TinyTorrent: Combining BitTorrent and SensorNets

by

Karsten Holger Fritsche, B.Sc.(Hons)

Dissertation

Presented to the

University of Dublin, Trinity College

in partial fulfilment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2005

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Karsten Holger Fritsche

September 10, 2005

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Karsten Holger Fritsche

September 10, 2005

Acknowledgments

I would like to thank my supervisor Ciaran McGoldrick for his invaluable help and guidance on this project.

KARSTEN HOLGER FRITSCHE

University of Dublin, Trinity College
September 2005

TinyTorrent: Combining BitTorrent and SensorNets

Karsten Holger Fritsche, M.Sc.
University of Dublin, Trinity College, 2005

Supervisor: Dr Ciaran McGoldrick

The first of the two goals of this research was to investigate the BitTorrent protocol to determine what features make it successful on an Internet scale. These features include its ability to rapidly replicate data across the network, while ensuring fair participation of peers in the network. This resulted in the design and development of the TinyTorrent protocol, which adapted these BitTorrent features for a SensorNet environment. TinyTorrent was implemented for the Crossbow MICA2 hardware platform using nesC. The second goal was to explore a means of exposing SensorNet data to external systems. This resulted in the development of a SensorNet Plugin for the Azureus BitTorrent client, based on a proxy architecture. The SensorNet Plugin enables SensorNets to publish data to the BitTorrent network, and also acts as a tracker for the SensorNet running TinyTorrent. The TinyTorrent system was evaluated by experiment and simulation to determine the protocol overhead, energy consumption characteristics, effects of interference and real-world throughput. The evaluation showed that the use of more complex algorithms is feasible provided these lead to a reduction in the number or size of messages.

Contents

List of Figures

Chapter 1

Introduction

SensorNets are networks of sensor devices, such as satellites, CCTV cameras, or seismographs, which are used to monitor and control an environment. Recent advances in microprocessor technology and electronic fabrication techniques have made it possible to design small, low-power, autonomous sensor devices, equipped with short range radio components. These devices can be embedded directly into an environment, and can form ad-hoc wireless networks. This allows them to cooperate and coordinate their data gathering functions, route data through the network, filter information, and make more detailed and accurate observations about the environment than other techniques could achieve.

Large scale production volumes offer the potential of producing wireless sensors at extremely low cost, thus making it possible to deploy them in far greater quantities, and far more densely into an environment. This also means that each individual sensor is expendable to the network, and thus makes the network potentially more robust in the face of failures. This offers exciting opportunities in a variety of fields, such as industrial monitoring, home automation, safety and security applications, and health monitoring. In all these application domains, the low cost, small size, and high density of sensors in an environment opens up new opportunities for high quality data gathering.

SensorNets have significant limitations, such as very strict energy constraints, simple processors and short range communications which pose serious challenges to researchers. These constraints require careful attention to issues such as resource management (bandwidth and energy), protocol design and algorithm choices. Traditional sensor devices do not usually operate under the same constraints, as they often have the luxury of relying on a fixed

infrastructure to support them.

Such issues have been addressed at a larger scale by systems such as BitTorrent, which is a content distribution network based on the peer-to-peer model. The properties that make BitTorrent successful are the direct interaction between peers, the rapid duplication of content across the peers, and encouraging fairness among peers. This allows content to be rapidly replicated from the original publisher, thus distributing the cost of publication to many peers, and also ensures that content is redundantly available from several peers in the network.

The goal of this research was to investigate in detail what aspects of BitTorrent lead to its success in managing resources at an Internet scale, and applying these techniques to a SensorNet, taking into account the resource constraints under which sensor devices must operate. TinyTorrent was developed as a result of this investigation by adapting BitTorrent concepts to a SensorNet based on the TinyOS platform. The properties of TinyTorrent were evaluated experimentally through both simulations and real-world applications which run on physical sensor devices. A further goal was to investigate the means by which SensorNet data could be exposed to the outside world, in order to allow external systems to benefit from the information gathering ability of the SensorNet.

The evaluation showed that several important goals were achieved. Firstly, a BitTorrent proxy for a SensorNet was built, based on the well-known Azureus BitTorrent client. This involved developing a custom plugin for Azureus, as well as making modifications to some of the existing core plugins. This system was successfully able to publish data gathered by the SensorNet to a website accessible to any browser, and subsequently made it possible for any standard BitTorrent client to access the information published in this way. Secondly, the experimental evaluation of TinyTorrent further showed that the application of BitTorrent concepts to SensorNets was feasible, and highlighted some interesting properties of such an approach. An important result was that the energy usage is heavily weighted towards the radio hardware, with roughly 63% of all energy being used for communication. This implies that more complex processing is feasible on the motes, provided this helps to reduce the number or size of message. This was supported by the protocol overhead measurements which showed that only between 22% and 37% of all data transferred by TinyTorrent was used by the actual data payloads. Throughput measurements showed that transfer rates of 175.86 bytes/sec with a single mote were possible, with the transfer rates for a SensorNet of 10 motes dropping to between 27.44 bytes/sec and 55.56 bytes/sec for the entire SensorNet.

This research has raised several questions, and highlighted future opportunities for inves-

tigation, such as the effects of compression on throughput and energy consumption, possible techniques for reducing protocol overhead, and the effects of security protocols on energy consumption.

Chapter 2

Introduction to SensorNets

In order for computer systems to interact with their environment, they need to be able to gather information about the environment. Although this information can be provided by direct human input, in many applications it is preferable to equip systems with automatic means of gathering information. This is achieved using networks of autonomous sensor devices (or SensorNets).

Recent developments in microprocessor design, electronic fabrication techniques, and wireless communication technology have made it possible to produce small, autonomous electronic devices with the ability to gather data about their environment, communicate with each other, and cooperatively monitor or control the environment, at very low cost. Such SensorNets have become the focus of intense research, due to the exciting opportunities for new applications that such system have opened up, and the interesting (and complex) challenges that SensorNets pose. In this chapter, the key technologies that make SensorNets possible (such as improved electronic fabrication techniques and wireless communications) will be discussed. The challenges faced by SensorNet researchers face will be outlined, as well as the potential application areas of SensorNets, and an introduction to some of the commercial SensorNet platforms that have been developed.

2.1 Sensors

Sensors can range from devices such as cameras (eg: CCTV systems) to satellites and radar stations ([1]). These are typically connected directly to a centralised computer system, usu-

ally by high bandwidth networks. The continual decrease in manufacturing costs of microprocessors, coupled with their increased processing power, now raises the possibility of producing small, cheap, low-power "smart sensors" ([2], p 31) which can be embedded directly into the environment. Such "smart sensors" make it possible to monitor an environment at a much finer granularity than previously possible. Smart sensors can collect information such as light and temperature readings, vibrations, moisture levels, etc., and report them directly to a computer system. The system may then record and possibly alert human operators about the status of some environmental condition, or could be designed to manipulate the environment through actuators.

Sensor devices are used in some form in most industries which involve automated systems ([2]). These sensors are the first step in making more intelligent computer systems, which can react to changing environmental conditions.

