnost. MUSICFX: An arbiter of group preferences for computer supported collaborative workouts. In *Proceedings of the 1998 Conference on Computer Supported Cooperative Work*, pages 363–372. ACM, 1998.

[72] Joseph F. McCarthy and Eric S. Meidel. ACTIVE MAP: A visualization tool for location awareness to support informal interactions. In *Proceedings of the International Conference on Handheld and Ubiquitous Computing*, pages 158–170, 1999.

[73] Paul V. Mockapetris. Domain names — concepts and facilities. RFC 1034, November 1987.

[74] John Nagle. Congestion control in IP/TCP internetworks. RFC 896, January 1984.

[75] Hani Naguib and George Coulouris. Location information management. In *Proceedings of Ubicomp 2001*. ACM, October 2001.

[76] T. Narten, E. Nordmark, and W. Simpson. Neighbor discovery for IP version 6 (IPv6). RFC 2461, December 1998.

[77] William Newman and Pierre Wellner. A desk supporting computer-based interaction with paper documents. In *Proceedings of CHI '92*. ACM, May 1992.

[78] Robert J. Orr and Gregory D. Abowd. The Smart Floor: A mechanism for natural user identification and tracking. In *Proceedings of CHI 2000*. ACM, April 2000.

[79] Christina Parsa and J. J. Garcia-Luna-Aceves. Improving TCP congestion control over internets with heterogeneous transmission media. In *Proceedings of the 7th IEEE International Conference on Network Protocols*. IEEE, 1999.

[80] Christina Parsa and J. J. Garcia-Luna-Aceves. Improving TCP performance over wireless networks at the link layer. *ACM Mobile Networks and Applications*, 5(1):57–71, March 2000.

[81] Jason Pascoe. The Stick-e note architecture: Extending the interface beyond the user. In *Proceedings of 1997 International Conference on Intelligent User Interfaces*, pages 261–264. ACM, 1997.

[82] Charles E. Perkins. Mobile networking in the internet. *ACM Mobile Networks and Applications*, 3:319–334, 1998.

[83] Philips. I^2C: a networking solution for integrated circuits. `http://www.semiconductors.philips.com/i2c/`, 1992.

[84] Jon Postel, Editor. Internet Control Message Protocol. RFC 792, September 1981.

[85] Jon Postel, Editor. Internet Protocol. RFC 791, September 1981.

[86] Jon Postel, Editor. Transmission Control Protocol. RFC 793, September 1981.

[87] Nissanka B. Priyantha, A. Chakraborty, and Hari Balakrishnan. The Cricket location-support system. In *Proceedings of the Sixth International Conference on Mobile Computing and Networking*, August 2000.

[88] Nissanka B. Priyantha, Allen K. L. Miu, Hari Balakrishnan, and Seth Teller. The Cricket Compass for context-aware mobile applications. In *Proceedings of the Seventh International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001. ACM/IEEE.

[89] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. RFC 1918, February 1996.

[90] Jun Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *Proceedings of Asia Pacific Computer Human Interaction '98*, 1998.

[91] Jun Rekimoto and Masanori Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *Proceedings of CHI '99*. ACM, May 1999.

[92] Tristan Richardson, Frazer Bennett, Glenford Mapp, and Andy Hopper. Teleporting in an X window system environment. *IEEE Personal Communications*, 1(3):6–12, 1994.

[93] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, February 1998.

[94] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the development of context-enabled applications. In *Proceedings of CHI '99*. ACM, 1999.

[95] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The ParcTab mobile computing system. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 34–39, Napa, California, USA, October 1993.

[96] Bill N. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.

[97] Prasun Sinha, Narayanan Venkitaraman, Raghupathy Sivakumar, and Vaduvur Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. In *Proceedings of the Fifth International Conference on Mobile Computing and Networking*, pages 231–241. ACM/IEEE, 1999.

[98] Thad Starner, Dana Kirsh, and Solomon Assefa. The Locust Swarm: An environmentally-powered, networkless location and messaging system. In *Proceedings of the International Symposium on Wearable Computing '97*. IEEE Computer Society, 1997.

[99] Mark Stemm and Randy H. Katz. Vertical handoffs in wireless overlay networks. *ACM Mobile Networks and Applications*, 3(4):335–350, 1998.

[100] Yasuyuki Sumi and Kenji Mase. Digital assistant for supporting conference participants: An attempt to combine mobile, ubiquitous and web computing. In *Proceedings of Ubicomp 2001*, pages 156–175, September 2001.

[101] N. Vaidya, M. Mehta, C. Perkins, and G. Montenegro. Delayed duplicate acknowledgements: A TCP-unaware approach to improve performance of TCP over wireless. Technical report, Computer Science Department, Texas A&M University, 1999.

[102] Zheng Wang and Jon Crowcroft. A new congestion control scheme: Slow Start and Search (Tri-S). *ACM Computer Communication Review*, 21(1):32–43, 1991.

[103] Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging physical and virtual worlds with electronic tags. In *Proceedings of CHI '99*. ACM, May 1999.

[104] Roy Want, Andy Hopper, Veronica Falcao, and Jonathon Gibbons. The Active Badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[105] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the Active Office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[106] Mark Weiser. Some computer science problems in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, July 1993.

[107] J. Werb and C. Lanzl. Designing a positioning system for finding things and people indoors. *IEEE Spectrum*, pages 71–78, September 1998.

[108] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

[109] Hao Yan and Ted Selker. Context-aware office assistant. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, 2000.

[110] Xinhua Zhao, Claude Castelluccia, and Mary Baker. Flexible network support for mobility. In *Proceedings of the Fourth International Conference on Mobile Computing and Networking*, pages 145–156. ACM/IEEE, 1998.