UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

Electronics and Computer Science

Delay Tolerance for Constrained IPv6 Networks

by

Tyler Ward

Thesis for the degree of Doctor of Philosophy

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

Doctor of Philosophy

DELAY TOLERANCE FOR CONSTRAINED IPV6 NETWORKS

by Tyler Ward

Low power sensor networks have traditionally been regarded as not having the capabilities required to connect them to the internet. New research into the Internet of Things has challenged this concept and is opening up new possibilities for sensor network capabilities. Environmental sensor networks are just one of the areas which will greatly benefit from this connectivity improvement. However, there are many challenges to be solved in order to make full and efficient use of these advancements.

One of the major challenges which has been identified is the lack of connectivity when sensors are in low power sleep states. Previous solutions for low power devices have relied on application layer gateways to proxy communications to the sensors, but this restricts the flexibility of the network as it is limited to the capabilities of the proxy. Delay Tolerant Networking (DTN) offers a solution to this problem by allowing sensors to respond and handle communications at their convenience.

This thesis presents and evaluates a novel method and implementation of Delay Tolerant Networking using IPv6 extension headers. The proposed DTN extension header is found to have a significantly lower packet size overhead than other DTN protocols. In addition, the protocol and systems to support it are entirely backwards and forwards compatible with the existing internet infrastructure allowing for it to be incorporated into existing deployments. The developed protocol forms a new state of the art for DTN on constrained sensor networks using end to end IP connectivity. Using this, a new range of low power IoT devices can be developed, featuring long battery lives and reliable connectivity.

Contents

List of Figures

List of Tables

List of Acronyms

6LoWPAN	IPv6 for Low power Wireless Personal Area Networks
AGU	American Geophysical Union
AH	Authentication Header
ASCII	American Standard Code for Information Interchange
CoAP	Constrained Application Protocol
CIDR	Classless Inter-Domain Routing
DGPS	Differential Global Positioning System
DNS	Domain Name System
DO	Destination Options
DTN	Delay Tolerant Networking
ELF	Extremely Low Frequency
ESP	Encapsulating Security Payload
EWSN	Environmental Wireless Sensor Network
FNC	Future Networks Conference
GPIO	General-purpose input/output
GPS	Global Positioning System
HbH	Hop by Hop
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ICMPv6	Internet Control Message Protocol version 6
IEEE	Institute of Electronic and Electrical Engineers

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

M2M Machine to Machine

MIB Management Information Base

MQTT Message Queue Telemetry Transport

MSWiM The ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems

MTU Maximum Transmission Unit

NAT Network Address Translation

NTP Network Time Protocol

POE Power Over Ethernet

PE-WASUN The ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks

PyDTN Python Delay Tolerant Networking

PWM Pulse Width Modulation

RPL Routing Protocol for Low power and Lossy Networks

RTC Real Time Clock

SDNV Self-Delimiting Numeric Values

SLF Super low frequency

SLIP Serial Line Internet Protocol

SNMP Simple Network Monitoring Protocol

SPI Serial Peripheral Interface

SSN Seismic Surface Node

TCP Transmission Control Protocol

UDP User Datagram Protocol

uIPv6	micro IPv6
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
WAN	Wide Area Network

Declaration of Authorship

I, Tyler Ward , declare that the thesis entitled *Delay Tolerance for Constrained IPv6 Networks* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- Parts of this work have been published as: Martinez et al. (2013), Ward et al. (2014), Ward et al. (2015) and Basford et al. (2016)

Signed:...

Date:...

Acknowledgements

I wish to express my utmost gratitude to my main supervisor Professor Kirk Martinez for all the support and advice that he has continuously given me throughout the years of my PhD studies. My gratitude goes also to my second supervisor Dr Tim Chown who has also given his time as well as his invaluable advice throughout. Thanks must also be given to the Glacsweb, Erica the Rhino and Mountain sensing teams without whom the deployments discussed in Chapter 3 would not have been possible.

Additional thanks to the research staff and PhD students of the Web and Internet Science (WAIS) research group for their advice and support during my studies. Again to the members of the Southampton University Wireless Society (SUWS) and Union Films for the friendship and outings and pub trips and more, which provided much needed relief from the research. Finally I wish to thank my family Brian, Carol and Tanis as well as my girlfriend Louise for being there for me when I wanted to talk things through or just have someone to listen to my moaning.

Chapter 1

Introduction

Low power and low cost wireless embedded electronic control and sensing systems are fast becoming a critical part of our global infrastructure. These systems are able to assist with automating tasks previously thought to be impractical to perform manually or far too costly to consider implementing using previous systems. Uses for these sensors vary widely from monitoring conditions in our environment for changes or potential dangers (Werner-Allen et al., 2005), to everyday tasks such as management of large scale distributed infrastructure (Zhou et al., 2010). At present the majority of systems have not embraced this technology, however, demand is growing due to the push for smart buildings, utilities, and cities. This demand is increasing the prevalence of these systems and strengthening the need for even lower power and more connected systems.

Sensor networks have traditionally been restricted to using specialized or proprietary network technologies. Gateway devices operating at the application layer are then used to interface with the sensor network and present its services to other networks. Figure 1.1 shows several potential connectivity options for sensor networks, these traditional networks typically fit in the 'not internet connected' or 'virtually connected' categories depending on the design of the gateway device.

The Internet of Things (IoT) is offering a new methodology for both communications within, and connection of, sensor networks. By using standard internet protocols across the sensor network boundary the limitations caused by the application layer gateway can be removed. New protocols and data formats can be added while maintaining full interoperability with present and future hardware, this is something that would usually require a replacement or upgrade of the gateway.

The phrase "the Internet of Things" has become a popular term but does not have a consistent definition, and as a result is used by different companies and organizations to mean different things. The phrase has become a useful marketing term and has been used by some to encompass any smart object, even devices that have no internet

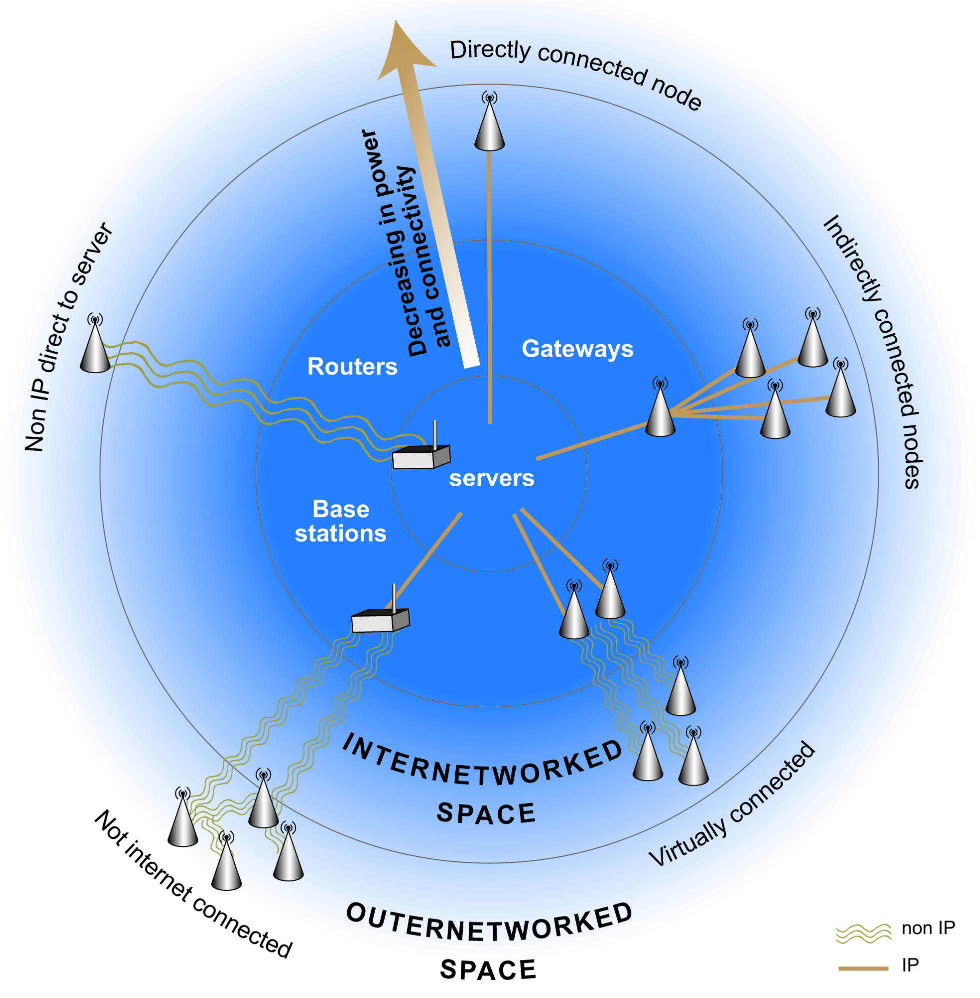


Figure 1.1: Types of sensor network connectivity republished from Hart and Martinez (2015)

connectivity such as Bluetooth controlled home appliances. In this thesis the "Internet of Things" is used to describe embedded devices with direct internet connectivity using the internet protocol. This fits within the wider scope of "connected devices" which includes devices and systems with other forms of connectivity.

It has long been thought that the requirements for internet connectivity exceeded the capabilities available on the end devices in low power sensor networks, however, this has been proven to no longer be the case (Durvy et al., 2008). With recent improvements in processing power and communication technologies for embedded systems, it is now possible to provide direct Internet Protocol (IP) connections to sensors. Using IP connectivity within these networks breaks down the barrier between the constrained sensor world and the rest of the internet, allowing for increased flexibility and access to the data these sensor networks produce. New devices with this capability are forming the IoT, a network of smart sensors, controllers and gadgets with the power of the internet behind them. At the moment, however, connecting systems to the internet comes at a higher financial and energy cost than using existing sensor network technologies. While the financial cost will come down as the market for these devices matures, reducing the power cost requires additional research.

The current generation of IoT devices have been unable to take advantage of many of the energy saving mechanisms used in present sensor network systems due to needing to maintain a continuous internet connection. This results in increased energy consumption and shorter battery lives compared to a non-IoT solution. If these energy saving techniques could be integrated into the IoT, then fully internet connected systems would lose one of their major disadvantages, the other being the increased cost, when compared to systems using other communication technologies.

The field of low power wirelessly connected internet enabled devices is currently relatively small, yet technological advances in this area have the capability to revolutionise a massive number of industries and fields. The variety of end applications that could be advanced with the use of such devices ranges from personalised healthcare (Jovanov and Milenkovic, 2011) and underground construction (Stajano et al., 2010), to energy efficient smart grids (Lu et al., 2011) and intelligent agriculture (Suryady et al., 2011). The move to IP connected systems will open up new opportunities for almost every type of sensor network system or low powered device. There will also be an array of new devices and techniques that will become feasible by the availability of low cost, highly connected sensor and actuator systems.

1.1 Research Questions

This thesis explores several research questions, a summary of each is outlined below.

1.1.1 Identification of Challenges to Internet of Things (IoT) Environmental Wireless Sensor Network (EWSN) Deployments

While IoT devices are starting to appear on the market, the majority of these have been as smart-home devices and there has been limited use of the IoT in sensor network deployments. Sensor networks have long struggled with connectivity issues between low power networks and other connected systems; the IoT offers a promising solution to this problem. Identification of the challenges and limitations of using IoT compared to other sensor network technologies is an important step on the route to combining these technologies and creating solutions to drive this technology forward.

1.1.2 Reducing Power Usage on IoT Sensor Networks

The energy consumption of IoT devices is often far greater than a traditional sensor network device with similar functionality. The increased energy consumption means that the nodes will need to have greater battery capacity or require more frequent battery replacement. In order to make IoT in EWSNs a viable alternative to existing networks, this energy consumption needs to be reduced.

Many of the current IoT devices maintain a constant connection to the internet. Thus the communications hardware must always be powered, and therefore consuming energy. The majority of sensor networks enter a low power shutdown state to save energy, however, powering off the communications hardware on IoT devices causes them to lose connectivity.

In order to enable this form of energy saving, IoT devices need to be able to use short bursts of connectivity rather than maintaining continuous connectivity. This is a method of operation that is not expected with existing internet connected devices, and as such, systems are not designed to support it.

1.1.3 Implementing Delay Tolerant Networking for Constrained IoT Networks

Following on from the previous research, Delay Tolerant Networking (DTN), a variant of store and forward was identified as a promising solution for enabling the use of sleep states on IoT devices. Current protocols for DTN are not well suited for use on the types of constrained devices that will be used on the IoT. Because of this, a new protocol suitable for enabling DTN functionality across the internet will need to be developed. While embedded IoT devices will be the main focus of the developed protocol, it should also be suitable for use on other networks operating on internet technologies in order to avoid a proliferation of standards.

1.2 Thesis Structure

Each chapter in this thesis covers a different element on the route to bringing delay tolerance to constrained devices and in so doing addresses the above questions. A summary of each chapter and the major points discussed in each is outlined below.

1.2.1 Chapter 2: Literature Review

The IoT has spun out of several other areas of research, and it is therefore necessary to provide an overview of the previous research in these areas. A large amount of the research on sensor networks is also relevant to the IoT as they share many common features and mechanisms. The technologies and systems of the wider internet ecosystem are also an inherent part of the IoT. Based on the issues raised in chapter 4, DTN research is discussed because it offers a potential solution to the identified challenges.

1.2.2 Chapter 3: Sensor Networks in the Internet of Things

This chapter covers the deployments of several sensor networks to evaluate the advantages and disadvantages of using the IoT as a platform for such networks. The differing style of each of the networks provides a different perspective on the challenges facing the IoT. The first deployment is a traditional non-IP sensor network for monitoring Icelandic glaciers called Glacsweb. Followed by an IoT sensor network, based on Internet Protocol version 6 (IPv6), used in an indoor environment as a platform for experimentation. Then an end-to-end IP sensor network for monitoring Scottish peatland in Glen Feshie. Finally, an interactive artwork called Erica the Rhino. Work on the Glacsweb network was used in the 'Using Internet of Things technologies for wireless sensor networks' poster presented at the American Geophysical Union (AGU) Fall Meeting (Martinez et al., 2013). A paper on Erica the Rhino titled 'Erica the Rhino: A Case Study in Using Raspberry Pi Single Board Computers for Interactive Art' was published in the MDPI Electronics journal (Basford et al., 2016).

1.2.3 Chapter 4: Sleeping Devices in the Internet of Things

The use of sleep states on internet connected devices conflicts with the expectation of continuous connectivity on the internet. In this chapter, investigations are carried out to allow the use of sleep states on IoT devices whilst still maintaining connectivity. A set of potential solutions are identified and evaluated against one another on several simulated network topologies. The results of this chapter have been published as 'Simulated analysis of connectivity issues for sleeping sensor nodes in the Internet of Things' at the The ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN) conference (Ward et al., 2014).

1.2.4 Chapter 5: Delay Tolerant Networking

As current Delay Tolerant Networking (DTN) protocols are not well suited for constrained IoT networks, a new DTN protocol which is suitable for use on such networks is required. This chapter covers the development, implementation and evaluation of this protocol. Implementations for Linux platforms as well as the Contiki operating system for low power devices were developed. These implementations are then tested on simulated networks and real world deployments, and the results of evaluations of the protocol and implementations are discussed. The development and initial evaluation of this protocol discussed in this chapter was published as ‘Adding support for delay tolerance to IPv6 networks’ at the Future Networks Conference (FNC) (Ward et al., 2015).

1.2.5 Chapter 6: Conclusions

The final chapter brings together and summarises the contributions from this thesis. It reiterates how the power-consumption challenges of constrained IoT sensor networks can be combatted using DTN, and ventures to suggest that the DTN protocol developed in this thesis be introduced as a standard. Suggestions for continuation of the research, and new opportunities that may be afforded by it, are also identified.

Chapter 2

Literature Review

Integrating the Internet of Things (IoT) into sensor networks is a solution to the communications obstacles present in current systems. The improvements in connectivity from the IoT has the potential to provide great benefits to sensor networks. However, power saving techniques used on sensor networks are not all compatible with the requirements of the Internet of Things.

There are many areas of research which will contribute towards the goal of integrating sensor networks into the IoT. The current state of the art in sensor networks has already developed effective solutions to many issues with small embedded devices making it a good starting point. Based on the research carried out in chapter 4, research into DTN has been investigated as a potential solution to resolving the issues identified. The current implementations for DTN are studied and analysed and as none of the existing solutions were viable in this situation, additional research was carried out in chapter 5 to develop an improved solution.

2.1 Sensor Networks

Sensor networks are collections of embedded devices that work together to gather data. The type and capabilities of these sensor networks can vary enormously depending on application, from a few sensors in a single room to thousands covering entire cities. These networks enable us to observe more information, with more frequent reporting from the environment than was previously possible with standalone sensors. This additional information gives us the ability to make better informed decisions or evaluate causes of incidents (Hart et al., 2011).

The majority of sensor networks rely on wireless connectivity for their communications. This is especially important for environmental sensor networks, as they often lack the physical infrastructure that is normally available in man-made environments. Wireless sensors are almost always power constrained as the lack of infrastructure often means that there is no external power available.

Many of the issues that will be faced by constrained IoT devices have already been encountered by sensor networks. The solutions used for sensor networks are therefore a good foundation for solving the same issues in the Internet of Things. In addition, sensor networks often have difficulty with communications to the outside world. These issues make sensor networks a prime candidate for connecting to the Internet of Things.

2.1.1 Power use in Sensor Networks

Nodes in a wireless sensor network usually have a finite amount of energy available to them. Because of this, management and reduction of power usage is critical to ensuring the network remains operational for as long as possible. Almost every aspect of the design of hardware and software has an effect on the power usage of the sensor node (Ko et al., 2012).

The majority of sensor network nodes are battery powered and will be required to last a substantial time with that limited amount of energy. While some sensors have the ability for their battery to be exchanged, there are many sensors, especially those in challenging or dangerous situations, where the option to have the battery replaced when the power runs out is not feasible.

Not all sensor networks, or nodes within a network, need to be power constrained. In some cases sensor networks will have some nodes with additional power resources, or potentially be entirely mains powered, or have the availability to be otherwise kept charged. An example is smart grid sensors in power meters (Farhangi, 2010). Whilst sensors such as this will not directly benefit from the use of low power technologies, the effects of their use may still benefit other nodes in the network.

One of the major drains on a sensor network's power is communications (Puccinelli and Haenggi, 2005). Considerable efforts have already been made in order to reduce the power used in this way. Energy savings in the radio can be made in several ways; improved radio designs and communications protocols, as well as intelligent power cycling of the radios themselves.

By using efficient wireless protocols with smaller packet sizes, a reduction in the transmission and reception times can be achieved; this reduces the energy required to communicate the stored information. Because of this, efficiencies in the radio protocol can make a significant difference to the power use of the node. Preprocessing or compressing data on the nodes can reduce the amount of data that needs to be transmitted, however, this comes at the cost of CPU time to process the readings (Kimura and Latifi, 2005). Also, if lossy compression is used, future options for analysing the data might be lost.

The majority of sensor networks are only required to perform tasks for a small fraction of the time, the rest of the time they are in an idle state. Reductions in power use while in this idle state will have a significant impact due to the large percentage of time spent in this state. An almost universal approach to reducing the energy usage is to turn off as much of the processor and peripherals as possible when not in use. A processor in a sleep state can consume as little as 0.9μA which is less than 0.1% of the power usage when in the operational state[1]. Other approaches include hard powering off the processor, until an external low power RTC wakes it up (Martinez et al., 2009).

In recent years there has been a push by chip designers to reduce the power draw of their processors while in low power modes, in order to facilitate a sleep wait cycle style of operation. For example, five years ago a typical sleep current with basic peripheral enabled might be around[2] 7.5μA, were as now it is only[3] 0.9μA.

2.1.2 Sensor Network Communications

Before the arrival of interconnected sensor networks, data had to be collected by standalone data loggers. These data loggers require manual intervention to retrieve the data from them. This is a time consuming task especially when the sensors are in remote or hard to access locations. While there are still many data logging systems in operation, this effort can be avoided by deploying connected sensor networks instead.

Communication is essential for the operation of a sensor network. The majority of sensor networks use two different network stacks, one for the low power network and another for the external interface (Akyildiz et al., 2002). This requires data to be converted between formats in order to pass the network boundary; this process adds additional complexity to the gateway devices and complicates communication with the nodes.

While some sensor networks are able to use wired networks to communicate, these are relatively few in number. Most are wireless networks which offer more flexibility in placement than wired sensors. In addition, sensor networks can be deployed in situations where there is no supporting infrastructure, for example, remote environments. Sensor networks deployed in such environments are often referred to as Environmental Wireless Sensor Networks (EWSNs).

A large number of networks utilise custom proprietary protocols for intra network communication, with a wide range of capabilities and complexity. These proprietary protocols allow for very lightweight, compact protocols at the cost of compatibility. In some cases this lack of compatibility might be considered advantageous by system providers as it ties users into their product ecosystem.

[1]http://www.silabs.com/products/mcu/lowpower/pages/efm32g-gecko.aspx
[2]http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC16F876
[3]http://www.silabs.com/products/mcu/lowpower/pages/efm32g-gecko.aspx

An alternative to developing and using a custom protocol is to use one of the standardised protocols such as ZigBee, WirelessHART or ISA100 (Nixon and Rock, 2012). These can provide advantages during development as tools and libraries for that protocol are already available. Many of the standardised protocols build on other protocols for the lower communication layers, for example Zigbee, WirelessHART and 6LoWPAN all operate on top of 802.15.4. This can cause issues, if they are unable to distinguish other 802.15.4 packets from their own. To help alleviate this newer protocols are including support for protocol detection on top of 802.15.4 (Mulligan, 2007).

The scope of these protocols vary from just defining the transport and network layers to defining everything, including the data formatting and device capabilities. Those systems that use a defined data encoding layer and set device functionality make it easier to design and test new devices for that technology. However, devices that do not fit the expected capabilities defined in the protocol will have to either alter their capabilities to conform to the provided standard or deviate from it to support its additional features. This is likely to mean these features can only be used by a limited number of other devices.

In order to improve reliability against network interruptions, many sensor networks employ a technique called store and forward. With this technique data can be passed between nodes in a network to move the data closer to a sink where the data can be sent off the network. This technique is discussed in depth in Section 2.3.1.

In order to make use of the data contained within the sensor network, the network needs to be able to communicate with the outside world. Most current sensor networks have one or more gateways that provide the interface between the network and the outside world (Martinez et al., 2010). The problem with such systems is that the gateway is required to know about everything on the network and how to interface with it in order that it can offer those services to devices on another network. As a result, the flexibility of what can be attached to the network is limited without requiring changes to the gateway. Although some flexibility can be designed in, this is unlikely to solve every case.

As sensor networks are becoming more commonplace and less bespoke, the need to standardise communications with sensor networks is becoming increasingly important. This could be achieved by standardising the way gateways interface with the outside world.

Another solution would be to use the same protocol on the network as that used to communicate with the network, and just having a transparent gateway. A standardised global communications infrastructure has already been developed around the protocols and technologies of the internet. If sensor networks can be connected into this infrastructure they will be able to communicate far beyond the range of their own network.

Research has been done into systems and technologies for internet connected sensor systems under the banner of Internet of Things, and fully developed network stacks for sensor network applications are available (Durvy et al., 2008).

2.1.3 Deployments of Sensor Networks

EWSNs have been deployed in a wide variety of locations, including volcanos (Huang et al., 2012; Song et al., 2009) and glaciers (Hart and Martinez, 2006; Martinez et al., 2009). The data gathered from such networks has helped us better understand these locations which other methods of monitoring and sensing are not practical or safe.

With traditional bespoke sensor network designs, deployment and configuration of the systems is usually a complicated procedure (Martinez et al., 2004). Domain specific knowledge is usually required in both the field being sensed as well as in sensor networks. There will always need to be some domain specific knowledge in the area being sensed in order to ensure that appropriate data is collected. However, in order to make this technology more accessible we must limit the level of sensor network knowledge required to deploy these systems.

This detailed sensor network knowledge is often needed to configure the sensor nodes. The configuration interfaces on many bespoke networks are usually achieved in such a way as to minimise development time rather than to provide easy to use interfaces. Although methods such as using audio interfaces to guide node placement have been developed (Costanza et al., 2010) these are rarely seen in actual deployments.

Many EWSNs in remote environments use an IP based long range backhaul to connect the remote sites to the remote servers where data is stored with an alternate protocol used for communications within the network (Bencini et al., 2010; Martinez et al., 2009; Huang et al., 2012). The use of this dual stack architecture requires the presence of a gateway to convert between them. However, the presence of an IP connection to the remote sites shows that offsite connections for IoT based networks should not be a major issue.

2.1.3.1 Glascsweb

Glacsweb is an EWSN developed to monitor the movement and changes within glaciers. Networks have been deployed in Norway on the Briksdalsbreen glacier (Hart and Martinez, 2006) and in Iceland on the Skalafellsjökull glacier (Martinez et al., 2009). The Icelandic deployment of Glacsweb operates using several separate networks linked by the base station. Figure 2.1 shows how the different segments of the network are connected together.

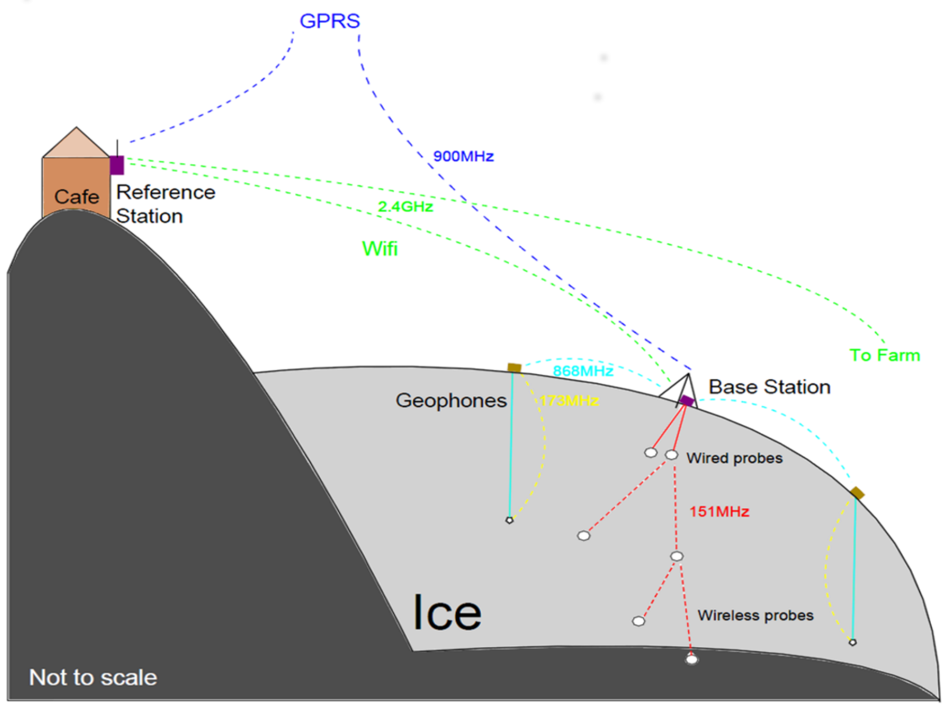


Figure 2.1: Overview of the Glacsweb Network republished from Martinez et al. (2013)

Two separate networks are used for communication on the glacier; these share a custom protocol but use different radio frequencies. For communications under the ice to the deep probes, 151Mhz radios are used in order to maximize propagation through the ice (Martinez et al., 2013). For surface communications to seismic surface nodes an 868Mhz network is used. These networks work on a master slave system where the base station is the master and communicates with nodes in turn. In order to conserve power communications with all nodes are performed in predefined time slots, allowing the radios to be powered off the rest of the time. Real Time Clocks (RTCs) on each node are used to allow all the nodes to wake up at the same time.

The communications to nodes on and under the ice are coordinated by the base station. Unlike the other sensors on the glacier, the base station has a single board computer running a Linux operating system as well as the low power system. While using such an operating system would usually require a permanent power supply, the Glacsweb systems avoids this by using a combination of techniques. While asleep, the low power microcontroller maintains the base station, which allows the Linux section of the station to be powered off when it is not needed. Several energy harvesting techniques are used for the base station, primarily wind and solar power, however, these do not function over winter as they become buried in snow. Even with the base station's considerable power reserves (170Ah of lead acid batteries), running its full compliment of features over the winter would quickly exhaust this supply. This is remedied by using an adaptive scheduling based on the available power reserves (Basford, 2015). The base station acts as a store and forward hub for the other sensors, this is discussed in detail in subsection 2.3.1.

For off-ice communications, IP networking is used. The base station is equipped with a WiFi interface, this communicates with a relay node situated on the roof of a café overlooking the glacier. The café node then relays the signal 3km down the glacier to a nearby farm which provides internet connectivity. Alternatively, if the WiFi link is unavailable, a cellular link can be used instead. Using one of these links the data is then sent back to a server which stores the data for later analysis.

2.2 The Internet of Things

The Internet of Things (IoT) is the concept of bringing internet connectivity to types of devices that have historically not been connected. Unlike the current range of internet connected devices for which users specifically intended to use the internet, IoT devices will use the internet as part of their normal operation often without user involvement. The phrase “the Internet of Things” has become a popular buzzword in recent years, and is often used as a way to describe other sorts of connected devices that can be controlled over the internet and other connected systems, even if they are not internet connected themselves. For the purpose of this thesis the “Internet of Things” will refer to devices with direct internet connectivity rather than simply any connected device.

Unlike many current smart devices, which use application layer gateways to handle communications to the device, Internet of Things devices have end to end communications. This provides many advantages over the gateway based connectivity of present systems.

2.2.1 Internet Protocols and Architecture

The internet protocol, like many other network technologies, uses a layered architecture. This allows for the Internet Protocol (IP) to be used on a variety of different link layers, and enables it to transport many different types of content with different connectivity requirements. Figure 2.2 shows the Transmission Control Protocol (TCP)/IP layer model that is commonly used for describing IP based communication systems (Tanenbaum, 1996).

In order to enable connectivity between embedded networks and devices with the existing internet, there must be a common interface layer between these new networks, and the existing internet. Sensor networks often achieve this using high level gateways, which convert between protocols at the application layer. These gateways can be replaced with more flexible devices by allowing the embedded networks to receive and process the network layer protocols used on the internet.

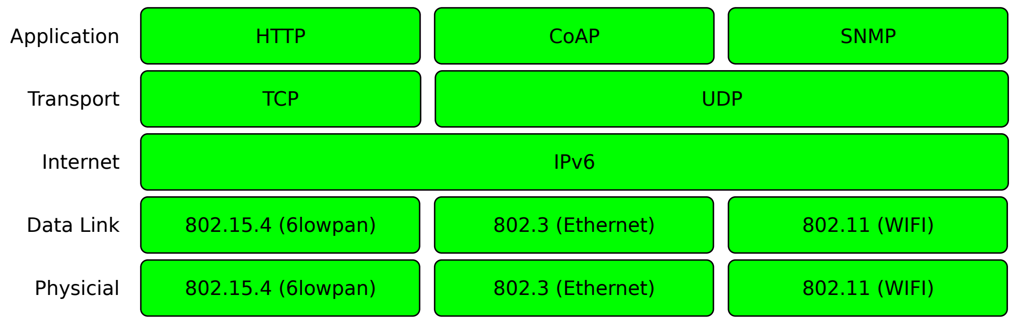


Figure 2.2: TCP/IP Layer model with example protocols for each layer

The core part of the internet infrastructure is the Internet Protocol itself. Operating at the network layer, this provides a universal layer across the internet's infrastructure. There are two versions of IP in use, Internet Protocol version 4 (IPv4) and the newer Internet Protocol version 6 (IPv6).

IPv4 (Postel, 1981) has been the standard network layer protocol on the internet for many years. This protocol, however, was developed before the popularity of the internet was fully realised and as a result there are insufficient addresses available to supply demand. This remains the case even with address efficiency and sharing measures such as Classless Inter-Domain Routing (CIDR) and Network Address Translation (NAT). The entire address space provided by IPv4 has now been allocated to regional registries, who themselves are now facing address exhaustion. At the time of writing, four of the five regional registries have exhausted their supply of general allocation IPv4 addresses, and are now only allocating addresses for IPv6 transition purposes.

IPv6 (Deering and Hinden, 1998) has been designed with much larger address space using 128 bits rather than the 32 used by IPv4. This increase will provide enough addresses so that each device can maintain a unique address. The adoption of IPv6 is not yet complete, with many systems and providers only supporting IPv4 (Nikkhah and Guérin, 2016). With complete IPv4 exhaustion imminent the need for more addresses will force the adoption of IPv6.

Some IP packets require special handling. To enable this the IPv6 specification (Deering and Hinden, 1998) includes the option for packets to have additional extension headers between the IPv6 header and the payload. Some of the features available as extension headers are present in the IPv4 header, such as packet fragmentation options, however, this introduces unnecessary bytes into the header when the feature is not needed.

While there are a few single purpose extension headers for the most common packet requirements, such as fragmentation and encryption, most of the options are implemented as part of two extensible option headers. These are the Hop by Hop (HbH) and

Destination Options (DO) headers; the HbH extension header being the only extension header to be evaluated while the packet is en route, all others are only evaluated at the destination given in the IP header. Options within these headers contain information in order to allow for predictable behaviour on devices which do not understand a given option. While most of the options currently in use allow for the packet to be processed if they are not understood, there are some such as the Jumbo Payload option that require the packet be dropped instead.

Due to the uncommon nature of IPv6 extension headers, there are currently issues with their handling on some devices, which can lead to packets being dropped (Gont et al., 2015). With increasing adoption of IPv6 it is likely that support for extension headers will improve as their use becomes more widespread.

2.2.2 Link Layer Technologies for the Internet of Things

The Data Link layer is used to transport the IP layer traffic across a network. Different link layers allow for disparate physical mediums to be used for transporting traffic in different situations. Many existing technologies have been used to connect current IoT devices, such as WiFi and Ethernet. These link layer technologies, however, are optimized for high bandwidth low latency applications and as a result are far more energy demanding and complicated than existing low power networks (Lu et al., 2004). Existing low power network protocols where not designed with the purpose of acting as a transport for IP. To resolve this, efforts were made to develop new systems to support IPv6 over 802.15.4, the result was 6LoWPAN (Kushalnagar et al., 2007).

2.2.2.1 Current Internet Link Layer Technologies

The current internet uses many different link layer technologies. The most common of these are the Ethernet, WiFi and cellular technologies. These have often been used for the current generation of IoT devices as supporting infrastructure is readily available. The capabilities of these technologies are often far more than are required for the operation of the device, although some data rich objects, or those that require low latencies for example, might require this level of performance. As these technologies are already common, objects with these connection technologies have the advantage of being able to interface with a user's existing network infrastructure without the need for extra hardware (Ostermaier et al., 2011).

The Ethernet protocol (IEEE 802.3) is being used to connect some static Internet of Things devices and base stations for indirectly connected sensor systems such as the hive hub[4]. These devices will almost always have a permanent power supply, either

[4]https://www.hivehome.com

from the mains or through the Ethernet cable using Power Over Ethernet (POE). As a result, these devices can contain more processing power as they are not as resource constrained. Being a wired protocol, Ethernet has the advantage of having a high connection reliability as well as having a high throughput. In addition, a wired interface allows for connection in environments where radio communication is unsuitable or unreliable. However, this reliability comes at the cost of installing cabling and the resulting restrictions on device placement.

The IEEE 802.11 series of specifications, more commonly known as WiFi, is used on many current Internet of Things objects. A major reason for this is that the near universal presence of preexisting WiFi infrastructure in domestic and commercial environments makes it easier to integrate the devices and reduces the barrier to entry for consumers. Whilst this is also true of Ethernet infrastructure, the majority of consumers only have Ethernet next to their router and so WiFi allows for more placement options even though a power supply is still required. WiFi is often also used as a long range backhaul for remote sensor systems (Martinez et al., 2012).

WiFi based devices require more energy to communicate small amounts of data than other systems. However, for transmitting large volumes of data they are more efficient than 802.15.4, or bluetooth networks (Lee et al., 2007). Due to the power use of WiFi connectivity most 802.11 based Internet of Things objects are mains powered or require regular charging.

Cellular technologies such as 3G are useful for several types of highly mobile Internet of Things devices such as vehicle trackers. Cellular technologies require an external data contract in order to operate, this has ongoing costs per sensor. These ongoing costs make using this technology on every device unattractive for many applications.

2.2.2.2 6LoWPAN

6LoWPAN (Montenegro et al., 2007; Hui and Thubert, 2011) is a compression layer which reduces the size of IPv6 headers when used on an 802.15.4 data link layer. The compression layer fits between the 802.15.4 protocol and the network layer containing IPv6, as can be seen in Figure 2.3. The compression is mainly performed by removing the data duplication between the 802.15.4 and IPv6 headers. Many fields in the IPv6 header can be derived from information in the 802.15.4 header, such as device addresses if auto configuration is used. This allows the header size to be heavily reduced without the loss of functionality. In addition, other parts of the header are elided if they are commonly used defaults; those that are not can be carried in line.

802.15.4 (LAN/MAN Standards Committee IEEE Computer Society, 2011) is the Institute of Electronic and Electrical Engineers (IEEE) standard for low-rate wireless personal area networks. The standard provides the data link and physical layers of the

6LoWPAN connection. As well as 6LoWPAN many other protocols use 802.15.4 for example, ZigBee and WirelessHART. The use of this standard for the lower layers is advantageous as there is already a range of low power radios which support this.

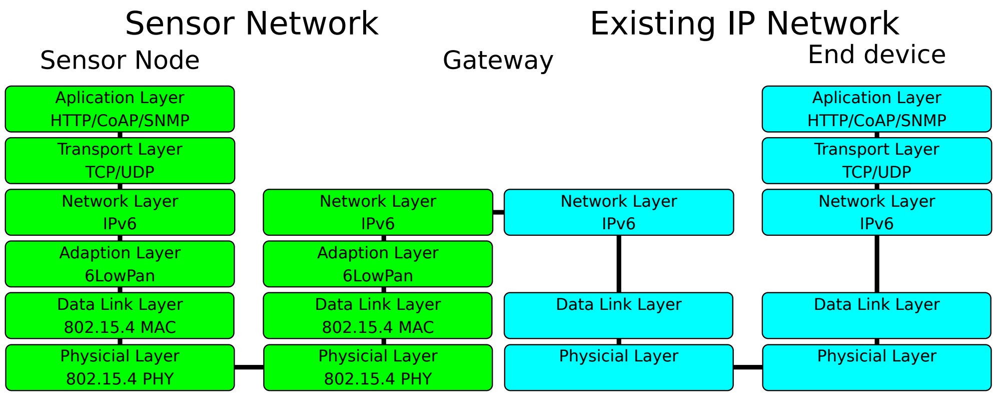


Figure 2.3: Integration of 6LoWPAN into the existing IP layer structure

The reduction in header size from the use of 6LoWPAN lowers the transmission duration, and hence the power usage. This is especially true for fragmented packets as the header must be repeated for each packet, as well as reducing the payload space available. Comparisons of the stack and program size of 6LoWPAN implementations show it to be approximately the same size as many existing propriety standards that operate on top of 802.15.4, such as ZigBee and Wireless HART; this can be seen in Table 2.1.

	6LoWPAN	ZigBee	WirelessHart
RAM	1.7-4K	8K	6K
ROM	11-22K	32-64K	34K
Header size	2-39	15	21+

Table 2.1: Memory usage and packet size of 6LoWPAN compared to other 802.15.4 protocols, data from Durvy et al. (2008); Mulligan (2007); Zhu et al. (2011); Konovalov (2010)

The new ZigBee standard (ZigBee IP) (ZigBee Alliance, 2013) is based upon 6LoWPAN technologies for its lower layers, meaning that it should be compatible with other 6LoWPAN based networks. However, with different standards being created based on parts of 6LoWPAN and its supporting technologies, it is likely that these standards will not be fully compatible with each other due to companies adding to the standards with their own technologies.

2.2.2.3 Other Link Layer Protocols

Serial Line Internet Protocol (SLIP) is a way of encapsulating IP traffic for transmission over a serial connection. Being a wired protocol, SLIP connections gain the same reliability and lack of contention advantages as Ethernet, however, SLIP has the advantage of lower overheads compared to Ethernet. As SLIP only requires a serial port for communication, the hardware to support it is already present on almost all microcontrollers. As the protocol does not specify the underlying connection, it is possible, by using serial protocols such as RS-485, for the cable to be up to 1.2Km long allowing for long distance communications to remote nodes.

2.2.3 Application Layer Protocols for the Internet of Things

In order to send information over an IP link, an application layer and data encoding are required. Although it would be possible to simply use common standards from the existing internet, these are likely to be inefficient.

There is a trade-off to be made between the usability and support of current technologies against the efficiency of new protocols designed for the constrained internet. In cases such as configuration where there is direct user interaction with devices, it may be better to use existing protocols, which will be familiar to users, over more efficient protocols.

2.2.3.1 Existing Internet Protocols

Hyper Text Transfer Protocol (HTTP) is one of the most widely supported and used protocols on the Internet as it forms the basis of the World Wide Web (Fielding et al., 1999). The widespread use and understanding of this protocol makes it easy for people to use without needing a high level of IT knowledge. A wide range of data encoding types can be used with HTTP, this allows some flexibility between the size of the transmitted data and how easy the data is to read and process.

HTTP works on top of the TCP layer, and as a result requires a connection to be established. For small volumes of data these TCP handshake packets can add significant overhead. However, for larger volumes of data, such as when transmitting a days worth of data readings at once, these handshake packets are a far less significant proportion of the overheads.

Simple Network Monitoring Protocol (SNMP) is a widely used protocol for network and data center infrastructure monitoring and management (Case et al., 1990). Rather than containing the descriptions of what responses mean within the packets, this information is stored in a Management Information Base (MIB) file. This file is stored on the device requesting the information. By doing this the amount of data that needs to be sent to get or set a value is heavily reduced. As a result it is possible for both the request and response, for up to three different values to fit within a single 802.15.4 frame. When using 6LoWPAN this avoids fragmentation.

2.2.3.2 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) is a newly developed protocol for constrained Machine to Machine (M2M) applications on Internet of Things devices (Shelby et al., 2014). The User Datagram Protocol (UDP) transport layer is used to avoid the connection setup overhead; packet acknowledgements are instead handled within COAP itself. A large amount of the features of HTTP, have been included in the protocol such as being able to support multiple data types. By doing this, stateless mapping of COAP requests into HTTP requests is possible, meaning more capable devices can provide proxies that allow access to COAP resources over HTTP. It also provides a level of familiarity to developers using HTTP, for example, the 'not found' error is 404 on HTTP and 4.04 on CoAP.

CoAP is more than a compressed HTTP for constrained devices, as it adds additional features suited to constrained M2M applications. The major additions in this regard, are asynchronous messages and support for multicast requests. CoAP is not yet widely deployed but implementations have been created for many embedded operating systems and programming languages (Kovatsch et al., 2011).