2.2 Wireless Communications

Wireless communication has made it possible to construct networks where they would otherwise have not been possible. Wireless communication allows devices to be mobile (such as satellites, vehicles, etc.) while still being connected to a network, and thus to reap the benefits that membership of a computer network can offer. Wireless communication also reduces the complexity of the physical infrastructure required to support a network, since physical proximity and physical access to a device are not required, as they are in traditional wired networks. Deploying a wireless network into an environment also requires less physical modification to the environment, due to the lack of physical constraints on the wireless communications medium. However, wireless communications also brings its own set of challenges for researchers.

2.2.1 Bandwidth and Power

Bandwidth in wireless networks is usually limited compared to the bandwidth of wired networks. It is only recently that such wireless communications standards as the IEEE's 802.11 suite for wireless LANs has reached 54Mbps, while traditional Ethernet commonly operates at 100Mbps ([3]). This is compounded by the fact that wireless communication requires more energy than wired communication, and usually has a smaller range.

Recently, the need for low-power low-bitrate wireless technologies has also gained attention. Several standards are making progress in this area, such as the IEEE's 802.15 standards, which operate at between 20Kbps and 250Kbps. These standards are designed for small area networks, called Wireless Personal Area Networks (WPANs). This technology is intended for connecting small devices (with limited power) which are carried around a person's body. A more well-known competitor in this field is the BlueTooth standard ([4]), which is designed for similar ranges and low power devices. BlueTooth devices can operate at different power levels, depending on the intended communication range, from 1mW to 100mW.

2.2.2 Medium Access and Interference

The wireless spectrum is a broadcast medium. This means that there is no way to control the propagation of a signal once it has been transmitted. A consequence of this is that access to the wireless medium must be carefully controlled to prevent messages being corrupted. This is compounded by the fact that in wireless networks, nodes may unable to see each other, yet still be able to communicate with a common neighbour. In general, access to the medium should be controlled so that only one device at a time attempts to broadcast at a given frequency. Also, there needs to be a means of detecting collisions in the medium, and possibly a means of recovering from collisions. This means that medium access protocols in wireless environments are far more complex than their wired counterparts.

The broadcast medium also means that interference from external sources can play a role in the effectiveness of wireless communications. Most commercial wireless technologies operate in the ISM (Industrial, Scientific and Medical) bands, which are unlicensed. However, this means that these technologies compete with each other for the same medium. This becomes a problem especially for low-power devices, which can easily be overwhelmed by more powerful devices operating in the same frequency bands.

2.2.3 Security

The broadcast nature of wireless communications also has serious implications for security. Since the propagation of a signal cannot be controlled, it is crucial to take the security of communications into account when developing wireless communications protocols. The IEEE 802.11 standards have attempted to include security in their designs, by trying to simulate

the same level of security as wired networks. However, the WEP scheme (Wired Equivalent Privacy) has been shown to have flaws which make it unsuitable for serious use. More complex algorithms such as AES are being employed in 802.11i, in an attempt to make the standard more secure. However, this will continue to be a problem for designers of wireless protocols.

2.3 Sensor Networks

Sensor networks, or SensorNets, are networks of sensor devices which are characterised by three principle features([5], [6]):

- the ability to *gather sensory data* about their surroundings
- perform *data processing* on this data
- *communication* with neighbouring sensor devices or other systems

Wireless SensorNets are the natural extension of the trend towards smart sensors (see Section 2.1) combined with modern developments in wireless communications technologies (see Section 2.2), specifically in low-power technologies. The main goal of SensorNets is to perform remote distributed sensing tasks which take advantage of the collaborative nature of such networks to improve data quality and reliability. They can also provide more detailed data than other techniques would achieve, since the sensors can be deployed directly into the environment, and can be deployed far more densely than other techniques. These features make it possible for SensorNets to collaboratively filter data, and thus improve the detail and quality of the data.

Recent literature focuses mainly on SensorNets involving wireless sensor devices deployed in ad-hoc network topologies, with limited energy supplies, and limited communication range. These devices are often called motes or Smart Dust ([7]), due to their small size. They can be manufactured cheaply in large quantities, and provide for a denser deployment of sensors in the environment, thus allowing a much richer variety of data to be gathered. However, such large scale networks involving low-power devices pose serious challenges to researchers.

In this section, the main components of a wireless SensorNet device will be identified, and an analysis of the strengths and weaknesses of SensorNets, the constraints under which

they typically operate, and the application areas to which they have been applied is provided. A particular emphasis will be placed on wireless SensorNets.

2.3.1 Components of a Wireless Sensor Device

The main components of a wireless sensor device are the sensory input component, the processor, the communications component, and a power supply. These are shown in the block diagram in Figure 2.1 ([6]).

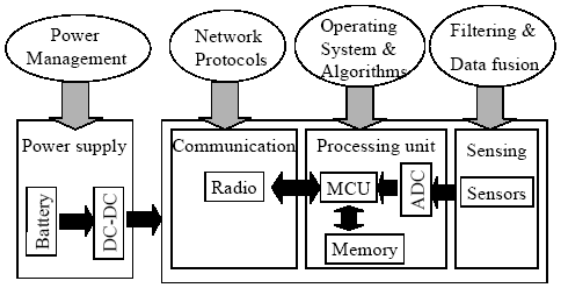

Figure 2.1: Main Components of a Sensor Device

In this section, the purpose of each of these components in wireless SensorNet devices will be discussed.

Sensor Input

Traditionally, sensors (such as satellites) are placed far from the phenomenon they are designed to observe ([5]). The wireless SensorNet approach effects a dense deployment of sensor devices in or near to the phenomenon, and uses ad-hoc networks for communication among the sensors. Sensor components typically produce an electrical response to changes in the physical environment ([6]), which can be converted into digital signals, processed by the sensor device, and possibly communicated to other sensors. Examples are accelerometers, light sensors, microphones, temperature sensors and pressure sensors.

Data Processing

SensorNets are often described as consisting of *atomic computation units* ([8]), where each node has some processing abilities and can function autonomously. In practice, microcontrollers are used as the data processing component, as these are usually general purpose and programmable, and thus can be adapted to different applications easily. The trend in recent times has been toward producing cheap, low power microcontrollers, which is ideal for a wireless SensorNet environment. In Section 2.3.2, the power constraints under which SensorNets often operate are discussed in more detail. In particular, the high energy cost of communication relative to local data processing is discussed. Microcontrollers are used to aggregate data locally, and thus save energy on communications.

Other alternatives to microcontrollers are "field programmable gate arrays", or FPGAs. These are devices which use gate array technology, allowing reprogramming of the device after manufacture ([9]). The advantage of such devices is that they can be far more flexible for development purposes due to their re-programmability, but they often consume more power. However, they do allow additional configurable control logic to be added to a device more easily ([10]).

Communications

In wireless SensorNets, the communication component involves a radio device. Due to the power constraints under which wireless sensor devices operate (Section 2.3.2) the radio is usually designed to use as little power as possible. This has an impact on communication range and bandwidth, both of which are often highly constrained in wireless SensorNets.

Power Supply

As has already been mentioned, energy is a prime concern in wireless SensorNets. Sensor devices are usually not connected to any physical data or power infrastructure, and must thus carry the power supply along with the device itself. The trend towards smaller sensor devices means that the power supplies which can be packaged with the devices are also highly constrained in terms of their size, and thus in terms of the power output. Typically, commercially available batteries are used, such as standard AA batteries in the case of the MICA2 mote ([11]) or coin cells in the case of the MICA2DOT mote.