2.2.3.3 Custom Protocols

Rather than use a standardised protocol, another option is to use a custom protocol on top of the network or transport layers. Using a custom protocol allows it to be heavily optimized for its specific task. This optimization comes at the cost of reduced interoperability between devices at the application layer. However, devices can still share the network layer resources making it significantly more interoperable than using a fully custom stack. When surveyed by Thoma et al. (2014) proprietary protocols where deemed to still be a dominant factor in the IoT, although this is expected to decrease in the future.

2.2.4 An Analysis of Current Issues on Internet of Things Devices

There are a growing number of IoT devices available; while some of these are smart products that have been badged with the IoT title, there are many true IoT products on the market. The successes and failings of these products are important to understand so that they can be improved upon and developed further for the next generation of IoT devices.

2.2.4.1 Loss of Remote Infrastructure

Many current IoT devices use cloud servers to provide their interface and control much of their functionality. This is done for several reasons, including lack of processing power, NAT punch-through or ease of software upgrades.

The reliance on this remote infrastructure has caused several commercial IoT devices to be made useless overnight when the remote servers have been shutdown. An early example of this is the Nabaztag smart object produced by Violet. The Nabaztag is an interactive companion in the shape of a rabbit, it was able to interact with a user via a combination of lights, ear movements and spoken announcements. Due to a lack of onboard processing, all the external interfaces and speech synthesis were carried out on Violet's servers. The bankrupcy of Violet resulted in the loss of the servers that provided the Nabaztags services, resulting in all of the Nabaztags becoming unresponsive (Volpe, 2011).

More recently, the company behind the Revolv smart home hub was purchased by Nest. Nest then decided to shutdown the servers linking the device and the smart phone app which controlled it (Price, 2016). Unlike other areas of technology, where as, when companies decide to stop supporting products it results in a lack of updates and unavailability of accessories and spares. These cloud controlled IoT devices, are reliant on the manufacturers service for basic functionality and the loss of that service leaves them as little more than paperweights. Whereas previous instances of this occurring have mainly been due to bankruptcy or closure of the company, Nest are still active and are maintaining other product lines meaning that this was a purely business decision.

2.2.4.2 Low Energy Radio Power off Systems

While the majority of IoT devices rely on larger energy reserves in order to provide their required connectivity, there are some which have implemented a low power sleep state in order to conserve power. One such example is the Amazon Dash button, this is a device that allows consumers to order consumables online by pressing a physical button.

The Amazon Dash connects to the WiFi network only briefly in order to send the ordering request. The rest of the time the device can enter a low power sleep state waiting for the interrupt generated by the button press. This method has the disadvantage that other devices cannot contact the button in order to query it for information. In the case of Amazon's Dash button this limitation is of little concern. However, for many other devices this could not be so easily ignored and as a result this style of implementation is rarely seen.

2.2.4.3 Operating Systems for IoT Devices

In the current internet the majority of embedded devices run some form of operating system based on the Linux kernel. The use of a pre-existing operating system allows for faster development and more consistency on how standards are followed. Even the extremely cut back Linux distributions such as openwrt[5] or microcore linux[6] require resources that are unavailable on all but the most powerful EWSN nodes. Current embedded IoT devices which are too small for Linux often use their own in-house operating systems, the development of which takes considerable time and resources.

There are several operating systems being developed or extended for use on future low power IoT devices. Many of these are open source operating systems, whereas others such as Windows 10 IoT Core are closed source systems. Whilst these operating systems are only just starting to appear in products, they are in common use in academic and hobbyist environments.

One of the more popular operating systems for IoT research is Contiki[7] (Dunkels et al., 2004). Contiki is an open source operating system designed to function on a variety of platforms and chip architectures. As low power microcontrollers do not have the support for operating systems that application processors do Contiki has to emulate these features. To provide multiple application support for example, Contiki uses a system called protothreads which emulates a multi-threading architecture on single threaded processors. Contiki supports several different network stacks which have been developed for it. The latest of which is the micro IPv6 (uIPv6) network stack which provides 6LoWPAN capabilities (Durvy et al., 2008). To help reduce energy usage Contiki uses a MAC layer addition called ContikiMAC, this shuts down the radio most of the time and only listens periodically, when transmitting the message is sent multiple times, this enables the receiver to have one of its periodic receive states during a transmission.

Another recently developed open source operating system is RIOT[8] (Baccelli et al., 2013). Like Contiki, it has been designed to support a variety of platforms and architectures. Unlike Contiki, RIOT features full real time preemptive multithreading support which was imported from FireKernel (Will et al., 2009).

Microsoft has released an IoT operating system called Windows 10 IoT Core[9]. Unlike the other examples discussed above, this is not designed for the microcontroller based platforms used by Contiki and RIOT and is instead a competitor to the cut down Linux implementations.

[5]https://openwrt.org/
[6]http://tinycorelinux.net/
[7]http://www.contiki-os.org/
[8]http://www.riot-os.org/
[9]https://developer.microsoft.com/en-us/windows/iot

2.3 Delay Tolerant Networking

Delay Tolerant Networking (DTN) is the application of store and forward methodologies within the wider networking space. Whereas store and forward works on blocks of data at the application layer, DTN works on the network level by buffering packets. This allows the creation of more resilient links in challenging scenarios, such as deep space communication and low power lossy networks.

The concept of using store and forward based technologies on an IP system was proposed by Vint Cerf as part of the interplanetary internet (Cerf et al., 2007). The majority of current research into DTN has been for use in the deep space networking area. Despite the lack of implementations suited for use on constrained IP systems, the concepts of Delay Tolerant Networking are a promising solution to the challenges faced by constrained networks (Ho and Fall, 2004).

2.3.1 Store and Forward

Store and forward is a method of sending data, where data is passed in stages towards its destination, each stage stores the data before passing it onwards (Butterfield et al., 2016). It breaks down a multiple hop connection into several shorter connections, allowing data to be sent even if simultaneous end to end connectivity cannot be attained. This is especially useful in situations where there are unreliable radio environments, or nodes are mobile (Martinez et al., 2004). Store and forward is widely used in EWSNs deployments as these networks are more susceptible to intermittent connectivity.

With end to end connectivity, disruption on any single hop on the path prevents any communication from occurring. With store and forward, however, progress can be made. Where interruptions are commonplace, end to end connectivity may only be attained infrequently. In these cases, store and forward will still allow for data to be transferred off the network.

There are many different methods of implementing store and forward. One implementation method is to split the network into several sections, with nodes linking the sections performing the store and forward operations. This is the method used in the Glacsweb network (Martinez et al., 2006). Where there are separate technologies used on different stages of the network the connecting gateway is often a convenient location to perform store and forward.

Another option is to build store and forward into the networking system used on the network. In this design, storage and forwarding of messages is part of the underlying protocols supporting the network. This allows for every node in the network to store and retransmit information which is advantageous in mesh networks. As store and forward needs to be built into the network when it is created, this approach is limited to operating on networks it was designed for.

The efficiency improvement that can be gained from using store and forward technologies varies based on the reliability of the connections within the network. However, for highly reliable networks, the overhead of performing the store and forward can outweigh the advantages.

2.3.1.1 Use of Store and Forward on Sensor Networks

Like many other EWSNs, the Glacsweb network makes use of store and forward technologies (Martinez et al., 2004). In the Glacsweb network, discussed in Section 2.1.3.1, node connectivity has seasonal patterns. The base station is used to perform the store and forward, when it collects the data from the sensors it stores the sensor readings locally until an external internet connection can be acquired, at which point the data is forwarded to the Glacsweb server where the data can be used.

The main advantage to the Glacsweb network of using store and forward is the protection of the sensor data. In the winter heavy snowfall buries the surface systems, preventing communication off the glacier. However, at the same time liquid water in the glacier freezes, improving radio propagation and allowing communication with nodes which were previously out of contact. When connectivity with nodes is established, the data files are transferred to the base station. By transferring the files up to the base station the risk of data loss is significantly reduced. The probes themselves are prone to run out of power, being crushed, or meeting many other forms of demise. The base station, however, is more robust and even if it fails, the data can be extracted by removing the storage card when a research team is next on the glacier. Assuming the base station survives the winter, the spring melt will uncover the WiFi antennas, allowing for the files to be transferred back to Southampton where they can be processed.

2.3.1.2 Differences Between Store and Forward and DTN

At first, DTN may seem very similar to store and forward. While this is true for the underlying principles, the way in which it is achieved is substantially different. Rather than passing blocks of data between nodes, DTN works by buffering packets that can not be passed onwards.

The majority of networks which use store and forward technologies have to be built around this technology, and as such store and forward implementations are limited to the networks which use them. This limitation prevents store and forward from operating across multiple interconnected networks such as those used in the IoT. DTN, on the other hand, is intended to operate with existing networks and fit within existing communication systems.

Store and forward implementations often require the data blocks to be passed to the next hop deliberately. This works when the supporting network and all of the devices connected to it are based around this principle. However, where the onward network technology is not known and a range of devices can be connected to it, this method becomes limiting. DTN allows for compatible devices in the communications chain to choose to buffer packets without requiring all devices to support it.

2.3.2 DTN Implementations

There are several protocols being developed that enable the use of DTN. While some of these are more for academic interest others such as the bundle protocol are being developed with the intention of forming an internet standard.

2.3.2.1 The Bundle Protocol

The bundle protocol is the most developed of the DTN protocols. The bundle protocol (Scott and Burleigh, 2007; Cerf et al., 2007) is an implementation of DTN with a focus on the interplanetary networking use case. The development of the bundle protocol has been carried out by the DTN working group of the IETF. As much of the interest around DTN networking has been in this area, a significant proportion of DTN research has been towards the development of the bundle protocol.

The architecture of the bundle protocol is designed as an overlay network which runs on top of existing network infrastructures (Fall and Farrell, 2008). This allows the use of the multitude of different link and network layers in existing installations. The data is encapsulated (with additional routing information that specifies the eventual endpoint), into bundles that can be transmitted over the existing links. The additional routing information in the bundles is processed by DTN enabled devices. These are used as the endpoints of messages sent on the underlying networks, similar to the linklayer endpoints of other networking protocols. Figure 2.4 shows the structure used by bundle protocol bundles.

The bundle protocol has been trialled as part of the CGBA5 (Commercial-Grade Bioprocessing Apparatus 5) unit on the International Space Station (Jenkins et al., 2010). In this trial the bundle protocol was used to replace the previous unacknowledged “transmit-in-the-blind system” which relied on repeated retries to ensure packet delivery. The bundle protocol was a significant improvement over the existing solution,

```
Primary Bundle Block
+----------------+----------------+----------------+----------------+
|    Version     |                  Proc. Flags (*)                 |
+----------------+----------------+----------------+----------------+
|                          Block length (*)                         |
+----------------+----------------+---------------------------------+
|  Destination scheme offset (*) |     Destination SSP offset (*)  |
+----------------+----------------+----------------+----------------+
|      Source scheme offset (*)   |        Source SSP offset (*)    |
+----------------+----------------+----------------+----------------+
|    Report-to scheme offset (*)  |      Report-to SSP offset (*)   |
+----------------+----------------+----------------+----------------+
|    Custodian scheme offset (*)  |      Custodian SSP offset (*)   |
+----------------+----------------+----------------+----------------+
|                    Creation Timestamp time (*)                    |
+---------------------------------+---------------------------------+
|             Creation Timestamp sequence number (*)                |
+---------------------------------+---------------------------------+
|                           Lifetime (*)                            |
+----------------+----------------+----------------+----------------+
|                        Dictionary length (*)                      |
+----------------+----------------+----------------+----------------+
|                  Dictionary byte array (variable)                 |
+----------------+----------------+---------------------------------+
|                      [Fragment offset (*)]                        |
+----------------+----------------+---------------------------------+
|              [Total application data unit length (*)]             |
+----------------+----------------+---------------------------------+


Bundle Payload Block
+----------------+----------------+----------------+----------------+
|  Block type    | Proc. Flags (*)|        Block length(*)           |
+----------------+----------------+----------------+----------------+
/                     Bundle Payload (variable)                     /
+-------------------------------------------------------------------+
```

Figure 2.4: Structure of a bundle protocol bundle republished from rfc5050 (Scott and Burleigh, 2007)

reducing the retransmissions of packets from several thousand to only a few times. However, the authors recognise that a large proportion of the reduction in data transmitted could also have been achieved by adding acknowledgements to the system.

The bundle protocol has also been trialled on wireless sensor networks (Pottner et al., 2012). In this trial, the network was created in two isolated segments connected by a moving node that would transport bundles between them. The bundle protocol implementation uDTN (Von Zengen et al., 2012) running on Contiki was used directly over an 802.15.4 network layer using the Compressed Bundle Header Encoding format (Burleigh, 2011) to reduce the header size. The packet size overhead of 802.15.4 and BP header came to 31 bytes compared to 23 when using UDP over 6LoWPAN on 802.15.4.

While the bundle protocol fulfills the requirements of NASA and other space agencies use cases, design decisions made in its development make it far less suitable for use in sensor networks. The main design decision which reduces suitability for constrained networks is that it has been implemented as an overlay network. There are several reasons for this, the first being the increase in packet size caused by the use of the bundle protocol. The additional bundle routing layer also adds additional complexity and requirements for low powered devices, as they now need to process an additional routing header. This additional processing requires both computational resources and additional code space, neither of which are abundant on low cost microcontrollers. Another problematic design choice with the bundle protocol is that all devices are required to know the current time (Fall and Farrell, 2008). While this has been deemed to be acceptable for the sorts of systems used in deep space DTNs, this is a significant burden for many constrained IoT devices.

The overlay network structure of the bundle protocol adds a significant amount of additional data which needs to be contained within the packet. The majority of this additional data is because the packet needs to contain additional addressing information, as the endpoints of the bundle might not be within the scope of the underlying network layers. Figure 2.5 shows a breakdown of the packet size when the bundle protocol is used compared to a standard IPv6 and 6LoWPAN packet. The use of the dictionary in the bundle protocol means the size of a packet can vary significantly based on the size of the URI scheme used. The developmental and changing state of the bundle protocol means that the sizes of the packets may vary as further development is carried out.

A large proportion of the size of the bundle protocol header is due to the packet addressing within the protocol. The addresses are included as American Standard Code for Information Interchange (ASCII) encoded Uniform Resource Locators (URLs), this has been done to allow transport over multiple network layers which will have different addressing systems. Using friendly URLs requires a lookup system such as Domain Name System (DNS); unfortunately for sensor networks reliable DNS access cannot be assumed and raw IP addresses would need to be used instead. An IPv6 address will

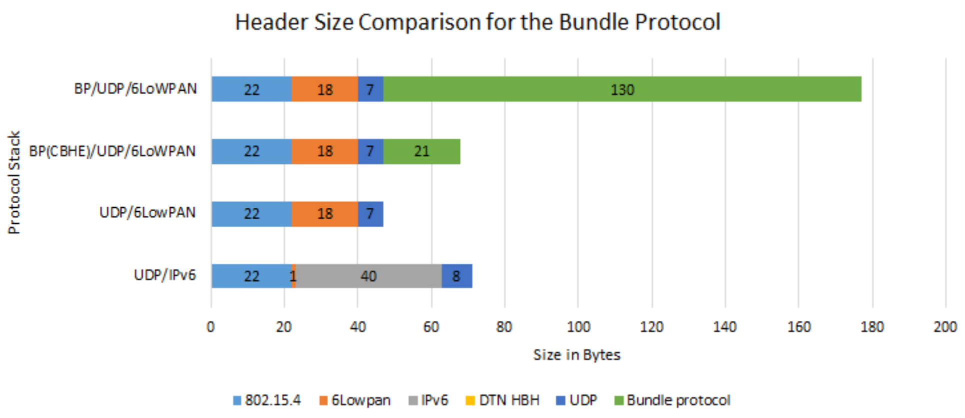


Figure 2.5: Comparison of Bundle protocol header size on IoT networks.

take between 7 and 39 bytes for either a minimal address such as AAAA::1 or a fully utilised address such as AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA. Yet if encoded in binary these addresses would require just 16 bytes. In addition, there is a protocol identifier which will identify the protocol being used for raw data, such as dtn:udp:address,port. This adds additional data, although the Uniform Resource Indicator (URI) scheme can be shared if it is used for both the source and destination. Combined together, this results in a very inefficient method of encoding IP addresses, the total size for the bundle packet headers comes to approximately 130 bytes with full addresses, depending on options.

The bundle protocol uses absolute timestamps counted in seconds past the start of the year 2000 in Coordinated Universal Time (UTC). The Self-Delimiting Numeric Values (SDNV) variable structure of the bundle protocol (Eddy and Davies, 2011) avoids needing to preallocate the variable sizes meaning that the timestamp field can increase as required. It currently takes 5 bytes to provide the timestamp in this format as only 7 bits are available for data in each byte, however, this format will expand a byte at a time when needed rather than needing to immediately use a 64 bit value. The lifetime is measured as seconds since the creation timestamp and uses the same SDNV structure, meaning that length increases with increased lifetime, 3 bytes are required to represent a lifetime of 1 day using this format.

Due to size or cost requirements many devices do not contain an RTC, or they may rely on intermittent harvested power that makes maintaining a reliable clock infeasible. While the bundle protocol has partial support for devices that do not know the time, it does not allow the nodes to determine when the packet will timeout.

2.4 Literature Summary

Bringing IoT technologies into the EWSN ecosystem incorporates research from a variety of areas, each bringing important considerations and solutions to the problem. Sensor network research has produced low energy systems which are able to collect data from our environment, however, these systems have struggled with interfacing with outside systems. The continued development and proliferation of the internet has created the IoT, bringing internet connectivity to embedded devices but at a higher power consumption than the communication techniques used in sensor networks. Combining these together give the option to get the best of both worlds, the capabilities of sensor networks but with the communication capabilities from the IoT.

After the research in chapter 4, additional investigation was needed to determine the best options for implementation. These revealed that the principles behind DTN offered a suitable methodology but current implementations were not well suited to constrained IoT networks. The choices, both good and bad, made in the existing implementations helped refine decisions in chapter 5 where a new protocol suitable for constrained networks was developed.

Chapter 3

Sensor Networks in the Internet of Things

The IoT has the potential to cause a major shift in how sensor networks are deployed and managed. There is the potential to resolve longstanding limitations with regard to user interaction and connectivity but will likely bring new limitations and restrictions to such networks. To understand the impact of these new limitations it is necessary to deploy several networks both with and without the use of the IoT.

Sensor networks have often been restricted as to how they can communicate as the gateway device needs to translate between networks. The end to end addressing and connectivity available through the IoT allows for many different types of data to flow across the network boundary, this has the potential to remove the limitation of the gateway as it will only need to perform routing functions rather than provide a full interface for external connections. Current user interaction factors require both specialist sensor network knowledge, and a knowledge of the domain being sensed, in order to deploy and setup a network. This issue is a major barrier to their deployment, which can be reduced by providing familiar configuration interfaces alongside the efficient data transmission protocols. This reduces the level of sensor network knowledge required to configure a network.

Through the use of the IoT on EWSN deployments the new limitations can be identified and the impact in real world deployments determined. The first of these trial deployments was a temperature monitoring network in the university research labs. Using the knowledge gained from this trial an EWSN was deployed in the Cairngorms at Glen Feshie. In contrast, an IoT based system was also used in the construction of an interactive artwork known as Erica the Rhino. By evaluating all of these deployments the barriers to the use of the IoT can be identified.

3.1 Deployment of a Traditional Wireless Sensor Network for Glacial Monitoring

The Glacsweb sensor network (Martinez et al., 2010)[1] is an Environmental Wireless Sensor Network (EWSN) used to monitor conditions within glacial formations. This section covers some of the work carried out as part of the deployment on the Skalafellsjökull glacier in Iceland in order to better understand the networking requirements of EWSNs. An overview of this network's previous systems and designs can be found in section 2.1.3.1.

The Glacsweb network uses several types of node developed specially for each application. The subglacial probes are used to report conditions under the glacier. There are two types of subglacial probe in use on the network. The first type of node sits at the bottom of the glacier and monitors conditions at the point where the glacier meets the till underneath. The second type is the seismic deep node, these sit inside the glacier and use geophones to pick up tiny earthquakes in the ice caused by the glacier's movement. These two types of subglacial node are also supported by a third set of nodes on the surface which collect DGPS data and act as relays for geophone nodes. The network is coordinated by the base station which sits on the top of the glacier. A custom low power protocol is used to collect readings from the nodes. These readings are then sent back to base over IP.

In addition to the networked elements of the deployment, there are several standalone sensor systems. These systems require the data to be manually retrieved from each device upon each visit to the glacier. These devices are therefore only included on surface nodes for obvious reasons. The networking behind Glacsweb relies on several different underlying network protocols which link together to form the sensor network. Figure 3.1 shows the protocols and frequencies used for each link in the network.

The low power links between the sensor nodes use the same custom protocol over different physical layers. The wireless links use different radio frequencies for subsurface and surface communications so as to minimize the losses incurred when the signal passes through the ice (Martinez et al., 2013). The geophone nodes are linked by a wired connection, which uses the same protocol as the wireless links but over an RS485 link. As well as allowing the geophone nodes to send data back to the base station, the wired link also provides additional power to the nodes, allowing them to maintain a physically small size whilst still permitting the high level of performance required for data collection. The 173 MHz link to the nodes was not installed in the deployed network.

As well as the low power links, the base station also features WiFi and GPRS interfaces, to provide IP connectivity to the outside world. The use of multiple interfaces to the

[1]http://glacsweb.org

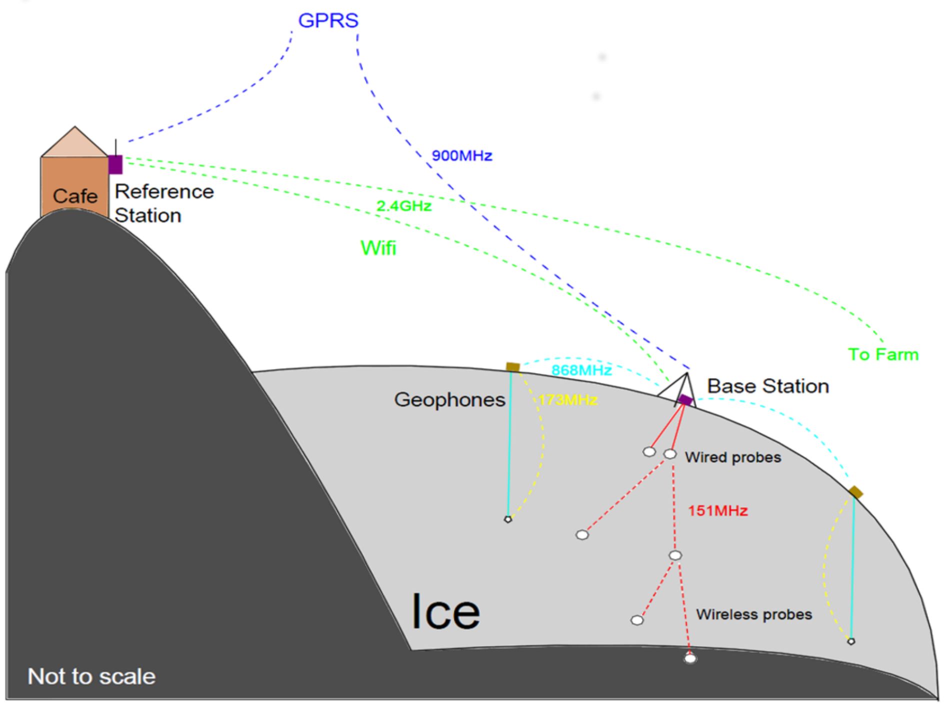


Figure 3.1: Connections in the Glacsweb sensor network(Martinez et al., 2013)

outside world provides redundancy, increasing the chances of transferring data off the ice during periods of adverse weather.

Configuration of the sensors can be achieved using a serial terminal over a wired connection. The nodes can also be configured by sending commands from another node within radio range. This method can be used by the base station to remotely send commands the next time it performs communications with the central server. While this is a functional solution, it requires operators to learn how to use multiple methods of configuration as initial setup has to be done with the direct connection.

The majority of the non networked data logging nodes are designed to collect GPS data. For the most recent deployment, several of the SSN were replaced with upgraded versions that could also collect DGPS data.

An external Ublox GPS receiver module was connected to the SSN node over a serial link. The Ublox was configured to transmit raw GPS data over this interface which the SSN logs to the SD card along with data received from the seismic nodes. To reduce power consumption the GPS receiver is hard powered off between GPS sampling sessions. The data which is captured by the GPS receivers is transferred back over the 868MHz sensor network using the same mechanisms as the data from the seismic nodes.

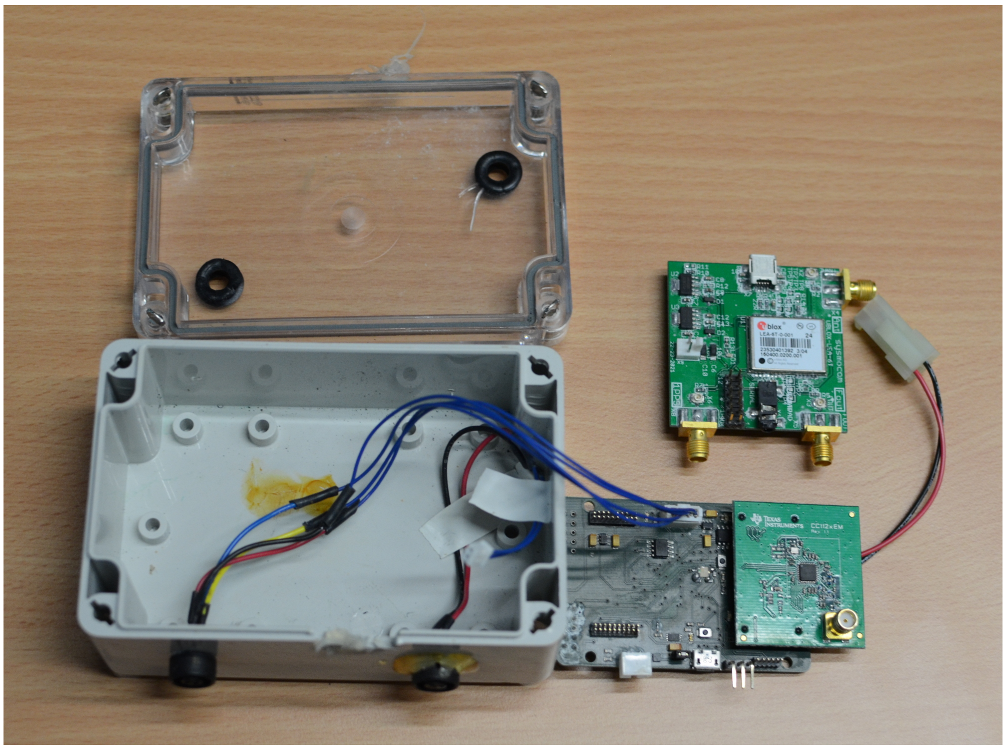


Figure 3.2: Photograph of recovered parts of a 2013 Seismic Surface Node (SSN) with DGPS module. Upper right is a Osmocom LEA-6T GPS module breakout board, lower right is a 2013 SSN board.

The loss of communications with the base station, due to unexpected power depletion, resulted in limited data being reported back through the sensor network. In addition, deep snowfall prevented physical recovery of the data in 2014. A recovery trip in 2015 to retrieve equipment loaned from NERC managed to extract one of the deployed DGPS nodes which had melted out of the ice. The data was then extracted from the internal micro SD card so the data contained within could be evaluated. The second live DGPS was unable to be recovered as it was still frozen in the ice.

An analysis of the GPS data collected from the enhanced SSN node showed an accuracy of 8cm. While this is less accurate than the Leica DGPS units in use, the new nodes come at a fraction of the cost and have the capability for data to be reported back rather than requiring collection on each visit to the glacier.

The style of sensor network deployment used in Glacsweb, is also common of other bespoke EWSNs. The use of this deployment style requires a level of knowledge about the sensor network and how it operates, as well as the domain specific knowledge of the environment that is being monitored. While this is sustainable for bespoke research systems the level of knowledge required results in a lack of scalability.

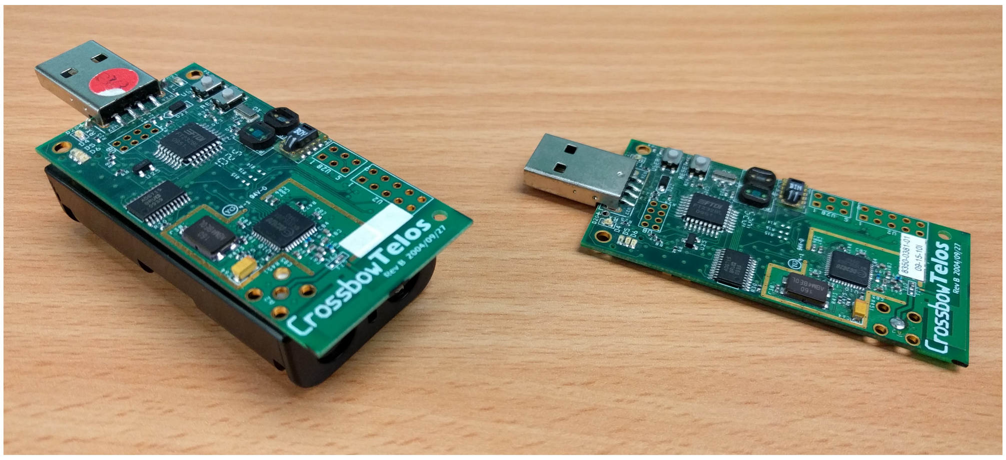


Figure 3.3: Photograph of two Telos sensor nodes, that were deployed in the network

3.2 Building Temperature Monitoring Network

In order to explore the capabilities of an end to end IPv6 based sensor network, a trial network which could be used as a test bed for experimentation was deployed. To enable easy physical access to the nodes in the network, the network was deployed in the research labs at Southampton University. This permits frequent access to the nodes so that firmware can be uploaded and diagnostics performed. The sensor network is based around the Telos B sensor node platform and the Contiki (Dunkels et al., 2004) operating system. The Telos B platform has temperature, humidity and light level sensors built in, which avoids the need for external circuitry. The sensors are connected together using 6LoWPAN(Hui and Thubert, 2011) over 2.4GHz radio links. The 6LoWPAN communications implementation used in this deployment is uIPv6 and is provided as part of the Contiki operating system (Durvy et al., 2008).

Each node in the network runs a web server which allows for the readings to be fetched using the HTTP protocol. As well as the readings from the temperature and light sensors, information on the current routes around the network is also available over the HTTP interface.

SNMP was also trialled as a method for data reporting. The use of SNMP reduced the amount of network traffic compared with using HTTP. Being UDP based, SNMP avoids the overhead associated with creating and ending TCP connections. In addition the use of the MIB allows for far more compact data transmission. This allowed three requests to be fitted into a query that fits within a 6LoWPAN packet. This meant that only four packets (two requests and two responses) were required to read all the sensor data. The disadvantage of this is that the values stored are only referenced by their IDs and the contents will not be useful without the MIB file.

The interface between the sensor network and the campus network was provided by an Alix board, industrial single board computer, with an attached Telos node. (the Alix board running the OpenWrt operating system and the Telos node running Contiki). The two devices were linked over USB over which a SLIP interface was used to provide connectivity. Routing of connections between the networks was managed by the Openwrt router with the Telos node handling translation from IPv6 packets to 6LoWPAN encoded IPv6. The ip6tables packet filter was used to process the routing of packets between the various interfaces available on the border router. While there was usually only one network managed from the border router, the system is capable of supporting several low power networks from a single border router. The ability for multiple 6LoWPAN networks to be attached to a single border router also allowed for experimentation with networks on other frequencies such as 868MHz.

One of the major advantages of using standardised network layers is that additional hardware, potentially from different manufacturers, can be added to an existing network without needing to make changes to existing infrastructure. As part of the work undertaken when experimenting with potential hardware platforms for the sensor network deployment in Glen Feshie, Zolertia Z1 nodes were added to the network. As both nodes were running a 6LoWPAN stack the Z1 nodes were able to join the network using the existing Telos B powered border router and nodes. The ease with which this was achieved shows how powerful a common IP based network layer for sensor networks can be in supporting deployments.

The research carried out as part of this deployment contributed to the 'Using Internet of Things technologies for wireless sensor network' poster presented at the American Geophysical Union (AGU) (Martinez et al., 2013).

3.3 Deployment of an Internet of Things Sensor Network for Monitoring Scottish Peat Land

The Glen Feshie sensor network[2] is being developed to monitor hydrological processes of a montane environment in the Scottish Cairngorms. The sensor nodes in this network feature end to end IPv6 connectivity, This allows direct communication between the nodes and remote servers. These nodes use the internet for reporting the readings as well as providing a user interface for configuring nodes.

The network used at Glen Feshie is largely based on the systems trialled in the lab temperature monitoring network discussed in section 3.2. Contiki was again used as the operating system for this network, with modifications to allow for the use of 868MHz CC1120 radios (Bragg et al., 2016). The sensing nodes used the Zolertia Z1 platform

[2]http://mountainsensing.org

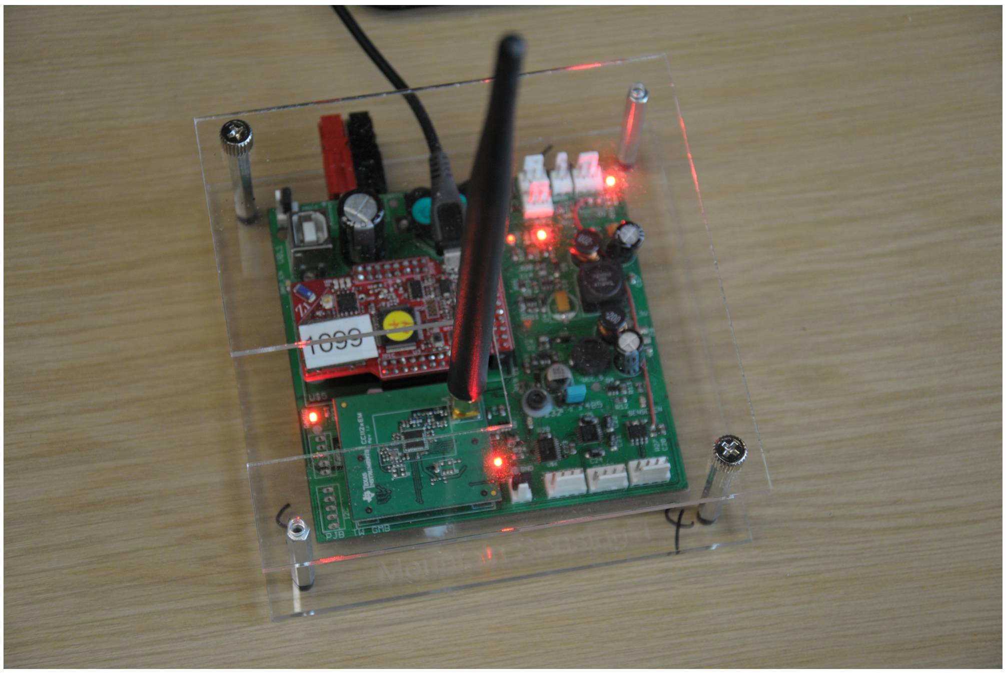


Figure 3.4: Photograph of the MS1 sensor node with the Z1 and CC1120 modules mounted

which were used in the later stages of the lab network experiments, as these have more memory and were more readily available than the Tmote Sky nodes. The Z1 and a CC1200 module was mounted onto a central PCB which also contains the external sensor interfaces. Figure 3.4 shows a photo of a node with the CC1120 and Z1 installed.

Similar to the Glacsweb surface network, the nodes at Glen Feshie communicate using CC1120 868MHz 802.15.4 radio modules from Texas Instruments. However, rather than a custom protocol, they use standardised 6LoWPAN connectivity above the radio layer. The system uses protocol buffers[3] to encode the sensor results. This data is then sent as a HTTP post message to the border router. While end to end connectivity could have been used, sending the data to the border router allows it to perform store and forward operations.

In order to provide a friendly configuration and status interface for the node, Hyper Text Markup Language (HTML) encoding over HTTP was used. This is a significant improvement over the serial console configuration method used for the Glacsweb network. Although the HTML/HTTP/TCP combination is not very efficient with regards to its communication requirements as identified in section 2.2, this is the standard combination used on the world wide web. This enables users to be immediately familiar with the configuration interface. It would allow new nodes to be added, or even entire new networks to be deployed, without requiring significant expertise in sensor networks.

[3]https://code.google.com/p/protobuf/

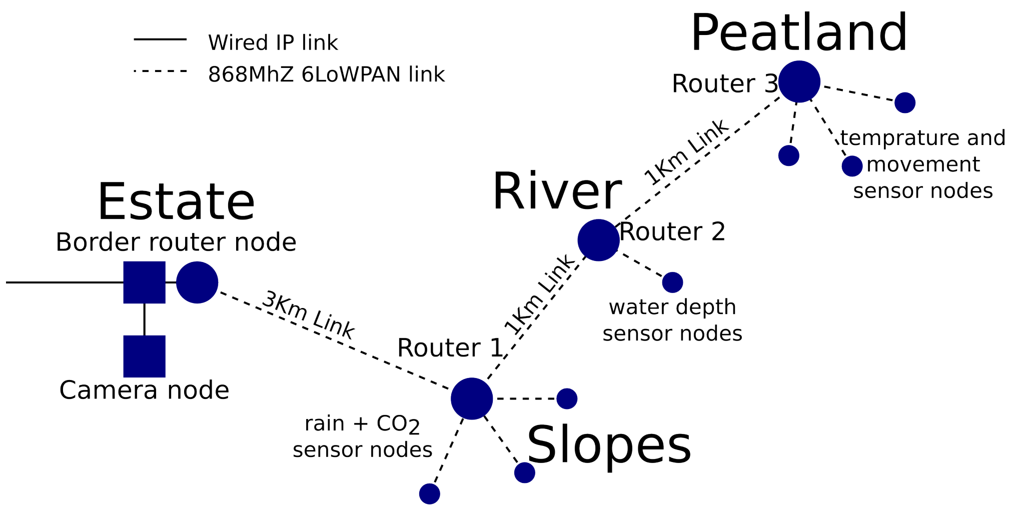


Figure 3.5: Deployment of Router and sensor nodes for the Glen Feshie sensor network

Unlike the Glacsweb system, the nodes do not go into a power down state for long periods of time, instead the central processor and radio remain active, however, the external sensors are still powered off. The main reasons for this decision stem from the choice of Contiki as an operating system. Contiki does not have inbuilt support for long duration sleep cycles, to introduce this would involve a significant amount of changes to enable, and increase the possibility of adding bugs which would not be seen until deployed. In addition there were concerns about how sleep states would affect the dynamic routing [Routing Protocol for Low power and Lossy Networks (RPL)] used on the network as available routes on the network might change while the node was asleep. The option to have the nodes always on is only possible as the nodes are not space constrained, and as a result can be large enough to contain the significant amount of battery power to remain on all the time. While a full shutdown was considered, limitations within the Contiki operating system and routing system made choosing an always on operational mode the preferred option. Although the node was unable to take long sleeps, the combination of Contiki and ContikiMAC allow for it to sleep for brief periods of time and still be able to respond to radio messages.

The lack of long sleep states in the Glen Feshie system has the advantage of improving the usability web based configuration interface. Putting nodes into sleep states would cause an issue with this method, as the configuration interface will be unavailable while the node is asleep. Having short time periods when devices can be configured adds an additional layer of knowledge and organisation required by the end users to configure the nodes.

In the summer of 2015 several modifications to the node software were installed. The most significant of these was the move to CoAP for upper layer communication. The

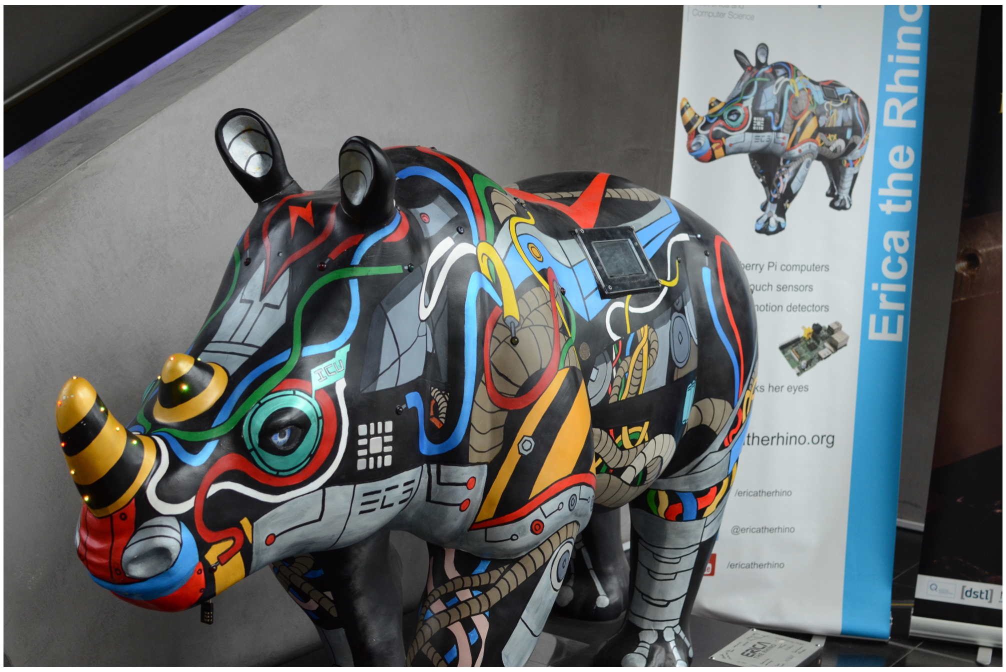


Figure 3.6: Erica the Rhino

move to CoAP removed the need for TCP connections, which greatly improved the performance of data transmissions over the low power network.

3.4 Erica the Cyber Rhino

Erica the Cyber Rhino[4] is an interactive artwork built around a network of IoT devices. Erica was created in order to promote rhino conservation as part of the Go Rhinos community artwork project from Marwell wildlife[5], as well as facilitating electronics and computer science outreach activities.

The internal systems of Erica consist of five Raspberry Pi single board computers. For the initial deployment these were the Pi model B rev2s but were later changed to Raspberry Pi 2s during a series of upgrades after the conclusion of the Go Rhinos deployment. Each Pi manages a separate part of Erica's systems and has a daughter board that contains the control hardware required for that system.

The network Pi manages Erica's connectivity capabilities, using both wireless and wired networks. A combination of Ethernet 802.3 and WiFi 802.11 are used as the physical and data link layers for both internal and remote links. Erica uses IP connectivity both internally, to communicate between Pi's which run the subsystems, and externally. The external connectivity is used to provide additional inputs to Erica's reactions such as social media mentions and weather data, as well as allowing remote monitoring.

[4]http://www.ericatherhino.org
[5]http://www.marwell.org.uk

Erica's ears and lighting are controlled through the Mechatronics Pi. The ears are controlled by General-purpose input/output (GPIO) driven stepper motors. In the initial version the lights were controlled using Serial Peripheral Interface (SPI) Pulse Width Modulation (PWM) controllers. In the upgrades, this was replaced by a system using the DMX512 lighting protocol. One Pi is used for each of Erica's eyes, these perform the vision processing and control the eyes mechanics. The image processing software on the eyes allows for Erica to react to things like faces and QR codes.