Environmental energy is an attractive alternative, in which the environment itself serves as a power source for the sensor device. Various techniques can be used to draw energy from the environment (such as the Prometheus system's use of solar energy, [12]) which can be combined with careful energy management to extend the lifetime of sensor devices to a matter of years, and possibly decades.

2.3.2 Strengths, Weaknesses and Constraints of SensorNets

SensorNet research is gaining in popularity, due to the exciting possibilities it opens up to new application domains. In this section, the strengths of SensorNets, which give rise to these new possibilities, and the weaknesses and constraints will be discussed.

Strengths

- **Low Cost and High Production Volumes:** SensorNet devices are typically composed of microcontrollers, small embedded radios, and simple electronic sensors. These electronic components are usually produced in large volumes, with a corresponding reduction of per-unit cost. Thus, the overall cost of producing such devices is continually dropping. At the same time, improved production techniques are leading to higher yields of better quality components, meaning that devices are becoming increasingly more powerful while using less energy.

  Ideally, SensorNet devices would cost a few cents to produce. RFID tags (discussed in Section 2.5.1, [1]) have already reached this milestone. Other forms of SensorNet devices (such as Smart Dust and Crossbow MICA2 motes) are still comparatively expensive, since the fabrication of these motes is experimental and small-scale. However, large-scale industrial production volumes would reduce the per-unit cost and make their practical use far more feasible.

  The potentially low cost of SensorNet devices is one of their key strengths. It would allow them to be individually expendable, and allow far more of them to be produced for deployment. This would mean that they can be deployed far more densely into an environment, with less regard for the particulars of deployment. The consequence of this is that the SensorNet as a whole is far more robust and fault tolerant, since the SensorNet is far more resilient to individual device failure.

- **Ad-hoc Deployment:** SensorNet devices are usually described as being deployed in ad-hoc topologies. This means that the SensorNet devices cannot rely on fixed neighbourhood sets or other infrastructure to work reliably. Instead, they must use various discovery protocols to cooperatively form into networks. This is useful when deploying them into environments such as natural settings, in which the devices may not necessarily be easy to place in exact locations (such as when dropping them from an airplane).

  This property of ad-hoc topologies and self-organising networks means that SensorNets require far less human intervention than other networks. Instead, the burden of network formation is placed on the devices themselves, making the environment itself appear smarter. This makes SensorNets potentially far easier to maintain.

  Ad-hoc deployment of SensorNets also means that more flexible means of deploying them is possible. It is possible to embed sensors during construction of structures, for example using fiber optic sensors embedded in concrete structure to measure strain ([13]). Wireless sensors can also be deployed *after* construction. Small SensorNet devices can also be dropped by airplane ([5]), or dropped into an environment by other means (such as a missile for military uses).

  Whereas traditional sensor networks (such as satellites) usually reside far from the phenomenon being observed, cheap wireless SensorNet devices can be deployed directly in the environment, close to the phenomenon. This can provide a much richer source of data than previously possible.

- **Fault Tolerance:** As mentioned above, the potentially low cost of SensorNet devices produced in large volumes, and the ad-hoc topologies typically used for communication mean that SensorNets can be equipped with far more redundancy than other networks. Each device can act autonomously, and is expendable as far as the whole network is concerned. This inherent redundancy makes SensorNets potentially far more tolerant of individual node failures.

- **Data Quality:** The redundancy inherent in SensorNets also allows for higher quality data to be gathered. This is possible because individual data items gathered by different nodes can be confirmed by cooperative filtering.

Constraints and Weaknesses

- **Energy Consumption** One of the defining constraints on modern wireless SensorNets is that they are extremely limited in terms of the energy available to them. This is because they are often untethered (ie: not bound directly to a fixed power or communications infrastructure), and are required to either carry a power supply with them, or derive power directly from the environment.

  Batteries are the obvious choice for the power supply, but these are often large compared to the electronic components which they power. Larger, more powerful batteries take up more space generally, and this negates the benefit of the small size of modern wireless SensorNet devices. Replacing batteries is also not always feasible, depending on where the SensorNet devices are deployed (for example they may be embedded inside machinery).

  Environmentally derived power is an attractive alternative, as this would reduce the size of the devices. One example is the Prometheus system ([12]), which used solar panels together with careful power management to extend the lifetime of a mote from an estimated 1 year (under 100% load) to 43 years (under 1% load). The solar panels could produce 40mA at 4.8V when under direct sunlight. Such technology requires far more complex power management facilities, but shows great potential to extend the lifetime of SensorNets.

- **Communication Limitations:** Traditional SensorNets are often equipped with sufficient power to communicate over hundreds of kilometers (such as satellites), or are directly connected to power and communication infrastructure, which enables high speed reliable communication, without much concern for power requirements. However, the limited power available to wireless SensorNets and their small form factor mean that they are usually equipped with simple low-power radios which do not have a range greater than several tens of meters in many cases. Also, wireless SensorNets often have to compete for access to the wireless medium, since they often use unlicensed portions of the spectrum (such as the ISM bands) which is shared with other more powerful devices (such as 802.11 and BlueTooth devices). This means that the ability of small wireless SensorNet devices to communicate reliably is limited to very short ranges.

Additionally, communication in wireless SensorNets usually incurs a very high energy cost compared to the energy cost of operating a microprocessor. Using a radio to transfer 1KB of data over 100m can require as much energy as executing 3 million instructions on a general purpose processor with 100MIPS/W ([14]). Most radios used in embedded systems can operate in several power modes, such as transmit, receive, idle and sleep modes. A power analysis of the Rockwell WINS sensor nodes ([15]) shows that there can be an order of magnitude difference in power consumption between operating a node in full active state (microcontroller in active state and radio in transmit state) and a node operating in low-power mode (microcontroller in sleep state and radio off). The ATmel microprocessor used in the Crossbow MICA2 series of motes draws 8mA when in full active mode (and less than 15μA in sleep mode), while the radio component draws 27mA when transmitting at full power ([11]).

Besides the energy cost and limited range of communication, SensorNet devices also have limited bandwidth. The Crossbow motes support a maximum transfer rate of about 38Kbps ([11]). This places a restriction on what sort of data can be transferred in a SensorNet, and affects how the network protocols are designed.

The network protocols used in SensorNets also need to be energy-aware. Conventional network protocols assume that the receive power is negligible compared to the transmit power. However, in wireless networks this assumption is no longer true. It can even be possible that the receive power exceeds the transmit power over short distances ([15]).

The conclusion is that SensorNets should use local processing as far as possible to reduce the use of radio communication, and thus to save power.

- **Network Topology and Mobility:** Wireless SensorNet devices are usually untethered, meaning that they are not connected directly to any fixed infrastructure. This opens the possibility that the devices could be mobile, whether accidentally or by design. Thus, wireless SensorNet devices need to be equipped with some form of location awareness, so that they can operate effectively, or report on their location. Without this information, the data gathered by a SensorNet device could be useless, since the information may only have meaning in the context of a location.

  Also, the ad-hoc deployment that makes SensorNets so attractive means that there is no way to determine before deployment what the physical location of the devices will

be, nor their eventual network topology. Thus, the devices need to be designed with the ability to determine their location, either individually or cooperatively, and to form into networks on their own.