The remaining Pi controls Erica's interaction responses and is referred to as the Brain Pi. This receives the responses to external stimuli detected by the other Pis and then triggers the necessary Pis to perform appropriate reactions. The first iteration of the system used HTTP to transfer messages, however, this was changed to use Message Queue Telemetry Transport (MQTT) during the upgrades. MQTT is a protocol often used for sending messages between different systems using a publish subscribe model. While using a message queue has more overhead than solutions such as HTTP, the networking structure and availability of computing resources means this is not a concern. Figure 3.7 shows how, after the upgrades, the subsystems used in Erica link together.

Erica presents a very different style of deployment to the low power deployments discussed earlier. Although only a single unit, from external perspectives the distributed nature of Erica's internal systems are similar to how data can be passed around in other IoT networks. The increased power and connectivity resources available on Erica when compared to the sensor network deployment scenarios, allows the use of far more demanding protocols and systems than would be possible on constrained systems. This resource rich environment is typical of many current IoT devices which rely on mains power and communicating to remote systems for device management.

The systems and development of Erica along with the results of her deployments have been published as 'Erica the Rhino: A Case Study in Using Raspberry Pi Single Board Computers for Interactive Art' in the "Raspberry Pi Technology" special issue of the MDPI Electronics journal (Basford et al., 2016).

3.5 Improving the Deployability of the IoT

Traditional bespoke sensor networks have long struggled with ease of deployment issues and data access difficulties across the sensor network boundary. Through the use of IoT technologies and frameworks it is possible to create networks with significantly less development time than creating custom bespoke systems. The layered architecture offered by the IoT can also aid deployment of networks by offering convenient configuration interfaces whilst simultaneously providing end to end connectivity for data transfer. These benefits come at a cost, the current IoT operating systems have not been designed for EWSN use, and because of this consume significantly more energy than bespoke systems.

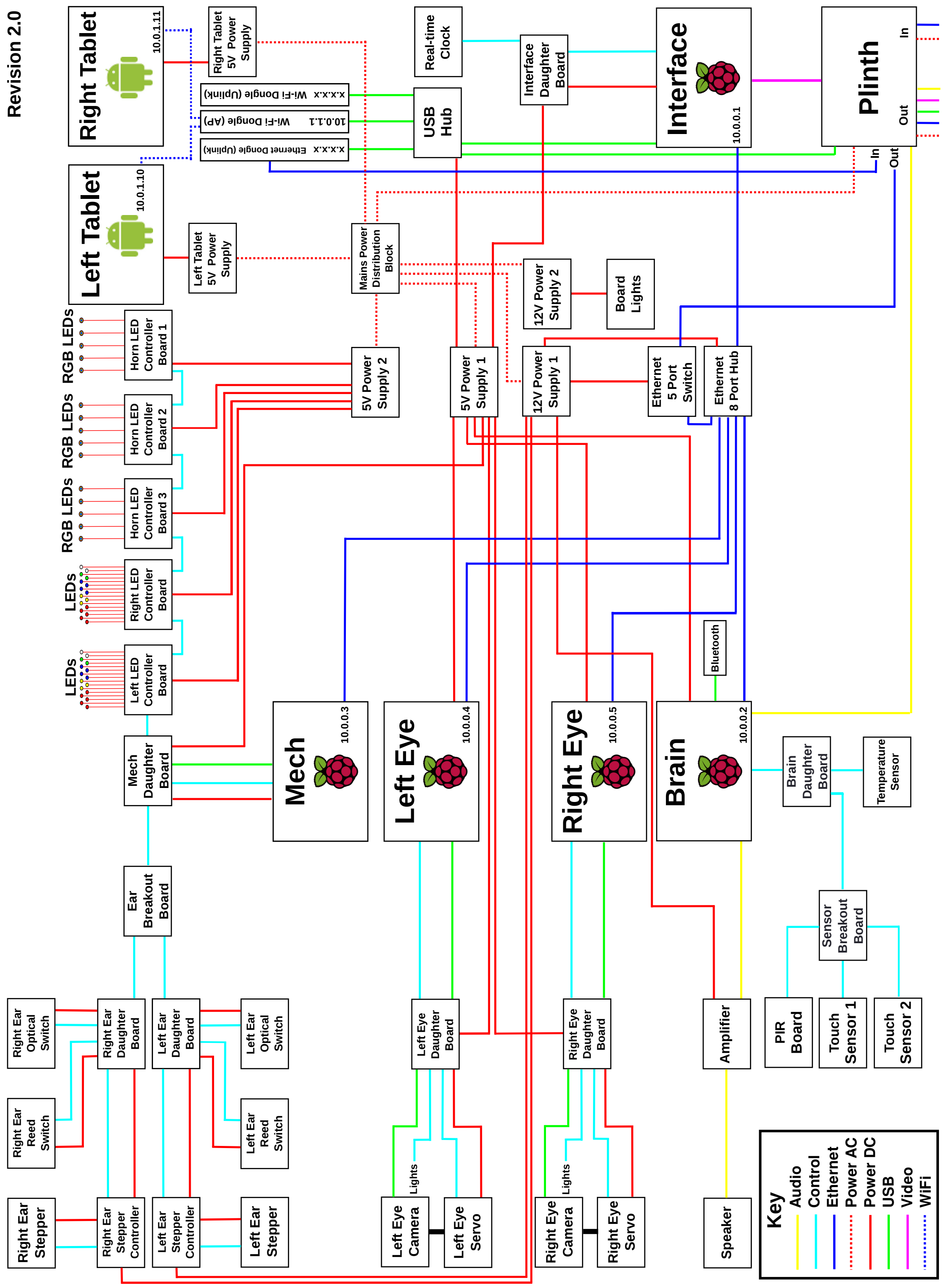


Figure 3.7: Hardware connections within Erica

The major factor limiting the usability of IoT based systems for EWSN applications such as in the Glen Feshie network, is the increased power consumption when compared with existing systems. In order to provide the small form factor and long lifetimes required for many sensor network applications, power consumption must be reduced. One way to make substantial reductions in power draw is to move away from the always on model and move to using sleep states like many sensor networks. Incorporating DTN capabilities into these sensor network frameworks would allow these sleep state savings without any additional development time.

Chapter 4

Sleeping Devices in the Internet of Things

The current generation of Internet of Things (IoT) systems mostly rely on using always on devices. Even with low power radio technologies, always on devices do not have the battery life necessary for IoT deployments such as environmental monitoring. For instance the low power nodes in the Glen Feshie network discussed in section 3.3 were always on devices. Despite being powered by large lead acid batteries, they only had enough energy reserves for around half a year.

A common method of reducing power draw in non IoT sensor networks is to put the sensor into a low power sleep state where the radio and other peripherals are turned off in order to minimize power usage (Akyildiz et al., 2002). In this configuration, the sensor nodes will wake up from a low energy sleep state to perform readings and communications and then return to a sleep state once those actions are completed. This energy saving method conflicts with the common expectation with internet connectivity that a device will be connected all the time it is in an operational state. Networks with these properties are often classified as challenged networks (Bormann et al., 2014).

4.1 Solution Requirements

Clearly a solution needs to be found to allow energy saving through the use of sleep states whilst still maintaining connectivity. Compatibility with existing systems is an important consideration given the quantity of existing network infrastructure currently deployed. While changes to end devices will inevitably be required regardless of the solution, it is unrealistic to require changes to the entire network infrastructure.

The internet operates over a variety of link layer technologies such as 802.15.4, 802.11, 802.3 and many more. Making alterations to existing link layers is impossible and limiting the solution to use only selected link layers has the same problem as current store and forward systems on sensor networks. Because of these reasons any potential solution will need to work with the existing link layer technologies, meaning that it needs to be implemented in the network layer or above.

Any solution will naturally lead to increased overheads or restrictions to the sensor or network. These overheads come in the form of both software complexity and additional power usage either through additional communications or increased packet processing. Minimizing the overheads and restrictions introduced is an important factor when developing solutions.

Unlike other sensor networks where there is a known deployment setup, IoT deployments can take many forms. To prevent fragmentation of the IoT ecosystem, solutions should work over a multitude of deployment styles; this will allow for a single solution rather than a mix of different solutions.

4.2 Network Topologies for Sleeping Nodes

In the Internet of Things there are a wide range of potential network topologies; different topologies lend themselves to different application areas and deployment styles. Each of the different topologies will have a different set of requirements in order to support sleeping devices.

The most simplified form of network for sleeping nodes is where there is an always on gateway which all the devices connect to directly. An example of this topology can be seen in Figure 4.1a. This is the type of network topology which would likely be used in home and office environments, where a border router can cover the entire deployment area. In this setup the border router and external devices will have reliable power and data connections. Sleeping nodes in this environment have reliable access to internet connectivity when they power up, as the rest of the network infrastructure is an always on system. Remote systems have reliable access to the gateway device, which will allow for it to take an active role in supporting the sleeping devices.

On the other end of the connectivity scale, environmental sensor networks are often in remote areas where reliable power and connectivity is not available. In order to overcome these issues, the gateway is often fitted with a long range connectivity option, for example, satellite modems, long range WiFi links or GPRS. These gateways are usually battery powered, so they also need to enter low power states when possible in order to conserve energy. This means that several hops in the communication chain need to go through these sleeping nodes, this can be seen in Figure 4.1b. An additional complication in environmental networks is that communications are often less reliable than the above example. For example, in the Glacsweb deployment, the long range links used to communicate between the surface stations and the outside world stop working once the antennas have been buried by the winter snowfall.

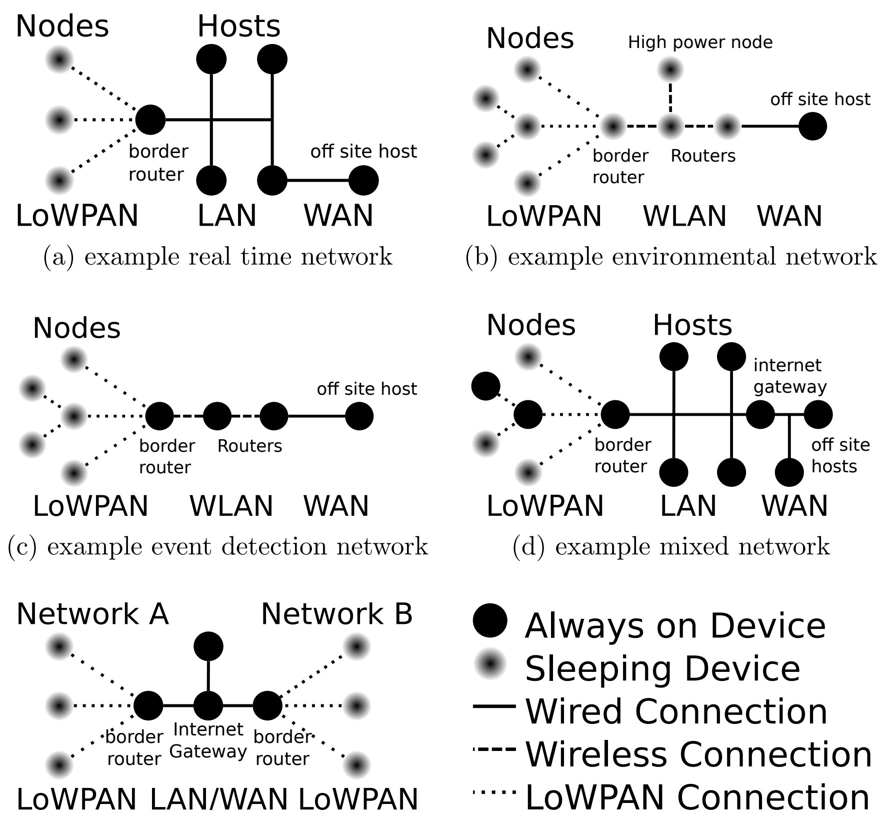
Sense and store sensor networks, like the ones mentioned above, tend to condense transmission of results into infrequent communications periods. For example Glacsweb performs communications once a day. Due to the reduced number of communication intervals in these networks, the communication intervals can be longer, allowing for more tolerance in time discrepancies in the network. The multiple hop network structure reduces the reliability of communications to the border router as there are more links which can go wrong. Hopping networks which sleep will need to remain synchronised in order to make use of other nodes during the communication interval. In addition, it is possible that an end to end communications link, where all links in the chain are up, may not be available.

Some environmental monitoring networks are used for event detection and reporting. These are used in situations such as disaster warning or other scenarios where knowledge of events occurring is time sensitive. Because of the time sensitivity of the data, these networks often have more reliable backhaul networks than conventional EWSNs. Figure 4.1c shows such a network topology. These networks allow for greater reliability of communications between the border router and external devices, avoiding many of the additional problems with EWSNs over home and office networks.

As IoT deployments begin to use common protocols we will see a rise in the use of mixed networks, where several types of device share the same border router and network infrastructure. For example, in a home environment there will be IoT devices which will have continuous power, such as household white goods as well as devices running on battery power. The mix of nodes in the network allows for always on nodes to act as a reliable backbone for sleeping nodes in the same network. Figure 4.1d shows an example of such a network.

A major challenge with current systems is internetwork communication, such as in Figure 4.1e. Currently, in such networks both nodes would need to be online at the same time in order to communicate. This requires synchronized communication periods, meaning that both networks would need to have been deployed with the knowledge of the schedules in the other network. Where one node communicates with several other nodes, it will need to coordinate its communications with the schedules of all of the other nodes, and these may be completely different from each other.

All of the topologies discussed above require different levels of support from any system to enable the use of sleeping nodes. To prevent fragmentation of the IoT space, a system to support sleeping devices would need to be able to fulfill all of the above cases. Without a common system for sleeping device support, a similar issue with current fragmented technologies for sensor network deployments is very likely.



(a) example real time network

(b) example environmental network

(c) example event detection network

(d) example mixed network

(e) example multiple network setup

(f) Key for the network layouts

Figure 4.1: Potential network layouts evaluated. Republished from Ward et al. (2014)

4.3 Communication Solutions for Enabling Sleeping Devices

In order to allow bi-directional communications with low power Internet of Things devices, a better solution than the workarounds discussed above will be required. For successful communications, messages will need to be sent and received in the device's communications window. The methods to facilitate this fall into two categories: correctly timing messages so that they fall into the communications window, and adding new networks technologies to allow for messages to be delivered during the communications interval.

It has been suggested that the way to resolve this issue is to have the sleeping device initiate all communications (Ostermaier et al., 2011). The node itself will inherently know when its communications are available. This method is also used in many always on IoT devices, but for reasons of NAT traversal rather than low power sleep states.

In this setup, low power devices will need a reliable connection to a remote server that will manage the devices communications. This can not always be relied upon, especially if the node communicates with a device over the internet; it would be unhelpful if your heating was unable to communicate with its thermostats because the internet was down. In several cases, such as Violet and their Nabaztag product or Nest and the Revolv product, the phone home server for such devices has been shut down rendering them inoperable (Volpe, 2011; Price, 2016).

Without the ability to communicate with the node directly, altering the configuration of the node would need to happen through the remote server. The remote servers of the nodes communication will need to act as application layer gateways for the low power node. As discussed in Section 2.1.2 application layer gateways provide many disadvantages and should be avoided if possible.

When there are lossy links in between the node and the remote server, the node will take the transmission burden of the failed communication attempts. This is because it will need to transmit the initial communication rather than waiting to receive one. The effect of this on the node's overall power budget will be based on the amount of energy used to transmit, and the reliability of the network. A similar scenario can occur if systems are no longer interested in data from a node for a period of time. If the system either does not, or is unable to, inform the node this is the case, then the node will not be aware and will continue to transmit data which is not being utilized.

Rather than having the node initiate communication, the remote server could initiate it instead; to do this it will need to send its messages during the communication window. In order to time the messages correctly, both endpoints of the communication will need to maintain a synchronised clock, as well as a record of when the link and any intermediate nodes with sleep states will be available. There are many methods to achieve a synchronised clock across devices, the most popular being the Network Time Protocol (NTP) which is in wide use in many internet connected devices. Unfortunately, maintaining a synchronised clock comes at additional hardware and power budget costs.

There are no current technologies which allow devices to communicate when a link will be available. This could be implemented in several ways, for example, a similar way to IPv6 Maximum Transmission Unit (MTU) discovery could be used where a feeler message is sent out and if the host is not available then the border router could return an error which includes the nodes schedule. Alternatively, link availability could be provided to the endpoints from an external service, for example a DNS record.

To support such a system, both ends of the link will need to support these features. This complexity could be reduced by having always on devices act as intermediaries; they would need to know the schedule but remote devices can avoid this complexity. By doing this you have an intermediary which needs to know how to fetch data from the nodes in effect an application layer gateway, removing one of the main benefits for moving to an end to end IP system.

Another method for supporting sleeping nodes would be to add the capability for packets to be held until a communications link comes back online. A similar feature called store and forward is often used in non-IP connected sensor networks. While this can be achieved by simply delaying the packet, almost all protocols have timeouts at the higher levels which will result in the packet being ignored or mishandled if too much time passes before a response is received. These packets would still end up consuming resources when sent and processed, even though the responses are no longer relevant. Picking a generic timeout for packets would result in some being delivered when they are no longer of use and some which are still relevant being discarded. In addition, the sender of the packets would not know whether the packet was being held or if it had simply been lost.

In order to resolve these issues, additional meta data will need to be transmitted along with the packet. This extra data would need to store how long the packet is useful for, and to request status updates. With this extra information, packets which are no longer of use can be identified and the sender can remain informed of the packet's progress.

4.4 Framework for Testing Network Implementations

In order to compare the two solutions above it is necessary to evaluate their performance on a test network. To evaluate all of the potential solutions in the appropriate test cases using real sensor network hardware, would take a considerable amount of time and resources. Instead of using physical hardware, simulated networks can be used instead. By using a simulated system the time and cost of deploying and configuring networks can be avoided. In addition, it is easier to simulate network interruptions and anomalies in a simulated environment than it is to create such an event in real life.

While there are many existing low level network simulation tools such as NS3 and Cooja, these require completed protocol implementations in order to be used. As protocols do not exist for either of the solutions, using such a tool would require the development of a protocol and an implementation for each system first.

The use of a higher level simulator simplifies the development work required to test the protocols, as it is only necessary to be concerned about the high level concepts of the protocol rather than the implementation specifics. In order to maintain compatibility with existing systems, any protocol developed for supporting sleeping devices would need to use the same MAC and physical layers as current internet systems. As these layers will not be altered by any systems for supporting sleeping nodes they can be abstracted to part of the cost for using a communications link. Abstraction of the lower layers makes the network layer the lowest layer which needs to be simulated.

The payload of the packet can also be abstracted from the simulation. While different nodes will have different application layers, the content of these layers will not be investigated by other devices on the packets path. In addition, as the payload will not be altered, the reaction of the node to a given packet will be the same. Therefore the payload can be abstracted down to a message type, for example a request, and a length.

As well as the abstraction of the payload and low level layers the network layer can also be abstracted. Rather than encoding and decoding the network header each time, it can be treated as a set of parameters. Further abstraction by avoiding using network addressing can be achieved, instead internal simulator identifiers can be used for message source and destinations. As the network header is still a fixed size we can treat this parameter list as a fixed size block of data when sending packets in the simulator.

The inclusion of additional features will inevitably have a processing and network transmission cost associated with it. However, until the protocols can be developed, the cost of these changes will not be known, as a result they can not be fully taken into account within the simulator but will need to be considered when analyzing the results.

There was not an existing simulation tool which fulfilled the requirements above. Therefore it was necessary to develop a new tool for this purpose. The simulator was developed using the SimPy discrete event simulation framework for Python (Muller and Vignaux, 2003). All of the elements in the simulation can be modelled using discrete event triggers which can be from internal timers or network activity.

In order to provide an accurate picture of the network, links and routers will need to be simulated as well. The developed simulator operates on the principle of interconnected nodes and links. A node represents a device such as a router or computer and the link will represent a communications channel. Some systems can be abstracted down to a single link, for example, a Wide Area Network (WAN) connection can be modeled as a single link even though there are several routers along the path. However, some links such as those with border routers on the path will require a node as we will want to simulate changes to this node.

As packets are represented in an abstract format they are able to store a log of activities that have been performed on them as they passed through the network. This allows tracking the path of every packet across the network and to evaluate metrics about the packet's transmission.

Links in the network are implemented as SimPy processes which take and deliver messages to connected nodes. The links were designed to support simulation of various network technologies. This is achieved with several alterable parameters such as latency, jitter and packet loss. The transmission time of packets is modelled using the average latency with a jitter added using a standard deviation model, which takes into account the nodes processing time for receiving a packet. Packet loss is modelled by randomly dropping packets based on the link's loss rate. The values for these parameters were taken from measurements conducted on various deployed networks. For wireless links there can only be one message on the channel at one time, this is modelled by delaying transmissions until the channel is clear, representing the use of a clear channel assessment system.

Nodes represent computers and other devices in the network such as routers. The basic model which the nodes are built on provides one or more interfaces to be connected to communication links. On top of this basic model, each node can implement different capabilities and features. Routers allow for received packets to be directed to the appropriate interface. The routers were configured with static routes as the node layout is known in the simulator and implementing a dynamic routing system in the simulator would provide no advantages over static routes. The low power nodes in the network were set up to send a response to request messages they received. These nodes also followed a sleep schedule which controls when they are able to process messages, messages received outside of its active periods are ignored. The request nodes were configured to send periodic requests to nodes and to monitor the success and failure of those requests. For some of the scenarios, nodes were given additional features such as proxy interfaces or store and forward capabilities.

4.5 Simulations of Potential Solutions

Using the simulator discussed above, initial experiments were carried out using the home/office setup discussed in section 4.2. In this trial network, sensor nodes were configured for a two second communication period every minute. This could be a potential configuration for a device such as an ambient temperature sensor. With this timing the nodes would be in a 1/30th duty cycle which would greatly reduce power consumption while maintaining frequent updates from the sensors.

The main statistics which were used for this initial analysis were the amount of traffic on the low power network and the success rate of the queries sent. Figure 4.2 shows these statistics for several potential solutions. This test was run for an hour of simulated node time. The upper of the two graphs represents the amount of requests sent by the host; requests in green had responses, and requests in red were not responded to. The lower of the graphs shows the amount of packets sent on the low power network; dark blue indicates messages sent from the border router, and light blue messages received by the border router.

Figure 4.3 shows the timeline of the first 200 seconds of the simulation. The cumulative count of packets over time is shown by the coloured lines, each line representing a different configuration, these use the left hand vertical axis. The shaded boxes in the

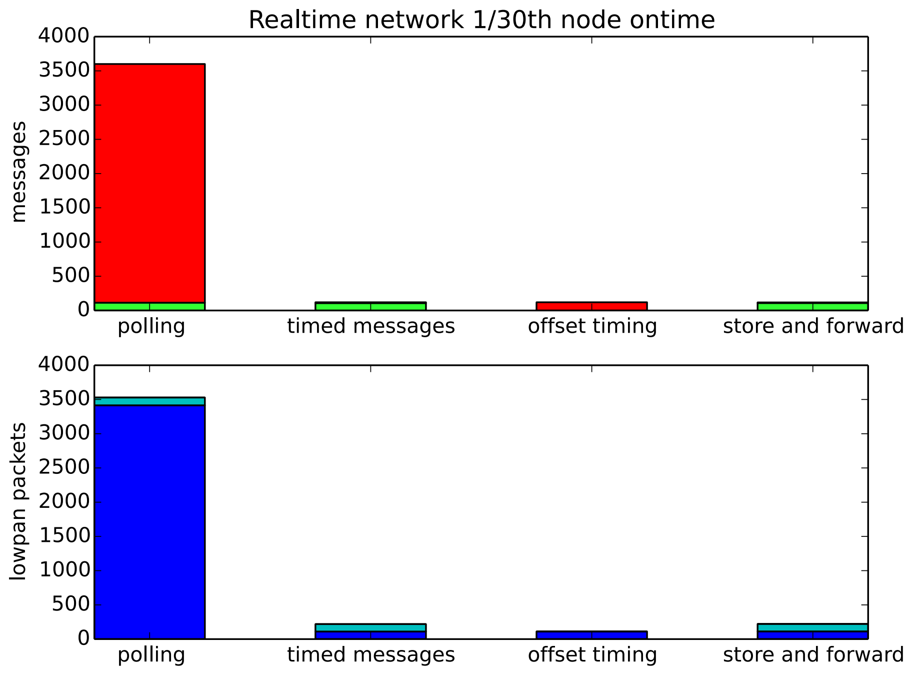


Figure 4.2: Success rates and quantity of messages sent on a real-time sensing network.

top graph show where the low power node is powered up and can be communicated with. Successful messages are shown as dots, using the same colour as the line for that configuration. The dots are placed at the point of transmission with ping times on right vertical axis.

For comparison, repeated polling of the node was included. Repeated polling of the node can be clearly seen to cause a lot of network traffic for an approximately 3% success rate. This is to be expected, given the nodes duty cycle and the fact that it needs to try at least every ten seconds to catch the nodes communication window. While this method manages to catch the node, it is very inefficient with regards to network utilization and will get worse with lower node duty cycles.

If the requester can match schedules with the sensors then very efficient communication is possible. If the systems become out of sync then communication breaks down. From the requesters point of view it is unable to tell the difference between a loss of synchronisation or the remote device having malfunctioned. Using a store and forward solution allows for successful communication whether timing is out of sync or not. When the timing is out of sync there is a delay on the message response as the packet is held until the next communication interval.

When creating an EWSN testbench for solutions the network is configured to use a single communications session per day. As there is more data to be sent in that communications

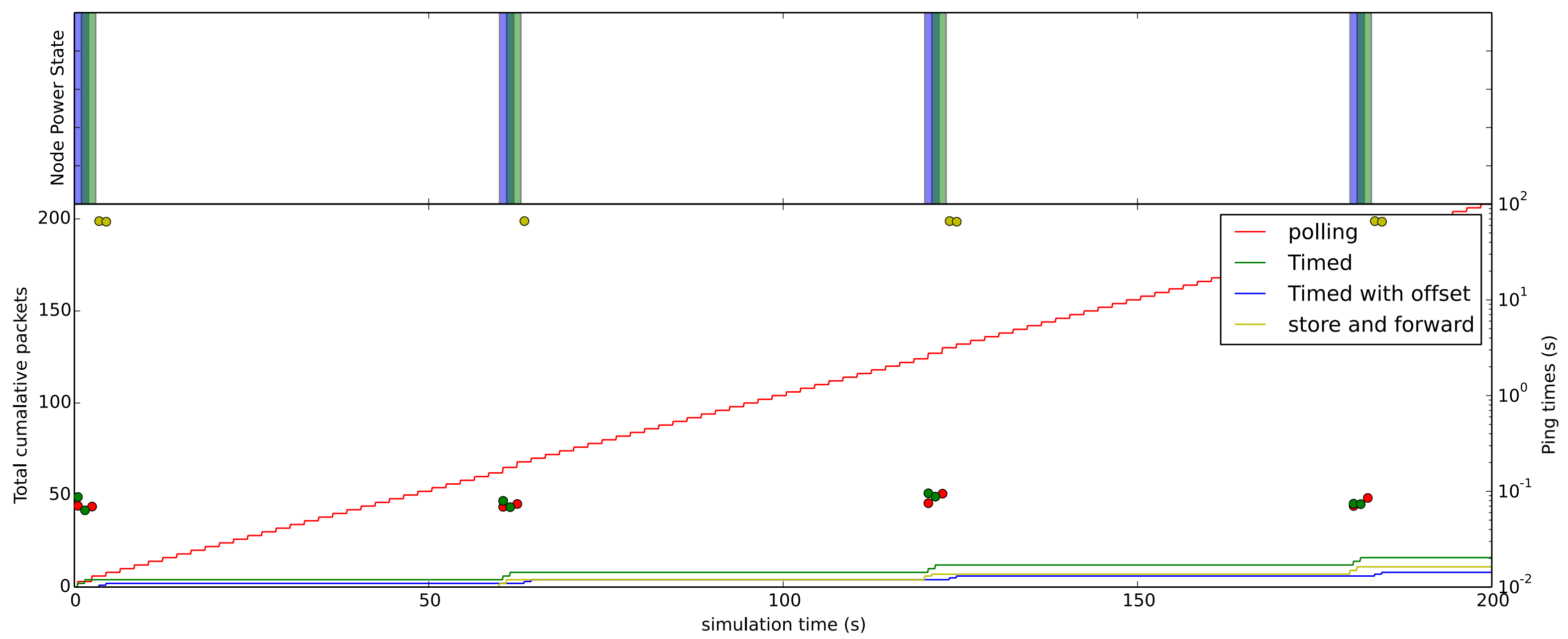


Figure 4.3: Timeline of a realtime network. A cumulative count of packets is shown as lines using the left axis. Successful messages are shown as dots at the time of transmission with ping times using the left axis. The coloured background bars on the upper graph show when the two low power nodes are powered up.

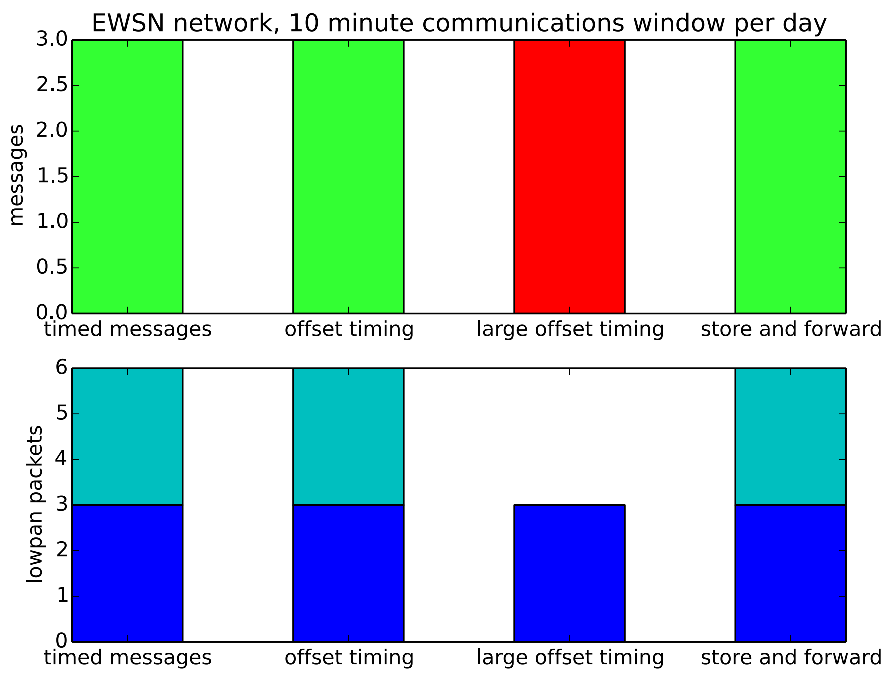


Figure 4.4: Success rates and quantity of messages sent on the environmental sensor network testbench using a single daily communication interval.

period it will take longer to transmit than the example above. With a reduction on the quantity of communications intervals, the time where the node is available can be increased without a major impact on the battery life. The longer communications interval provides greater slack on timing intervals when attempting to match schedules.

Running the simulation on an environmental network produces results that can be seen in Figure 4.4. As expected, the longer communications interval avoids small discrepancies in timing calibration from causing a communications breakdown.

Although not shown in Figure 4.4, environmental sensor networks often have links that become unavailable for periods of time. For timed solutions these would prevent communication, however, with a store and forward system these messages will wait to be transmitted until the link is back up. When there are many interruptions like this, a complete end to end communication path may only be available infrequently.

Event networks have a lot of the deployment traits of environmental sensor networks; as a result they have a similar outcome to the EWSN testcase. Figure 4.5 shows the results of the simulation.

When dealing with communication between multiple low power networks, synchronisation becomes difficult as each network will be running on their own schedule. Figure 4.6 shows what happens when two networks on differing schedules attempt to communicate. It can be seen that only the store and forward based solution is able to support

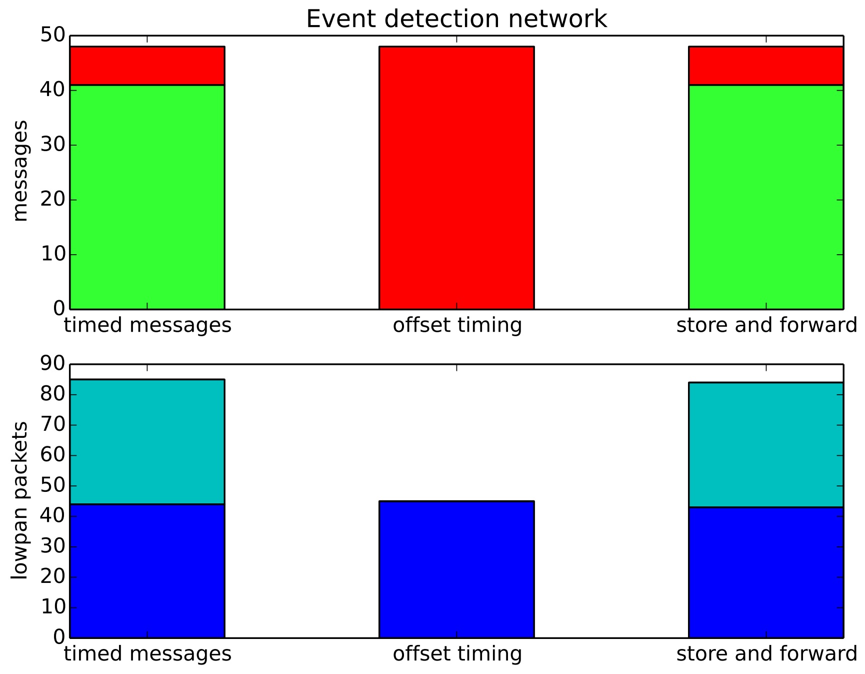


Figure 4.5: Success rates and quantity of messages sent on an event detection network.

communications between these networks. Matching the schedules of one network to the schedules of the other, as seen in Figure 4.7, allows for timing based solutions to work, however, this restricts the options for the network deployment. When there are multiple data sources feeding into a single low power data collector, (for example, several soil moisture sensors feeding into an irrigation controller), the node receiving the data would need to turn on for each of the other nodes communication periods or have all the other nodes follow the same schedules.

Mixed networks incorporate features from several of the above cases, and as such need a solution which can work for all of them. Throughout the examples above it has been assumed that the schedule of the sensor node is predictable. For some devices, such as those using energy harvesting, their schedules might not be known in advance as they may be altered based on available energy reserves (Basford, 2015). With such implementations, the timing solutions discussed above would need an indication that the schedule has been altered.

Figure 4.8 shows a summary of the results from the simulation framework during a simulated 24 hour period. The top and bottom rows of the graph show different measurements from the network. The top row of graphs shows the amount of requests and those that received responses, the green bar shows requests which received a response while the red bar shows ones that did not. The bottom row of graphs shows the amount of messages traveling over the sensor network; dark blue indicates messages sent from

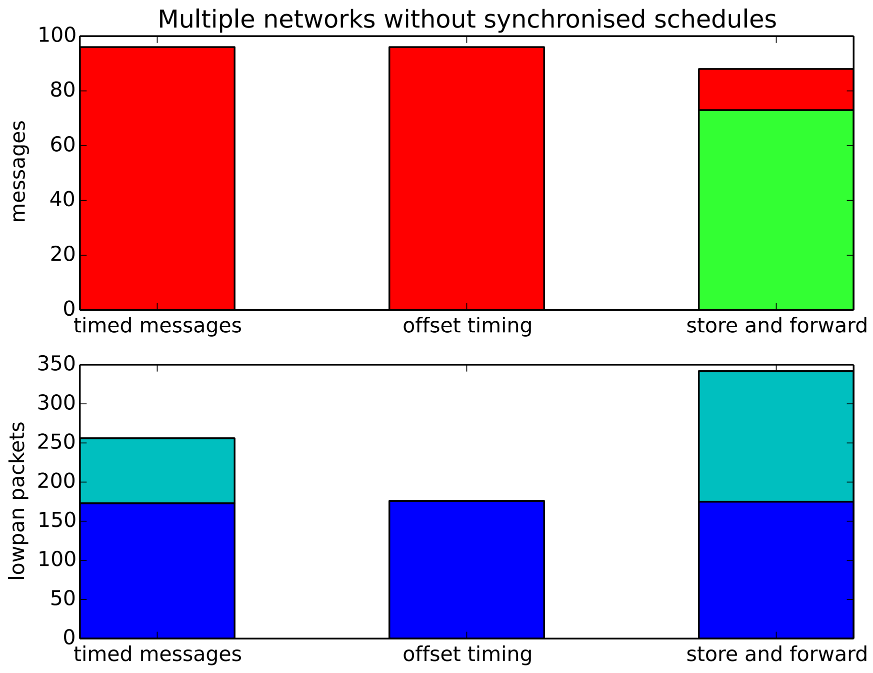


Figure 4.6: Success rates and quantity of messages sent on a multiple network testcase were the networks schedules where not synchronized with each other.

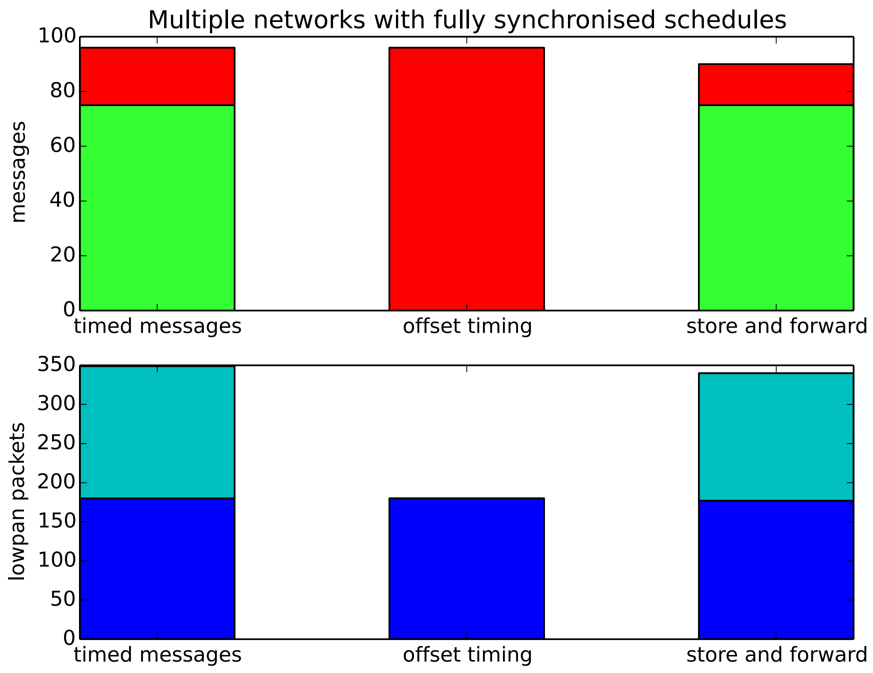


Figure 4.7: Success rates and quantity of messages sent on a multiple network testcase were the networks schedules where synchronized with each other.

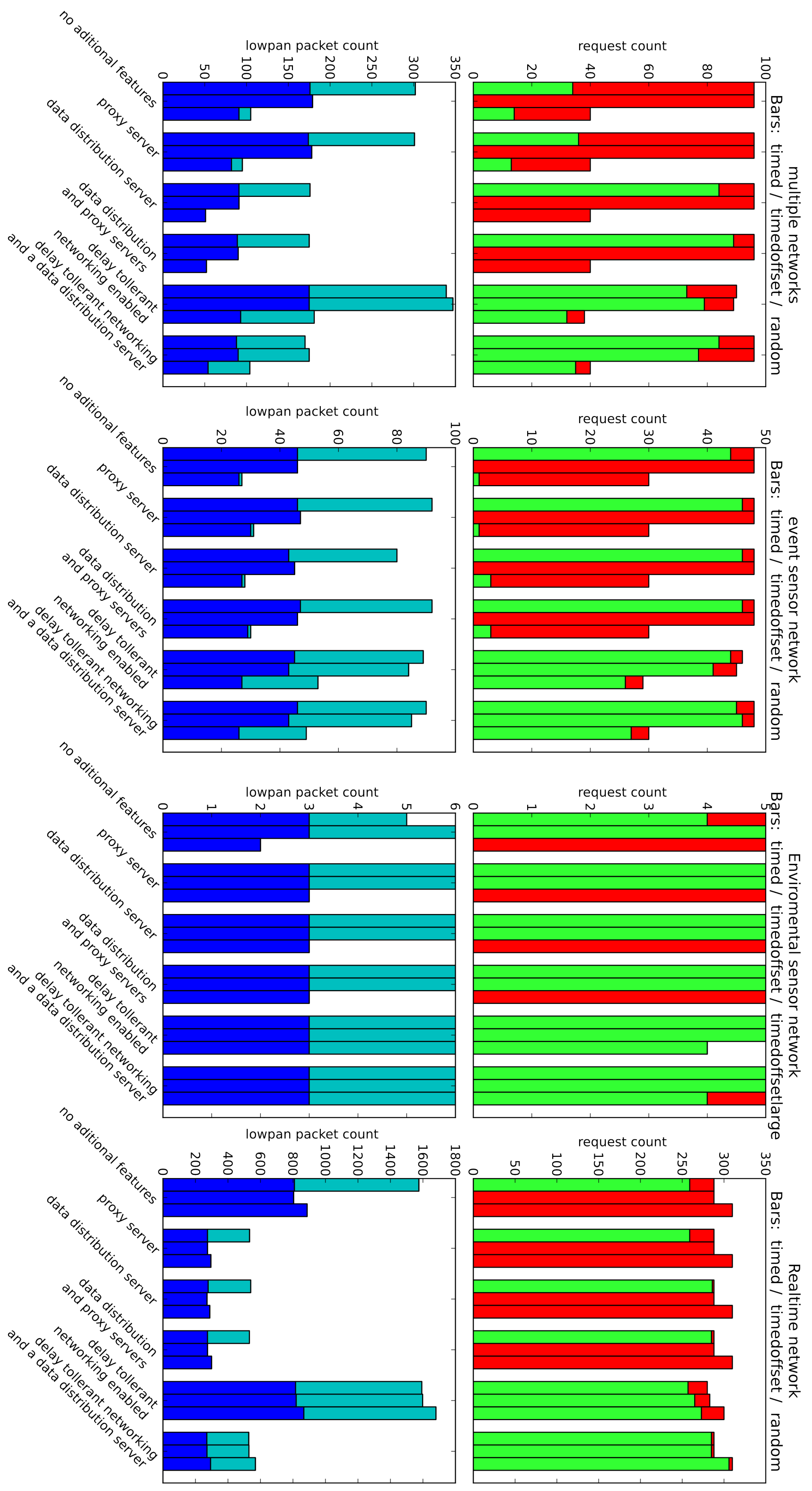


Figure 4.8: Summary of request success and packets sent and received during a 24 hour simulation run. Republished from Ward et al. (2014)

the border router, and light blue messages received by the border router. Each vertical pair of graphs shows a different network configuration as described in section 4.2. Within each vertical pair, several network technologies are included. These are shown as a group of bars, the features in use are shown in the bottom legend. Each of these groups contains a series of timing possibilities for the requests, these are shown at the top of the graph.