- **Scale:** The low cost of SensorNet devices means that they can be deployed densely into an environment, and in extremely large numbers, up to tens of thousands. This scale of network, operating on such tiny and simple devices, makes it crucial that network protocols take this potential size of network into account. Techniques include imposing hierarchies onto the network, clustering, and attempting to keep communications localised to portions of the network ([16], [17]).

2.3.3 Applications of SensorNets

The original motivation for the development of SensorNets was military research. In recent times, advances in electronics and communications technology (such as the development of the IEEE 802 and BlueTooth low-power wireless networking standards) have made it possible to develop SensorNets consisting of small, self-contained radio devices, which are cheap to manufacture and deploy. This has opened up a range of possible applications for SensorNets. Some of these applications will be discussed in this section.

Military Applications

Research into SensorNets was driven in the early stages by military research, as a means of surveillance and information gathering. These sensor networks consisted of radar stations, satellites, or acoustic sensors (such as those placed on the ocean floor to detect submarines during the Cold War, [18]). Vehicles such as airplanes are also used to carry sensors such as radar systems, which can monitor large geographic areas. Modern research into SensorNets began with the Defence Advanced Research Projects Agency (DARPA), which wanted to investigate the communications technologies that networks of small, low-cost sensors would require. This project was called the Distributed Sensor Networks project (DSN). The idea of using such low-cost expendable devices appealed to military researchers, who saw an opportunity for deploying such devices into enemy territory for surveillance and tracking. SensorNets may even be interfaced to traditional weapon systems, and provide information to help with targeting. This approach is known as network-centric warfare ([18]).

Security

The most obvious extension of SensorNets to a non-military setting is to infrastructure security, in which key pieces of infrastructure (such as buildings or communications facilities) need to be protected from attack or infiltration. SensorNets can be deployed around key pieces of infrastructure to monitor and detect potential threats. The data from many sensors can be combined to provide better coverage and to reduce false alarms [18]. They could also be used to detect chemical or biological attacks if equipped with appropriate sensors.

Medical Monitoring

Small wireless sensors are also seeing application in medical monitoring, where such sensors can provide far more detailed information about patients. Their small size and wireless communications also make them less of a hindrance to daily life. In [19] and [20], a SensorNet system is described in which patients can be monitored remotely for a fall, enabling doctors to detect problems earlier, especially in the case of elderly or frail patients. SensorNets can also be used in other areas of the medical industry, such as locating doctors and nurses, and monitoring movements of drugs and medical equipment in a hospital.

Home Automation

Home automation is an area of potential SensorNet applications that people will be most be most likely to identify with, as the convenience of a "smart home" is a naturally appealing application of technology.

Wireless sensors can be used to make home and working environments more adaptive to people. In this way, SensorNets allow homes and offices to become smarter, in that they can detect environmental conditions and (when equipped with suitable actuators) can adjust the environment accordingly. For example, air conditioning and ventilation or light levels can be automatically adjusted. This could also be used to make homes more responsive to the needs of the handicapped or elderly. SensorNets can be deployed for security and safety monitoring, such as fire and hazard detection systems.

However, considering the cost and likely applications, together with the (currently) limited market for such systems, it is likely to remain a niche. Simply put, applications of SensorNets to industrial, medical, or military settings currently dominates the literature, since these are areas where SensorNets can provide the greatest return on investment.

Environmental Monitoring

SensorNets are ideally suited to monitoring the natural environment, since they can be deployed over wide geographic areas ([18]), and can form ad-hoc networks. This provides much better coverage of an area than more directed methods would be able to achieve, and can lead to a very high sensor density. This means that information gathered can be correlated to remove noise, and provides higher confidence in gathered information. Examples of SensorNets used in such applications can be found in [21], in which a 65-node SensorNet was deployed over 2 acres in a vineyard. The nodes were arranged in a grid pattern, and used wireless ad-hoc communication to establish connectivity. The sensors measured temperature variations in the vineyard, which provided early warning of potential frost damage and provided useful insights into the temperature variation across the vineyard. They also found that the cost of ownership was less than for a wired network. This information is useful for determining what variety of grape will be best suited to a particular plot of land.

SensorNets can also be deployed in disaster areas such as earthquake zones. Networks of seismographs are already deployed around the world to detect earthquakes. Wireless SensorNets could be deployed after an earthquake in a localised area to detect further quakes, or to detect other vibrations such as those from collapsing buildings, or potential survivors. This would allow rescue workers to direct their efforts far more accurately, and could provide valuable insights into how to manage disasters more effectively when they occur.

Industrial Monitoring and Control

Industrial applications of SensorNet technology aim to improve performance and lower costs, by allowing the early detection of potential problems in machinery (such as vibration sensors or detecting wear and tear on a part, temperatures, etc.), or monitoring the movement of stock, vehicles or other assets. Sensors can be placed in areas which humans have no access to, for example in areas that are hazardous (eg: inside nuclear reactors) or can be embedded in machinery or structures at the time of manufacture in order to monitor their health ([13]).

Wireless sensors are particularly useful in industrial settings ([18]), since sensors can be deployed in an industrial environment some time *after* construction, without incurring the costs and inconvenience of wire-bound solutions. An example of an industrial monitoring application is the electricity grid, which relies on a variety of sensors to monitor the state and

health of the entire network. Other applications are stock tracking and management. Here, recent developments such as RFID tags make it possible for stock to be tracked cheaply and reliably, and sensors deployed around a factory can detect stock movement and react accordingly. The price of RFID tags is on the order of a few cents and are small enough to be embedded into price tags ([1]). RFID tags have almost no processing power, and usually are restricted to only transmitting an identifying code, and possible adjusting an internal counter. Close physical proximity to an RFID reader also impose restrictions on uses. However, RFID tags represent the low end of the SensorNet spectrum. More complex sensors can be used to provide early warning of machine failure, or provide health alerts (eg: air quality, fires etc.).

Traffic Control

SensorNets have also been applied to monitoring the traffic flow on roads. Information gathered with such SensorNets can be used to detect potentially dangerous traffic conditions, or detect congestion. For example, the California Department of Transport uses a SensorNet of about 10,000 sensors to monitor the traffic on Californian highways, and makes this information publicly available ([22], [23]).

In [24], the author outlines a variety of traffic control techniques used around the world. The systems described use sensors built into roads (such as induction coils), CCTV cameras, or overhead radar to monitor traffic flows. This data is used to detect traffic accidents, speed, volume, congestion and weather conditions, and can be used to adjust the speed limits on roads accordingly, thus improving safety and increasing traffic flow. In this area, wireless technologies can play a very important role, as they can be deployed *after* the road is constructed (as opposed to technologies that must be embedded in the road at construction time), and can thus be added to a traffic monitoring system as the need arises.

2.4 Research Challenges

In [18] the authors identify several research challenges facing SensorNet research, including network discovery and routing, collaborative processing, tasking and querying, and security. Further challenges include fault tolerance, data propagation, location or position awareness, and energy constraints. These are discussed in this section.