In the graph it can be seen that when time synchronisation cannot be maintained, connectivity is lost or is very unreliable. This is avoided when store and forward has been enabled on the network. The effect is even more noticeable in the multiple networks scenario, where there are multiple networks with different schedules that need to be traversed.

Although it would appear at a first glance that the more reliable solutions are less efficient as they had more packet transmissions, this is because they are getting replies to more of the requests for data, hence more packets are transmitted. Although messages from the router to the node which are not received, due to the node being in a low power state, do not cause any load on that sensor, they can affect other sensors on the network. A node that is on will still receive the message and will need to expend energy to determine if that message is intended for itself. In addition this increases the likelihood of sensor nodes needing to back off before transmitting, which consumes additional power.

4.6 Analysis of Solutions

While the correct timing of messages is successful in many cases, it falls down when dealing with multiple networks that need to communicate between one another. The method of transferring this scheduling information will need to be carefully considered. While in small networks scheduling information can be broadcast out, on larger networks this becomes troublesome, and by the time you get to the scale of the internet it is simply impossible. Therefore the remaining options are to publish schedule information or to allow other devices to query the schedule from upstream nodes.

Throughout the above simulations using the store and forward solutions allowed for reliable data transfer. The store and forward option avoids the need for end to end schedule and clock synchronization, although some synchronization might be needed on a hop by hop basis. While this avoids end to end time synchronization it will add an overhead to the network communications. This overhead comes in the form of the additional timeout information. This is required to prevent stale packets remaining in the network.

The lack of support for multi scheduled networks in the timing based solution breaks the requirement for a single solution for all network types. The store and forward solution is compatible with the requirements identified in section 4.1 although consideration will need to be made to ensure support on multiple link layers. Delay Tolerant Networking (DTN) uses the same concepts as store and forward but is designed to work at the network layer, rather than the application or data link layers and as such it offers a promising method for implementing store and forward for the IoT.

The simulations and results discussed in this chapter have been published as part of the "Simulated analysis of connectivity issues for sleeping sensor nodes in the Internet of Things" paper (Ward et al., 2014). This paper was presented at The ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM) and can be found in The ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN) conference proceedings which was co-located with MSWiM.

Chapter 5

Delay Tolerant Networking for Constrained Devices

As previously discussed, one of the areas in which IP falls behind other network technologies used on sensor networks, is its lack of store and forward functionality. The concept of Delay Tolerant Networking (DTN) was originally proposed by Vint Cerf (Cerf et al., 2001) (Cerf et al., 2002) as part of creating an interplanetary internet which offers a solution to this. In the interplanetary internet environment, there is likely to be significant time periods between transmission and reception of messages, as well as periods where communications are not possible due to orbital positioning or shared use of directional communications systems such as the deep space network. The challenges faced on the proposed interplanetary internet are very similar to those being faced by low power sensor networks.

The IETF standards track, has been developing the bundle protocol (Scott and Burleigh, 2007) to allow delay tolerant packets to be transported over IP and other networks in use on the deep space network. As identified in section 2.3.2.1, there are several issues when using the bundle protocol on networks with limited energy and bandwidth capabilities. The main issue with using the bundle protocol on constrained terrestrial networks is that it is an encapsulating protocol, this causes it to have an increased packet size as well as requiring intermediate devices to support the bundle protocol to correctly handle the packet. Despite its unsuitability for IoT networks the bundle protocol provides a useful reference for design choices that need to be made while developing an improved solution.

The target environment for a DTN protocol for the IoT are microcontroller based devices such as those supported by the Contiki operating system. These devices have limited memory capacity and processing power, for example the Tmote Sky nodes have 10k of RAM and 48k of Flash and run at 8MHz clock speed. There is also limited bandwidth and restricted packet sizes, 128 bytes per packet for 6LoWPAN. Further restrictions are placed on the protocol by a lack of resources common on current computing platforms such as RTCs as well as needing to run for long periods on battery power.

5.1 Implementing DTN in the IP Layer

As identified in section 4.1, any solution needs to be implemented in the network layer or above. A feature of the Internet Protocol is that it can operate on a range of lower layers such as Ethernet or 6LoWPAN and requiring changes to all of them would be impossible. Previous implementations of store and forward and DTN have either used additional layers on top of existing technologies or completely different network stacks and as a result have suffered with the problem of backwards compatibility. This lack of backwards compatibility makes it harder to integrate DTN functionality into existing systems, often requiring replacement of existing devices with a compatible system. By incorporating DTN into an existing layer the benefits of DTN can be attained without requiring an entirely new system.

5.1.1 Use of Extension Headers

The IPv6 specification (Deering and Hinden, 1998) provides the capability for additional information to be optionally included within the IP layer with the use of extension headers. These headers are included between the IPv6 header and the transport layer header. The extension headers have been designed so that the packet can still be delivered if devices do not understand the headers, although it might not be processed as intended. This allows compatibility with present sensors and routers, that may not understand the headers to support a new DTN protocol.

With encapsulated protocols, data often needs to be duplicated so that it appears in both the IP layer as well as within the encapsulated protocol. Extension headers avoid this repetition by keeping all the options in the same layer, thus avoiding the need to decode the higher levels to access the required information. Many of the other additional features of protocols such as the bundle protocol are already available in the existing standardised IP headers, meaning they are already supported on many networks and IP implementations. For example, the functionality of the bundle security protocol within the bundle protocol is already provided by the Encapsulating Security Payload (ESP) and Authentication Header (AH) extension headers which can provide packet encryption and signing. As a result of this the DTN extension header should contain the minimal amount of additional information required to perform its function. This will help keep packet sizes small when operating on constrained networks.

Within the extension headers there are several ways to implement DTN. The first would be to add a new extension header which complies with rfc6564 (Krishnan et al., 2012).

This was the approach taken by Gomeimoto and Matsumoto (2013) in their implementation of DTN for IP networks. With this method, however, unrecognized headers may not be processed in the same way by different devices, this adds an element of uncertainty to how the packet will be handled (Carpenter and Jiang, 2013).

Alternatively, the DTN status information could be added as an option in one of the two extensible option headers available. These headers contain a series of additional options on how the packet should be handled. A major advantage over a new dedicated header is that within the option type identifier, information on how to handle the packet if the option is not recognised is encoded. This is beneficial for DTN as the behaviour of devices along the route that do not understand the protocol is predictable and as such a better choice than using a new standalone extension header. Using the Hop by Hop header adds a potential 2 byte overhead for the extensible option header's identifier and length should it not already be in use on a packet. The added reliability of knowing what will happen at an intermediate node makes this a worthwhile trade off, in addition new protocols are encouraged to use the options headers rather than creating a new header where possible.

The two option headers available are the Hop by Hop (HbH) options header and the Destination Options (DO) header. The Hop by Hop (HbH) option header is evaluated by each router along a packets path, the destination options are either evaluated by nodes in the routing header or only at the final destination depending on the option headers location (Deering and Hinden, 1998). In the DTN case the Hop by Hop option header is the best location for adding the option header with a minimum of additional overhead, however, the destination options combined with the routing header can also be used for explicit control of which DTN nodes to use en route.

The most critical element of a DTN protocol is the time at which packets expire. In addition to this expiry time some additional flags will need to be provided in order to identify any special requirements the packet might have. Figure 5.1 Shows an overview of how this can fit into an option header and how that fits within the packet.

5.1.2 Packet Lifetime Encoding

The main piece of additional data that needs to be added to a packet to enable DTN capabilities is a lifetime of the packet. While there is a hop limit in the IPv4 and IPv6 protocols which is often referred to as its lifetime, this counts the amount of routers that a message can pass through before it is discarded and is intended to prevent network loops. With DTN a time at which the packet is no longer relevant and may be discarded is also required, this prevents a packet being held in the network indefinitely.

The Encoding of this lifetime needs careful thought in order to minimize the requirements of implementing the protocol on constrained devices. These requirements fall into

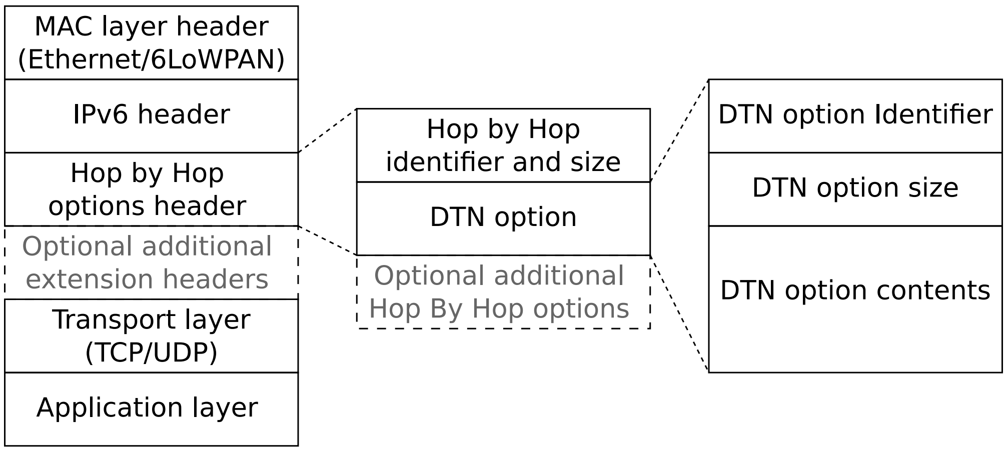

MAC layer header (Ethernet/6LoWPAN)
IPv6 header
Hop by Hop options header
Optional additional extension headers
Transport layer (TCP/UDP)
Application layer
Hop by Hop identifier and size
DTN option
Optional additional Hop By Hop options
DTN option Identifier
DTN option size
DTN option contents

Figure 5.1: IPv6 DTN option header overview

two categories minimizing transmission and processing overhead and minimizing node timekeeping requirements. As the timestamp is the significant proportion of the data being added as part of the header, it is important that it is transmitted with a minimum of required bytes. In order to keep packet processing time down and to conserve node processing resources, the timestamp should avoid requiring significant processing for the required operations. In addition the nodes will need to be able to provide the timekeeping capabilities necessary to support the protocol. Because of this the format should minimize the additional capabilities the nodes will need to be able to provide.

Within most Unix based computer systems, timestamps are stored as the number of seconds since the first of January 1970. This has historically been a signed 32bit value, but with this set to expire in the year 2032 systems will need to migrate to a 64bit timestamp. The use of Unix style timestamps poses several obstacles for their use in a DTN header.

The first of which is their size, using a 32bit value is not feasible as this would limit the lifetime of the protocol. Choosing a different offset would still place a limit after which the protocol would no longer be functional. Even though 130 years could be achieved with an unsigned 32bit value and a recent start point, the potential use of DTN in long term infrastructure projects necessitates that the technology should be viable for even longer. The options for this are to use a 64 bit Unix timestamp or use a variable length encoding method like SDNVs which are used on the bundle protocol.

Both the bundle protocol (Scott and Burleigh, 2007) and the implementation by Gomeimoto and Matsumoto (2013) use absolute timestamps. The use of an absolute timestamp will require that every node in the network maintains a synchronised time. There are several issues with this requirement. Many low power embedded systems do not have a real time clock module within them. For many devices the additional cost or battery drain of adding an external module will be prohibitive. In addition the time needs to be kept in sync, Table 5.1 shows the potential drift over time for various clocks. Without regular

			max error/day	
Type	Accuracy	Power Use	@25°C	@0°C
32.768k cyrstal and RTC on-board processor[1] [2]	20 ppm@25C	0.4μA	1.7s	3.9s
temperature compensated standalone RTC [3]	2ppm 0-80C	110μA	0.2s	0.2s
temperature controlled crystal [4]	50ppb	300mA	$43 \times 10^{-3}s$	$43 \times 10^{-3}s$

Table 5.1: RTC drift over time

synchronisation these inaccuracies can lead to messages being detected as expired when they still have time to be delivered. These differences are also magnified when devices are subjected to voltage fluctuations or changes in temperature. The periodic clock synchronisation that would be required to keep nodes in sync adds additional overhead to the node.

Instead of using a fixed timestamp, a relative timestamp can be used. The relative timestamp works by counting down the number of seconds until the packet is no longer required.

The use of a relative timestamp removes many of the obstacles of using a fixed timestamp. While maintaining a clock synchronised to an external time reference can be problematic, devices are almost always able to track the passage of time locally. While this local time reference might not be perfectly accurate, the problems of this inaccuracy are drastically reduced. Where as before these inaccuracies would build up over time, with a relative timestamp these inaccuracies are mitigated as each message starts the count afresh. The disadvantage of this approach is that the timestamp needs to be rewritten when the packet is retransmitted by a custody holder and that the custody holder needs to keep track of when it received the packet in order to appropriately reduce the lifetime.

Where as a fixed Unix style timestamp requires a 64bit variable, a relative timestamp can be represented in a much smaller space. Using a 32 bit value as a relative timestamp provides over a hundred years of potential packet lifetime regardless of the initial transmission date. Using any less than a 32bit timestamp, however, gives less than a year which would be unsuitable for systems with seasonal connectivity. Using larger timesteps such as counting the lifetime in minutes or hours reduces the accuracy of short duration timeouts significantly. A comparison of timestamp formats can be seen in Table 5.2 and Figure 5.2.

[1]EFM32 Datasheet (https://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32LG880.pdf)
[2]NC15LF crystal Datasheet (http://www.foxonline.com/pdfs/ncseries.pdf)
[3]DS3231 RTC Datasheet (https://www.maximintegrated.com/en/products/digital/real-time-clocks/DS3231.html)
[4]AOCJY1 Crystal Oven Datasheet(http://www.abracon.com/Precisiontiming/AOCJY1.pdf)

Encoding Scheme	Length	Accuracy	Max Lifetime (s)
Count 1s steps	16 bit	1s	65,536 (18 hours)
	32 bit	1s	4,294,967,296 (136 years)
Count 10s steps	16 bit	10s	65,5360 (7.5 days)
Count 60s steps	16 bit	60s	3,932,160 (45 days)
Exponent 1s steps	3+13 bit	1s at 1 hour	1,048,448 (12 days)
		16s at 1 day	
	4+12 bit	1s at 1 hour	134,184,960 (4.25 years)
		32s at 1 day	
	5+11 bit	2s at 1 hour	4×10^{12} (139,393 years)
		64s at 1 day	

Table 5.2: Comparison of relative timestamp formats

This 32bit value is still a noticeable amount of data to carry around with each packet. By leveraging the ability to use less accurate timestamps for longer duration expiries the packet size can be reduced further. The principle behind storing floating point numbers was applied to the timestamp to achieve this reduction. As negative or sub second precision timestamps are of little benefit, traditional floating point representations can be improved upon for this use case.

The improved timestamp format would be represented as an exponent with a multiplier component. The sizes of these elements need to be carefully considered in order to provide reasonable accuracy while still ensuring sufficiently long timeouts can be achieved. Table 5.2 shows a comparison of different potential sizes of the exponent and multiplier parts. As can be seen in the table a four bit exponent combined with a 12 bit multiplier provides the capability for long lifetime packets while maintaining a high level of accuracy. Figure 5.2 compares the new format against other non exponent based formats. The figure shows that the developed format maintains the high accuracy at low values and maintaining a worst case accuracy of 0.03% for longer duration timeouts. Using this format the variable format shown in Figure 5.3 was created.

5.1.3 Custody Transfer Mechanisms

One of the major aspects of DTN operation is how to perform the custody transfer from one device to another. This transfer involves the receiving node informing the sending node, which currently holds the custody, that the message has been received and it has taken responsibility for the onward delivery of the packet. The implementation of this feature should be done in such a way as to minimize the additional overheads required to support it. These overheads may appear as additional packets sent or additional data included in the DTN header. There will also be additional processing and state tracking overheads to process and manage any custody features.

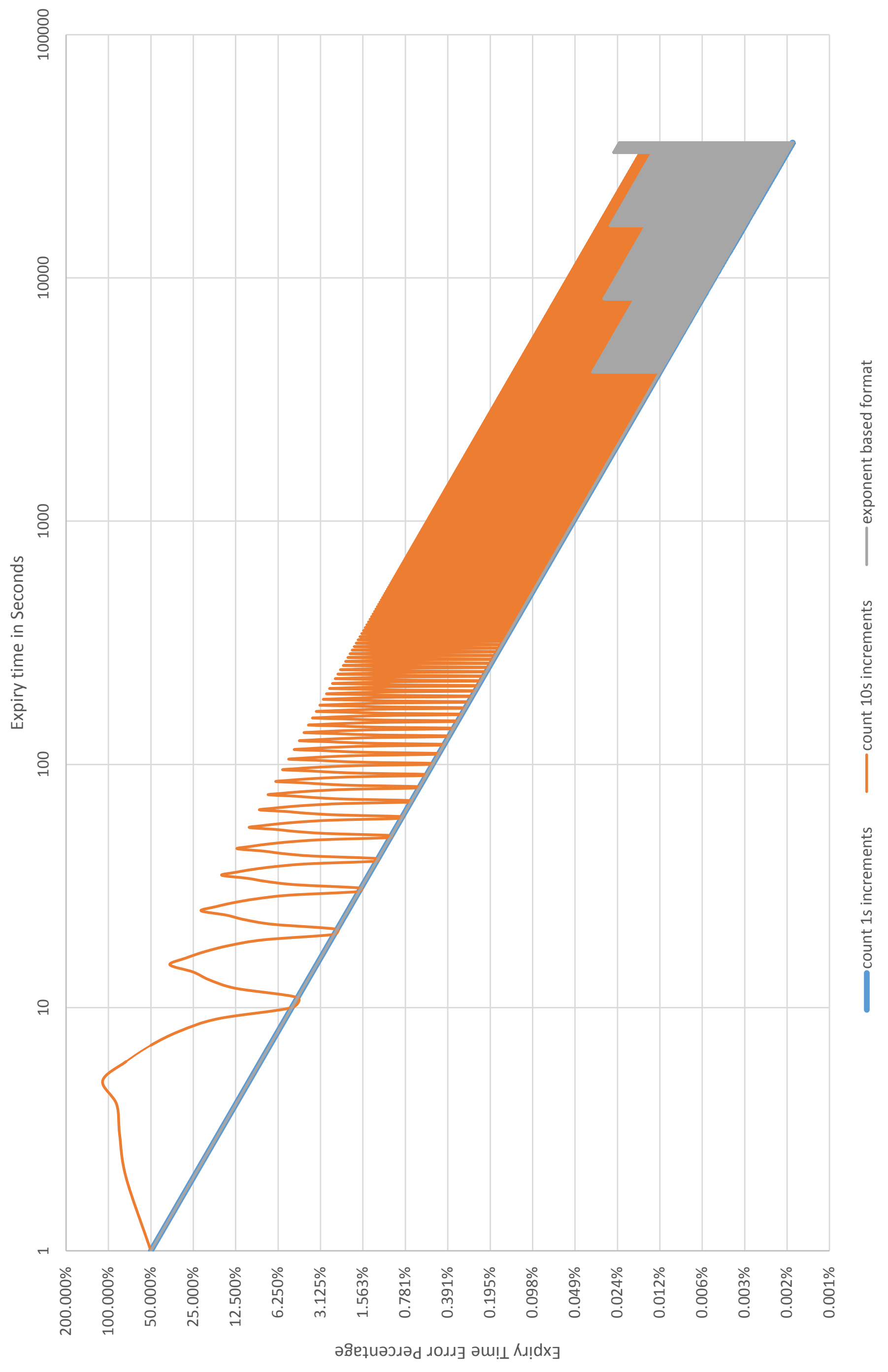


Figure 5.2: Inaccuracies for relative timestamp representations

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Exp  |       Multiplier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

$$Lifetime = Multiplier \times 2^{Exponent}$$

Figure 5.3: Variable precision timestamp format

Informational Responses	
Code	Name
1	Custody acceptance
2	Temporary custody acceptance
3	Custody Rejection

Error Responses	
Code	Name
1	Lifetime Expiry
2	Unable to deliver within lifetime
3	DTN packet deleted

Table 5.3: ICMPv6 Codes for DTN informational and error responses

In order for the previous custody holder to know it no longer needs to hold the packet, it needs to receive a status update from the new custody holder. The Internet Control Message Protocol version 6 (ICMPv6) (Conta et al., 2006) is used in IPv6 networks to transmit status information relating to packets. This protocol supports expansion so could also be used for sending DTN status information. ICMPv6 messages are split into a type field and a code field, with additional information included in the body if required. The type field is split into two ranges, one for error messages and one for informational messages. As DTN custody actions are informational, this range was chosen for the type code. The experimental use value has been used during development and testing as per rfc4443 (Conta et al., 2006). Within this type several codes have been chosen to describe these actions as well as for other cases where a response would be required, these can be seen in Table 5.3.

The custody status update packet, needs to be communicated to the previous custody holder. In many store and forward implementations the custody holder of a packet is known to be the last link layer hop and so the acceptance can be sent to that device. In an IP environment where some routers will not support DTN functionality this strategy does not work and another method must be used. The IP address of the last custody holder could be carried as part of the DTN header. Including this data in the DTN header will add 16 bytes of additional information which is roughly 12% of the packet length of 6LoWPAN. This is a significant overhead for such constrained systems and as such if this data could be avoided it would be beneficial.

In symmetrically routed networks transmitting the packet back towards its destination will cause it to pass back though the routers it visited previously. This could be used to deliver the custody message to the previous custody holder. To avoid every DTN enabled router from needing to examine every ICMPv6 frame passing through, a more efficient way of detecting DTN custody transfer acknowledgements is required. As these routers already need to detect and process the DTN header contained in the packet, a flagged DTN header can be used to signify that this is a custody message. A bit in the flag byte can be used to mark DTN messages as requiring analysis by each node en route to facilitate this requirement. In addition, as the custody acknowledgements are time sensitive, the expiry timeout can be set to zero so that they do not remain in the system should the link to the previous custody holder be unable to transfer the acknowledgement. In some cases a DTN enabled router receiving a custody acknowledgment might not be the custody holder for that message, this could be due to a variety of reasons such as being unable to store the message because of space or policy constraints. In this case the message can be sent onwards so that the current custody holder receives the message.

The proposed return path based custody acknowledgments require tree based routing to work. While routing paths in low power networks will change in relation to network activity, this will be infrequent and not affect the majority of custody responses. In larger multipath networks such as the current core internet backbone, the DTN nodes would need to be pushed to the edge. This is a minor issue as very limited benefit will be gained by placing DTN nodes within these reliable networks, especially when the cost of retransmission across the entire network is so low.

In asymmetrically routed networks, the proposed bounce back DTN custody transfer mechanism would not be suitable. However, the majority of networks which fall into this category are based on multiple unidirectional links, for example satellite, downlinks with dial up uplinks. In this case DTN acknowledgements can not be sent back up the chain of reception. Instead DTN devices would need to be placed at the point where the two unidirectional systems reconverge, Figure 5.4 shows an example of this.

The bounce back method has an increased chance of network situations occurring, which may cause it to fail, but also offers smaller header sizes. On the other hand, including the network address of the last custody holder is likely to be more stable but comes at the cost of adding 16 bytes to the header. Because of the reduced size of the bounce back method, and the ease of falling back to include the previous custody holder address should problems arise, the bounce back method will be tested in the initial implementation.

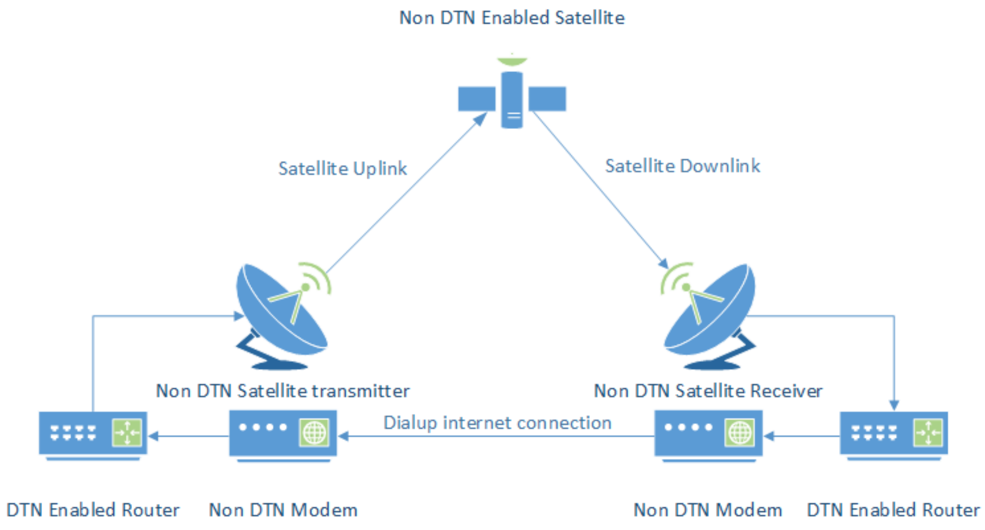

Non DTN Enabled Satellite
Satellite Uplink
Satellite Downlink
Non DTN Satellite transmitter
Non DTN Satellite Receiver
Dialup internet connection
DTN Enabled Router
Non DTN Modem
Non DTN Modem
DTN Enabled Router

Figure 5.4: Placement of DTN nodes in a unidirectional satelite system

5.1.4 Combining These to Form a Protocol

Combining the features and developments discussed above, a reliable and efficient DTN format has been developed. Figure 5.5 provides a breakdown of how components fit together to form a header.

The DTN option identifier's first three bits are used to signify how the packet should be handled (Deering and Hinden, 1998). As DTN packets are expected to pass though non DTN routers the 00 skip option is required for the first two bits. The DTN header will change en route so the third bit needs to be set to 1 so that the header is ignored when calculating any authenticating value. subsection 5.1.5 discusses the security implications of this. The rest of the header is made up to match the experimental value as defined in rfc4727 (Fenner, 2006). The next byte contains the option length which is set at four bytes.

The next two bytes are used for option flags. While only a single byte is required for the currently considered flags two were allocated. This is because extension header lengths are specified as a number of 8 byte blocks, meaning a padding byte would need to be added in the HbH header to make it the correct length. Rather than add the padding byte at the end, the option flags were extended to provide more space for experimentation should more data be needed. At the moment the only flag that is used is a flag to mark the packet as an informational response. The final two bytes are used for the 16bit relative timestamp discussed in subsection 5.1.2. If needed an additional 16 bytes can be used for the previous custody holders address if the bounceback method does not work.

The layout of the protocol can be seen in Figure 5.5. A comparison of packet size against using the bundle protocol can be found in Figure 5.6