2.4.1 Ad-hoc Discovery

SensorNets are unplanned networks, meaning that the exact topology of the network is unknown before deployment. For SensorNets to function, the devices need to be aware of their network surrounding (ie: who their neighbours are, routing information) for cooperation to be possible. SensorNets are complicated by the fact that the devices may be mobile, and thus the topology of the network may be changing. This poses a serious challenge to designers of SensorNets, as discovery protocols in such ad-hoc environments take on a far more important role than in traditional tethered networks.

2.4.2 Routing

In traditional wired networks, the topology is usually fixed. This means that node identity is often used as a means of routing information between nodes. However, due to the ad-hoc nature of wireless SensorNets, node identity does not often provide enough information to make routing decisions. Variability in the topology also means that routing information may not be stable, and must be constantly updated in a SensorNet to keep information flowing effectively. Routing in SensorNets is often based on physical location, as this usually makes the most sense in such an environment, or on data-centric principles, such as Directed Diffusion ([25]).

2.4.3 Security

SensorNets are designed to operate largely autonomously using self-organisation. Also, they usually consist of small embedded devices with limited processing and communication facilities. These factors make security of SensorNets is a very tough challenge. Implementing complex security protocols may not be feasible on small microprocessors, and the limited communication bandwidth means that security measures should not add excessively to the size and quantity of messages sent between nodes. Many techniques have been proposed which attempt to provide strong security without incurring a heavy cost on the SensorNet, such as key predistribution ([26]) or simple key management for SensorNets ([27]).

2.5 Commercial Implementations of SensorNet Devices

2.5.1 RFID Tags

Radio frequency ID tags (or RFIDs) exist at the extreme low end of the SensorNet spectrum. RFID tags are usually very simple devices, capable of transmitting a simple value (such as an ID) and possibly executing a few instructions ([1]). They usually have a very small communication range, since they are passively powered by proximity to an RFID reader. However, they are extremely cheap to manufacture (costing a few cents each), and are widely used for applications such as toll systems and stock tracking, where these limitations are not a huge problem, and where the cost per RFID tag makes them a very attractive alternative to more fully featured SensorNets.

2.5.2 Intel Motes, Smart Dust and Crossbow Motes

There are many commercial and experimental implementations of SensorNet devices. Intel produces both experimental motes which use the TinyOS platform, as well as RFID tags and microprocessors.

The DARPA-funded "Smart Dust" project ([7]) aims at producing cubic millimeter sized sensor devices. Such small scales push the limits of microfabrication technologies, and place a very tight bound on energy usage. The circuit fabrication discussed by Pister et al ([7]) is a 0.5- to 0.2-micron process, requiring about 0.2 to 0.3V. Radio communication was rejected for this platform, as it requires to much power and current fabrication technology cannot shrink radios down to below a cubic millimeter. Thus, optical communication was chosen, as the components are far smaller and require far less power than comparable radio circuitry. The latest mote produced by the Smart Dust project occupies a mere 63-mm^3, and uses a hearing aid battery for power (see Figure 2.2).

Crossbow also produces a popular range of motes suitable for experimental and research use ([11]). These motes use the TinyOS platform, and support applications written in nesC. They use an ATmel microprocessor, and support the addition of sensor boards (which are also available from Crossbow). These motes were used in this project as a means of evaluating the TinyTorrent protocol, and are described in more detail in Section 5.1.2.

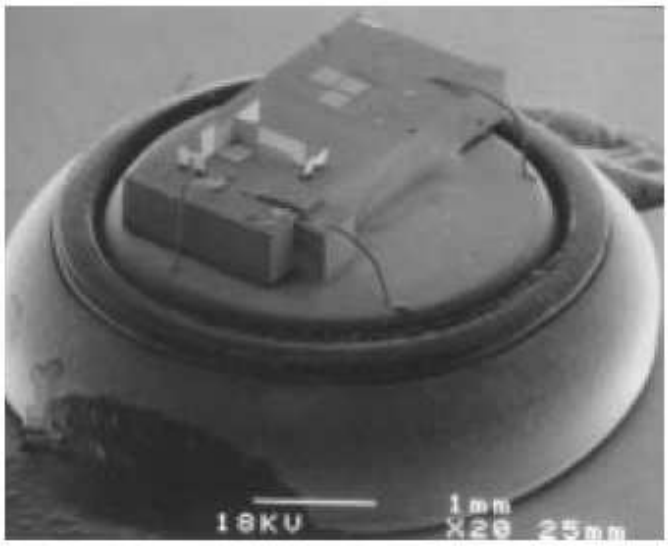

Figure 2.2: Smart Dust mote, occupying a volume of only 63mm^3

Chapter 3

BitTorrent

3.1 Introduction to Peer-to-Peer Networks

The traditional architecture for Internet applications has been dominated by the client-server paradigm (see Figure 3.1). In this architecture, a central server provides a service (such as publishing content) to a variety of clients. The central server acts as the resource manager, and is thus a potential bottleneck in the system.

In recent years, there has been a growing trend towards peer-to-peer (P2P) architectures, which are characterised by moving the resource management to the edge of the network, ie: to the "peers" ([28]). This is achieved by allowing direct communication among peers (see Figure 3.2), which allows them to cooperate amongst each other in managing the resource (which could be bandwidth, data or computing power). This means that the central node is relieved of much of this burden. Ideally, a P2P network has no central nodes at all ([29]). However, many P2P networks employ a small set of special nodes to manage the network, for example Gnutella and Kazaa ([28]).

The P2P architecture spreads the cost of managing the resource across the peers in the network, instead of placing the burden of resource management on a centralised node ([28]). Thus, P2P systems generally scale better than corresponding client-server systems. This is most clearly seen in file-sharing applications ([29]) in which the cost of publishing and transferring data is spread across all the nodes in the P2P network.

Peers in a P2P system operate autonomously, without relying as heavily on a centralised server. This means that such systems exhibit greater reliability and availability, since they

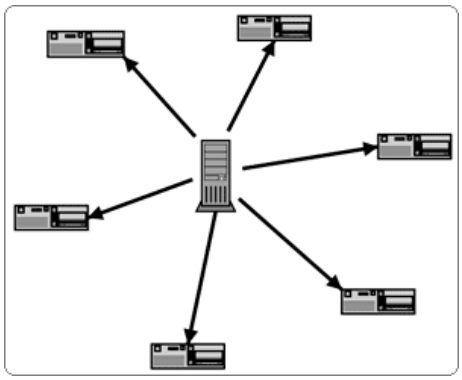

Figure 3.1: Client-Server Architecture

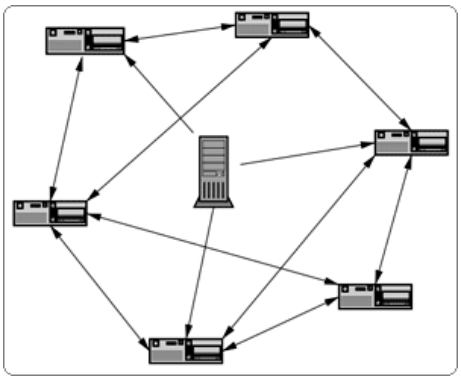

Figure 3.2: Peer-to-Peer Architecture

(Images from http://www.bittorrent.com/introduction.html)

are more resistant to individual node failures ([29]) and (ideally) do not have central points of failure. Peer autonomy also means that peers are often equipped with discovery protocols, which allow them to find functioning peers in the network when neighbouring peers have failed. This ability of "self-organisation" greatly adds to the reliability of P2P systems.

3.2 Introduction to BitTorrent

BitTorrent is a peer-to-peer (P2P) protocol intended for efficient data dissemination on the Internet, originally designed by Bram Cohen ([30]). Its key components are the peers (who cooperate in replicating content across the network) and a tracker (which provides a means for peers to discover each other). BitTorrent allows peers to cooperate directly in transferring data, and ensures fairness among peers using a tit-for-tat algorithm. Content is also broken into pieces, each of which are treated independently. This allows different portions of the same file to be downloaded at the same time, and maximises bandwidth utilisation. The goal of BitTorrent is to replicate the content as quickly as possible among the peers, and thus to reduce the burden on the original publisher of the content.

The design of BitTorrent was prompted by several observations regarding content hosting on the Internet: the "flash crowd" problem, and the "free rider" problem.

3.2.1 The Flash Crowd Problem

"Flash crowds" form when content hosted at a server on the Internet becomes extremely popular, resulting in many simultaneous requests for that content to be sent to the server. This can often overwhelm the server or soak up all the bandwidth available to the server, and can lead to short term unavailability of content. In a sense, the flash-crowd phenomenon punishes popular data by making it unavailable. The "flash crowd" problem is described in further detail in [31] and [32]. This observation is at the core of BitTorrent, which attempts to establish an overlay network of peers who can download portions of the file from each other, reducing the burden of content distribution on the original host by spreading the burden to the peers ([30]).

Figure 3.3 shows how a flash crowd appears from a BitTorrent tracker perspective. These graphs are based on the analysis of a BitTorrent tracker log file over 5 months ([31]). The graph on the right shows the first 5 days only. The flash crowd phenomenon is clearly visible as a sharp spike during this short time span, after which the participants in this download quickly drop off as the popularity of the content wanes. In BitTorrent, the tracker does not actually host the content. However, if it did, then this initial flash-crowd would probably overwhelm the tracker and make the content unavailable.

The success of BitTorrent as a data distribution system is that it attempts to alleviate the "flash crowd" problem by removing the dependency on the original content host and to achieve high availability of the content as quickly as possible. The means by which this is achieved are discussed in Section 3.4.

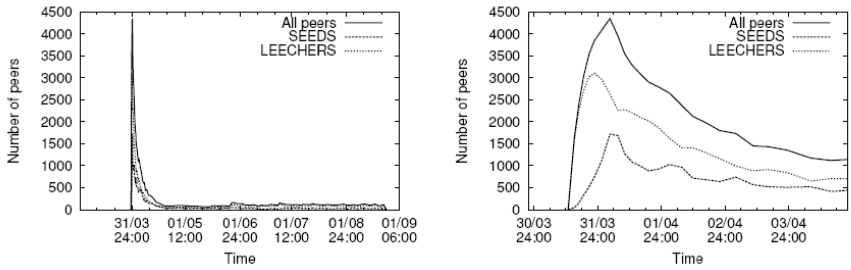

Figure 3.3: Example of the Flash Crowd phenomenon

3.2.2 The Free-rider Problem

Another problem observed in many P2P systems is the "free-rider" problem. Free-riders are peers who contribute no resources in a P2P network. These peers use a large amount of the resource for their own benefit without contributing back to the network. Pouwelse et al ([32]) describe research conducted into this phenomenon, saying that as many as 70% of peers in a P2P network have been observed to be free-riders. Furthermore, in the Gnutella network, the top 1% of hosts satisfy approximately 50% of all requests ([33]). This means that a small proportion of the nodes in such systems are providing most of the benefit, and the remaining nodes are non-contributors, or free-riders. Free-riders can also be understood in economic terms as nodes who fail to take the benefit of others into account when making use of the public good (in this case the P2P system resources). This leads to what is known as the "tragedy of the commons", in which self-interested consumption of resources destroys the public utility of those resources ([28]).

The free rider problem can be reduced by using peer approval schemes (in which peers rate the contributions of other peers to detect free riders) or token schemes (peers earn tokens when they contribute to the network, and spend them when using the network) ([33], [28]). BitTorrent attempts to prevent free-riders using a peer approval scheme, in which fairness is encouraged using a "tit-for-tat" protocol. BitTorrent peers favour those peers which have contributed more bandwidth to the network. This is achieved using the concept of choking, by which peers can temporarily refuse to upload to other peers which appear to be behaving like free riders. In this way, peers are encouraged by their neighbours to behave fairly. The choking mechanism is described in greater detail in Section 3.4.3.

3.3 BitTorrent Concepts and Architecture

The main elements in the BitTorrent architecture are the torrent file, the tracker, the set of peers, and the seed peer (see Figure 3.4). The purpose of each of these is discussed in this section.

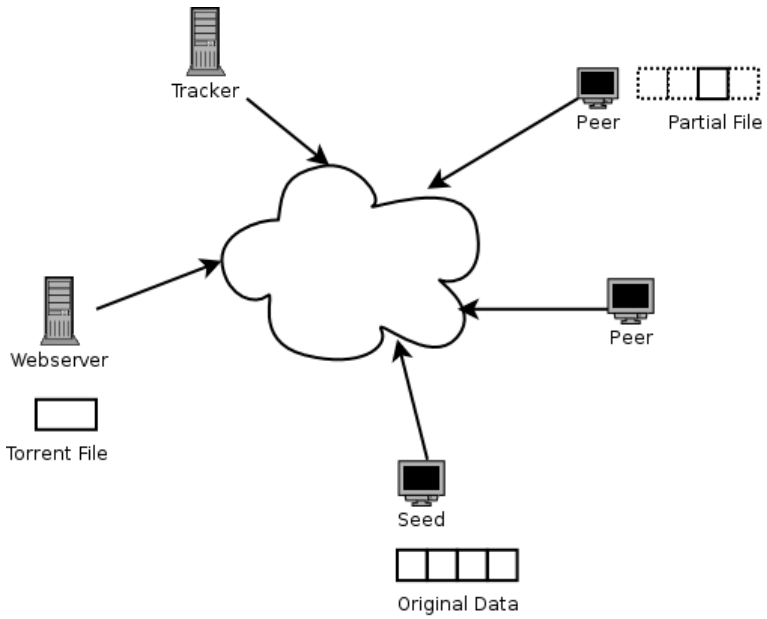

Figure 3.4: BitTorrent Architecture, showing the main components

3.3.1 Peers, Seeders and Leeches

BitTorrent distinguishes peers into two groups: seeders and leechers ([31]). Peers which have a complete copy of the file being distributed are called seeders. Leechers are peers which do not yet have a complete copy of the content. Interactions between peers is described in Section 3.4.3.

3.3.2 Tracker

The tracker is a centralised component which is responsible for keeping track of all peers currently involved in distribution of a file ([32]). It is the means by which peers discover one another, using the tracker protocol, which is layered on top of HTTP (see Section 3.4.2). The tracker is not directly involved in distributing the content ([30], [31]).

3.3.3 Torrent File

BitTorrent does not include a file search mechanism, as other P2P content distribution networks do. Instead, it relies on users acquiring a “torrent” file, which contains metadata about

an item of content published through BitTorrent ([32]). The torrent file is made available by some means outside of the BitTorrent system, for example on a standard HTTP webserver or through email ([30]). Conceptually, the content is broken into equal sized pieces, usually 256kB each ([31], [30], [34]), and each piece is treated separately by the BitTorrent protocol.