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  - - - - - - - - - - - - - - - -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                 |  Option Type  |  Opt Data Len |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |         Option Flags          |           Timestamp           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Additional Data ...
 +-+-+-+-+-+-+-+-+-+-+-+-+- - - - -
```

Figure 5.5: Hop by Hop DTN option header layout

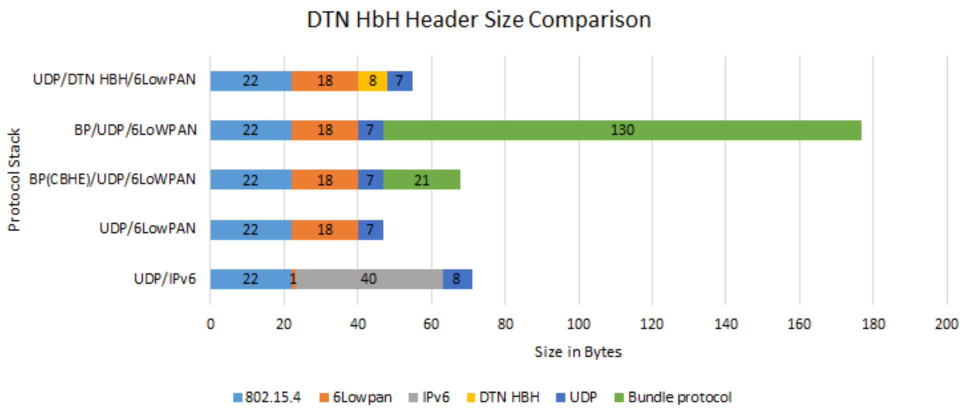


Figure 5.6: Comparison of DTN header size against bundle protocol.

5.1.5 Security Considerations

Enabling Delay Tolerant Networking on a network creates several security considerations regardless of the technology used to support it. The storage of packets opens new opportunities for denial of service attacks on the resources of the devices supporting DTN. An example of an attack that could be performed is to send a large volume of traffic to a non responsive destination with a long expiry time. This will cause DTN nodes to store that information, occupying the storage buffer and preventing its use for legitimate traffic. Alternatively, the packet could be deliberately dropped by a malicious device while valid custody acceptance messages are still sent. This leads to packet senders believing that a packet is en route when it is not, this has higher impact than in a non DTN enabled network, as it will take significantly longer for the device to realise there is a problem than would be the case with direct end to end communication.

The use of relative timestamps causes the DTN option header to change during transmission. This means that the DTN header must be ignored when calculating an authenticating value should an authentication header (Kent, 2005) be present. This requirement is signified by the third bit in the extension option identifier being set to one. This means that there is the potential for malicious actors in the messages path to amend the HbH header to cause unintended operation even on authenticated packets. While this reduces the effectiveness of using the authentication header to validate traffic before it reaches a constrained network, it will reduce the potential attack surface as replay attacks can still be detected and blocked. Even if the header could be secured, malicious actors in the packet path still have other methods of causing undesired operation, such as dropping the packet as discussed above. The risks of allowing this header to be altered as required for relative timestamps is still a concern but is outweighed by the benefits discussed in subsection 5.1.2.

5.2 Implementations of the new DTN Protocol

In order to further develop and evaluate the protocol it is necessary to trial it on real networks. This necessitates creating working implementations of the protocol. The majority of the use cases as identified in section 4.2 link constrained devices with more capable devices. To fully understand the benefits of the protocol these use cases must be evaluated, which means implementations for both embedded microcontroller based platforms and full featured devices with application processors are required.

For the full featured device implementation of the protocol the Linux operating system was used. This will allow it to be run on small devices such as border routers as well as more powerful devices. This implementation was used to help refine choices in the protocol development as it provided a fast turnaround on experiments before the development of the embedded implementation. For the embedded implementation, DTN was integrated into the Contiki operating system which can run on low power constrained hardware. These were then used together to evaluate and analyse the benefits of the protocol.

5.2.1 Linux Python Implementation

The full featured implementation of the DTN header was implemented on the Linux operating system. As the amount of resources available are orders of magnitude higher than embedded systems this avoided any concerns about optimisation during the early experiments. In addition, improved access to debugging output is available compared to embedded systems.

At first it might seem that implementing DTN handling in the kernel in a low level language would be the way forward. While this would be the appropriate way for a deployment version, a different route was taken during development. Rather than operate in kernel space, the DTN compatibility was instead implemented in user space. The use of user space over kernel space allows for a faster development cycle reducing the time between iterations of the protocol. The Python programming language was

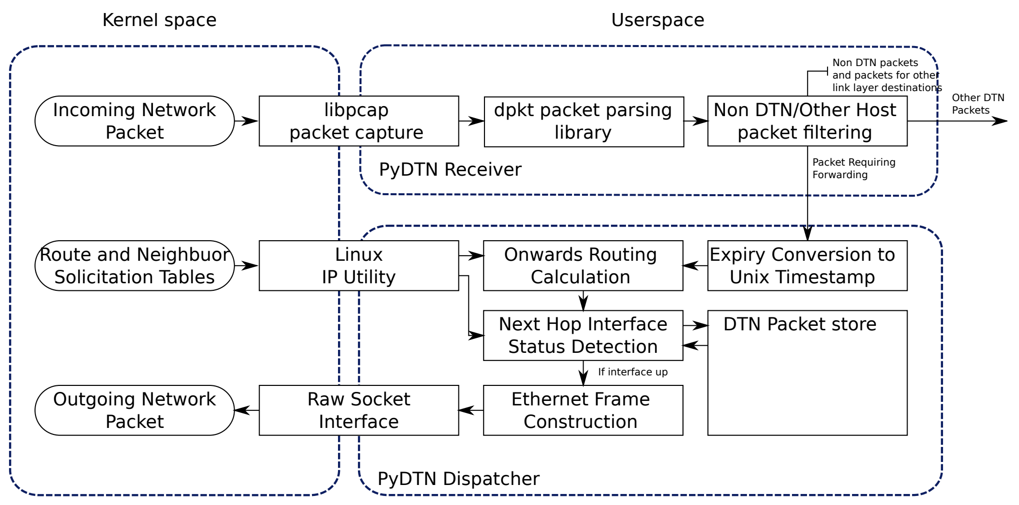

Kernel space
Userspace
Incoming Network Packet
libpcap packet capture
dpkt packet parsing library
Non DTN/Other Host packet filtering
Non DTN packets and packets for other link layer destinations
Other DTN Packets
PyDTN Receiver
Packet Requiring Forwarding
Route and Neighbuor Solicitation Tables
Linux IP Utility
Onwards Routing Calculation
Expiry Conversion to Unix Timestamp
Next Hop Interface Status Detection
DTN Packet store
If interface up
Outgoing Network Packet
Raw Socket Interface
Ethernet Frame Construction
PyDTN Dispatcher

Figure 5.7: Python DTN processing for a router implementation without custody transfer

used to develop this implementation. The use of Python allows for faster prototyping as it has access to a range of libraries that can simplify the soloution.

The Python implementation known as PyDTN was created as a series of modules and utility functions that can be combined together to create different types of DTN applications such as transmitters, routers or responders. Figure 5.7 shows how the modules discussed below interface together to form a DTN router implementation which acts as an intermediary node on the path of a DTN packet.

The PyDTN receiver module handles the reception of DTN enabled packets and directs the packets to be processed locally or forwarded onwards. The first section of the receiver module is extracting the packet from the network interface. The Python pcap package was used to provide raw packet capturing, this preserves the headers of the link and IP layers for analysis. Libpcap only allows for listening on a single network interface, to allow for bidirectional routing a separate instance of the receiver module is required for each interface on the machine.

To avoid processing every packet received, a pcap filter for the HbH extension header was used. This filters out packets that do not contain the HbH options field, greatly reducing the amount of packets that need to be processed. This filter still leaves non-DTN packets present that use other Hop by Hop options. Also, packets directed to different hosts are still accepted by this filter, these include packets transmitted on that interface as well as broadcast traffic.

The packets that pass through the filter are further filtered and analysed to remove the unwanted packets. The dpkt package for Python was used to decode the packets received from the packet capture utility. Dpkt breaks down the packet into a series of separate binary fields allowing for easier analysis. The destination MAC and IP fields are checked to ensure the packet is intended for this host at the link layer. The packet is then inspected for the presence of a DTN header. If any of these tests fail the packet is discarded by the DTN processor, the standard kernel packet processing will handle the packet in this case. Depending on the destination of the packet, the packet is then either sent for onwards transmission or delivered to a local process.

For packets requiring onwards transmission the PyDTN dispatcher module manages this process, this module is also used for packets being sent by the host. The dispatcher module manages the storage and transmission of DTN packets. In order to insert the DTN header into the packet, the packet needs to be sent as a raw packet as the other socket modes do not allow for extension headers to be added. This requires that the IP and link layer headers need to be computed manually. In the IP layer the packets TTL and DTN expiry counter are the only variables that need updating, the link layer, however, needs to be generated entirely from scratch.

Before the link layer can be generated, the outgoing interface and next hop needs to be calculated. This is done by querying the routing table with the IP route command and then parsing the output. This returns the interface and destination IP address of the next hop. The link layer address is then extracted by looking up the IP in the neighbours table and performing a neighbour solicitation request if missing. This step also works as part of the next hop availability checking, devices that do not respond to solicitation requests are unavailable.

When the next hop is unavailable the packet is stored in the packet store. The packet store maintains a copy of the packet along with the Unix timestamp at which it will expire. When the next hop is available and can be contacted the packet is passed to the transmission tool. This tool opens a raw socket on the required interface and loads the MAC address of that interface into the MAC layer of the packet being dispatched. The decoded packet is then compressed back to a binary form before being transmitted down the interface.

The early version of PyDTN implementation was initially created without a mechanism for custody transfer. As a proxy for a custody transfer mechanism, next hop availability detection using IPv6 neighbour solicitation was used. Unlike a proper custody transfer mechanism, the next hop detection can not guarantee reception of the packet or detect communication breakages after the first hop. In the PyDTN test setup which consisted of VMs connected by virtual Ethernet cables it could be accepted that if the next hop was available then a message would be received as the connections were very reliable. In addition as every node in the network was running PyDTN the next hop was known to be a DTN enabled device so further link availability was not a concern.

In order for Contiki to confirm that the packet has been delivered the Python implementation needs to respond with a custody packet. A utility was added that generated the custody response based on the incoming packet. The source and destination are

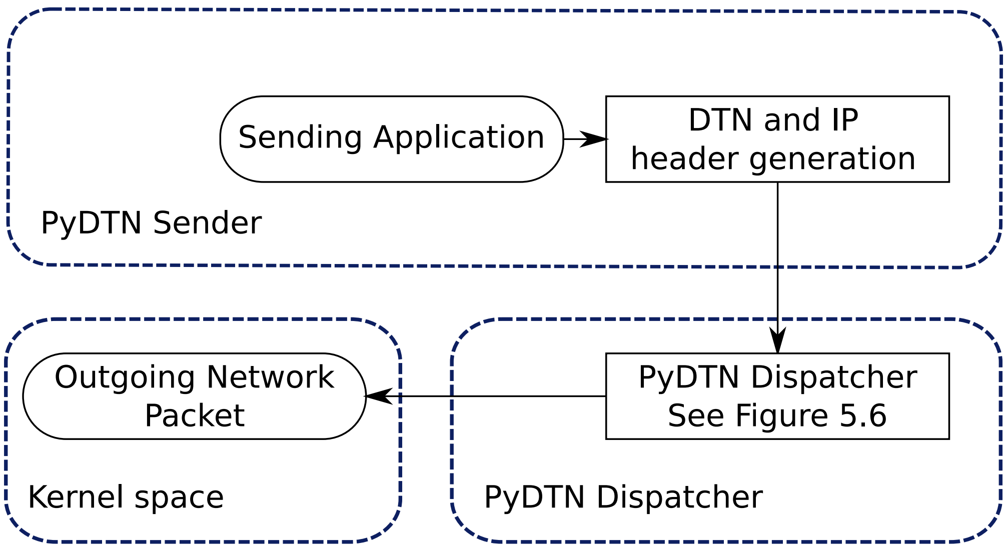


Figure 5.8: Overview of the PyDTN sender application

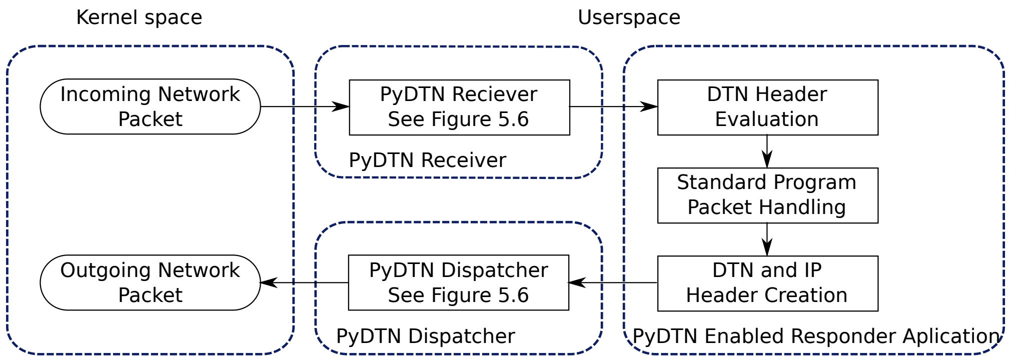

Kernel space
Userspace
Incoming Network Packet
PyDTN Reciever See Figure 5.6
PyDTN Receiver
DTN Header Evaluation
Standard Program Packet Handling
Outgoing Network Packet
PyDTN Dispatcher See Figure 5.6
DTN and IP Header Creation
PyDTN Dispatcher
PyDTN Enabled Responder Aplication

Figure 5.9: Overview of the PyDTN replier application

extracted from the incoming packet and used to generate the custody response, this is then sent using the dispatch module like any other DTN packet.

Transmitter and responder programs can also be built using the modules discussed above. Figure 5.8 and Figure 5.9 show how the modules are used as part of these implementations. As the sender and replier modules include the application logic they are customised for each application.

For a DTN enabled sender application, rather than using a standard socket to send messages, the application sends the payload through the dispatcher module. A DTN header is generated for the packet by the sender module. The sender then uses dpkt to generate a complete IPv6 frame which is then dispatched to the dispatcher module. The replier modules are similar to the sender application but also handle processing of the incoming packets. Reception of DTN packets can be achieved through standalone applications as these can natively receive DTN enabled packets although they will be unable to see the packets DTN header.

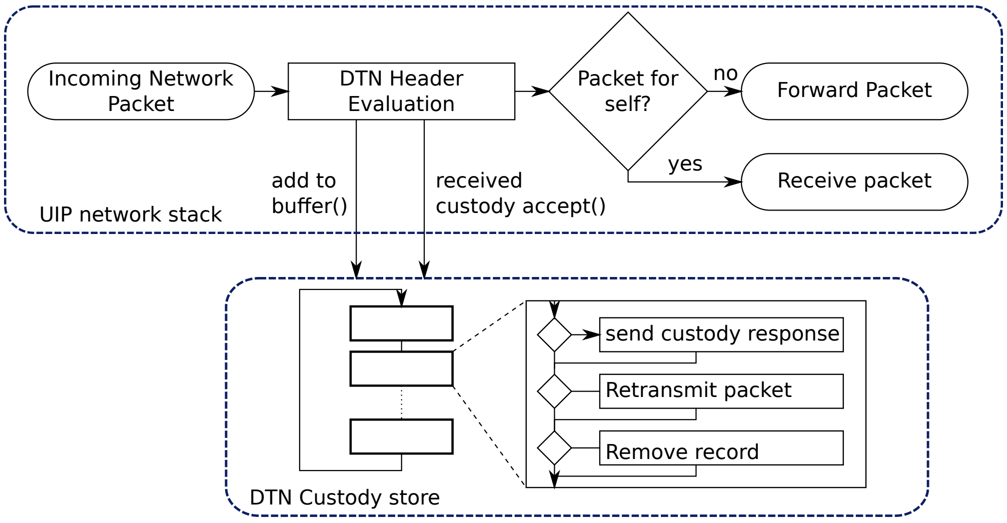


Figure 5.10: Overview of the modified Contiki networking stack republished from (Ward et al., 2015)

5.2.2 Contiki Implementation

The Contiki operating system (Dunkels et al., 2004) was chosen as the platform for the constrained implementation. Contiki is a lightweight operating system designed to run on platforms with limited resources such as the Tmote Sky and Zolertia Z1 platforms. This is the same operating system which was used for the Lab sensor network and for the Feshie deployments discussed in chapter 3. Contiki is discussed in more detail in Section 2.2.4.3.

Contiki's IPv6 network stack is known as uIPv6 and was developed by Cisco for the operating system. The network stack operates on a single buffered packet, passing it through several processing steps before either sending it to the communications libraries for transmission, or passing it to packet reception code which handles the processing of the packet's contents. By modifying a preexisting operating system and network stack, the time to develop a working DTN enabled system is significantly reduced compared to creating a new implementation. Figure 5.10 shows how the elements of the Contiki implementation fit together to provide DTN functionality to Contiki.

As one of the uIPv6 processing steps, the packets are checked for extension headers. Several extension headers are already supported by uIPv6 these include the HbH and DO option headers. Support for these is included in Contiki because the RPL (Winter et al., 2012) routing system uses Hop by Hop options to transmit data regarding routes around the network.

An additional case statement was added to the option header processing code to detect DTN enabled packets. Packets containing DTN options were identified as either an incoming packet or a custody response. Incoming packets are sent for storage within the DTN custody store. These packets are not dropped, so continue through the uIPv6 system to be retransmitted or directed to the appropriate handler for that packet type. Custody responses are passed to the custody store where they are checked against stored packets. If a match is found uIPv6 is instructed to discard the packet otherwise it is allowed to continue onwards to the node holding custody of the linked packet.

Contiki's uIPv6 networking stack is limited to a single packet being processed at any one time meaning custody responses can not be sent while the incoming packet is being processed. Because of this a separate custody management system was required. This custody management process is implemented as a Contiki protothread.

The custody manager maintains a list of packets in the custody index. This index stores selected details about the received packet, which allows for efficient checking of any actions to be carried out on the packet without needing to reprocess the packet to extract the required information. Along with the index, the packets themselves are stored so that they may be retransmitted if necessary. On the current implementation the packets are stored in RAM, the use of the index allows the use of other storage mediums such as external flash memory, whilst still being able to process the packets in a limited timeframe.

When a new DTN enabled packet is received, a custody management function is called from the uIPv6 network stack. This function extracts the required details from the packet and stores them in an empty slot in the custody index and stores the packet in the packet store. The network processing is then told to continue processing the packet and send it onwards to its destination. When the custody management process next runs it will detect the newly stored packet and create a custody acceptance message for the previous custody holder. This is then sent off to the previous holder to confirm the transfer.

When custody acceptances are received by the networking code another one of the custody managers functions is called. In this function the incoming acceptance is matched against the appropriate record in the custody store, that record is then flagged as having been delivered, the network stack is then told to drop the packet. In the event that the packet cannot be matched against a record then the network stack will continue to forward the packet to its destination as another device is holding custody of that packet.

Once a packet has been accepted by a downstream node and the custody acceptance has been sent to the upstream node the packet no longer needs to be kept. The custody management process can then delete the packet from the packet store and clear the index entry to make way for another packet.

In the event that no custody acceptance is received then the packet will be retransmitted. It is up to the node to decide when to retransmit, in the Contiki implementation a two minute retry was used as this worked well in the lab environment where connectivity is usually restored quickly. In other applications, however, a longer timeout or alternate

	Size Bytes	
Component	ROM (with debug statements)	RAM
DTN Custody store	676 (1140)	1724
DTN Header processor	484 (894)	0
Total	1160 (2034)	1724

Table 5.4: Space requirements of the Contiki DTN implementation with a 10 packet buffer

	Size Bytes	
Component	ROM	RAM
Core	4998	1184
DTN	1160	1724
elfloader	1899	3098
other	53451	8857
rime	3159	1382
sky	3925	209
uip	12234	3010
Total	80826	19464

Table 5.5: Code size breakdown of the DTN enabled border router running on a Z1 node

retry logic may be more appropriate. When it is time to retransmit a packet the expiry timestamp needs to be updated. The packets expiry is stored as a fixed timestamp within the index using the internal Contiki time counter which is measured as the number of seconds since boot. This avoids needing to update the time in the index on a regular basis. This timestamp is then used to recompute a new relative timestamp that will be included in the retransmitted packet.

If a packet continues to receive no acceptance messages despite the retries then the packet will eventually expire. This is detected by comparing the timestamp against the current nodes clock. When the time is greater than the stored expiry time the packet will then be erased by the custody management process.

The Contiki implementation was tested in the Cooja simulator (Osterlind et al., 2006) which emulates nodes in a virtual radio environment. Tmote SKY nodes along with Zolertia Z1 nodes are used simultaneously in the simulator in order to ensure operation on multiple platforms.

Table 5.4 shows the amount of resources consumed by the Contiki DTN implementation, Table 5.5 compares this against other contiki components. The slight increase in size caused some of the Contiki examples which were already very close to the maximum allowed size, such as the Sky websense demo, to no longer fit within the space available on the device.

The Contiki implementation, while providing a good basis for testing DTN capabilities has some areas where improvements can be made to increase its functionality and performance. The first of these is the packet store. In the current implementation the packets are stored in RAM, this severely limits the amount of packets that can be stored. An improved method would be to use the EEprom IC on the Z1 and SKY nodes to store packets while they are being delayed. Newer processors might also be able to improve upon this situation, as they have a much larger amount of memory available for use. The CC2538 used on the Zolertia RE-Mote has 32K of ram which is an eight fold increase on the MSP430F2617 used on the Z1. On such a device the extra RAM available could allow for the storage of up to 100 extra packets in the packet buffer without resorting to flash based storage.

Another area which could be improved is the retransmission logic. The current version uses a fixed retransmission interval which was chosen for the lab environment used for testing. An exponential backoff would provide an improved retransmission time for short interruptions while reducing the amount of retransmissions for longer duration outages. Previous knowledge about network interruptions could also be used to influence future retry attempts. This would allow for more efficient retries when the reliability of the network changes over time.

5.3 DTN Extension Header Trials and Evaluation

In order to evaluate the protocol which has been developed and implemented above, trials need to be carried out. While simulated trials will provide a useful initial evaluation, the protocol must also be tested on deployed networks in order to ensure operation outside of the controlled conditions present in a simulated setup.

5.3.1 Simulated Tests

For the initial trials during development of the DTN extension header, simulated nodes and virtual machines were used. The Cooja simulator was used for the low power network and a set of Ubuntu virtual machines acted as Linux based router nodes. Within this test bed a simulated sensor network was created.

The Linux PyDTN implementation provides detailed debugging and status information on each packet that is being processed. This information is printed to the console of each node running PyDTN, which can also be piped to a file for later analysis if required. Debugging within the Contiki implementation is far less plentiful. In order to maintain a fast processing time and to fit within the code size limitations, only the most important status messages are output over the debug interface.

Within this low power network, the packets still need to be tracked as they travel between nodes in the network. This was achieved by the use of the Wireshark program to inspect the radio traffic within the Cooja simulator. The 802.15.4 radio packets are exported by Cooja to a file which is read by Wireshark. Wireshark then decodes the contents of the packets sent between each hop across the network and presents them as a time ordered list of packets. While Wireshark does not have support for decoding the DTN header itself, it can decode the rest of the packet while leaving the header contents to be decoded manually. This allows for the in-flight changes to the DTN header as it passes over the 6LoWPAN network to be inspected.

One of the Linux VM's running PyDTN was configured to request data from the sensor nodes at a fixed interval using a sender application. While this node was requesting data from the nodes, network interruptions were introduced into the network. The flow of the packet in response to the interruption was then monitored.

The simulated network was tested with and without DTN capabilities on the nodes. Without the DTN capabilities the packets were silently lost whenever interruptions occurred, which is exactly what would be expected of current networks. With the DTN functionality added, the Hop by Hop custody transfer ensured packet reception for those packets that could be delivered within the allowed lifetime. The work on the implementations and protocol development as well as the initial evaluation have been published as 'Adding support for delay tolerance to IPv6 networks' (Ward et al., 2015) at the FNC.

5.3.2 Trials on Deployed Networks

In addition to the simulated tests, experiments were carried out on physical hardware devices. Physical Z1 and Sky nodes where programmed with the same firmware which was used for the simulated experiments. With the physical hardware there is less diagnostic information available regarding packet flows and node actions.

The flow of packets on the border router were monitored using Wireshark. Unlike the simulated experiments where every hop was captured, the physical hardware tests just covered messages traveling over the tun interface used for the SLIP connection to the border router. Despite this limitation, the transmission of the messages can be monitored as well as the first custody transfer between the Linux based system to the Contiki implementation. A visual indicator was added to the nodes so that they turn on one of their LEDs when holding custody to provide additional insight into the flow of messages around the network.

To test the resilience of the network, deliberate connection interruptions were introduced. This was achieved using two main methods. The first was to replace the antennas connected to devices in the network with dummy loads. This prevents radio waves from

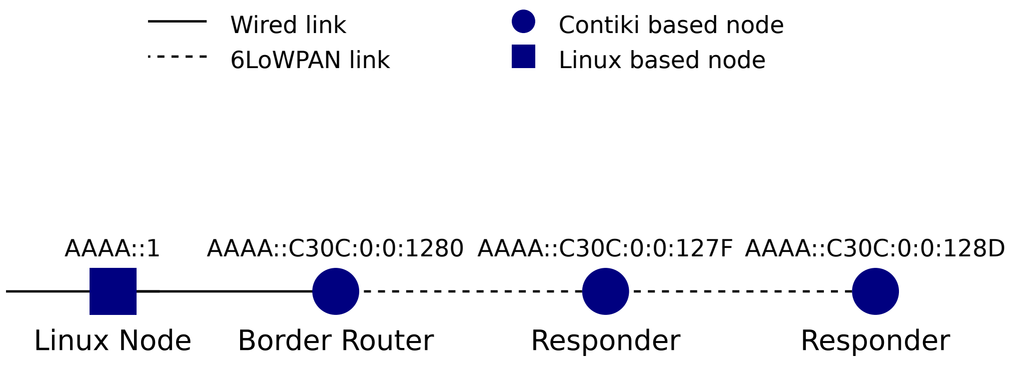


Figure 5.11: Network structure used for the DTN header evaluation

being transmitted or received while keeping the device online. The other method was to power off the node by removing the batteries or disconnecting it from the power supply. The physical implementation provided the same improvements in connectivity as was seen in the simulated network. Where the nodes holding custody were powered off, any packets in their custody store were lost. This is an inherent limitation of storing the packets in RAM. Fortunately, the failure of the node is a far less common occurrence than a loss of communications therefore reducing the impact of this limitation.

The next stage of the real world tests was to run a prolonged test of a DTN network to evaluate how it functions with naturally occurring network disturbances. A small network was deployed in the research lab similar to that deployed for the lab temperature sensor network discussed in Section 3.2. The network consists of three nodes, two acting as targets for DTN enabled messages and the other operating as the border router which was connected to a computer running Linux which was running the PyDTN code. The nodes were positioned around the lab so that they formed a single line routing path allowing for testing of the protocol over multiple hops. Figure 5.11 shows the network layout and structure used.

The target nodes in the network were programmed with a simple responder application. This was based on Contiki's unicast UDP receiver application. The responder application responds to UDP requests by returning the original contents along with additional diagnostic information including a unique response ID which is incremented with each response.

A python program was written using the modules discussed in Section 5.2.1. The created program sent scheduled requests to each of the nodes in the network. The requests contained a unique ID as well as diagnostic information to allow packets to be identified and tracked. When combined with the information added by the UDP responder program on the nodes, the packet flow across the network can be tracked and analysed.

The network was evaluated over a one week deployment period. Overall the network increased the reliability of packets although this came at the cost of a noticeable amount

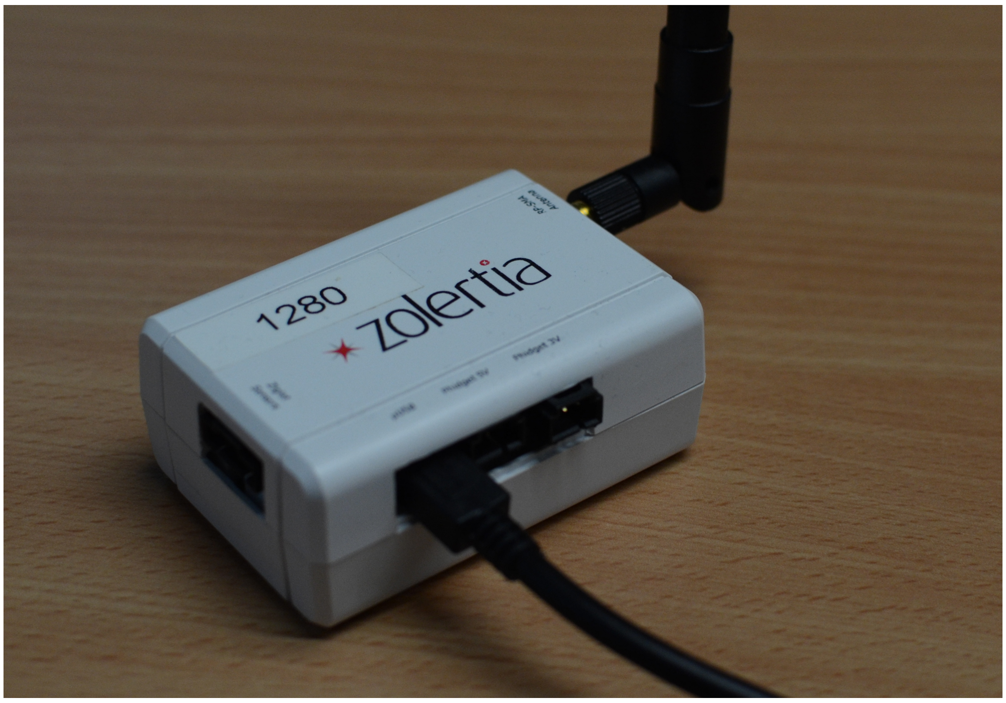


Figure 5.12: Photograph of a Z1 sensor node used for the DTN trial

of repeated messages. These repeated messages were caused when the custody acknowledgement was either not received or not processed correctly by the previous custody holder causing it to retry transmission at a later time.

Over the week 1974 messages where sent out to the nodes on the network. Of these messages 1940 received immediate responses, 29 received responses after a delay and 5 received no response. On each node there was a pair of consecutive requests which received no response and the remaining request being only a few requests later than the other errors on that node. For all of these lost packets there were successful requests within the packets lifetime, this shows the link was not down for long enough to cause the packets to reach their expiry time meaning the loss of these packets is outside the expected DTN operation.

There are several ways in which those packets could become lost within the network. As the DTN packet buffer and index is stored in volatile memory on the nodes, the contents of the DTN packet buffer are lost if the node reboots, however, as the nodes did not reset the response count this can be ruled out as the reason for the loss of these packets. Another possibility is that the responder node received custody of the outgoing packet successfully but did not generate a response packet. The final option is that the packet was mishandled and was deleted from a nodes custody store without a custody response having been received from the next node in the chain. This indicates that these lost packets will be due to errors in the handling of the DTN packet or due to issues on the responder program. The responses broken down by node are shown in Figure 5.13.

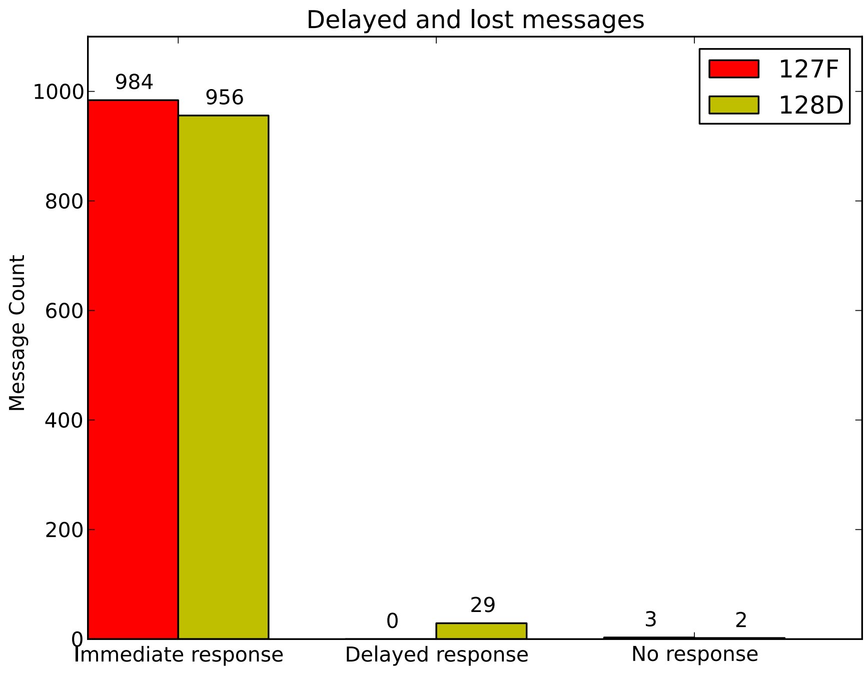


Figure 5.13: Responses when using DTN in a real world network

While almost all messages received a response, a large number received multiple responses. Figure 5.14 gives a breakdown of the number of responses received for each request. These duplications were not seen in the simulated or early trials with physical hardware. The message duplications are almost always happening on the return path after the response has been generated. If a message is duplicated when travelling from the computer to the node then a separate response ID will be generated for each. As there was almost always a single response ID per message the duplications must have occurred when traveling back from the node to the computer. Figure 5.15 shows a count of unique response IDs received for each request. It can be seen that the node with the additional communications hop (128D) had significantly fewer duplicated packets than the node with a direct link to the border router (127F).

Within a DTN network a packet can be come duplicated if a custody response is lost. In this situation two nodes can end up both holding custody over a packet, resulting in a duplicated packet being sent after the retry delay. The most probable cause of the high number of duplicated messages is that the custody messages from the response packets are lost. Despite being a regular occurrence on the real world deployment these retransmissions were not present on the simulated networks. It is probable that the more predictable nature of the simulator or the lack of external radio influences avoids the issue manifesting itself.

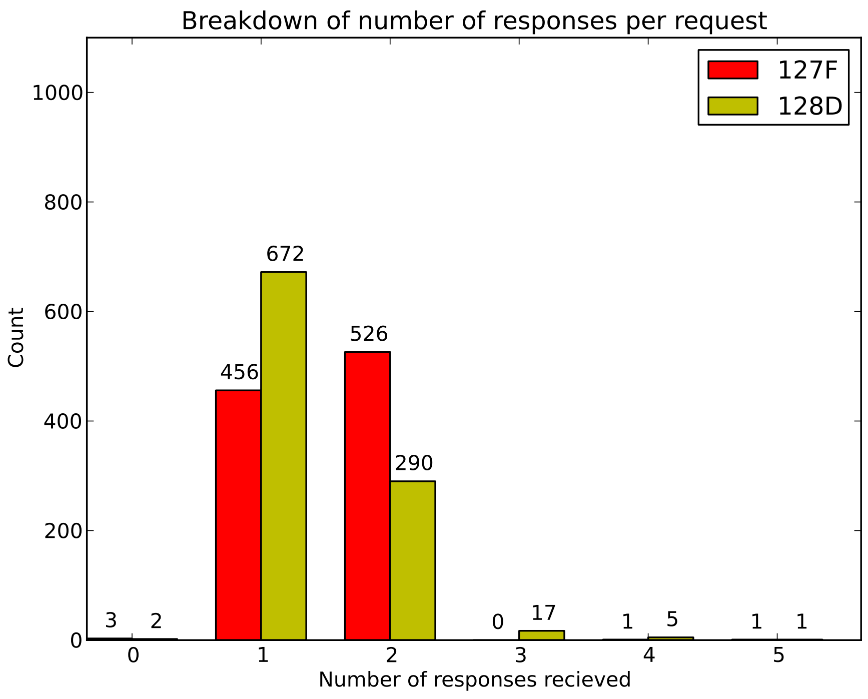


Figure 5.14: Amount of responses received per request on the trial DTN deployment

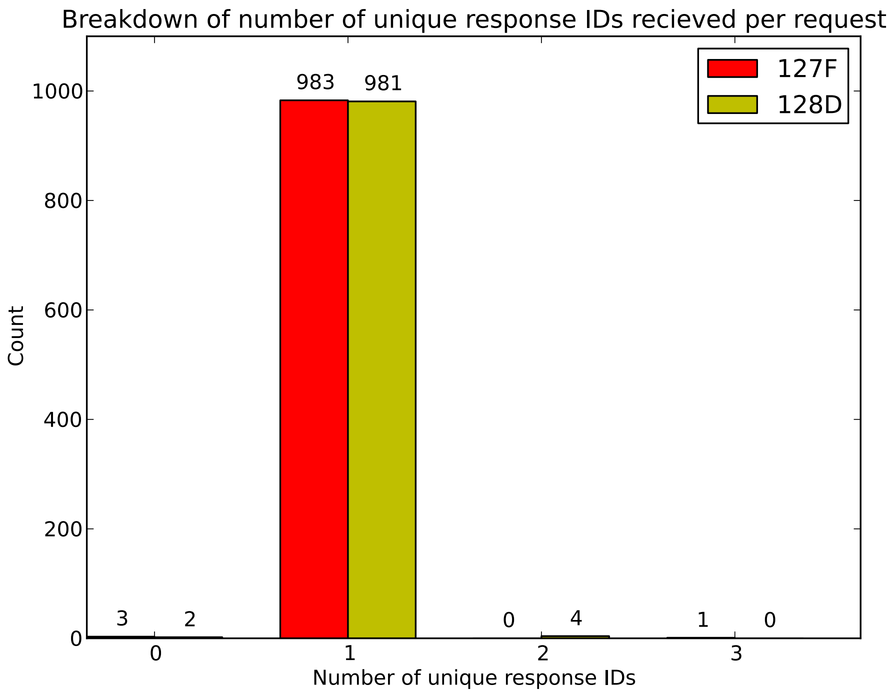


Figure 5.15: Breakdown of number of unique response IDs per request

5.4 Summary

The previous DTN implementations which were primarily created for interplanetary networking are inefficient and cumbersome, especially when considered for use on constrained networks. The DTN Hop by Hop option header developed above, implements the core functionality required for DTN without duplicating features available in other extension headers. This allows for creation of an efficient protocol which takes a minimum of space. With careful consideration of data packing and data formats a compressed relative timestamp format was chosen, this avoids placing any additional hardware requirements on nodes while maintaining long lifetimes and allowing fine control of short lifetime packets.

This protocol has been shown to be capable of being integrated into current operating systems with a minimum of changes to existing network software and with the minimum of impact on the processing times for non DTN enabled packets. Through a series of trials it has been shown that DTN can increase reliability even on networks without sleep states, although the benefit is obviously far greater on networks with sleep states. In deployed networks the developed DTN system resulted in a number of repeated packets, which consume bandwidth.

The possibility of the loss of custody acceptance messages cannot be entirely eliminated even with reliable link layer transport as connections could become unavailable in the middle of a DTN custody transfer. Therefore in order to reduce the number of duplicated packets a deduplication method needs to be integrated into the protocol. The detection of duplicated packets can be achieved by comparing the contents of received packets to that of previous ones. The addition of a DTN ID field in the DTN flags would reduce the amount of comparisons that would need to be made as this could be checked first before comparing the entire packet. Constrained nodes, such as those used in this test, would only have the resources to perform deduplication of packets if they are also holding the custody of the cloned packet. However, more capable devices could store a history of packets which will allow for deduplication of packets they have previously held custody of.

Despite this issue, the capability for integrating DTN into the IP layer using extension headers has been clearly demonstrated and many benefits can be gained by doing so. This makes it an obvious choice for bringing DTN capabilities to the next generation of sensor networks and embedded devices.

Chapter 6

Conclusions

With the demand for smarter buildings, utilities, and cities, the need for low power internet connected systems continues to grow. Current technologies for such markets have not been able to offer full IP access to sensors, which complicates access to the data contained within. New IoT technologies are mainly being held back due to the energy usage of the end devices in the networks. In order to make end to end IP systems a reality, the energy cost of these networks must be reduced.

The most effective method of reducing the power is to put devices into a low energy, or sleep state, for as much time as possible. This is a technique used on many existing low energy sensor network deployments. Unfortunately, enabling such sleep states prevents the nodes from maintaining continuous network connectivity. Many of the communication technologies used on the internet expect continuous end to end connectivity between devices, which these challenged networks are unable to achieve. In existing non-IP networks this has been dealt with using a sensor network gateway which proxies messages to and from the nodes on the network. While proxies could be added to IP based systems this reintroduces the limitations created by the gateway in previous systems.

Without using a gateway device, messages will need to be timed to match the devices communication periods. This can be achieved either by timing the message transmission from the source or by delaying the message on route to align it with these intervals. Delaying the message during transit using store and forward has several key advantages over other solutions. By storing and forwarding messages the sender does not need to maintain synchronisation with the sensor's schedules. When a continuous end to end connection can not be established (such as when two networks with different schedules wish to communicate) there are not any time slots in which a message could be sent. However, using store and forward allows the packet to progress through the network in stages, removing the need for end to end connectivity.

Delay Tolerant Networking (DTN) has been developed to solve similar issues that are encountered in interplanetary networking, and the principles behind this offer an effective way of implementing store and forward as part of the network architecture. Current DTN technologies which have been designed for interplanetary networking are not well suited for use on constrained terrestrial networks. A new DTN protocol has been developed to bring these features to terrestrial networks. The protocol has been designed to allow full backwards and forwards compatibility with existing and future systems, while also maintaining a small overhead. The developed protocol has been implemented as an IPv6 HbH option header which allows for seamless integration into the network layer while allowing the packets to be processed by any device in the communications chain.

Many constrained systems are unable to maintain an accurate clock which is synchronized to an external time reference. Because of this, the protocol has been designed to use a compressed relative time-stamp format which only requires nodes to be able to record the passing of time, this method significantly increases the number of devices that will be able to support the protocol. Combined with an efficient method for informing the previous custody holder of packet reception, the protocol is able to offer a very low overhead for both intermediate and end devices in the chain.

6.1 Contributions Made

The major contributions from this thesis are aligned with the research questions identified in section 1.1. The questions have been repeated below.

- Identification of Challenges to IoT EWSN Deployments
- Reducing Power Usage on IoT Sensor Networks
- Implementing Delay Tolerant Networking for Constrained IoT Networks

Through the series of sensor network deployments discussed in chapter 3 several challenges to the use of IoT systems in sensor networks were identified. The most significant of these is the difference in power use between existing low energy systems and new constrained IoT networks. The increased power usage has the effect of either reducing sensor lifetime or increasing the required size of the energy storage or harvesting options, neither of which are an option on the majority of devices. To avoid resorting to these options the power usage of the nodes needs to be reduced. The lack of sleep states on IoT devices is by far the single largest cause of the difference in energy consumption between existing sensor network systems and internet connected systems. Enabling these sleep states will bring the energy consumption of IoT down to the levels expected of current sensor network technologies.

The barrier to implementing these sleep states is the loss of connectivity caused by powering off the radio hardware. Current internet technologies have not been designed to support challenged devices without continuous connectivity, this results in connection problems with sleeping devices. Chapter 4 investigated options for providing connectivity to challenged sensor network devices. A high level network simulator was created to evaluate options for resolving these issues. Through those tests, storage and forwarding of packets has been shown to be an effective method of enabling reliable packet delivery on such intermittently connected networks.

In chapter 5 a new DTN protocol for use on constrained IoT networks has been developed and tested. Current DTN solutions were not well suited for use on constrained IoT deployments, so investigations were carried out to design a protocol which would meet those requirements. Through this work a new IPv6 HbH option header was developed to bring DTN capabilities to challenged IPv6 networks. This header forms the new state of the art for DTN technologies for IP networks.

The header was designed to use the minimum number of additional bytes in order to minimise overhead. To achieve this a new relative time-stamp format was developed for the header; with this new format times can be compressed down to two bytes while maintaining an accuracy of better than 0.1% and lifetimes of over four years. The developed header has been tested in real world and simulated networks in order to test its capabilities. Implementations for both constrained devices (in the form of a Contiki implementation) and more capable devices (in the form of a Linux implementation) were used in these trials. These form the first end to end DTN implementation for constrained IP networks, and while they still have some bugs they clearly demonstrate the potential of such a solution.

6.2 Use of the Research

The contributions discussed in this thesis will have a significant effect on the development of future internet connected sensor systems. Bringing DTN into embedded internet based sensor networks instantly resolves many of the connectivity and energy problems faced by those systems.

With support for delayed messages, new options for innovative link layer technologies or deployment options which do not provide immediate data transfer are possible. While the only current link layer protocol to feature extended transmission times in normal operation is rfc1149 ‘A Standard for the Transmission of IP Datagrams on Avian Carriers’ (Waitzman, 1990) there are many more serious use cases where this capability can be used. Examples of such systems might be an optical link that only works at night, communication links using satellites in low earth orbit, or links in the Extremely Low Frequency (ELF) or Super low frequency (SLF) radio bands which have extremely low bandwidths causing messages to take a significant time to transmit.

The research conducted in chapter 4 has already been used as part of other research and has been cited in those authors publications. examples include: Young et al. (2015) in their paper on “Image analysis techniques to estimate river discharge using time-lapse cameras in remote locations” and Neto et al. (2015) in their paper on “A Robust and Lightweight Protocol for Efficient and Scalable Automatic Meter Reading using an Unmanned Aerial Vehicle”.

6.3 Future Work

As with almost all technological research, the work carried out in this thesis has identified new avenues for investigation. Several examples of which are suggested below.

The use of DTN on a network transfers the choice of when to retransmit messages from the sender to each node in the chain. These nodes have the potential to make better informed decisions as to when to retry as they will be closer to the source of the problem. In order to do this, however, intelligent methods of determining the likelihood of successful retransmissions are required. Efficient algorithms for different types of networks must be developed, and different approaches will likely be needed for different network technologies. For some networks which have an active reconnection system the availability of devices can be determined by monitoring which devices are connected to the network at any one time. Other networks, however, will need alternate approaches to know the schedules of devices on their network in order to minimise retransmission attempts. While this may seem to be inefficient in terms of development time, the reduction in packet retransmissions by improving the retransmission algorithms has the potential to improve the efficiency gains from DTN significantly.

Current trials with the DTN option header have focused on unicast traffic. This however is not the only kind of network addressing scheme available in IPv6. In addition to unicast traffic there is also Broadcast, Multicast and Anycast. These flows serve important functions within the IP networking space. In order to decide how DTN packets should be handled for these use cases additional research needs to be carried out.

With the increased transmission times available through the use of DTN there is an increased potential for devices to move their physical location while a message is in transit. If they change the network they are connected to they will get a new IPv6 prefix resulting in a change of their global IPv6 address. If the address changes, then the address used to send packets already in flight will no longer be valid. A potential solution to this problem might be to use Mobile IPv6 (Perkins et al., 2011) where a node can have a fixed home address as well as their mobile address. Research is needed to determine how such a technology could work alongside DTN and to determine what new features or capabilities are required to do so.

So that the protocol can gain traction in commercial devices, the protocol would need to go through some form of standards track in order to create a fixed point for implementations to conform to. The Internet Engineering Task Force (IETF) would be the logical place to publish the header as a standard.

6.4 Final Summary

In the future we will be looking for ever more automation and intelligent decision making by our everyday devices. In order to provide this we will need low power sensing and actuator systems with reliable and easily interfaced communications systems. The addition of DTN to the IP networking space provides the required communications capabilities to support these challenged deployment scenarios.

Applying these systems to EWSNs greatly improves the flexibility of the system compared to traditional non-IP sensor systems. The increased flexibility and reduced development times for deploying sensor networks make it far more economically viable to deploy such networks. Rather than requiring teams of engineers to develop and deploy bespoke sensor networks, networks will be able to be designed with off the shelf sensors and networking components from a variety of manufacturers.

With the work presented in this thesis the capability for ultra low power sensor networks to use resilient end to end IP networks is now a possibility. Previously IoT enabled sensor networks needed to be power-hungry systems in order to maintain reliable IoT communications. Existing DTN solutions generated a significant overhead burden which was problematic to fulfill on embedded platforms. The DTN header developed and evaluated in this thesis resolves this issues by providing an efficient method to enabling reliable connectivity on challenged networks.

Appendix A

Publications

In the course of this research contributions have been made to several published papers accepted for publication, these are included on the following pages.

A.1 Using Internet of Things technologies for wireless sensor networks

Martinez, K.; Hart, J. K.; Basford, P. J.; Bragg, G. M.; Ward, T.

Numerous authors have envisioned the future internet where anything will be connected: the Internet of Things (IoT). The idea is an extrapolation of the spread of networked devices such as phones, tablets etc. Each device is expected to have its own Internet address and thus be easy to access. The key building blocks of any IoT system are networking, hardware platforms and node software - so they are similar to wireless sensor network requirements. Most existing IoT demonstrators and applications have been gadget-style objects where power and connectivity problems are not too restricting. Environmental sensor networks can benefit from using some of the technologies involved in IoT development. However it is expected that tuning the networking and power management will be necessary to make them as efficient as state of the art wireless sensor networks. Some IoT assumptions such as always-connected nodes and full IP capability need to be considered. This paper will illustrate the advantages and disadvantages of IoT techniques for environment sensing drawing on a range of employment scenarios. We also describe a glacial 'Internet of things' project, which aims to monitor glacial processes. In particular we describe the IoT developments in a deployment in Iceland to examine glacier seismicity, velocity and provide camera images.

Simulated Analysis of Connectivity Issues for Sleeping Sensor Nodes in the Internet of Things

Tyler Ward, Kirk Martinez, and Tim Chown
Web and Internet Science, Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
t.ward@ecs.soton.ac.uk, km@ecs.soton.ac.uk, tjc@ecs.soton.ac.uk

ABSTRACT

The growth in wireless sensor network deployments requires a move towards more standardised systems to improve compatibility and to reduce development times. The technologies being developed as part of the Internet of Things, such as 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks), can greatly assist with this aim. Connecting low power wireless sensor network devices to the Internet of Things presents certain challenges. One of these challenges is the lack of constant connectivity to sensor nodes with sleep states. Current internet technologies expect that devices are always contactable which is not the case in sensor networks. We simulate and evaluate several solutions to this problem in a multitude of different scenarios. We conclude that delay tolerant networking is an effective solution to the challenges created when dealing with sleep states while minimising overheads. However, current standardised delay tolerant technologies are not easily applicable for use with sensor networks, so a new standard needs to be created to meet the requirements described in the paper.

Categories and Subject Descriptors

C.2.0 [**Computer-communication networks**]: General—*Data Communications*

Keywords

Sensor Networks; Internet of Things; Wireless Sensor Networks; Delay Tolerant Networking

1. INTRODUCTION

In the past, wireless sensor networks have often used customised hardware, software and network protocols for each deployment[1]. The continuing growth in the need for wireless sensor network deployments limits the sustainability of this approach. This methodology also causes problems when trying to connect sensor networks to each other, or other devices to sensor networks [5]. One example is future home networks with devices connected over a mixture of low power and traditional links. Connecting sensor networks to the Internet and making them part of the Internet of Things is one solution to this problem. This also has the potential to unlock additional capabilities using the improved connectivity. Much work has already been done under the Internet of Things banner to make this a possibility[8].

However, in order to benefit from migration to the Internet of Things there are several areas which still need further research in order to support the requirements of wireless sensor network devices. Many sensor nodes need to spend most of their time in low power sleep states with communications turned off[1]. Although improvements in low power radio technology are allowing greater use of the radio for the same sensor lifetime[4], these improvements are unlikely to alleviate this issue entirely, especially for nodes that will remain battery powered for years. This restriction is incompatible with the expectation of continuous end-to-end connectivity as assumed by the majority of Internet connected systems.

In previous non IP (Internet Protocol) systems this problem is handled by an application level gateway which could respond in various ways, such as informing the enquirer that the node is asleep or serving cached data. With current IP systems when a host attempts to communicate with a device that is in a sleep state the message is lost as it cannot be delivered to its destination. There are methods in place to allow senders to be informed that a message could not be delivered, such as the destination unreachable message, however, this only gives a rough idea of why a node is unreachable. There is no guarantee that such a message would even be sent or that they will not be dropped by the network.

Whilst the issue of communicating with sleeping nodes has been identified as a challenge[2, 13], as yet there has only been a limited amount or research into this area. Current solutions rely on the node initialising all communication with the outside world, with the inherent expectation that the rest of the world is always accessible[13].

PE-WASUN'14, September 21–26, 2014, Montreal, QC, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3025-1/14/09 ...$15.00.
http://dx.doi.org/10.1145/2653481.2653490.

2. IMPROVING COMMUNICATIONS WITH SLEEPING INTERNET OF THINGS DEVICES

Traditional wireless sensor networks are not usually expected to receive traffic from Internet devices. However, the Internet of Things has the expectation of the same bidirectional communication capabilities that also exist on the non constrained Internet.

There are many different solutions to overcome the communications issues faced when dealing with sleeping Internet of Things devices. All of the current potential solutions come with an attached overhead which needs evaluating. Many of the solutions require firmware changes on the devices or software changes in the software interfacing with the devices. Sleeping nodes are not yet commonly deployed on the Internet so there is little backwards compatibility that needs to be maintained. There are existing software tools for management and control which can not be modified easily for which maintaining compatibility would be advantageous.

2.1 Node Initiates all Communication

The current suggested solution to this issue is to rely on the node to start all communication[13]. This avoids the issue of needing to know when the node is on. Whilst this method is appropriate in certain circumstances, it imposes limitations that can hinder certain tasks.

Where a node needs to have changes to its configuration parameters performed remotely it will need to periodically poll a control server to check if updates are required. It is worth noting that as the node initiates the communication, what it needs to talk to and how often becomes part of this configuration. The frequency of this checking becomes an important decision, too frequently and the node uses unnecessary power, too infrequently and changes to the node's configuration take a long time to apply.

In the situation where communications are not reliable or are not guaranteed to be available, there is the potential for wasted power. For example, A relay node may be unable to relay traffic due to power loss, or damage. During this period the node will need to continue to attempt to communicate as it will be unable to tell if there is a working connection until it sends a message. In environmental sensor networks such as Glacsweb[9] these communication outages can last many months. By having the request come to the node it will avoid transmitting when there is no connection as no requests will be received. A similar scenario can occur if systems are no longer interested in data from a node for a period of time. If the system either does not or is unable to inform the node this is the case, then the node will not be aware and will continue to transmit data that is not being utilised.

One scenario where this has a significant ease of use benefit over other technologies is when the node is behind access controls such as firewalls or NATs (Network Address Translations). For this reason this method is used in many Internet of Things smart home devices. They phone home using IPv4 to avoid users needing to make network configuration changes. The use of IPv6 can remove the limitations of NAT, facilitating easier communication between devices, however, IPv6 has not yet reached the stage of widespread adoption [3]. Firewalls will still need configuring for use with IPv6 however the user will have more control than with IPv4 firewalls using NAT. It should be noted that in several cases such as Violet and their Nabaztag product the phone home server for such devices has been shut down rendering them inoperable [17].

2.2 Polling the node

The most basic solution available is to repeatedly send requests until a response is received. This is a particularly bad solution with regards to conserving sensor network resources. In order to catch the node when it is awake, the host needs to poll the node frequently enough so that the node cannot wake up and go to sleep in between messages. The nodes will need a reasonable amount of on time to avoid needing to poll too frequently. Even with a long amount of time with the radio on, this method generates a lot of unnecessary traffic. This traffic will impose an increased load on other nodes that are awake as they will need to evaluate whether that packet is for them or if they should forward it on. Many systems doing this will consume a significant proportion of a low bandwidth communication link, this is especially true when the bandwidth bottleneck is the sensor itself rather than the channel bandwidth as the sensor may miss other important messages.

This method does not require any changes to be made to either the network hardware or to any node firmware. Providing that the software on the computer making the queries can be configured to tolerate intermittent connections then there should not be an issue. Many pieces of software, however, are unable to be configured like this.

2.3 Time Coordination Between Data Sources and Sinks

If the periods when a node is able to receive communications are known in advance then it is possible for hosts that want to request data from a sensor node to time their requests so that they fall within the communications window. This solution does not require any additional equipment or always on devices.

The major downside to this solution it that the software running on the remote host has to be configured with the node's schedule. Software that is unable to be scheduled or nodes with unpredictable schedules cannot utilise this approach.

In order for the devices to keep their schedules in sync both devices need to maintain an accurate clock. Depending on the capabilities of the sensor device it may also be necessary to add an RTC (Real Time Clock) module to maintain an accurate enough time. A sensible method of maintaining the clock is to use NTP (network time protocol) or similar clock synchronisation protocols. Depending on the quality of the RTC this could be done quite infrequently, this adds some additional network traffic and processing but not a huge amount. This method also requires that a node is powered up for long enough to accommodate any time drift between them and the remote host.

2.4 Transparent Proxy

With some sensor systems a proxy server can be used in order to make it appear as if the nodes are always on. With this solution, when the hosts request the data from the node they are instead sent to a proxy server. This server can forward the request for the data when the nodes are on-line and store the responses for later. When the nodes are offline it can reply with cached copies of the data. Some protocols such as CoAP (Constrained Application Protocol) have inbuilt support for this function [16]. Although the proxy can only understand some types of request, unrecognised requests or communication protocols can be passed onto the network to provide compatibility with other systems which are on the same network.

Multiple requests to a single node for data can be responded to without the need to request data from the node again. There is a risk of data not being refreshed or not being available if data is not requested while a node is on. This can be solved by the proxy automatically updating its cache of data by talking to the nodes itself, however, with this the proxy begins to act more like the data distribution system solution.

Messages to control a node will need to be passed through the proxy in order to be received by the node. This means that those messages will need to be correctly timed in order to be acted on.

2.5 External Data Distribution System

On the majority of non Internet connected wireless sensor networks, the data from the sensor nodes is gathered together in one place where it can be requested by networked devices. While this was almost essential in previous systems the Internet of Things mainly relies on direct node communication which will scale badly when there are more interested parties. By having a device perform this data management it can drastically reduce the complexity required for the majority of devices. As they are requesting the data from an always on system they no longer need to worry about communication with sleeping nodes. Although this solves the problem for the majority of the hosts it does not solve the problem for itself. As there is usually a single data store, it could have the nodes push data, or could request data from the nodes using another of the solutions discussed.

With this solution other hosts are limited to the data collected by the store and the rate at which the data store collects it. As the data store needs to understand how to request the data from a node there is a requirement that the data store and node share a common protocol and data format. Multiple hosts interested in data from a single sensor do not increase the amount of communications to a node.

The server can also convert the data into multiple formats and consolidate sensors with otherwise incompatible data formats or protocols into a common format presented to interested parties. In order to do this the data distribution server needs to understand the format of the data in order to present it to other interested parties.

There may initially seem to be little difference between using a data distribution system and the existing proprietary systems. While a data distribution system might serve the majority of requests for data on a network, other IP based systems which would like direct access can still achieve this. A caveat with data distribution systems is that while they are suitable for allowing many devices to get data from a system they are unable to handle sending messages the other way, thus making this solution redundant for actuator nodes.

2.6 Message Queues

Message queues are commonly used in systems where multiple processes need to pass information to one another either on the same machine or somewhere else on the network. This technique could also be used to coordinate communication between sensor nodes[7].

In this set-up, nodes would operate by pushing data to a queue on a central machine whilst also checking for any messages that may be waiting for it. Hosts can subscribe to the queues published by the node, in order to send a message it can be added to the nodes queue to be collected later. The data sent would encapsulate the higher level protocol or raw data in a message queue packet, this breaks the end to end IP connectivity that is gained by moving to the Internet of Things.

Similar to data distribution systems and proxies the message queues allow for many data consumers without increasing load on a node. Unlike data distribution systems however it can also handle messages being sent to a node. In order to do this a node will need to send a message to find out if there is a message waiting for it, although this can also be done as part of sending data in.

2.7 Delay tolerant networking

A common solution in non Internet connected sensor networks is to use store and forward technology to allow data to be sent when a link is reestablished. IP does not have support for these features, although it can be added at a higher layer usually referred to as delay tolerant networking.

While it is possible to simply hold the messages intended for a node until it wakes up, issues begin to arise as the times between on periods increase. As the time extends beyond a few seconds hosts will assume that the message has been lost, in order to solve this, there needs to be some information sent with the message in order to define how it should be handled. For example, most messages sent to sensor networks have a finite time in which they are relevant, after said time there is no point in sending them any more as all they will do is use unnecessary bandwidth and battery life.

The delay tolerant Bundle Protocol[15] can encapsulate IP traffic to allow store and forward operation. This encapsulation adds another layer that needs to be understood and decoded in order to process the message. Although the Bundle Protocol has successfully been ported to sensor networks[14] it is not a perfect solution. It is designed to support a large range of protocols and results in a long header; this is obviously not ideal for use in power constrained devices.

A potential improvement to this solution would be to have a delay tolerant IP header that can allow equipment supporting delay tolerant features to handle the packet appropriately without the need for significant extra overhead. A useful feature of using an IP header over adding an additional layer is that if a node cannot understand the header it can still process the message thus removing many compatibility problems that would otherwise arise.

3. TESTING SCENARIOS

Due to the scope of potential Internet of Things sensor deployments the potential solutions will need evaluating in a mixture of different practical network scenarios.

We review five scenarios, with figures 1 to 5 showing example layouts of what the different network topologies look like.

3.1 Real-time Sensing Network

In many cases the data from sensors is needed for processing as soon it is recorded. Examples of such sensors are thermostats and water level sensors[6] where their data is used to keep an environment or system within certain parameters.

Some sensors of this type will be always-on devices, however, this will cause battery powered nodes to consume their energy reserves faster. In order to extend the battery life, these sensors can turn on for short periods regularly to

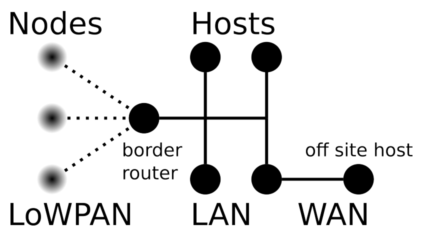


Figure 1: example real time network

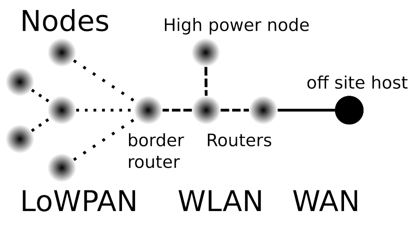


Figure 2: example environmental network

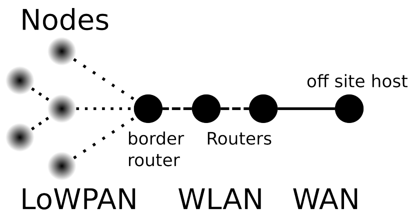


Figure 3: example event detection network

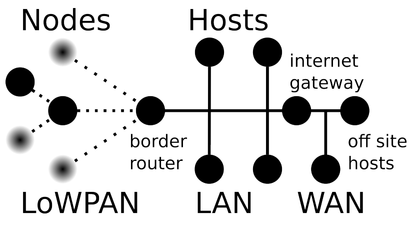


Figure 4: example mixed network

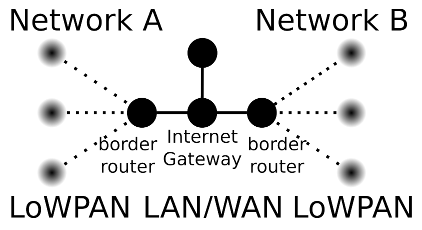
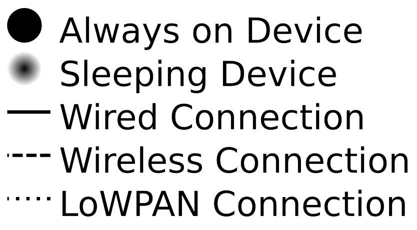


Figure 5: example multiple network setup



Key for the network layouts

record and transmit data and spend the rest of the time in a sleep state. These networks are likely to have a reliable connection to the wider Internet with a border router that is probably an always on device.

3.2 Environmental Sensing Network

Like the real-time sensing networks these networks will be recording data about their environment, however, there is no need for the data to be processed immediately [10]. This allows the data to be reported back in chunks with longer intervals in between. This saves power as the radio systems are used less frequently.

These 'log and send later' sensor nodes are regularly used when studying the environment or recording data for historical or evaluation purposes. Often replacing the battery on these types of device is difficult or even impossible.

Networks of this type often have routers which are themselves power constrained and likely to need sleep states in order to conserve power. There may be several of these low power routers between the sensor network and the Internet for networks in remote locations.

3.3 Event Monitoring Network

These sensors are intended to report events back as they happen rather than polling the environment. Examples of these sorts of networks are detection of natural hazards such as landslides or volcanic activity[18], similar principles would also apply to devices in more everyday scenarios such as fire alarms or security systems.

While these networks will usually push information when events happen there are many cases where it is necessary to communicate with the node, for example, to configure its sensitivity or confirm that it is still working. Such networks are likely to have an always on base station and attached communication infrastructure in order to ensure messages are relayed to their destination and never lost.

3.4 Mixed Networks

Due to the flexibility of the Internet of Things, a single network may include devices with different or multiple capabilities. An example of this is a HVAC (Heating Ventilation and Air Conditioning) system that shares a network with a security system.

Multiple network types sharing common infrastructure reduces the cost of providing multiple types of sensing systems or changing an existing deployment as new routers aren't required. In these networks sleeping nodes may coexist with non sleeping nodes that are not so power constrained. In such networks, the border router and its upstream connectivity will need to cope with the requirements of all of the nodes on the network, this will mean that it is probably going to be an always on device.

3.5 Internetwork Communication

The Internet of Things allows for far greater flexibility compared to existing sensor networks for machine to machine communications [19]. Internet technologies make it easier for devices to communicate to other devices inside the same network or in another network. There is a major issue when trying to do this for sleeping devices as the schedules for two devices might be completely different.

An example of this type of network could be an irrigation control system. An intelligent device that informs a user if a

Table 1: Solution applicability in different environments

	Test cases				
Solution	Real-time Sensing	Environmental Monitoring	Event Monitoring	Mixed Network	Multiple Networks
Polling	Yes	Yes	Yes	Yes	No
Time Synchronisation	Yes	Yes	Yes	Yes	No
Transparent Proxy	Yes	No	No	Partial	No
Data Distribution System	Yes	Yes	No	Partial	Yes
Message Queues	Yes	Yes	Yes	Yes	Yes
Delay Tolerant Networking	Yes	Yes	Yes	Yes	Yes

sluice gate is in the wrong position might use data from the network on the field as well as data from a sensor network monitoring water flow on a nearby river.

With multiple sleeping nodes direct communication becomes impossible if devices are running on different schedules. This communication could be coordinated by devices on the wider Internet or a local server. These solutions would require a reliable Internet connection or stable power supply respectively in order to operate, which might not be practical in extremely remote areas. Alternatively, using delay tolerant networking border routers could buffer packets until the other device is contactable.

3.6 Solution Applicability

While some of the solutions proposed earlier will work with any of the test cases, many are situational and either provide no benefit or cause issues for other test cases. For example, a proxy setup provides no benefit for environmental sensor networks as each data upload is different and ideally is only transmitted once.

Table 1 shows the solutions that will work with different test cases. Although some of these are obviously bad solutions, such as polling for a node that only turns on for a few seconds each hour or day, these were still included for comparison.

4. SIMULATION SOFTWARE

To evaluate all of the potential solutions in the appropriate test cases using real sensor network hardware would take a considerable amount of time and resources. Although the nodes could be emulated at a low level using tools such as Cooja [12] to avoid hardware costs this would still require a significant investment of time for the creation of device firmware. Instead a network level simulator was used, which simulates the sensors at a high level avoiding the complexity of hardware interfaces and encoding or decoding software protocols. Using a simulator that only focusses on the interested layers to simulate the nodes will still produce realistic results.

Different nodes may have different application layers and sensing capabilities which will put different requirements on a node. It has been assumed that the effect on the node from the higher layer protocols and sensing cycles can be broken down into a fixed load that the node will perform regardless of what happens on the network, and an additional load for every request. As the fixed load is independent of the effects of the network it can be ignored. The additional load introduced for each request is assumed to use the same amount of resources and as such can be included as part of the load for the node to process a message.

Any alterations to the network layer are not going to affect the way the lower layers or the underlying hardware will behave. As a result it can be assumed that the load imposed by the lower layers for any volume of network traffic, will be the same as any other traffic.

The changes to some of the node systems as required by solutions such as message queues and delay tolerant networking will add some additional load onto the node. Given how little of the total energy required is needed to service the packet handling logic for a request, any changes to this logic required to manage different solutions will probably have a negligible power impact. The additional bytes that might need to be transferred for those systems, however, will increase power usage of the radio which will need to be taken into account.

There will also be some load introduced onto the node whenever a packet is detected by the radio, the packet might not be intended for that node but the node will still need to confirm this. With these assumptions the total load of the node can be evaluated by using the amount of packets a node sends and receives, the other loads on the node are proportional to this or are a fixed drain on resources.

As all of the elements in the system can be modelled as events the simulator was created using the SimPy[11] discrete event framework. Each of the elements in the simulation would be triggered by events from either an internal timer or from other events such as local processes or network activity.

In addition to simulating the end devices on the network the links and routers between devices will also be simulated. The simulations will model the flow of every packet across the network and record important metrics about the packet's transmission. Factors such as latency, jitter and packet loss are taken into account at each hop across the network. The transmission time is modelled using an average transmission time with a jitter added using a standard deviation model. Packet loss is modelled by randomly dropping packets based on the link's loss rate. For wireless links there can only be one message on the channel at one time. This is modelled by delaying transmissions until the channel is clear, representing the use of a clear channel assessment system.

The solutions were separated into two categories, those that altered the timing of messages and those that added features. Along with the timings that would be a potential solution, additional timings were added, for example, ones chosen randomly or offset by a small amount of time. These timing options can then be used when testing the network with and without different additional features.

The simulator records information about if and how quickly each of the devices are responding to requests as well as the

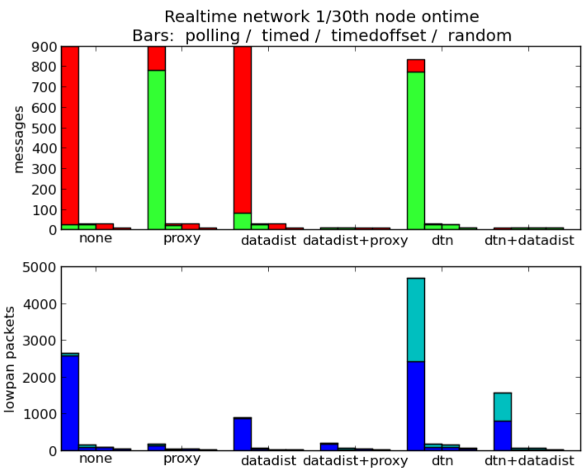


Figure 6: Success rates and quantity of messages sent on a real-time sensing network.

amount of traffic being sent over each of the network connections. The simulations can be run on each of the test cases to determine the effectiveness of solutions with different network and sensing configurations.

5. SIMULATION RESULTS

The two main statistics that were used for initial analysis were the amount of network activity generated on the lowpan network and the success or failure of queries sent by other devices.

The amount of network traffic used to communicate with the nodes is a good measure of how effective the solutions are, providing that these messages are also successful. If the required communication can be achieved with less traffic on the lowpan network then less power will be needed to send and receive thus allowing a greater sensor life and reducing network congestion.

Figure 6 shows multiple potential solutions on a real-time sensing network with a node on for a 30th of the time for 15 node cycles, for example a node on for 10 seconds every 5 minutes for an hour and a quarter. The data distribution server was set up to collect data using the request timings that would normally be used by the hosts. The hosts collecting data from the server were instead set to use a random request schedule as hosts using a data distribution system shouldn't need to use specific timings.

The Y axis is the amount of requests for data and message passed on the lowpan network. The request count bar shows how many requests where sent from hosts and how many where responded to. Light green indicates a response was received, the darker red that no response was received. The lowpan packet count bars show the directionality of the data, the lighter blue was sent by a node where as the darker blue was sent from the border router. The bars are grouped based on what features are enabled on the network, labelled on the X axis. Within those groups they represent the request timing used, labelled above the graph.

As would be expected, polling causes a massive amount of network traffic on a real-time sensing system when compared to the other solutions. This will increase as the percentage of time when the node is off increases such as in environmental sensor networks. Timed coordination of nodes works well when the schedules are aligned, however, should they become out of synchronisation then communications to the node are completely lost until the devices can regain synchronisation.

Figure 7 shows an example amount of traffic on the test case networks during a day. The polling solution has been removed from this graph to allow easier visibility of the other solutions. The colour scheme and layout is the same as used in Fig 6. The small amount of failed messages encountered with the majority of systems was due to the packet loss programmed into the simulation, this packet loss was modeled at 5% for the lowpan network with regular wireless links at 1% and wired links at 0.1%.

In the case where many hosts are wanting to request data from a single node the systems that could redistribute data had a major impact on the amount of data. In order for the proxy server to operate it requires requests that are correctly timed in order to update its cache, if nothing is requesting on the correct schedule then it will be unable to service requests that fall outside those times. Data distribution systems while effectively managing the amount of queries being received from other devices, were only truly effective when combined with other solutions. Message queues also resolved this issue, however, they have the unfortunate requirement that messages need to be in the message queue protocol, this breaks the end to end IP level connectivity and hence removes a lot of the benefits from moving to an Internet of Things platform. Nodes also need to query the message queue to determine if there are any messages for them. While this can be included when sending data to the message queue, should no data be being sent then a separate message will be required.

In the multi network test case it can be seen that only those solutions able to buffer either data or packets are able to maintain communications with multiple data providing nodes due to the mismatched schedules. Alternatively it would be possible to have the node fetching the data turn on for each schedule to request the data, this requires that the node knows all connected schedules and requires the node to use more energy. Schedules could be aligned so that there is a single schedule for all data sources, this avoids requiring additional wake cycles for the node requesting the data. Unfortunately this solution imposes quite a few constraints on those deploying a network and as such should not be relied upon.

Delay tolerant networking universally improves the situation for sleeping nodes allowing reliable communications without the need for end to end time synchronisation. In order to describe how the packets should be treated, and how much delay is acceptable, extra information needs to be transmitted with the packet. This could be done using IP headers, this will add a few additional bytes to any message sent with this feature, messages that do not require this will not have any additional overhead. While it is possible to implement delay tolerant features without adding any overhead this gives no information about how packets may be delayed and can lead to situations like the one that follows.

It was found when hosts were trying a polling style request with delay tolerant networking enabled and the node off period was extended, that the success rate of requests would drop off. This is caused by the traffic from the entire

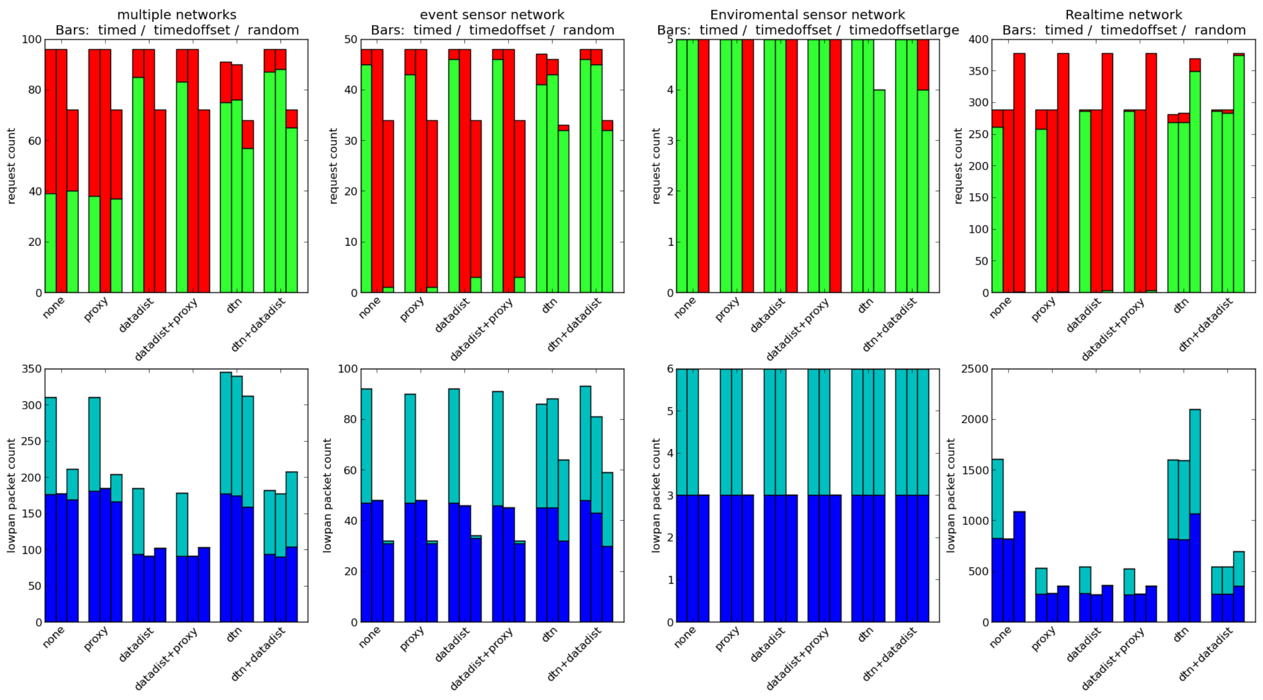


Figure 7: Network utilisation during a 24 hour simulation modelled with different test cases and solutions

period when a node is off being condensed into the nodes on period, causing the network to become overloaded similar to a denial of service attack. This emphasises the need for appropriate precautions to be taken such as detailing the useful life of a packet to prevent such an issue whether accidental or deliberate. With the bandwidths available to a single normally networked host it would be easy to disrupt a large number of lowpan networks.

In order to forward the packets at the correct point the border router with delay tolerant capabilities will need to know when the node is on-line. Depending on the network technology and configuration used this can be done in one of several ways. Most of these methods will not require any additional communication from the node. The router can be informed of the node's schedule as in the time synchronisation solution and use this to know when to forward the messages. This obviously inherits the synchronisation issues faced by the timed solution, however, due to their proximity maintaining synchronisation of clocks and schedules is an easier task and could potentially be included in a DTN header.

For some networks the node will need to communicate with the border router in order to connect to the network when it turns on. The border router can use this to detect that the node is able to receive communications. As the router is on the same network as the sensor node it can detect any communications the node may initiate, even if those messages do not leave the network, as a way of determining if the node is on-line. If none of these are available it is also possible for the node to inform the border router when it turns on similar to message queues. However, this will be more efficient than message queues as the communication delay to the border router will be reduced, in addition should the border router not have upstream connectivity the message can still be collected providing it has already made it to the router.

6. CONCLUSIONS AND FUTURE WORK

As the need for very low power sensor devices connected to the Internet of Things increases, the ability to communicate with devices in sleep states will become increasingly important. While it is possible for nodes to initiate all communication this limitation will restrict what can be achieved with the network. Needing to poll for configuration changes leaves them vulnerable to having an invalid or out of date configuration, which may waste power.

Delay tolerant networking has been shown to have the capability to massively reduce issues with communications between nodes and outside services due to the effects of sleeping devices in sensor networks. Although existing delay tolerant protocols do not lend themselves for use on constrained IP based sensor networks due to their size and the fact that they encapsulate the message, the principles can be used to develop protocols which are. In order for these protocols to be efficient they will need to use a minimum of bytes as well as avoiding any extra communications wherever possible.

The next stage will be to develop and evaluate using IP based delay tolerant networking technologies on real world networks and systems. This will be compared with existing technologies and deployments through field trials and further simulation at a lower level. During this a design for a lightweight delay tolerant networking header suitable for use on both constrained and unconstrained networks will be developed. This protocol will need to be designed to minimise the amount of bytes transmitted, from both bytes added to messages and new messages, in order to operate efficiently on constrained networks, whilst still having the capabilities available in the larger implementations. It should also allow end devices and intermediate routers which do not understand the protocol to be able to process the message. It is believed that the use of IP headers to add the delay tolerant features will achieve these requirements.

7. ACKNOWLEDGMENTS

The authors would like to thank the teams from the Glacsweb and Mountain sensing projects.

8. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications magazine, IEEE*, 40(8):102–114, 2002.

[2] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[3] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey. Measuring ipv6 adoption. Technical report, Technical Report TR-13-004, ICSI, 2013.

[4] A. Dunkels. The contikimac radio duty cycling protocol. 2011.

[5] M. Durvey, J. Abeillé, P. Wetterwald, B. O'Flynn, Colin Leverett, G. Eric, M. Vidales, M. Geoff, N. Tsiftes, N. Finne, et al. Making sensor networks ipv6 ready. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 421–422. ACM, 2008.

[6] J. Gustafsson, H. Mäkitaavola, J. Delsing, and J. Van Deventer. Integration of an ip based low-power sensor network in district heating substations. In *The 12th international symposium on district heating and cooling*, 2010.

[7] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. Mqtt-s a publish/subscribe protocol for wireless sensor networks. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 791–798, Jan 2008.

[8] L. Mainetti, L. Patrono, and A. Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*, pages 1–6, Sept 2011.

[9] K. Martinez, P. J. Basford, D. D. Jager, and J. K.Hart. Poster abstract: Using a hetrogeneous sensor network to monitor glacial movement. In *10th European Conference on Wireless Sensor Networks*, February 2013.

[10] K. Martinez, J. K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8):50–56, 2004.

[11] K. Muller and T. Vignaux. Simpy: Simulating systems in python. *ONLamp. com Python Devcenter*, 2003.

[12] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648, nov 2006.

[13] B. Ostermaier, M. Kovatsch, and S. Santini. Connecting things to the web using programmable low-power wifi modules. In *Proceedings of the Second International Workshop on Web of Things*, page 2, 2011.

[14] W.-B. Pottner, F. Busching, G. von Zengen, and L. Wolf. Data elevators: Applying the bundle protocol in delay tolerant wireless sensor networks. In *Mobile Adhoc and Sensor Systems (MASS), 2012 IEEE 9th International Conference on*, pages 218–226, 2012.

[15] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), Nov. 2007.

[16] Z. Shelby, K. Hartke, and C. Bormann. Constrained Application Protocol (CoAP). Active Internet-Draft, June 2013.

[17] J. Volpe. Mindscape pulls the server plug on nabaztag, hands source code to developers. *http://www.engadget.com/2011/07/28/mindscape-pulls-the-server-plug-on-nabaztag-hands-source-code-t/*, 2011.

[18] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *Internet Computing, IEEE*, 10(2):18–25, March 2006.

[19] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. Johnson. M2m: From mobile to embedded internet. *Communications Magazine, IEEE*, 49(4):36–43, April 2011.

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2015) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

The 10th International Conference on Future Networks and Communications (FNC 2015)

Adding support for delay tolerance to IPv6 networks

Tyler Ward[a], Kirk Martinez[a], Tim Chown[a]

[a]*Web and Internet Science,Electronics and Computer Science, University of Southampton, United Kingdom*

Abstract

As we continue to connect ever lower power and more power constrained devices to the Internet of Things the problem of maintaining constant end to end connectivity becomes harder. Accepting that continuous end to end connectivity cannot be maintained, we are forced to seek solutions to allow good operating function. Delay Tolerant Networking, an evolution of existing store and forward systems is a candidate for resolving this issue, however, current implementations are not ideal for use in constrained Internet of Things environments. We propose a solution to this by integrating the capabilities of Delay Tolerant Networking into the IP layer, in such a way as to maintain compatibility with existing and future systems and minimising additional overhead. This has been achieved by developing a new IPv6 Hop by Hop option header which contains the information required for messages to be delayed. This solution is then demonstrated to be implementable within the limitations of current Internet of Things hardware.

Keywords: Delay Tolerant Networking ; IPv6 ; Sensor Networks

1. Introduction

Maintaining continuous connectivity with Internet of Things devices is not always possible. This causes issues as continuous connectivity is the expected behaviour for internet connected devices. The reasons for the lack of connectivity can be due to a range of factors both deliberate and incidental.

The main reason for deliberately causing a lack of connectivity is to save power. Keeping communication hardware in a receive state is a significant drain on the energy resources of small battery powered devices. A method to reduce the power use of devices in sensor networks is to put the device into a low power sleep state for as long as possible. In this state the device has its communications powered down so is unable to be contacted but will consume substantially less energy. An alternative solution to this is to provide more energy resource to the device. Doing this limits the options for placement of these devices as they will either require access to a power source or be physically larger to accommodate the additional batteries. As a result this has become a trade off between connectivity and power consumption.

* Corresponding author. Tel.: +44 (0)23-8059-4583.
E-mail address: tw16g08@ecs.soton.ac.uk

With regard to incidental loss of connectivity, the environment in which the devices operate can cause breakdowns in communication. As an example, in remote sensing systems the weather can cause communications links to become unavailable. In the Glacsweb network, the winter snowfall often buries the equipment and prevents surface radio communication and in the summer, increased water levels within the ice can interrupt probe communications[1]. These issues are difficult to resolve and must be anticipated as a part of the deployment challenges.

Rather than trying to solve these issues individually the lack of low level connectivity can be addressed at a higher level. Currently many internet connected devices that have the issue of limited power reserves rely on the device initiating all communication rather than listening for any incoming communications. This relies on having an always available route to an endpoint which they communicate with, and requires all communication with the device go through that endpoint. While this has worked for current devices, this solution reimposes many of the limitations that had been removed by using direct internet connectivity rather than proprietary systems. Through previous research, Delay Tolerant Networking (DTN) had been shown to have potential but required additional research to make it suitable for IP based networks[2]. This paper covers the implementation of IP extension headers as a means of implementing DTN and discusses the initial testing of the system in Cooja simulated nodes then on Zolertia Z1 hardware.

2. Delay Tolerant Networking

With Internet Protocol (IP) networking technologies a message is either immediately delivered to its destination or is discarded. This is inconvenient when there may be many interruptions in connectivity as the originating device will need to keep retrying the transmission. While some link layers support retransmissions, these are intended to protect against packet corruption and occasional packet loss rather than loss of connectivity. In many cases the contents of the packet do not need to be delivered immediately and are still useful if they are delivered at a later time. The one solution to this is to allow nodes on the route to buffer packets and send them on when connectivity to the next hop has been restored. This is similar to existing store and forward technologies used in current sensor networks[3].