The metadata contained in the "torrent" file describes the content, and consists of ([35], [36] [31]):

- the filename and length of the content
- a SHA-1 hash for each piece of the file, used for integrity checking
- the address of the Tracker responsible for managing this content
- additionally, some optional content for various extensions to the protocol

3.3.4 Torrents or Swarms

The collection of peers (including both leechers and seeders) which a tracker is managing, in relation to the content itself, is called a "swarm" or a "torrent". Torrents form when a "torrent" file is published and a seeder is established, together with a tracker to manage the torrent. Peers can join and leave torrents, and the tracker is responsible for tracking the membership status for all peers and allowing peers to discover each other. The formation of swarms is described in Section 3.4.

3.4 The BitTorrent Protocol

The main goal of the BitTorrent protocol is rapid replication of content to the peers. This lowers the effect that flash crowds will have on the original seed, since portions of the content are available more quickly from multiple peers. This means that the content is also less likely to become unavailable due to individual peer failures. A further goal of peer interaction is to encourage fairness among peers, in order to reduce the effect of free riders on the system.

3.4.1 Publishing Data

In order to publish content in BitTorrent, one needs to construct the ".torrent" file. This ".torrent" file is made available outside of the BitTorrent network, typically by hosting it on

a standard web server. The "torrent" file contains details of the file itself (such as the SHA-1 hashes of each piece, the length, and the filename), and also the address of the tracker responsible for managing the torrent. Thus, a tracker is also required, which will track peer membership of the torrent, track peer progress with the content, and provide a means of discovering other peers.

Finally, there must exist at least one peer which has the full content. This peer is the seed for the torrent. The goal is to replicate all the content from this single seed peer as soon as possible, in order to remove the cost of publishing the content from this peer and spreading the cost to all the peers in the network as quickly as possible.

3.4.2 Tracker Protocol

The tracker protocol is used between peers and trackers, and signals the desire of a peer to join a torrent. The tracker keeps track of the peer membership in the torrent. It is the first step in the establishment of the overlay network of peers through which the content distribution takes place.

Once the content has been published (as described in Section 3.4.1), peers can attempt to join the torrent which has been established. The first step in doing so is to locate and download the ".torrent" file [31]. This is usually hosted on a web server, although other delivery means such as email are also possible ([35]).

The ".torrent" file must be passed to the BitTorrent peer software, which extracts the metadata necessary to start downloading the content. In particular, the address of the tracker responsible for the torrent is required. The BitTorrent peer contacts the tracker using slightly modified HTTP GET requests ([30]), in which the content of the messages is encoded using a technique called "bEncoding" ([36]). Once a peer has contacted a tracker, the tracker draws a randomly chosen set of peers from the torrent and passes this back to the newly joining peer as a response. The decision to use random selection of peers is significant, since random graphs inherently display high robustness properties ([30]). This set of peers is known as the peer set.

Once a peer has joined a torrent in this fashion, it will periodically inform the tracker of its status and progress, using the same protocol (HTTP and "bEncoded" messages). This ensures that the tracker's global view of the torrent is kept up-to-date ([31]).

3.4.3 Peer Interaction

Once a peer has joined the torrent and obtained a list of peers, it proceeds to establish TCP connections with its peer set over which all peer to peer interaction will take place.

All peer connections are bi-directional: information can flow in either direction. The peer interaction begins with a handshake process. The handshake messages contain a fixed header, followed by the SHA-1 hash of a portion of the ".torrent" file, which uniquely identifies the content. If both peers do not agree on this hash, they sever the connection ([30], [36]).

Once the handshake is complete, the peers begin exchanging data with each other. They do this by transferring pieces of the file (indexed from zero) with each other.

The BitTorrent peer protocol ensures that several pieces are requested at once, a process called pipelining ([30]). Pieces are further broken down into sub-pieces (usually of 16KB each). This ensures that peers maintain the maximum possible bandwidth utilisation with minimum delays.

Whenever a peer completes a piece, it informs all the peers in its peer set. This is important, as it allows peers to keep track of the status of other peers and thus make better decisions regarding piece selection ([31], described below).

When a peer completes a download, it has the option of re-seeding the content. If it does so, then it informs the tracker that it is a seed, and the tracker updates its status in the torrent accordingly. It is considered good BitTorrent etiquette to do so, as it gives other leechers in the torrent more opportunity to obtain rare pieces.

Two important aspects of the peer protocol are discussed below, namely: piece selection and fairness.

Piece Selection

Cohen ([30]) points out the importance of a good piece selection strategy. If pieces are chosen poorly, the chances of peers already having all pieces currently on offer (or conversely not having pieces which other peers want) rise. Also, if the piece selection strategy is poor, then it could take a long time before the network as a whole contains redundant copies of each piece.

In order to select which piece should be downloaded next, peers use a *rarest first* strategy. This means that peers choose pieces which the fewest of their own peers have already. In order for this to function, peers must inform each other when they obtain a new piece ([30]).

The *rarest first* approach makes sure that peers always have pieces which their peers are likely to want (meaning that peers always have something to offer to the network), as well as ensuring that the pieces are rapidly duplicated across the network in such a way that it becomes increasingly unlikely that pieces will become unavailable due to node failure. The *rarest first* approach also means that the burden on the initial seed is reduced far more quickly, since redundant downloads of popular pieces would impair its ability to get the complete file into the BitTorrent network quickly ([30]).

As noted in Section 3.4.3, pieces are broken down further into sub-pieces for pipelining. When a sub-piece of a piece has been fetched, then the remaining sub-pieces of that piece are prioritised ahead of other pieces or sub-pieces. In this way, complete pieces are constructed as quickly as possible.

When a peer first joins a torrent, it gets the first piece using a random selection. Thereafter it reverts to the rarest first strategy ([31], [30]). This makes sure that the algorithm initially gets a few pieces distributed quickly at a time when few pieces have been replicated to other node, since the *rarest first* strategy might not make good piece selections at this time.

Fairness: Choking and Un-choking

In BitTorrent, peers essentially barter with each other for pieces of the content ([32]). Each peer-to-peer connection in BitTorrent is associated with two pieces of state at each end: choked or un-choked, and interested or not ([36]). The interested status indicates if the peer at the other end of the connection has a piece that this peer still needs.

Peers decide to whom they should upload using a tit-for-tat algorithm. They do this using "choking", which is a temporary refusal to upload to a peer ([30]). A peer can choke another peer, meaning it won't upload to that peer, although it will still download from that peer if it can. Once a connection is un-choked, then uploading can take place again. In this way peers reciprocate by uploading to peers which upload to them, thus encouraging fairness.

Data transfer between two peers will only occur if the sending end of the connection is not choked, and if the receiving end is interested. Maintaining the choking state is the key to BitTorrent achieving its goal of fairness, as it allows peers to control each other and enforce a fair participation in the network.