Store and forward is often used in sensor networks to allow packets to travel through the network one hop at a time. These existing systems are usually limited to a specific link layer protocol or application layer protocol. This is a workable technology when the data is constrained to a single network or the data will always be sent with a specific protocol. Removing these limitations, however, is one of the main benefits of connecting such systems to the internet.

Connection interruptions are also an issue in other networks such as interplanetary communications. In the interplanetary use case, connection interruptions are mainly caused by alignment issues between nodes. Planetoids might be in the way or dishes not pointed in the right direction for immediate communication. These extreme distance communications also need to support the delays while a packet is in flight. There is currently an effort to create a solution to these issues known as Delay Tolerant Networking[4,5].

Delay Tolerant Networking aims to improve the connectivity between devices by providing a consistent methodology for managing delays that can be used over multiple link layers and containing any sort of data. While in most store and forward applications the communications system needs to be a specific type, in Delay Tolerant Networking this is no longer a requirement.

As well as improving the reliability of communications DTN can also improve network efficiency. With DTN networks, attempts at retransmitting messages that were lost can be achieved far more efficiently than is possible with end to end retransmissions. There are several reasons for this, as the DTN node is closer to the problem it has a better knowledge of when it is appropriate to retry. This avoids retrying when the message would still be unable to make it through; in addition the retries do not need to travel the successful part of the journey again. These combine to make retransmission far more efficient than would previously be possible, also as the origin does not have to be involved in the retransmissions it can perform other tasks or enter a sleep state itself. While the retransmissions will be more efficient, this comes at the cost of additional overhead of transmitting and processing the delay tolerance of the packet. For more reliable networks there will be less of an advantage as retries will be required less frequently.

While Delay Tolerant Networking is intended to be compatible with existing protocols and networks it is not simply a drop-in solution. As well as supporting the delay tolerant packets, the software on the endpoints will need to be able to deal with packets that have been delayed. Software which has been designed to wait for an immediate response from a device before continuing might be challenging to retrofit DTN compatibility into. While any upper

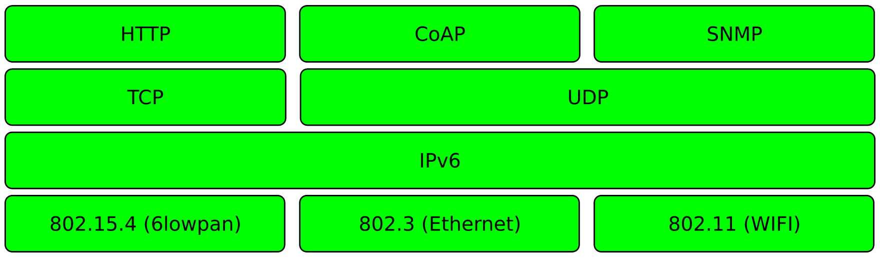


Fig. 1. IP as the common layer in the internet of things.

layer protocol can be sent with Delay Tolerant Networking enabled, the protocol might not be capable of handling the potential delay or require careful thought on its use. The Transmission Control Protocol (TCP) is one example of this, while there is no inherent problem with using TCP in a DTN environment the time-outs need to be at least as long as the DTN packet lifetime to avoid resending a packet that is still being transfered through the network.

Using Delay Tolerant Networking opens up several security and quality of service considerations mainly related to denial of service. As packets are buffered for a node, a large amount of packets can be built up, which can cause several issues. Intermediate routers can become saturated with messages that will be held waiting for other devices to reappear, this can prevent other users getting access to them. Alternatively this can overwhelm devices when they regain connectivity as they receive the entire buffer of packets waiting for them. These situations can occur either maliciously or as an unexpected side effect of the intended use.

2.1. The Bundle Protocol

Most of the current work on implementing Delay Tolerant Networking has been on the Bundle Protocol[6,7] which is an implementation of DTN for the deep space network use case. The Bundle Protocol works as an overlay network implementing its own system on top of existing networks and encapsulating the data within it. This makes it possible to transmit packets over multiple network types along its route, and allowing it to use the existing deep space networks which use a mixture of different network stacks as well as using the internet. This significantly increases the size of the packet headers, as information must be contained in the encapsulated protocol as well as the networks own headers.

The encapsulating nature of the Bundle Protocol puts additional demands on the nodes that need to access data within the encapsulated packet. These demands come as both additional code to decode the protocol and the resulting code space requirement on the device, there will also be the processing required to decode an additional protocol. These demands would be required on any node which would be able to delay the packet, but would not be required for nodes that just route the underlying network.

The features of the Bundle Protocol make it a highly versatile protocol at the cost of overhead. While this reduces its suitability for use in constrained networks, the principles of Delay Tolerant Networking can still be applied.

3. Integrating Delay Tolerant Networking into IP based networks

In order to use DTN on a network, it needs to be added to a layer common across all network types that the data will flow though. With store and forward this would typically require the use of a specific application protocol or low level network, which as previously discussed prevents flexibility. Previous attempts at implementing Delay Tolerant Networking technologies on IP, such as the Bundle Protocol, operate by adding another networking layer above the current IP framework and using it as if it was a link layer[8]. This encapsulating layer includes the packet data and routing information but cannot be processed by devices that do not support the protocol. In addition this requires duplication of data already contained in the packet as it must be included in the encapsulated version as well as the packet headers. To allow for seamless integration with existing systems both end devices and routers without DTN support need to be able to handle DTN packets in a predictable manner. DTN nodes will also need to be capable of handling non DTN enabled traffic. For these reasons DTN can not be implemented as an encapsulating layer or as a new lower layer protocol.

The alternative is to add the DTN functionality to a common layer across the network, for internet communications IP is the only layer which meets these requirements, this can be seen in Figure 1. IPv6 was created with the ability to accept optional extension headers which can add additional features or information to packets[9], examples of this include the routing header which specifies the route through the network or the authentication header which provides cryptographic authentication of the packet. These extension headers can be used to add the required information for using DTN to packets traveling across the network. Adding a new type of header can cause unpredictable responses from devices that are unable to process it, there are, however, two existing expansible option headers, the hop by hop options and destination options headers which can be used. These headers explicitly state how packets should be handled by devices which are unable to recognise one or more of the options. This guarantees a predictable response from devices which do not have support for delay tolerance, however, some parts of the internet infrastructure just drop packets with these headers[10] which could cause problems for protocols like this. The hop by hop options header is evaluated by each router along the route whereas the destination option is only evaluated by the final destination so the hop by hop options header is the one to use.

Adding a hop by hop option to a packet incurs a two byte overhead, one for the option type and one for the option length. There is an additional 2 byte overhead to add the hop by hop header itself if it is not already present. The actual data of the packet will contain a set of control flags and an expiry timestamp. This will provide the features required for the processing and handling of delayed packets. The total size of the DTN hop by hop options including type and length details is 6 bytes. While other DTN systems include additional features within the DTN data format these would be best served by a separate extension header. Many of the additional features of the Bundle Protocol already have equivalent IP headers available to perform those functions such as the fragmentation header. Figure 2 shows how the new DTN header fits in with the existing IP headers.

3.1. *Packet expiry timestamps*

The most important piece of data required for DTN operation is the expiry time at which a packet should be discarded. Without this, a packet could remain held in the network consuming storage resources indefinitely, or delivered too late to be of any use. In order to maintain a small packet overhead an efficient method of storing the time is required.

Storing a time as a fixed Unix style timestamp will require 64 bits, for small packets on constrained networks this is a noticeable amount of overhead. While 32 bit timestamp values could be used, this would make the protocol obsolete in the near future so is not a sustainable option. Using a fixed timestamp would also require every DTN node to maintain a reasonably accurate clock in order to determine packet expiry, which will not be possible for low power devices without an on-board real time clock.

Instead of using a fixed timestamp, a relative timestamp could be used. By doing this the value would be the amount of time remaining before the packet expired. This would require each node to update the remaining lifetime on route but avoids the need to know the exact time, just the difference in time between reception and transmission. Using a relative timestamp, the use of a 32 bit timestamp variable becomes feasible and reduces the packet size compared to a fixed timestamp by half.

The size of a relative timestamp can be decreased further by reducing the accuracy of the timestamps for long duration delays. As the delay lifetime extends into the order of hours or days then defining the lifetime down to the exact second is no longer required. By using this property an exponent based timestamp was developed, this allows for high precision for small lifetimes while still providing far longer timeouts. A 4 bit exponent combined with a 12 bit multiplier was chosen as this gives a high level of accuracy whilst still allowing packet lifetimes up to eight and a half years which should be plenty of time for any application. This solution reduces the amount of timing information that needs to be transmitted to 2 bytes, a 4 fold improvement over the fixed timestamps.

With relative timestamps, the time taken while the packet is in flight needs to be considered. In most cases this will be less than a second and can be ignored, however in some cases the time between transmission and reception will be significant. While this is not the case with almost all current IP carrying systems, the ability to accept delayed packets opens up such options to protocol designers. In cases where there could be an unknown time between transmission and reception it will be necessary to use a fixed timestamp as part of that link layer's headers in order to determine the transmission time.

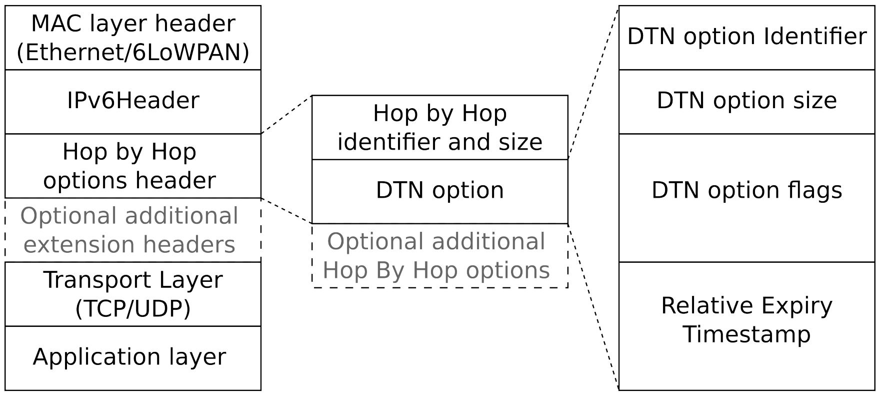

MAC layer header (Ethernet/6LoWPAN)
IPv6Header
Hop by Hop options header
Optional additional extension headers
Transport Layer (TCP/UDP)
Application layer
Hop by Hop identifier and size
DTN option
Optional additional Hop By Hop options
DTN option Identifier
DTN option size
DTN option flags
Relative Expiry Timestamp

Fig. 2. How the DTN option header fits into the network layer structure.

3.2. *Custody transfer*

To pass packets from one DTN enabled device to another, a process known as custody transfer is required. To do this the sender of a DTN packet needs to know that the next DTN router has received the packet and has taken responsibility for its onwards transmission. This is achieved by the recipient sending a custody response to the current custody holder. There may be several non DTN hops between two DTN routers so the custody acceptance response cannot be sent to the last link layer hop as that might not be the previous custody holder. While it would be possible to add the IP address of the last DTN node to the packet to give a place to return the custody report to, this adds additional overhead. This can be avoided by sending packets back towards the message's origin during which they will travel back through the previous DTN hop. To allow DTN nodes to identify these packets they need to be tagged, which will allow the router to know that they need to evaluate their contents rather than just letting them pass though. This tag can be implemented as another bit of the flag byte in order to avoid additional overhead.

As well as ensuring that the DTN routers can detect the custody packets it is necessary to send the details of the custody transfer in some format. Internet Control Message Protocol (ICMP6) makes a good candidate to transfer status information around as it is already widely used for that purpose on the internet.

Depending on the situation there will need to be different types of responses. In the case where the custody has been accepted and can be transfered an acceptance message will be sent. The acceptance message completes the handover process at which point the transmitting node can delete its copy of the packet, it is then the job of the receiving node to ensure forward transmission. As this message is intended for the DTN router holding the custody this message should not be passed beyond that router.

In some cases the next DTN router will not be able to accept custody of the packet for the entire remaining lifetime in which case a temporary custody acceptance can be used. It indicates that a node has taken custody over the packet for a given period of time but the original custody holder should maintain custody. The original custody holder can then retry after the temporary custody has expired, should a full custody transfer not be received in the meantime.

The case where the next DTN router cannot accept custody is an important one. This situation could occur for many reasons, such as running out of storage space or only accepting custody in certain conditions. The way to deal with this is to send the packet onwards if it is able, just like a normal router would, otherwise the packet should be silently dropped. The lack of a custody response for the dropped packet will cause the current custody holder to perform a retransmission at a later time when it might be possible to accept custody again.

4. Implementations

In order to investigate the potential of adding DTN technologies into the IP layer, two versions of the handler were created, one to run on the Linux based border routers and one to run on the constrained sensor network hardware itself. These implementations can be used together to provide a working delay tolerant system which is capable of handling delays both on and off the low power network.

4.1. Implementation for Linux systems

In order to provide delay tolerance on the 6LoWPAN border router a Linux implementation of the delay tolerant header processing is required. While handling such a protocol is much better suited to a kernel module this would be time consuming for initial development. To avoid this a userspace program was created, Python was chosen for this in order to leverage existing packet analysis libraries and tools for faster development.

As the Linux kernel provides an abstraction from the low level network when using network sockets a different solution was required to get access to the raw option headers. This is done by capturing complete packets and performing any required actions in userspace, the packet capture library (py-pcap) was used to perform this. These raw packets then need to be decoded in order to extract the required information from the packet. The dpkt library was used to decode the packets, allowing the packed binary data to be separated into individual parameters.

With the packets decoded, they are then filtered to remove those which did not contain a delay tolerant header. At this point the DTN header was evaluated and the packet passed to the packet store. As the usual Linux routing has been interrupted to allow the analysis of the DTN packets, the packet's next hop needs to be calculated manually so it can be sent over the correct interface. Initially the Linux code was implemented without custody transfer so Ethernet ARP tables were used to provide a guess as to whether it was possible to deliver the packet or not. Later on custody transfers were added to provide a guarantee that the packet had been received. The use of ARP tables provides a good example of where additional local network knowledge of a DTN node can avoid unnecessary transmissions.

In order to send the packets onwards with a DTN option header the standard sockets library could not be used. Raw sockets were used to send the packets, this allows the userland code to control the DTN header but also requires the software to calculate the packet routing and link layer addressing as well.

The Python Linux implementation provides a functional demonstration of DTN working in a significantly less constrained environment. In order to realise the full benefits of Delay Tolerant Networking the protocol would also need to be supported on the low power network as well.

4.2. Implementation for constrained hardware

With a Linux implementation created, the next step was to implement the protocol on a constrained 6LoWPAN system. To develop this the Contiki operating system running on Zolertia Z1 nodes was used. The code was also tested on Tmote sky nodes, however, due to the code space limitations some of the Contiki example programs would no longer fit when the DTN processing was also present. The DTN code requirements came to approximately 1Kib of code space memory, however, with some optimisation this could be reduced, The increased amount of ram and flash on newer chips are likely to make this a non issue. A test DTN network was created using the Cooja emulation tool for Contiki to allow for faster development and easier access to the contents of in-flight packets. The Cooja tool uses mspsim to emulate the node hardware, which will provide realistic results without needing to manually retrieve and program hardware for each test.

The Contiki implementation was separated into two sections. Part of the implementation would be integrated into the network stack to provide fast processing of incoming packets and acknowledgements. The other part would be implemented as a separate process that would manage the stored packet records and perform any retransmissions required. An overview of the DTN implementation can be found in Figure 3.

Contiki's IP networking uses the uIP network stack to handle the processing of the IP layer frames. The modularity of this code meant that the DTN code could be directly integrated into Contiki's network stack. This still allows for nodes that did not support DTN to be built with the same codebase without incurring any processing or code space overhead. The RPL routing system used by Contiki uses the hop by hop option for some of its information exchanges, this meant that Contiki already had support for detecting hop by hop options and processing them. This code was expanded to include the DTN header, allowing the DTN options to be inspected as part of the packets reception.

The additional code in the network stack decodes and validates the DTN option included in the packet. If the packet is a new DTN enabled packet the packet is stored, if possible, in a new record in the DTN custody store and timers are set for the packet expiry. The packet then continues through the Contiki uIP networking stack as usual and is forwarded onto the next node or passed to the receive code as required. This allows packets to be passed through the network without additional delay if the network is up. As this is done regardless of whether the packet could be stored or not, it also allows packets to bypass nodes that have filled up the custody packet buffer.

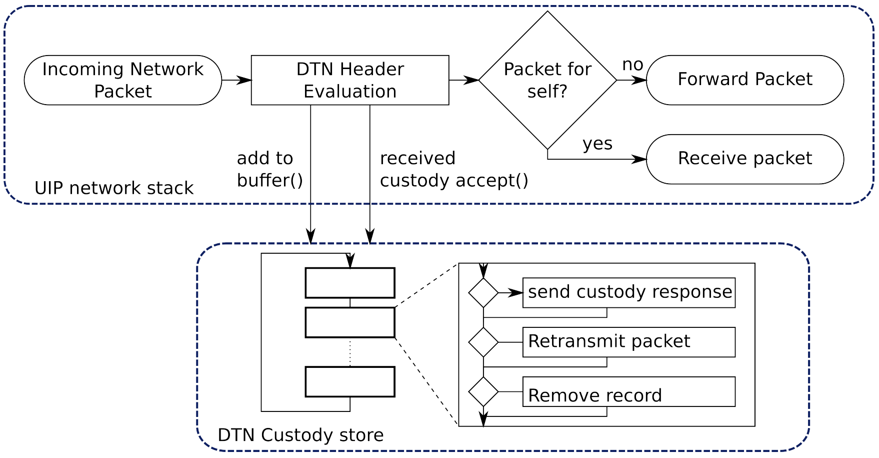


Fig. 3. DTN processing within the Contiki operating system.

If the packet is a custody acknowledgement it is matched to the appropriate record in the custody store, that record is then marked for removal in the next store cleanup, the packet is then dropped to prevent it being forwarded any further. If there is no matching record the packet is sent onwards as a previous node might be holding custody of the packet.

The custody store management process iterates though each of the records in the custody store and performs any actions required. The data storage in the custody store is implemented in two parts, the index and the packet store itself. The index stores the state of the packet and basic information about the packet, adding this information to the index avoids the need to access the packet except when performing retransmissions. The packet store, stores the actual packet, although this was implemented in ram it does not have to be, use of the coffee file system within Contiki would allow for packets to be stored in flash allowing for the space saved to be allocated to a larger index.

The main actions of the custody store are to send custody acknowledgements, perform retransmissions, and remove packets that have been delivered or expired. Due to limitations in the network stack of Contiki it is not possible to generate a custody response while processing the message in the network so this is done as one of the actions of the store. An ICMP6 message is built and sent out in the normal way, the addition of the DTN header and flagging as an informational DTN packet is implemented as part of the network send routine.

Where a custody message has not been received the custody store process will retransmit the message. In the trial network a retry was attempted after a fixed time if no response had been received. In order to perform a retransmission the packet was copied from the packet store into the network buffer. As the packet has been in storage the expiry timestamp was updated using the information in the index to give the new time until expiry. The network buffer with the updated packet was then transmitted.

In order to prevent the custody store filling up with unused packets, completed records need to be removed. Records that have been flagged for deletion by a custody response having been received are deleted as part of the stores operation. In addition the expiry timeout is checked to see if the packet has passed its maximum lifetime.

4.3. Preliminary Testing and Evaluation

Initial testing was carried out using the Cooja[11] tool for Contiki. A multi hop network was created comprising of a mix of Z1 and sky nodes with and without support for delay tolerance. This network was connected using a SLIP interface to a Linux border router which was running the Python DTN code. Several of the nodes were programmed with a DTN enabled UDP responder which would be used as the target for the test communications. The network traffic was then extracted for analysis by feeding the cooja radio logs into the wireshark packet capture and analysis tool.

Using this testbed framework, several experiments were carried out in order to see how the DTN nodes would behave. Nodes along the packets route were deliberately disconnected from the network in order to represent a loss of connectivity. Using the results from wireshark, it was possible to see the retransmissions of packets and the custody acknowledgements after a successful transmission. With the DTN features added to the network, packets were successfully retransmitted and delivered when the disconnected devices were reconnected to the network.

After testing on a simulated system, the same code was loaded onto physical sky and Z1 nodes which were deployed in our research lab. Instead of trying to access the radio messages in the network, wireshark was used on the link between the border router and the network. While this does not give hop by hop information on the packet's progress it still provides enough information to confirm the successful delivery of the packets even with artificially injected connectivity breaks, such as removing antennas. From this deployment the DTN implementation was shown to work in a real world deployed network.

5. Conclusion

Meeting the current expectation of continuous connectivity for all internet connected devices is infeasible with the new generation of ultra low power Internet of Things devices. Previous attempts at working around those issues leave major limitations that counteract many of the benefits gained from using the Internet of Things over other connection technologies. Delay Tolerant Networking makes it possible to maintain reliable communications without the burden of providing constant low level connectivity or limitations on what can be sent.

Previous delay tolerant technologies have required significant overhead making them unsuitable for constrained devices. It is, however, possible to provide delay tolerance with a far lower overhead and without breaking compatibility with existing systems. This has been solved by adding delay tolerance to packets using the IPv6 hop by hop extension header. A protocol has been created around this and implemented on Linux and Contiki operating systems. Through these implementations this technology has been demonstrated to be viable for use on constrained hardware systems.

Bringing Delay Tolerant Networking to IP, will allow a new wave of ultra low power internet connected devices to be created. This opens many opportunities that were previously unavailable to such devices.

This protocol demonstrates that IP layer DTN is viable even on constrained Internet of Things hardware. To make the most from the protocol some additional refinement will be required to resolve any edge cases that may exist, but care must be taken to avoid bloating the protocol with additional features as these will increase the overhead of the protocol. With this complete the huge potential from using DTN in the internet of things can be unlocked.

References

1. Martinez, K., Padhy, P., Elsaify, A., Zou, G., Riddoch, A., Hart, J., et al. Deploying a sensor network in an extreme environment. In: *Sensor Networks, Ubiquitous and Trustworthy Computing*. IEEE Computer Society; 2006, p. 186–193. URL: `http://eprints.soton.ac.uk/262701/`; event Dates: 5-7 June 2006.
2. Ward, T., Martinez, K., Chown, T.. Simulated Analysis of Connectivity Issues for Sleeping Sensor Nodes in the Internet of Things. In: *The Eleventh ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Network*. 2014, .
3. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.. A survey on sensor networks. *Communications magazine, IEEE* 2002; **40**(8):102–114.
4. Cerf, V., Burleigh, S., Hooke, A., L.Torgerson, , R.Durst, , K.Scott, , et al. Interplanetary Internet (IPN): Architectural Definition. Active Internet-Draft; 2001. URL: `http://www.ietf.org/id/draft-irtf-ipnrg-arch-00.txt`.
5. Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., et al. Delay-Tolerant Networking Architecture. RFC 4838 (Informational); 2007. URL: `http://www.ietf.org/rfc/rfc4838.txt`.
6. Scott, K., Burleigh, S.. Bundle Protocol Specification. RFC 5050 (Experimental); 2007. URL: `http://www.ietf.org/rfc/rfc5050.txt`.
7. Fall, K.. A delay-tolerant network architecture for challenged internets. In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*; SIGCOMM '03. New York, NY, USA: ACM. ISBN 1-58113-735-4; 2003, p. 27–34. URL: `http://doi.acm.org/10.1145/863955.863960`. doi:10.1145/863955.863960.
8. Bruno, L., Franceschinis, M., Pastrone, C., Tomasi, R., Spirito, M.. 6lowdtn: Ipv6-enabled delay-tolerant wsns for contiki. In: *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. 2011, p. 1–6. doi:10.1109/DCOSS.2011.5982154.
9. Deering, S., Hinden, R.. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard); 1998. URL: `http://www.ietf.org/rfc/rfc2460.txt`; updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.
10. Gont, F., Linkova, J., Chown, T., Liu, W.. Observations on IPv6 EH Filtering in the Real World. Active Internet-Draft; 2015. URL: `http://www.ietf.org/id/draft-gont-v6ops-ipv6-ehs-in-real-world-02.txt`.
11. Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.. Cross-level sensor network simulation with cooja. In: *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*. 2006, p. 641–648.

electronics

Article

Erica the Rhino: A Case Study in Using Raspberry Pi Single Board Computers for Interactive Art

Philip J. Basford *, Graeme M. Bragg, Jonathon S. Hare, Michael O. Jewell, Kirk Martinez, David R. Newman, Reena Pau, Ashley Smith and Tyler Ward

Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, UK; g.bragg@ecs.soton.ac.uk (G.M.B.); jsh2@ecs.soton.ac.uk (J.S.H.); moj@ecs.soton.ac.uk (M.O.J.); km@ecs.soton.ac.uk (K.M.); drn@ecs.soton.ac.uk (D.R.N.); R.Pau@soton.ac.uk (R.P.); ads04r@ecs.soton.ac.uk (A.S.); tw16g08@ecs.soton.ac.uk (T.W.)

* Correspondence: pjb@ecs.soton.ac.uk; Tel.: +44-23-8059-6657; Fax: +44-23-8059-2783

Academic Editors: Steven J. Johnston and Simon J. Cox
Received: 4 May 2016; Accepted: 24 June 2016; Published: 30 June 2016

Abstract: Erica the Rhino is an interactive art exhibit created by the University of Southampton, UK. Erica was created as part of a city wide art trail in 2013 called *"Go! Rhinos"*, curated by Marwell Wildlife, to raise awareness of Rhino conservation. Erica arrived as a white fibreglass shell which was then painted and equipped with five Raspberry Pi Single Board Computers (SBC). These computers allowed the audience to interact with Erica through a range of sensors and actuators. In particular, the audience could feed and stroke her to prompt reactions, as well as send her Tweets to change her behaviour. Pi SBCs were chosen because of their ready availability and their educational pedigree. During the deployment, 'coding clubs' were run in the shopping centre where Erica was located, and these allowed children to experiment with and program the same components used in Erica. The experience gained through numerous deployments around the country has enabled Erica to be upgraded to increase reliability and ease of maintenance, whilst the release of the Pi 2 has allowed her responsiveness to be improved.

Keywords: Internet of Things; interactive art; Raspberry Pi; open data; image processing

1. Introduction

Interactive art involves its spectators in more than just a viewing capacity. This interactivity can range from spectators perceiving that they are interacting with a passive art piece to pieces where input from the spectator influences the artwork [1]. Over the years, interactive art has evolved from simple mechanical contraptions [2] to installations involving some form of computer processing [3,4] or that are completely virtual in their output [5,6].

Since its introduction, the Raspberry Pi Single Board Computer (SBC) has provided an all-in-one platform that allows artists to carry out processing and hardware interaction on a single low-cost piece of hardware. This has led to it being used in many interactive art installations and the Raspberry Pi foundation have dedicated a section of their website [7] to documenting artistic works that incorporate Raspberry Pi SBCs.

The *Go! Rhinos* campaign was a mass public art event run by Marwell Wildlife in Southampton, UK for 10 weeks during the summer of 2013 [8]. The event involved 36 businesses and 58 schools placing decorated fibreglass rhinos along an 'art trail' in Southampton City centre, with the aim of raising awareness of the conservation threat faced by wild rhinos, and showcased local creativity and artistic talent.

The event provided an opportunity to promote Electronics and Computer Science at the University of Southampton and act as a platform for electronics and computing outreach activities.

A team of electronic engineers, computer scientists, marketing specialists and artists from within the University were brought together to design and develop a unique interactive cyber-rhino called Erica, shown in Figure 1. Erica was designed to be a Dynamic-Interactive (varying) [9] art piece where her behaviour is not only determined by the environment that she is in but also by her physical interactions with viewers—very much like a cyber-physical toy or Tamagotchi [10]. Internally, Erica is powered by a network of five Raspberry Pi SBCs connected to a series of capacitive touch sensors, cameras, servos, stepper motors, speakers, independently addressable LEDs and Liquid Crystal Displays (LCDs). These devices were carefully chosen to implement the desired features.

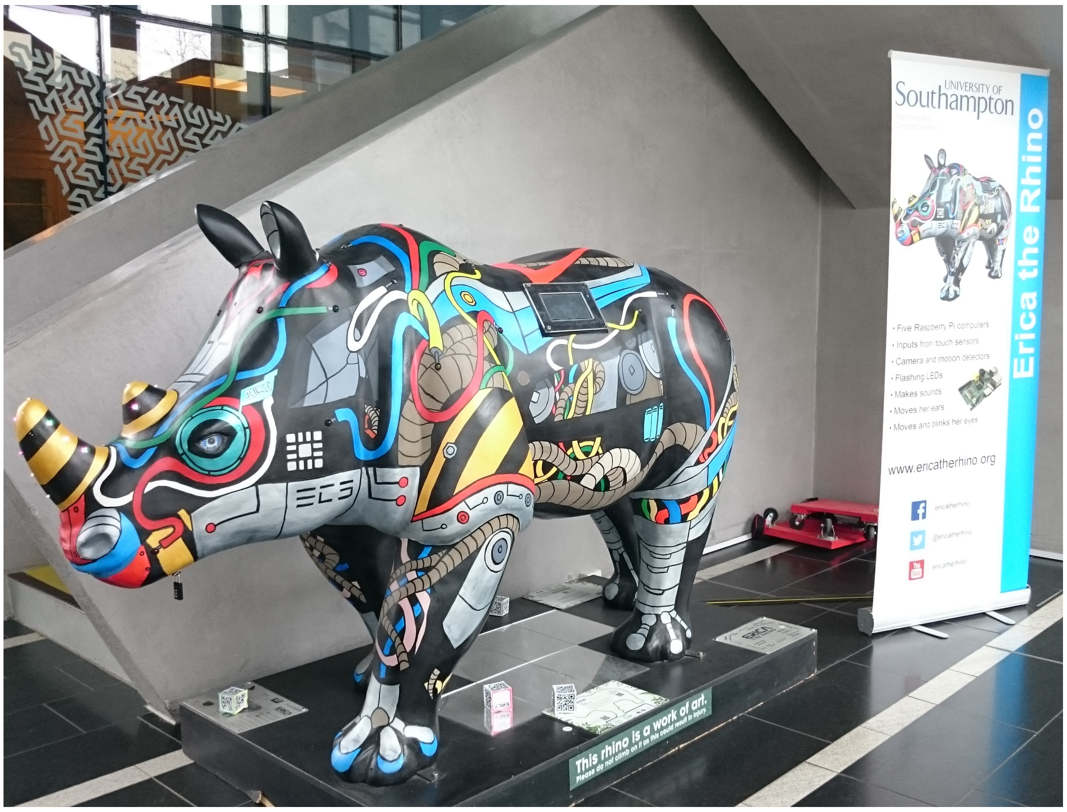

UNIVERSITY OF Southampton
Erica the Rhino
• Five Raspberry Pi computers
• Inputs from touch sensors
• Camera and motion detectors
• Flashing LEDs
• Makes sounds
• Moves her ears
• Moves and blinks her eyes
www.ericatherhino.org
This rhino is a work of art.

Figure 1. Erica the Rhino in her permanent home at the University of Southampton.

This article discusses in depth the impact and considerations of installing a piece of interactive art using Raspberry Pi SBCs in a public setting as well as the implementation methods. The paper is organised as follows. Section 2 discusses the features of Erica that brought her to life. Section 3 describes the initial implementation of Erica and the lessons learned, while Section 4 goes on to discuss the deployment of Erica into the wild. Section 5 describes the upgrades and maintenance after Erica's time with the general public. Section 6 demonstrates the impact of Erica with regards to public engagement and outreach while Section 7 provides a concluding statement.

2. Features

The initial concept of Erica was as a cyber-physical entity that merged actions inspired by natural behaviours with a showcase of the different facets of electronics and computer science in an interactive way. The Raspberry Pi was the platform of choice for its novelty, popularity with hobbyists and schools and its wide availability. The media awareness of the Raspberry Pi also helped to promote Erica. Additionally, the availability of the Raspberry Pi and open-source nature of Erica's design would permit interested people to inexpensively implement aspects of her at home. After several brainstorming sessions, an extensive list of desirable features that could be implemented was compiled. Each of these features was classified as either an input (a 'sense'), an output (a 'behaviour') or both as shown in Table 1. A broad range of features was selected to cover different areas of electronics and computer science, ranging from sensors and actuators to image processing and open data analytics, leading to an initial design drawing as shown in Figure 2.

Table 1. Features that were considered for inclusion in Erica. Those in italics were considered but not implemented.

Input	Output	Both Input & Output
Touch Sensor (capacitive)	RGB illuminated horn	Eyes (moving webcams)
Presence Sensor (PIR)	Animated body LEDs	Twitter
Temperature sensor	Moving ears	*SMS text messaging*
Open data	Side information displays	*Bluetooth presence detection and messaging*
QR codes	Sound	
Sound Level	*Simulated snorts (compressed air)*	
Speech recognition	*Ticker tape printing of tweets*	
	3D Printing	
	Projected Output	

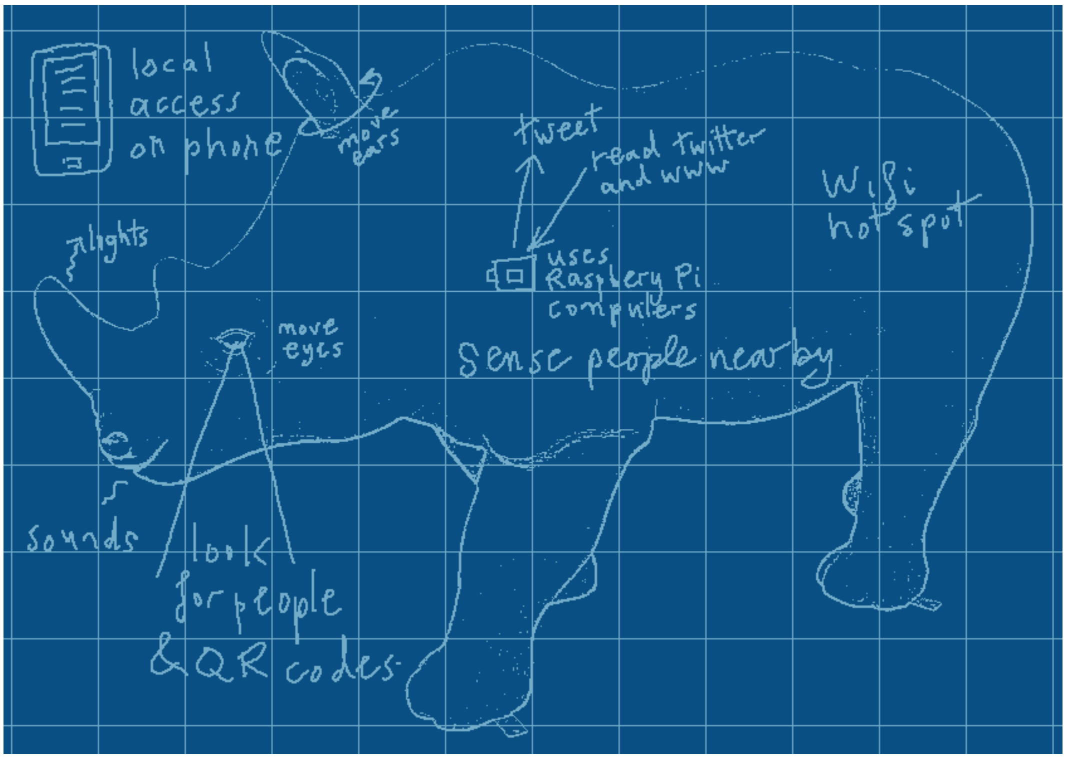


Figure 2. Initial ideas for Erica's features

When an input occurs, it is processed and an appropriate response is generated. These responses can be broken down into two categories: 'reactive' and 'emotive'. Reactive behaviours occur almost immediately and are a direct response to an interaction, such as a grunt being generated when a touch sensor is touched. This immediate feedback provides a strong link between the interaction and the response, which is beneficial when demonstrating how the sensors and actuators are connected. Erica's reactive behaviours can be thought of as being similar to reflexes in humans; however, they cover a broader range of interactions.

Rather than each interaction having a static response, it was decided that Erica should also have several emotive responses. This was achieved by implementing four emotive states, each with seven distinct levels, that triggered additional output events and influenced the outcome of future interactions. Emotive responses are based on a cumulative time-decaying set of 'emotions' as shown in Table 2 alongside the input sensors that contribute to their level and output events. When Erica is left alone for an extended period of time, she goes to sleep and recovers energy, but her interest, fullness and mood decay.

Table 2. The four emotive states used within Erica, together with the two 'extreme' cases, the inputs that affect them, and the outputs that are caused.

State	Level 1	Level 7	Affected By	Causes
Energy	Asleep	Overexcited	Interaction (or lack of)	Idle behaviour Web statistics
Mood	Sad	Happy	Cheek sensor	Idle behaviour Web statistics
Interest	Bored	Very interested	Cheek sensor Presence sensor	Web statistics
Fullness	Starving	Overfed	Mouth sensor	Energy Web statistics

The 'emotion' that turned out to be a favourite with adults and children alike is fullness. Fullness automatically decreases over time and is incremented every time she is fed (by touching the chin sensor), accompanied by a grunt noise. If Erica is fed too many times in quick succession, a more juvenile sound is also played.

2.1. Visual System

It was desired that Erica should be able to see like a real rhino so a visual system consisting of two cameras (one for each eye) was conceived. At the time of development, the Pi Camera [11] was not available so two USB webcams were chosen. Even if the Pi Camera had been available, they would have been less suitable than webcams due to mounting and cable length/flexibility issues. Initially, it was planned that the eyes would have two-axis pan-tilt; however, this proved impractical in the limited space available within the head. As such, a single servo was used to enable left-right panning about the vertical axis.

Software was built using the OpenIMAJ libraries [12] developed at Southampton—the use of cross-platform Java code and the inbuilt native libraries for video capture, combined with the use of commodity webcams. This portability ensured that it was possible to test the software on various platforms without need for recompilation or code changes, which substantially helped with rapid prototyping of features. Additionally, this had the added benefit of improved accessibility of the public to experiment with image processing using Erica's open source examples.

The original idea for the visual system was that it would perform real-time face tracking and orientate the cameras such that the dominant detected face in each image would be in the centre of the captured frame. The restriction to panning on a single axis and performance limitations of the Raspberry Pi meant that the tracking was not as smooth and apparent as desired. Therefore, it was decided that the visual system should be used for interactions that did not require immediate feedback to the user. In particular, the software for the eyes was setup to process each frame and perform both face detection (using the standard Haar-cascade approach [13] implemented in OpenIMAJ) and QR-code detection (using OpenIMAJ with the ZXing "Zebra Crossing" library [14]). This achieved recognition at a rate of a few frames a second (specifically, using the Raspberry

Pi model B, the frame-rate achieved was around five frames per second, while the Raspberry Pi 2 managed around ten frames per second).

2.2. Open Data

Open Data, specifically Linked Open Data [15], is a subject in which the University of Southampton has a rich research history. Linked Open Data is, in summary, information made available in a computer-readable form with a license that allows re-use. It was decided that Erica should both consume and publish Linked Open Data. Erica periodically checks a number of online data sources in order to get an idea of her environment. The most novel use for this is a function for checking the current weather conditions and reacting accordingly. Erica will get cold if the temperature drops, and will sneeze if the pollen count is too high.

Every hour, a script runs that takes a copy of Erica's current emotive state and converts it into an open format known as RDF (Resource Description Framework). This is then published to Erica's website and can be queried by any programmers who wish to interact with Erica. If an internet connection is not available, the script silently exits and tries again the next hour. The data in its RDF form is held on the website [16] rather than on Erica herself, so that it is always available even when Erica has no internet connection.

2.3. Features Summary

Having worked out a list of features to be included in Erica, how they were implemented needed to be carefully considered. The design choice of using Raspberry Pi SBCs as preference over a small form factor PC caused some additional challenges that would not otherwise have been faced.

3. Initial Implementation

During initial development, it was quickly found that a single Raspberry Pi was not sufficient to handle all of the processing required for the desired features. As such, a distributed system of five Raspberry Pi SBCs was conceived with each one being responsible for a different aspect of Erica's operation. Figure 3 depicts a block diagram of the initial implementation and shows how all of the inputs and outputs are connected to the SBCs.

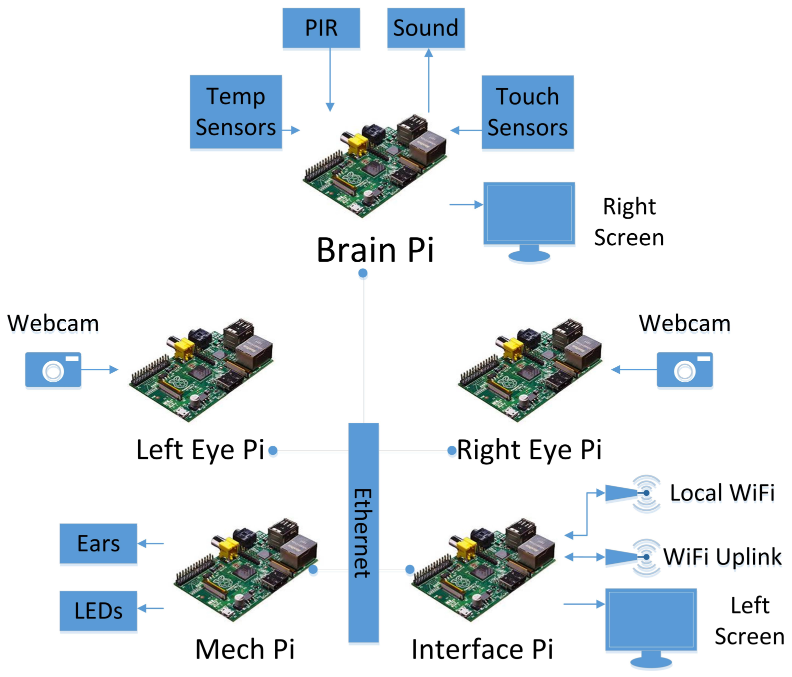


Figure 3. Erica's Pi architecture as initially implemented.

The overhead of the visual system required one Raspberry Pi per eye to give acceptable performance. LED and servo control required a number of I/O pins so one Raspberry Pi was dedicated to this task. To co-ordinate the actions of these Pi SBCs into a coherent entity, another Pi, the Brain Pi, was dedicated to controlling the whole system and was responsible for the PIR sensor, touch sensors, temperature sensor and sound output. Details of the operation of the Brain Pi are discussed further in Section 3.3. Finally, a fifth Raspberry Pi was used to provide network connectivity to the outside world.

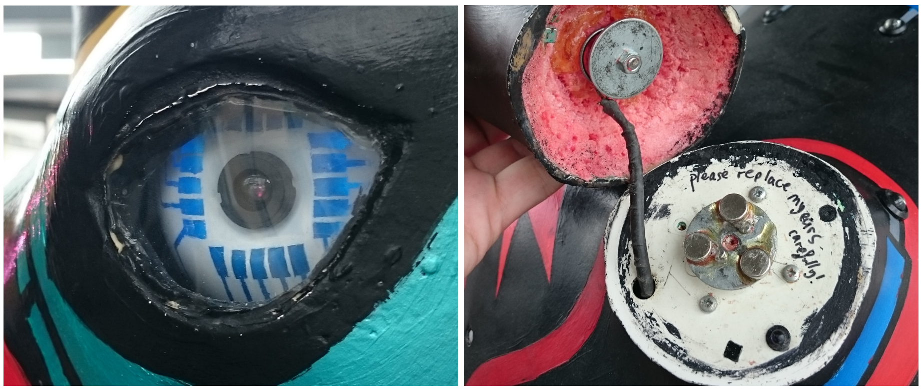
Erica included two HDMI-connected 7″ displays, one on each side. These were each connected to a separate Raspberry Pi and used to display information about Erica's mood, the *Go! Rhinos* campaign and rhino conservation. These displays were deliberately positioned at different heights to allow for easy viewing for both adults and children alike.

3.1. *Physical Design*

Erica was delivered as a sealed white fibreglass shell with no access to the interior. The artistic design of Erica was outside the areas of expertise of the authors, so talent was sought elsewhere in the University. A design competition was run at the Winchester School of Art where undergraduate students submitted potential designs. The winning artist was invited to paint Erica in their design, which was then displayed in the Southampton city centre for 10 weeks.

Rather than hiding the electronics inside Erica, it was felt that being able to see what was driving her would add to her appeal and general intrigue, so it was decided to make them a visible feature. This was achieved by making the access hatch that was cut in Erica's belly out of clear perspex (formed to the same shape as the fibreglass that was cut out) and placing mirror tiles on the plinth beneath to allow viewers to see inside easily. The Raspberry Pi SBCs were mounted upside down on a board suspended above the perspex window and illuminated by two LED strip lights.

The webcams chosen to act as Erica's eyes had a ring of LEDs around the lens designed to be used to provide front-light to the webcam image. A digitally controllable variable resistor allowed software brightness control, so the LEDs could be used to simulate blinking. The webcams were then inserted inside a plastic hemisphere that was painted to resemble an eye with an iris. For installation, the eyes for the fibreglass moulding were carefully cut out so that the webcams would be in an anatomically correct position. Once mounted, it was noted that the eye mechanism was vulnerable to physical damage, especially as the eyes were at a child-friendly height, so colourless, domed perspex protective lenses were formed to fit within the eye socket and sealed to prevent external interference, as can be seen in Figure 4a.



(**a**) (**b**)

Figure 4. (**a**) detail of Erica's eye assembly; (**b**) detail of Erica's detachable ears.

Erica's moving ears were implemented by cutting off the fibreglass ears and remounting them to stepper motors so that they could be rotated freely. As the only external moving components, specific care was needed to prevent injury to people and to ensure that the mechanism could withstand being investigated by curious bystanders. This was achieved by mounting the ears magnetically to the stepper motor shafts, limiting the available torque. This, however, made it relatively easy to remove the ears so they were tethered to prevent them from being dropped and to discourage theft, as shown in Figure 4b.

Two distinct groups of LEDs were also inserted into her shell: RGB LEDs on her horn and mono-colour LEDs of differing colours on her body. The horn LEDs were installed in differing patterns on her short and long horns. The body LEDs were incorporated into her artistic design, being placed at the ends of her painted wires.

3.2. *Networking and Monitoring*

By choosing to use multiple SBCs to provide the compute power needed to run Erica, a means to interconnect these was essential. As Erica would need to be moved between locations, it was decided to run an internal network to provide this connectivity, which could then connect out to the Internet at a single point. There were two options considered for this, either an off-the-shelf router or to use a Pi. The USB ports available on a Pi gave the flexibility required to add both additional wired Ethernet, as well as wireless interfaces. This arrangement would give more flexibility in configuring these interfaces (for DNS, DHCP, NAT, routing, firewalling, etc.), whereas an off-the-shelf router with its generic firmware may have not been sufficiently configurable.

The initial design of the network ended up with the Interface Pi having three separate interfaces, which was facilitated by connecting a powered hub to one of its USB ports to provide the required capacity both of ports as well as power. These three interfaces consisted of:

- A wired internal interface to connect to the other four SBCs over an internal network.
- A WiFi uplink interface to connect to the Internet provided by a USB wireless dongle and high-gain antenna.
- A WiFi access point interface to allow those in the vicinity to interact with Erica using smartphones, provided by a USB wireless dongle with a standard antenna.

It was decided that no internet access would be available on the WiFi access point, as this would be a publicly available unprotected network and therefore any Internet access was liable to be abused. It was recognised early in the development process that remote access to monitor if various electronically controlled aspects of Erica were behaving as expected was essential. This also allowed certain features to be fixed when they were not working. This needed to be achieved in a way that was independent of the parent network providing the uplink to the Internet.

This remote access was facilitated by two separate means, which had both previously been investigated in earlier sensor network deployments [17]. This first of these techniques was to create an SSH tunnel out from the Interface Pi through the parent network to a device on the Internet that could accept SSH connections. This tunnel would allow SSH connections from this device directly onto the Interface Pi without needing to know either the current (private) IP address of its uplink or the IP address of the parent network's gateway.

The second technique was to register for an IPv6 [18] tunnel with a tunnel broker. SixXS [19] provides a variety of IPv6 tunnel options for which AICCU (Automatic IPv6 Connectivity Client Utility) meets the key requirement for Erica, to facilitate as simply as possible, routeable global IPv6 addresses for each Pi, allowing them to be connected to directly (rather than requiring a proxy via the device that maintained the IPv4 SSH tunnel previously described). These IPv6 addresses could then be assigned hostnames using DNS AAAA records for the ericatherhino.org domain, significantly simplifying the task of accessing and monitoring the Pi SBCs remotely. However, for security reasons this was carefully firewalled, and SSH was only allowed using public key authentication.

Monitoring of Erica was implemented using Icinga [20], a scalable open source monitoring system. At its most basic level, Icinga allows monitoring of hosts and common services (e.g., Ping, HTTP, SSH, DNS, etc.). It also allows dependencies between hosts and services to be defined, so some hosts or services are only monitored when other hosts or services are available. Through this means, Erica's infrastructure could be represented in the status map shown in Figure 5.

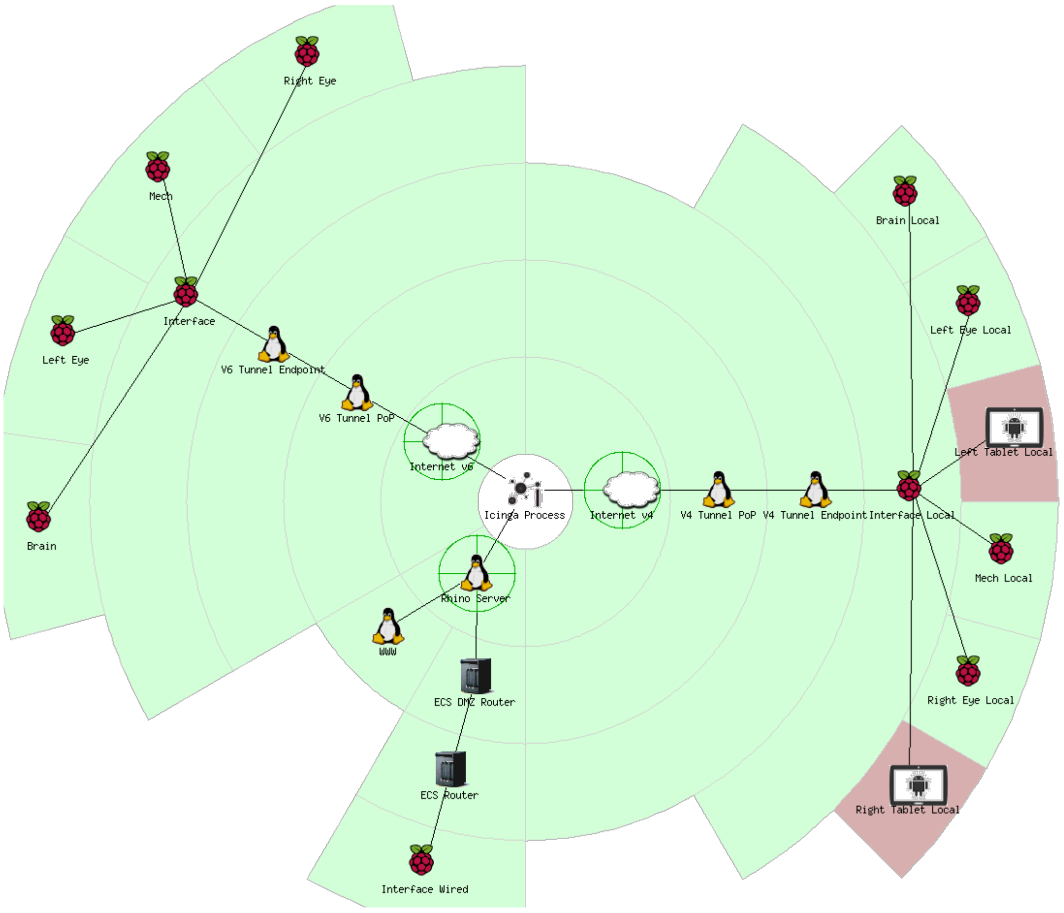


Figure 5. Example output from Erica's status monitoring, showing her current hardware as detailed in Section 5.

One particular feature of Icinga is that an accompanying application can be run on monitored hosts to allow scripts to be run to test bespoke features, when prompted by Icinga. This was made use of to ensure that particular applications were running on specific Raspberry Pi SBCs, prompting 'rhino engineers' to restart the required programs when necessary. It also made it possible to observe when particular interactions were not occurring in near real-time. This provided the ability to remotely determine if a feature was broken in a timely manner, allowing remedial action to be taken. It also provided information on simple trends that helped resolve regularly occurring faults or to evaluate the popularity of different interactions.

3.3. *Brain Development*

Due to the distributed nature of Erica's hardware, it was important that there should be a middleware capable of both receiving events from the various sensors and triggering commands to cause a reaction. This was deployed on the Brain Pi, with the sensors that require fast responses (touch & presence) also connected to it.

The software itself was implemented using the Django [21] web framework, and provided a RESTful API (REpresentation State Transfer Application Programming Interface) to the other Pi

SBCs. Each Pi that wants to send events runs a RhinoComponent web service, using the lightweight CherryPy [22] for simplicity. This registers with the Brain Pi on boot, indicating its name (e.g., 'left-eye') and a URL that is able to receive commands.

When a sensor is triggered, the Pi responsible for that sensor sends an event to the brain. This has the structure of 'source.component.action', e.g., 'interaction.chin.press'. The source indicates what caused the event (e.g., a sensor interaction, twitter, environment, or the brain itself); the component informs the brain as to the originating component (e.g., the chin, or the left eye); and the action gives the interaction that was actually performed (e.g., press, scan, detect). A dictionary of key/value pairs can also be sent, giving extra information (e.g., which side of the chin was pressed).

As soon as an event arrives, a collection of scripts are executed, known as 'behaviour scripts'. These are intentionally simple and small, giving the entire team the ability to add new behaviours without having to modify the underlying server code. They can read and modify Erica's emotional states, described earlier, and trigger commands. A short-term memory (capped at 100 items) and a long-term memory (holding counts of events) are also available. For example, if a face is detected by one of the eyes, Erica's mood and interest are increased, the appropriate eye is told to blink, a sound is played, and the website is updated. As a side-effect, the short-term memory will include the face detection event, and the long-term memory will show that one more face detection event has been handled.

The blink and sound playback actions in this example are performed by triggering commands. When the behaviour script triggers 'lefteye.lights.blink', the command is sent to the Left-eye Pi via the URL that it registered earlier. The component on the Pi can then affect the webcam's LED.

There are also some events that are not caused by external stimuli: an idle event is triggered at set intervals, so Erica's hunger and tiredness can be updated; and an event is triggered every hour, allowing Erica to send messages at appropriate times.

3.4. Electronic Interface Hardware

Each of the Pi SBCs in Erica has an interface board mounted to it. The interface boards were made using the Humble Pi prototyping boards, which are designed to fit on top of the Pi. Each of the Pi SBCs had a different interface board providing the necessary electronics. The Interface Pi is the simplest just requiring an RTC to allow Erica to maintain accurate time when no internet connection is available. The Pi SBCs responsible for eye control were provided with the hardware required to drive servos and simulate blinking as discussed in Section 3.1. The Brain Pi has a digital temperature sensor (TMP102) along with connections for the touch and PIR sensors. The Mech Pi interface contained the connectors to link to the ear control boards drivers and master control hardware for the LED subsystem. An example of an interface board from this generation of hardware is shown in Figure 6a.

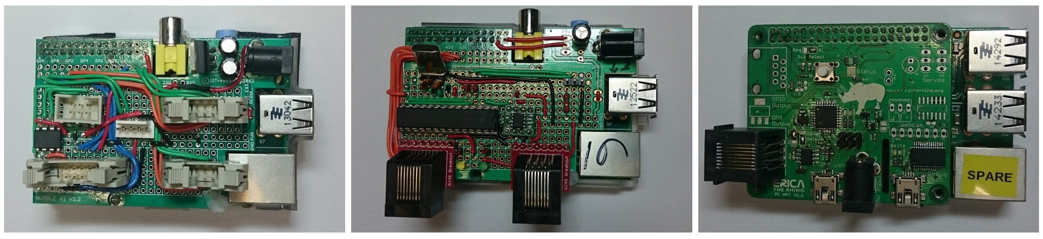

(**a**) (**b**) (**c**)

Figure 6. (**a**) an original electronics interface board; (**b**) second generation hardware interface; (**c**) current hardware interface HAT.

Erica's other circuit boards were initially made using strip board to allow for fast iterative development of the electronics required. Whilst this enabled Erica to be made quickly, the process of making spares was extremely time consuming, leading to only a couple of spares of the most common parts being made. This led to a change in approach after the initial deployment, as at least six of each of the main boards were required.

An example of a circuit board that was used widely in Erica's construction was the LED controller. Erica has 32 single colour and 15 RGB LEDs distributed around her body. Rather than have all these LEDs connected to a single controller board, a distributed control system was used. This simplified the cabling required inside Erica. Each LED (or colour, if RGB) had a separate PWM control channel. The distributed dimmers were connected together using a shared SPI bus which originates from the Mech Pi. The structure of the the lighting control can be seen in Figure 7.

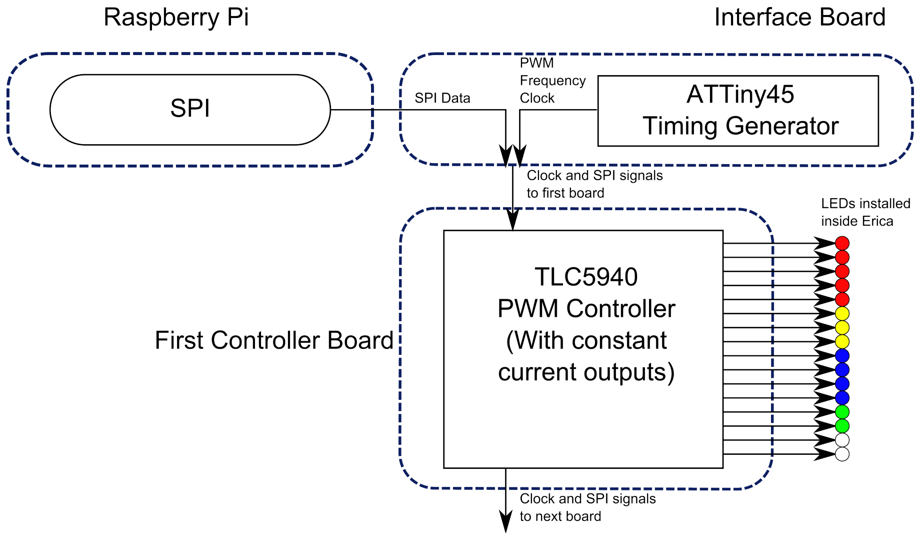


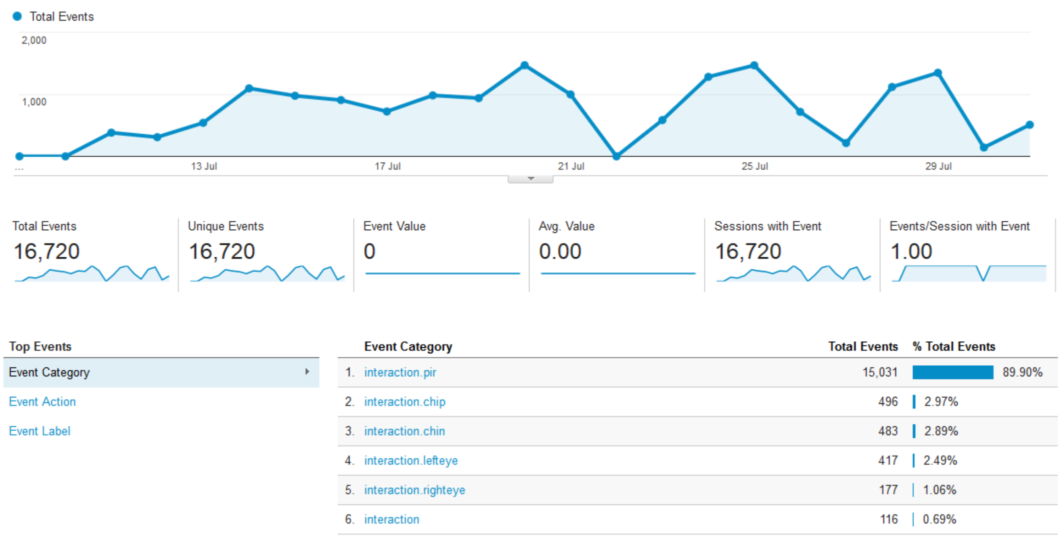
Figure 7. Original design for Erica's LED controllers.

4. *Go! Rhinos* Deployment

In summer 2013, Marwell Wildlife organised an 'art trail' around the city of Southampton, UK. The 36 life size rhinos and 62 smaller rhinos were on display for 10 weeks, and enjoyed by approximately 250,000 residents and visitors [8]. At the conclusion of the art trail, the life size rhinos were sold by auction raising a total of £124,700 for three charitable causes.

Unlike the other life-size rhinos on the art trail, Erica was located inside a local shopping centre. This location was chosen because of the availability of power, network and the realisation that making Erica both rain proof and resilient to vandalism would not be feasible. There was one particular unforeseen problem. The location had a large skylight which acted like a greenhouse and allowed direct sunlight to illuminate Erica's mostly black paintwork resulting in her internal temperature exceeding 45 °C on several occasions. While the Raspberry Pi SBCs handled this without issue, it was found that the glue holding circuit boards and cables inside of Erica was not able to cope, turning a series of neat cable looms and mounted hardware into a mess, leading to hardware failures.

Despite the thermally induced hardware failure, Erica's deployment was a success with Erica's analytics, as seen in Figure 8, showing that a significant number of people interacted with her. This shows that the majority of the interactions observed were from the PIR sensor. This has been attributed to the fact that this sensor did not require visitors to actively engage with Erica, meaning a substantial proportion of the count could be people passively observing her or just walking past. It was also observed that the other interactions available were not immediately obvious. Whilst there was signage describing the different ways that Erica could be interacted with, this was not presented in a child accessible way. Children would approach Erica and start randomly touching and stroking her, until their carers explained the interaction functionality available, having read the signage provided.



Total Events	Unique Events	Event Value	Avg. Value	Sessions with Event	Events/Session with Event
16,720	16,720	0	0.00	16,720	1.00

Top Events

- Event Category
- Event Action
- Event Label

	Event Category	Total Events	% Total Events
1.	interaction.pir	15,031	89.90%
2.	interaction.chip	496	2.97%
3.	interaction.chin	483	2.89%
4.	interaction.lefteye	417	2.49%
5.	interaction.righteye	177	1.06%
6.	interaction	116	0.69%

Figure 8. Daily counts of interactions with Erica during the *Go! Rhinos* deployment, recorded using Google Analytics.

Areas where public expectations differed significantly from the design were at the screens embedded in Erica's sides. These displays cycled automatically through a set content sequence including: details of the other rhinos on the art-trail, Erica's mood, and details of conservation efforts in the wild. The public expectation, however, was for these to be touch screens that would provide additional methods of interacting with Erica.

One of the features built into Erica that is immediately accessible is the window in her belly to view the electronics. Whilst adults tended to use the mirrors to save having to bend down, children were much more likely to crawl around underneath Erica herself in order to get the best view possible. The team of "Rhino Engineers" responsible for maintaining Erica were all issued with bright red branded t-shirts. Whilst wearing these t-shirts, team members were approached by members of the public who were wanting to know more, or provide feedback including bug reports. This feedback combined with team members' own observations were used to steer decisions behind the upgrades detailed in Section 5.

5. Upgrades and Maintenance

After the *Go! Rhinos* deployment, Erica returned to the University and the opportunity was taken to carry out general maintenance, perform upgrades based on feedback received and repair damage sustained during the deployment. The majority of the upgrades, with the exception of upgrading to Raspberry Pi 2s, were done in order that Erica could be taken to the 2014 Big Bang fair [23] as part of the University exhibit.

5.1. Physical Changes

Whilst performing maintenance on Erica during her time on the art trail, it was found that removing the main board was a time consuming task, due to the numerous connections to the body electronics. To address this, the electronics were redesigned to use a limited number of category 5 network cables for all signals connected to the Raspberry Pi SBCs. The new design of cabling infrastructure is shown in Figure 9, with changes to the electronics discussed in Section 5.3.

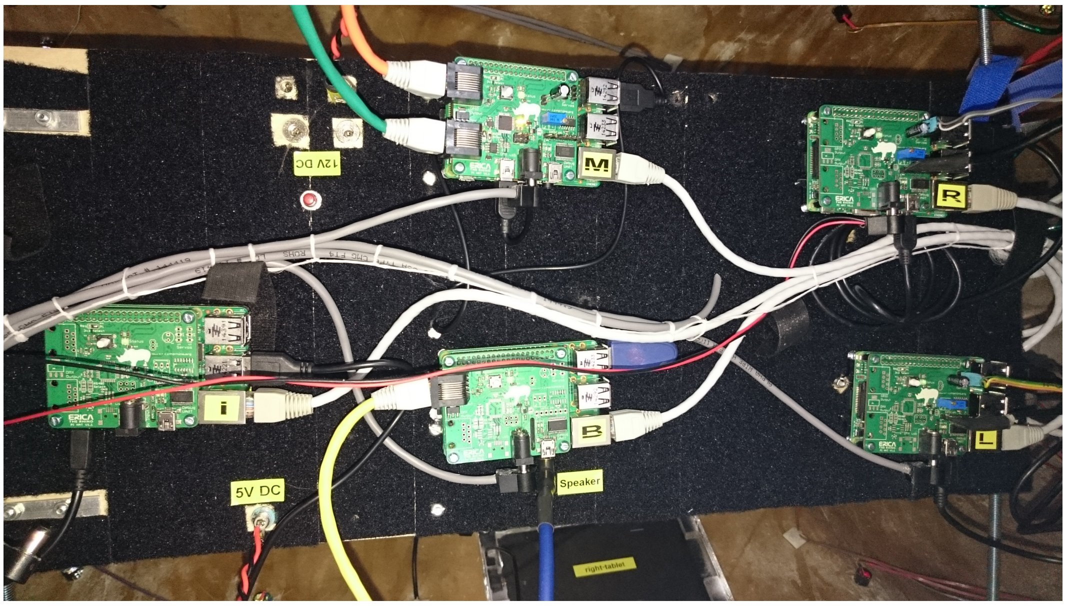


Figure 9. Erica's main computing board.

The cabling redesign was also extended into the plinth on which Erica is mounted. During the *Go! Rhinos* deployment, Erica's only physical external connection was the main power. This was ideal when Erica was left unattended in a shopping centre but was limiting for exhibition use and debugging. To improve usability, the plinth was fitted with PowerCon input and output power connections, network connections for both internal and external network, audio outputs (for when her internal speaker is not loud enough) and HDMI & USB connections to the interface Pi for debugging. All these connections were carried up from the plinth through Erica's legs, but can be unplugged to enable the plinth to be removed for transport.

5.2. *Processing Upgrades*

The performance offered by the original Raspberry Pi model B proved to be a significant limitation and affected all stages of the project, influencing architecture decisions and limiting responsiveness to interactions. When the Raspberry Pi 2 [24] was announced in 2015, it was an obvious decision to upgrade all Erica's Raspberry Pi SBCs to this new model to increase performance. Erica's overall responsiveness improved and allowed for more complex interactions, but the biggest difference observed was in the improvement in the performance of her eyes. Face detection now happens significantly faster and it was possible to implement the 'QR Cubes' and 'See what I see' as discussed in Section 5.4.

During the deployment, issues with SD card reliability were encountered. These issues have been explained by the fact that, in all deployment scenarios, the power has occasionally been cut off without performing a graceful shutdown first. This has been a recurring problem through Erica's multi-year lifetime. In order to simplify the process of recreating SD cards when needed and keeping the systems up to date, Puppet [25] scripts were created allowing the images to be rebuilt on replacement cards.

The LED subsystem had proven to be unreliable and susceptible to RF interference during the *Go! Rhinos* deployment. This was primarily due to the use of 3.3-volt SPI signals over excessively long cables. The replacement communication protocol chosen was DMX512 [26] as this is designed to cope with cable lengths significantly greater than needed. Given this change, a new design of hardware was needed, as shown in Figure 10. The hardware required for the main control interface is shown in Figure 6b. Having learnt from the scalability issues of using stripboard and having more development time, a PCB was created and the interface on the Pi was replaced with an open source DMX controller.

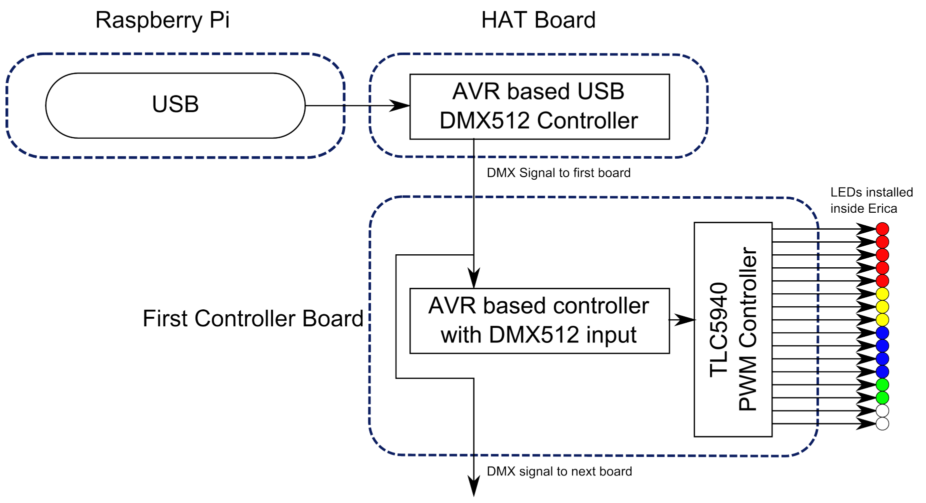

Raspberry Pi
HAT Board
USB
AVR based USB DMX512 Controller
DMX Signal to first board
LEDs installed inside Erica
First Controller Board
AVR based controller with DMX512 input
TLC5940 PWM Controller
DMX signal to next board

Figure 10. New DMX based design for Erica's LED controllers.

5.3. *LED Hardware Upgrades*

The form factor change of the Pi 2 when compared to the model B Pi required a redesign of the hardware interface boards. This new generation of boards was designed to be HAT-compliant (Hardware Attached on Top) [27]. Rather than create a separate HAT for each function, it was decided that a single modular HAT design (as shown diagrammatically in Figure 11 and built in Figure 6c) would simplify deployment and maintenance. These HATs contain an RTC, eye control hardware, a DMX controller and GPIO (General Purpose Input/Output) breakout. The designs for these HATs and all the associated software is Open Source and is available from the Erica github [28].

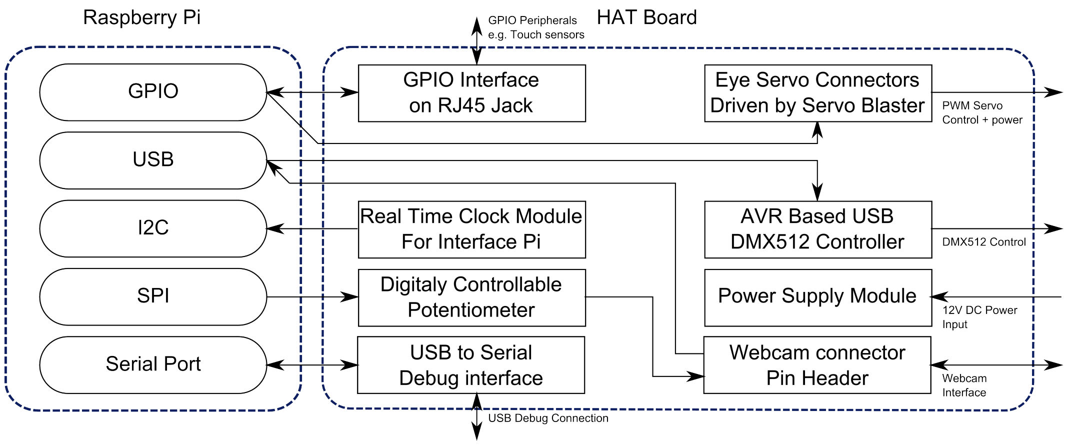

Raspberry Pi
GPIO Peripherals e.g. Touch sensors
HAT Board
GPIO
USB
I2C
SPI
Serial Port
GPIO Interface on RJ45 Jack
Eye Servo Connectors Driven by Servo Blaster
PWM Servo Control + power
Real Time Clock Module For Interface Pi
AVR Based USB DMX512 Controller
DMX512 Control
Digitaly Controllable Potentiometer
Power Supply Module
12V DC Power Input
USB to Serial Debug interface
Webcam connector Pin Header
Webcam Interface
USB Debug Connection

Figure 11. Block Diagram of Erica HATs.

5.4. *Screens & Interaction*

Initially, the 7" screens mounted in Erica's sides were HDMI monitors attached to the Brain and Interface Pi SBCs. These were intended to display a loop of static pages for visitors to consume. However, shortly after deploying them onto the art trail, several passers by commented that they were expecting them to be touch screens with interactive content. This reaction continued throughout the deployment. Therefore, it was decided to make an architectural change and replace these screens with Android-based tablet devices connected to Erica's local wireless network to provide touch interaction with dynamic content. This was done before the Pi touch [29] displays were available, and if this were to be done now, these displays would be the more obvious choice.

The tablets display a web-based menu system in a kiosk-mode browser that allows visitors to interactively view information about Erica, her mood, rhino conservation and the other Rhinos from the *Go! Rhinos* campaign. They also allow visitors to trigger Erica's eye movement, ear movement, horn LED colour change and body LED animation. The decision was also taken to allow people to see what Erica could see as it was a requested feature. A screen shot of the web interface is shown in Figure 12.

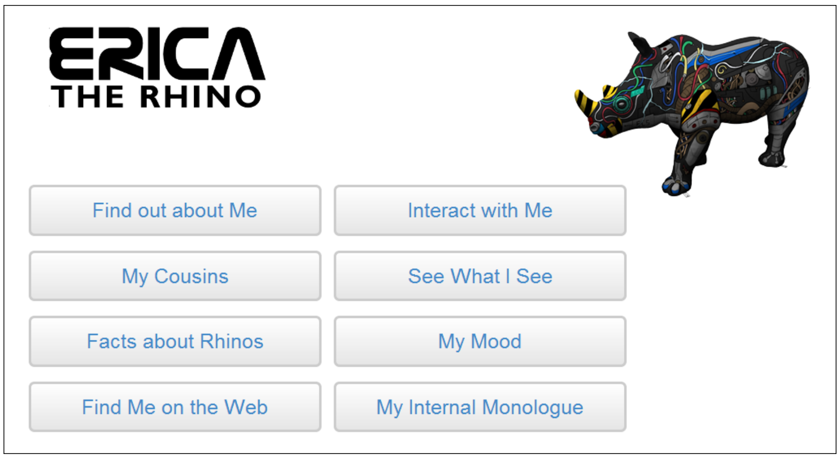


Figure 12. The home screen on the tablets in Erica's side.

Even after introducing interactive touch screens, it was still felt that the range and ease of interactions with Erica was lacking. The ability to identify QR cubes using Erica's two webcam eyes had never been fully exploited in a way that was simple and intuitive to an average visitor. Therefore, in June 2015, prior to the University of Southampton's open days, a number of cardboard cubes were constructed, as illustrated in Figure 13.

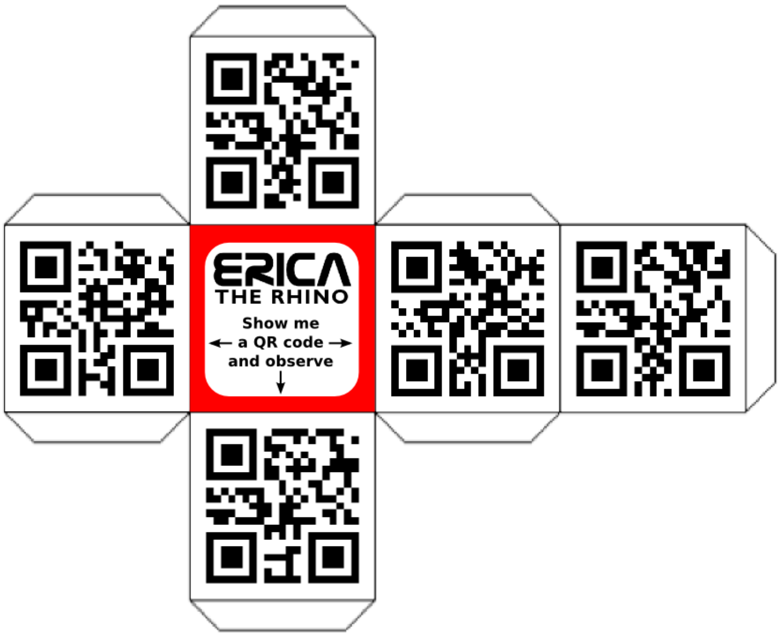


Figure 13. A two-dimensional net for a QR cube developed to aid interacting with Erica.

Each of the five QR codes on each cube represent a word within a theme and will lead to some reaction from Erica. One set of codes plays samples of music across a particular theme. Another set allows all of the body LEDs of one colour to be switched on or off depending on which eye the QR code is presented. The final set express one of five different emotions that involve at least two separate outputs.

These cubes have been particularly useful in increasing the amount of interactions with Erica on a day-to-day basis in her permanent home at the University of Southampton. The amount of time a right eye QR code scan has spent in Erica's short time memory has increased twenty-fold, and, for the left eye, this has increased over one-hundred-fold.

Whilst these improvements were being developed and deployed, Erica was touring the country and receiving visitors at her permanent home in Southampton.

6. Public Engagement and Impact

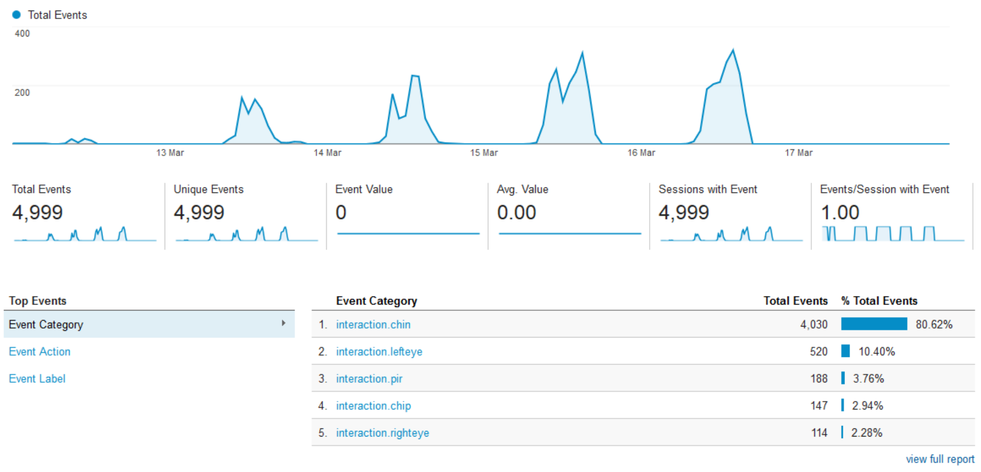
In terms of public engagement, there were three key outcomes from developing Erica.

While Erica was being initially developed, nine classes (approximately 230 pupils) were invited to see Erica at the University and discuss the sorts of interactions that they could imagine having with her. The pupils then learned about the basics of programming and how the hardware and software inside Erica worked. This was evaluated using questionnaires given to all students, which showed that the classes enjoyed understanding the potential of technology. All of these classes have since returned to the university for follow-up computing workshops.

Evaluating feedback from the general public, the mirror tiles placed underneath Erica to allow people to easily see the technology inside and the visits from the 'rhino engineers' when things needed fixing were both received positively. In particular, it helped make the public aware that this was a research project rather than a commercial one and gave people the opportunity to find out how Erica functioned.

While on display during the *Go! Rhinos* campaign, a pop-up classroom was run that taught programming to almost 1200 young people from the local community. They were told about Raspberry Pi computers, and how they could use software to build their own rhino components. Several participants made return visits to this workshop and parents were impressed at how much their children had learnt and carried on learning at home. These sessions were run in addition to the outreach sessions organised by Marwell Wildlife as part of the wider *Go! Rhinos* campaign. This campaign proved so successful that Marwell Wildlife are organising a follow on event this year focusing on Zebras [30].

As a result of the project, the authors have been approached by the organisers of science public engagement events to take Erica on tour. Erica was on display at the Big Bang Fair in March 2014 where there were approximately 5000 interactions over the four-day long event. Approximately 4000 of these interactions were people "feeding" Erica by touching her chin sensor as shown in Figure 14. Erica was also at the 2015 Cheltenham Science Festival. In addition to external visits, she has been part of the internal university science days for the last three years, which see approximately 4000 to 5000 people through the door each year.



Total Events	Unique Events	Event Value	Avg. Value	Sessions with Event	Events/Session with Event
4,999	4,999	0	0.00	4,999	1.00

Top Events

- Event Category
- Event Action
- Event Label

	Event Category	Total Events	% Total Events
1.	interaction.chin	4,030	80.62%
2.	interaction.lefteye	520	10.40%
3.	interaction.pir	188	3.76%
4.	interaction.chip	147	2.94%
5.	interaction.righteye	114	2.28%

view full report

Figure 14. Daily counts of interactions with Erica at the Big Bang fair, recorded using Google Analytics.

No matter where Erica has been displayed she has received interest from parents and children alike, with conversations ranging from electronics and programming to rhino conversation via her artwork. She was a finalist in the UK Public Engagement Awards, obtained a University of Southampton Vice-Chancellor's award, appeared in international media [31] and has been used as an example of Pi outreach by companies such as RS [32], and Rapid Electronics [33].

Erica is now on permanent display in the foyer of the Mountbatten building at the University of Southampton where she has regular interactions with staff and students of the University along with members of the local community. It is safe to say that Erica is now a local celebrity!

7. Conclusions

Erica the Rhino was created as a piece of interactive artwork to promote Electronics and Computer Science and the University of Southampton. In order to achieve this, an inter-disciplinary team was brought together from across the University. A feature set was decided upon and implemented, along with an artistic design for Erica's exterior. In order to implement these features, it was decided to use Raspberry Pi SBCs, as this way anyone interested in the technologies in use could acquire the same hardware cheaply to enable them to experiment themselves. Furthermore, as all the software and custom hardware created for this project is Open Source, other parties could develop their own art pieces using the same foundations.

The choice of using Raspberry Pi SBCs inside Erica to provide the compute power has influenced the entire design of Erica, both in terms of features available and how they are implemented. The same features could have been implemented using a less complicated architecture by combining a few Arduinos [34] with a small form factor PC. The outreach and engagement benefits of using the Pi have vastly outweighed the additional complication that it brought. In terms of outreach, Erica has been seen by several thousand young people and has prompted conversations on a wide variety of topics, some of whom have been inspired to continue learning at home. Overall, the entire project has been very successful, surpassing any expectations that the team had when the project was started.

Acknowledgments: Erica would not have been possible without support from the Marwell Wildlife *Go! Rhinos* team, Chris Clancy for the graphic design and painting and the rest of the Erica team as listed at http://www.ericatherhino.org/the-team/.

Author Contributions: Philip Basford: external networking and logistics, Graeme Bragg: animatronics, Jon Hare: visual system and animatronics, Michael Jewell: brain and personality, Kirk Martinez: project concept and lead, David Newman: Internal networking, Reena Pau: outreach Coordinator, Ashley Smith: linked open data, Tyler Ward: Electronics, firmware and construction.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Soler-Adillon, J. The intangible material of interactive art: Agency, behavior and emergence. *Artnodes* **2015**, *16*, doi:10.7238/a.v0i16.2744.
2. Paul, C. *Digital Art*; World of art, Thames & Hudson: London, UK, 2003.
3. Shortess, G. An application of a microcomputer as an interactive art medium. In Proceedings of the IEEE 1982 Symposium on Small Computers in the Arts, Philadelphia, PA, USA, 15–17 October 1982; IEEE: New York, NY, USA, 1982; pp. 75–77.
4. Muller, L.; Edmonds, E.; Connell, M. Living laboratories for interactive art. *CoDesign* **2006**, *2*, 195–207,
5. Bannai, Y.; Okuno, Y.; Kakuta, H.; Takayama, T. A mixed reality blow interface for the interactive art "Jellyfish Party". *Trans. Inf. Process. Soc. Jpn.* **2005**, *46*, 1594–1602.
6. Patridge, M.; Huntley, J. Fluxbase, An Interactive Art Exhibition. *Vis. Lang.* **1992**, *26*, 221–227.
7. Available online: http://www.raspberrypi.org/blog/tag/art-installation/ (accessed on 29 April 2016).
8. Available online: http://web.archive.org/web/20131221185340/http://gorhinos.co.uk/ (accessed on 29 April 2016).
9. Edmonds, E.; Turner, G.; Candy, L. Approaches to interactive art systems. In Proceedings of the 2nd international Conference on Computer graphics and Interactive Techniques in Australasia and South East Asia, Singapore, 15–18 June 2004; ACM: New York, NY, USA, 2004; pp. 113–117.
10. Dorin, A. Building Artificial Life for Play. *Artif. Life* **2004**, *10*, 99–112.
11. Available online: http://www.raspberrypi.org/products/camera-module/ (accessed on 29 April 2016).
12. Hare, J.S.; Samangooei, S.; Dupplaw, D.P. OpenIMAJ and ImageTerrier: Java Libraries and Tools for Scalable Multimedia Analysis and Indexing of Images. In Proceedings of the 19th ACM International Conference on Multimedia, Scottsdale, AZ, USA, 28 November–1 December 2011; ACM: New York, NY, USA, 2011; pp. 691–694.
13. Viola, P.; Jones, M. Robust real-time face detection. *Int. J. Comput. Vis.* **2004**, doi:10.1023/B:VISI.0000013087.49260.fb.
14. Available online: http://github.com/zxing/zxing (accessed on 29 April 2016).
15. Bizer, C.; Heath, T.; Berners-Lee, T. Linked data-the story so far. *Semant. Serv. Interoper. Web Appl.: Emerg. Concepts* **2009**, 205–227.
16. Available online: http://data.ericatherhino.org (accessed on 28 June 2016)
17. Martinez, K.; Basford, P.; Jager, D.D.; Hart, J.K. A wireless sensor network system deployment for detecting stick slip motion in glaciers. In Proceedings of the IET International Conference on Wireless Sensor Systems 2012, London, UK, 18–19 June 2012; Curran Associates, Inc.: Red Hook, NY, USA, 2012; pp. 14–16.
18. Deering, S.; Hinden, R. RFC 2460: Internet Protocol Version 6 Specification, 1998.
19. Available online: http://www.sixxs.net/ (accessed on 29 April 2016).
20. Available online: http://www.icinga.org/ (accessed on 29 April 2016).
21. Available online: http://www.djangoproject.com/ (accessed on 29 April 2016).
22. Available online: http://www.cherrypy.org/ (accessed on 29 April 2016).
23. Available online: http://www.thebigbangfair.co.uk/ (accessed on 29 April 2016).
24. Upton, E. Available online: http://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/ (accessed on 29 April 2016).
25. Available online: http://puppet.com/ (accessed on 29 April 2016).
26. DMX512-A – Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories, 2013.
27. Adams, J. Available online: http://www.raspberrypi.org/blog/introducing-raspberry-pi-hats/ (accessed on 29 April 2016).
28. Available online: https://github.com/ericatherhino/ (accessed on 28 June 2016).
29. Hollingworth, G. Available online: http://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/ (accessed on 29 April 2016).
30. Available online: http://zanyzebras.org.uk/ (accessed on 29 April 2016).
31. Schmidt, J. Von bastlern und erfindern. *Raspberry Pi 2 Handbuch* **2015**, *148*, 65.

32. Available online: http://www.rsonline.biz/raspberry-pi/9-Best-Things-Raspberry-Pi-Infographic.html (accessed on 29 April 2016).
33. Available online: http://www.rapidonline.com/News/The-rhino-powered-by-Raspberry-Pi-3543 (accessed on 29 April 2016).
34. Available online: http://www.arduino.cc/ (accessed on 29 April 2016).

References

Akyildiz I. F., Su W., Sankarasubramaniam Y., and Cayirci E. A survey on sensor networks. *Communications magazine, IEEE*, 40(8):102–114, 2002.

Baccelli E., Hahm O., Gunes M., Wahlisch M., and Schmidt T. C. Riot os: Towards an os for the internet of things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, April 2013.

Basford P. J., Bragg G. M., Hare J. S., Jewell M. O., Martinez K., Newman D. R., Pau R., Smith A., and Ward T. Erica the rhino: A case study in using raspberry pi single board computers for interactive art. *Electronics*, 5(3):35, 2016.

Basford P. J. Using an energy aware adaptive scheduling algorithm to increase sensor network lifetime. February 2015.

Bencini L., Manes A., Di Palma D., Manes G., and Collodi G. *Wireless sensor networks for on-field agricultural management process.* INTECH Open Access Publisher, 2010.

Bormann C., Ersue M., and Keranen A. Terminology for Constrained-Node Networks. RFC 7228 (Informational), May 2014.

Bragg G., Martinez K., Basford P., and Hart J. 868MHz 6LoWPAN with ContikiMAC for an internet of things environmental sensor network. 2016.

Burleigh S. Compressed Bundle Header Encoding (CBHE). RFC 6260 (Experimental), May 2011.

Butterfield A., Ngondi G., and Kerr A. *A Dictionary of Computer Science.* Oxford Quick Reference. OUP Oxford, 2016. ISBN 9780191002885.

Carpenter B. and Jiang S. Transmission and Processing of IPv6 Extension Headers. RFC 7045 (Proposed Standard), December 2013.

Case J., Fedor M., Schoffstall M., and Davin J. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

Cerf V., Burleigh S., Hooke A., Torgerson L., Durst R., Scott K., Fall K., and Weiss H. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007.

Cerf V., Burleigh S., Hooke A., Torgerson L., Durst R., Scott K., Travis E., and Weiss H. Interplanetary Internet (IPN): Architectural Definition. Active Internet-Draft, May 2001.

Cerf V., Burleigh S., Hooke A., Torgerson L., Durst R., Scott K., Travis E., and Weiss H. Delay-Tolerant Network Architecture: The Evolving Interplanetary Internet. Active Internet-Draft, August 2002.

Conta A., Deering S., and Gupta M. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443 (Draft Standard), March 2006. Updated by RFC 4884.

Costanza E., Panchard J., Zufferey G., Nembrini J., Freudiger J., Huang J., and Hubaux J.-P. Sensortune: a mobile auditory interface for diy wireless sensor networks. April 2010.

Deering S. and Hinden R. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

Dunkels A., Gronvall B., and Voigt T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462, 2004.

Durvy M., Abeillé J., Wetterwald P., O'Flynn C., Leverett B., Gnoske E., Vidales M., Mulligan G., Tsiftes N., Finne N., and Dunkels A. Making sensor networks IPv6 ready. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 421–422, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-990-6.

Eddy W. and Davies E. Using Self-Delimiting Numeric Values in Protocols. RFC 6256 (Informational), May 2011.

Fall K. and Farrell S. Dtn: an architectural retrospective. *Selected Areas in Communications, IEEE Journal on*, 26(5):828–836, June 2008. ISSN 0733-8716.

Farhangi H. The path of the smart grid. *Power and Energy Magazine, IEEE*, 8(1): 18–28, January 2010. ISSN 1540-7977.

Fenner B. Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers. RFC 4727 (Proposed Standard), November 2006.

Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., and Berners-Lee T. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

Gomeimoto and Matsumoto T. Implementation using the ipv6 extension header of the dtn protocol. *Information Processing Society of the 75th National Convention*, 3:7, 2013.

Gont F., Linkova J., Chown T., and Liu W. Observations on IPv6 EH Filtering in the Real World. Active Internet-Draft, March 2015.

Hart J. K. and Martinez K. Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, 78:177–191, 2006.

Hart J. K. and Martinez K. Toward an environmental internet of things. *Earth and Space Science*, 2(5):194–200, 2015.

Hart J. K., Rose K. C., Waller R. I., Vaughan-Hirsch D., and Martinez K. Assessing the catastrophic break-up of briksdalsbreen, norway, associated with rapid climate change. *Journal of the Geological Society*, 168(3):673–688, 2011.

Ho M. and Fall K. Poster: Delay tolerant networking for sensor networks. In *First IEEE Conference on Sensor and Ad Hoc Communications and Networks*, page 3, 2004.

Huang R., Song W. Z., Xu M., Peterson N., Shirazi B., and LaHusen R. Real-world sensor network for long-term volcano monitoring: Design and findings. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):321–329, Feb 2012. ISSN 1045-9219.

Hui J. and Thubert P. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), September 2011.

Jenkins A., Kuzminsky S., Gifford K., Pitts R., and Nichols K. Delay/disruption-tolerant networking: Flight test results from the international space station. In *Aerospace Conference, 2010 IEEE*, pages 1–8, March 2010.

Jovanov E. and Milenkovic A. Body area networks for ubiquitous healthcare applications: Opportunities and challenges. *Journal of Medical Systems*, 35(5):1245–1254, 2011. ISSN 1573-689X.

Kent S. IP Authentication Header. RFC 4302 (Proposed Standard), December 2005.

Kimura N. and Latifi S. A survey on data compression in wireless sensor networks. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 2, pages 8–13 Vol. 2, 2005.

Ko J., Klues K., Richter C., Hofer W., Kusy B., Bruenig M., Schmid T., Wang Q., Dutta P., and Terzis A. Low power or high performance? a tradeoff whose time has come (and nearly gone). In *Proceedings of the 9th European Conference on Wireless Sensor Networks*, EWSN'12, pages 98–114, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-28168-6.

Konovalov I. A framework for wirelesshart simulations. Technical report, Swedish Institute of Computer Science Box 1263, SE-164 29 Kista, Sweden, 2010.

Kovatsch M., Duquennoy S., and Dunkels A. A low-power coap for contiki. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 855–860, Oct 2011.

Krishnan S., Woodyatt J., Kline E., Hoagland J., and Bhatia M. A Uniform Format for IPv6 Extension Headers. RFC 6564 (Proposed Standard), April 2012.

Kushalnagar N., Montenegro G., and Schumacher C. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), August 2007.

LAN/MAN Standards Committee IEEE Computer Society . IEEE Standard for local and metropolitan area networks — Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). Technical report, Institute of Electrical and Electronics Engineers, 2011.

Lee J.-S., Su Y.-W., and Shen C.-C. A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *33rd Annual Conference of the IEEE Industrial Electronics Society, 2007. IECON 2007*, pages 46–51, 2007.

Lu C.-W., Li S.-C., and Wu Q. Interconnecting zigbee and 6lowpan wireless sensor networks for smart grid applications. In *Fifth International Conference on Sensing Technology (ICST), 2011*, pages 267–272, Nov 2011.

Lu G., Krishnamachari B., and Raghavendra C. S. Performance evaluation of the ieee 802.15.4 mac for low-rate low-power wireless networks. In *2004 IEEE International Conference on Performance, Computing, and Communications*, pages 701–706, 2004.

Martinez K., Hart J. K., Basford P. J., Bragg G. M., and Ward T. Using Internet of Things technologies for wireless sensor networks. *AGU Fall Meeting Abstracts*, December 2013.

Martinez K., Hart J. K., and Ong R. Environmental sensor networks. *Computer*, 37(8): 50–56, Aug 2004. ISSN 0018-9162.

Martinez K., Hart J. K., and Ong R. Deploying a wireless sensor network in iceland. *Lecture Notes in Computer Science, Proc. Geosensor Networks*, 5659:131–137, July 2009.

Martinez K., Padhy P., Elsaify A., Zou G., Riddoch A., Hart J., and Ong H. Deploying a sensor network in an extreme environment. In *Sensor Networks, Ubiquitous and Trustworthy Computing*, pages 186–193. IEEE Computer Society, June 2006. Event Dates: 5-7 June 2006.

Martinez K., Basford P., Ellul J., and Clarke R. Field deployment of low power high performance nodes. In *The Third International Workshop on Sensor Networks (SN 2010)*, 2010. Event Dates: 21st June 2010.

Martinez K., Basford P. J., Jager D. D., and Hart J. K. A wireless sensor network system deployment for detecting stick slip motion in glaciers. June 2012.

Martinez K., Basford P. J., Jager D. D., and Hart J. K. Poster abstract: Using a heterogeneous sensor network to monitor glacial movement. In *10th European Conference on Wireless Sensor Networks*, February 2013.

Montenegro G., Kushalnagar N., Hui J., and Culler D. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007. Updated by RFCs 6282, 6775.

Muller K. and Vignaux T. Simpy: Simulating systems in python. *ONLamp. com Python Devcenter*, 2003.

Mulligan G. The 6LoWPAN architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, pages 78–82, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-694-3.

Neto J. R. T., Yokoyana R. S., and Villas L. A. A robust and lightweight protocol for efficient and scalable automatic meter reading using an unmanned aerial vehicle. *IEEE Latin America Transactions*, 13(9):3044–3050, Sept 2015. ISSN 1548-0992.

Nikkhah M. and Guérin R. Migrating the internet to ipv6: An exploration of the when and why. *IEEE/ACM Transactions on Networking*, 24(4):2291–2304, Aug 2016. ISSN 1063-6692.

Nixon M. and Rock T. R. A Comparison of WirelessHART and ISA100. 11a. *white paper, July*, pages 1–36, 2012.

Osterlind F., Dunkels A., Eriksson J., Finne N., and Voigt T. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648, nov 2006.

Ostermaier B., Kovatsch M., and Santini S. Connecting things to the web using programmable low-power wifi modules. In *Proceedings of the Second International Workshop on Web of Things*, page 2, 2011.

Perkins C., Johnson D., and Arkko J. Mobility Support in IPv6. RFC 6275 (Proposed Standard), July 2011.

Postel J. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.

Pottner W.-B., Busching F., Zengen von G., and Wolf L. Data elevators: Applying the bundle protocol in Delay Tolerant Wireless Sensor Networks. In *2012 IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 218–226, Oct 2012.

Price R. Google's parent company is deliberately disabling some of its customers' old smart-home devices. *Buisness Insider*, 2016.

Puccinelli D. and Haenggi M. Wireless sensor networks: applications and challenges of ubiquitous sensing. *IEEE Circuits and Systems Magazine*, 5(3):19–31, 2005. ISSN 1531-636X.

Scott K. and Burleigh S. Bundle Protocol Specification. RFC 5050 (Experimental), November 2007.

Shelby Z., Hartke K., and Bormann C. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), June 2014. Updated by RFC 7959.

Song W.-Z., Huang R., Xu M., Ma A., Shirazi B., and LaHusen R. Air-dropped sensor network for real-time high-fidelity volcano monitoring. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, MobiSys '09, pages 305–318, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-566-6.

Stajano F., Hoult N., Wassell I., Bennett P., Middleton C., and Soga K. Smart bridges, smart tunnels: Transforming wireless sensor networks from research prototypes into robust engineering infrastructure. *Ad Hoc Networks*, 8(8):872 – 888, 2010. ISSN 1570-8705.

Suryady Z., Shaharil M. H. M., Bakar K. A., Khoshdelniat R., Sinniah G. R., and Sarwar U. Performance evaluation of 6lowpan-based precision agriculture. In *The International Conference on Information Networking 2011 (ICOIN2011)*, pages 171–176, Jan 2011.

Tanenbaum A. *Computer Networks*. Computer Networks. Prentice Hall PTR, 1996. ISBN 9780133942484.

Thoma M., Braun T., Magerkurth C., and Antonescu A. F. Managing things and services with semantics: A survey. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–5, May 2014.

Volpe J. Mindscape pulls the server plug on nabaztag, hands source code to developers. *http://www.engadget.com/2011/07/28/mindscape-pulls-the-server-plug-on-nabaztag-hands-source-code-t/*, 2011.

Von Zengen G., Büsching F., Pöttner W.-B., and Wolf L. An overview of μdtn: Unifying dtns and wsns. *Proceedings of the 11th GI/ITG KuVS Fachgespräch" Drahtlose Sensornetze"(FGSN)*, 2012.

Waitzman D. Standard for the transmission of IP datagrams on avian carriers. RFC 1149 (Experimental), April 1990. Updated by RFCs 2549, 6214.

Ward T., Martinez K., and Chown T. Simulated Analysis of Connectivity Issues for Sleeping Sensor Nodes in the Internet of Things. In *The Eleventh ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Network*, 2014.

Ward T., Martinez K., and Chown T. Adding support for delay tolerance to IPv6 networks. In *The 10th International Conference on Future Networks and Communications (FNC 2015)*, 2015.

Werner-Allen G., Johnson J., Ruiz M., Lees J., and Welsh M. Monitoring volcanic eruptions with a wireless sensor network. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pages 108–120, Jan 2005.

Will H., Schleiser K., and Schiller J. A real-time kernel for wireless sensor networks employed in rescue scenarios. In *2009 IEEE 34th Conference on Local Computer Networks*, pages 834–841, Oct 2009.

Winter T., Thubert P., Brandt A., Hui J., Kelsey R., Levis P., Pister K., Struik R., Vasseur J., and Alexander R. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), March 2012.

Young D. S., Hart J. K., and Martinez K. Image analysis techniques to estimate river discharge using time-lapse cameras in remote locations. *Computers & Geosciences*, 76:1 – 10, 2015. ISSN 0098-3004.

Zhou B., Cao J., Zeng X., and Wu H. Adaptive traffic light control in wireless sensor network-based intelligent transportation system. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–5, Sept 2010.

Zhu X., Han S., Mok A., Chen D., and Nixon M. Hardware challenges and their resolution in advancing WirelessHART. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 416–421, 2011.

ZigBee Alliance . ZigBee IP Specification. *ZigBee Public Document 13-002r00*, 2013.