Chapter 4

Research Goals

The goals of this research project are to investigate BitTorrent and to attempt to apply the techniques which make it successful in content distribution on an Internet scale to a wireless SensorNet. A further goal is to expose the data in a SensorNet to the outside world through a BitTorrent interface. Finally, the properties of this implementation will be evaluated experimentally, in order to determine the merit of these techniques and algorithms in a SensorNet environment. Properties which are of particular importance to SensorNets (such as energy consumption and protocol overhead) will be the main focus of the evaluation. These goals are discussed now.

4.1 BitTorrent Concepts Applied to SensorNets

BitTorrent has shown itself to be extremely successful on an Internet scale as a means of efficiently managing the distribution of (typically large) content. It does so using a variety of techniques, aimed at rapidly replicating data across a network, and spreading the cost of replication to the peers.

The techniques which achieve this are the breaking of files into pieces (allowing multiple parallel downloads), the piece selection strategy (rarest piece is chosen for download first), and the cooperation between peers (which reduces the reliance on a centralised node).

These techniques could prove useful in energy- and bandwidth-constrained environments such as SensorNets, where the peers need to work cooperatively to ensure the maximum possible useful lifetime of the network. By spreading the energy cost of data distribution

across the network, it may be possible to extend the life of a SensorNet. Bandwidth is also highly constrained in SensorNets, and the techniques that make BitTorrent successful at transferring large amounts of content on an Internet scale may be useful in transferring data at a SensorNet scale.

BitTorrent also achieves rapid redundancy of data, thus ensuring that content does not become unavailable as its popularity grows. One of the main strengths of SensorNets is that the sensor devices are cheap and thus individually expendable. However, this means that data stored at only one device could be lost if that device fails. Thus, the rapid replication of content in the BitTorrent network would be useful for SensorNets, where individual node failure must be taken into account. Efficient replication of content would thus help make SensorNets more robust.

In this project, some of these techniques were implemented for the TinyOS platform using nesC. The implementation was then evaluated using a combination of TOSSIM and PowerTOSSIM simulators, as well as by deploying the application onto real MICA2 motes from Crossbow. The experimental evaluation is presented in Chapter 6 and the development environment is described in greater detail in Section 5.1.

4.2 A BitTorrent Interface to a SensorNet

SensorNets cannot operate in isolation, but must be connected by some means to an external system, where data can be analysed, or from which the SensorNet can be manipulated. They can be thought of as data storage networks, which store the data gathered by each of the nodes which makes up the network. However, the question that arises is how to design the interface between the SensorNet and the external systems. Dunkels et al ([37]) discusses three principal designs for building an interface between a TCP/IP network and a SensorNet :

- Proxy design, in which a gateway is deployed between the SensorNet and the external system, through which all communication takes place.
- Delay Tolerant Networking, which is a communication model used for networks which have high bit error rates or long, unpredictable delays (such as satellites).

- Direct implementation of TCP/IP on SensorNet devices, which is currently not believed to be feasible.

For this project, the proxy design was chosen. The proxy was built on top of the Azureus BitTorrent client, which is a well-known BitTorrent client written entirely in Java. It supports the development of custom plugins, and also has a built-in Tracker component. These features make it an ideal platform for designing a SensorNet gateway, allowing external entities to access the information stored in a SensorNet over the standard BitTorrent protocol.

4.3 Experimental Evaluation

The experimental evaluation aims to determine to what extent the BitTorrent algorithms and techniques (as implemented) make sense in a SensorNet environment. The evaluation used a combination of simulations and real-world experiments to determine such parameters as protocol overhead, energy consumption and throughput. These are presented in Section 6.

Chapter 5

System Design and Implementation

5.1 Development Environment

The development environment used during this research consists of several components. Development was done using the nesC language (Section 5.1.1), which is designed for embedded network development. The experimental evaluation used a combination of simulations (using TOSSIM and PowerTOSSIM, described in Section 5.1.3), as well as deployment of compiled nesC applications on Crossbow MICA2 motes (described in Section 5.1.2). The BitTorrent interface to the SensorNet was built on top of the Azureus BitTorrent client (described in Section 5.1.4). These components are described in the following section.

5.1.1 The nesC Language

The nesC language is designed specifically for embedded network development, such as SensorNet development. The design of nesC recognises the particular demands which this form of development places on the language, and thus provides language constructs to support a simple concurrency model, event driven design, and modular composition of software components to form applications. The design of nesC is extensively described by Gay et al ([38]).

The nesC language is an extension of the C language. This was chosen because it is possible to generate efficient code for microprocessors with C code, and because most developers are already familiar with the C language.

Concurrency in nesC is based on events, which can be signalled from a component or

directly by a hardware interrupt. Events can preempt each other, and can execute simultaneously. Thus, it is important for nesC to deal with potential data race conditions which may result from concurrent access to shared data by two different events. The approach nesC takes is to perform a static analysis of the program code to classify code blocks into one of two categories:

- synchronous (meaning code which is reachable only from tasks or other synchronous blocks)
- asynchronous (meaning code which is reachable from an event)

Any access to shared variables within asynchronous blocks is flagged by the compiler as a potential problem, which the programmer is required to address manually. This is done by wrapping code with the `atomic` keyword, which indicates that the block may not be preempted by any other execution thread. This static analysis of code for safety and concurrency issues is made possible by the typically small size of nesC applications, and the lack of function pointers or dynamic memory allocation.

Applications in nesC are composed of components. These components are defined by their interfaces, which describe the commands they support, and the events which they can signal to other components. Applications consist of components which are wired together to achieve the desired functionality. Since TinyOS was written in nesC, this is exactly the same application model supported by TinyOS.

5.1.2 Crossbow Motes

The motes used in this research project are the "MICA2 Motes" produced by Crossbow (http://www.xbow.com). Crossbow produce a range of motes suitable for SensorNet applications and for experimental and proof-of-concept development. The latest in this series are the MICA2 and the MICA2DOT motes, which differ mainly in the form factor: the MICA2DOT mote is offered in a coin sized form, while the MICA2 is about as large as the two AA batteries required to power it (see Figure 5.1).

Processor

The processor used in the MICA2 motes is the Atmel ATmega 128L low power microcontroller. During active processing, the current draw is quoted as 8mA ([11]), although it

Figure 5.1: A Crossbow MICA2 mote and sensor board
(image from http://www.sequoia.co.uk/wireless/index.php)

supports a low-power sleep mode with a current draw of less than 15μA. In sleep mode, the MICA2 motes are expected to have a battery life of over 1 year using 2 AA batteries. However, active processor usage will shorten this life time considerably. The microprocessor runs applications from an internal flash program memory, which has a capacity of 128Kbytes. Additionally, the motes are equipped with a further 512 Kbytes of non-volatile Flash storage which can be used to store sensor data, or other information ([11]). The motes also have 4 KBytes of random access memory, used during program execution.

Sensor Input

The MICA2 also has a 51 pin expansion connector, which can be used to attach sensor boards to the mote. Crossbow supply a variety of sensor boards for use with the MICA2 motes. The data from the sensors is made available to the TinyOS application via a 10 bit digital-analogue converter ([11]). The sensor board which comes with the Crossbow development kit has a light sensor, temperature sensor, microphone and speaker. See Figure 5.1.

Communications

The radio built into the MICA2 mote comes in several different frequencies, namely 315 MHz, 433 MHz and 868/916MHz, which are all part of the Industrial Scientific and Medical (or ISM) bands ([39]). The radio has a maximum advertised data rate of 38.4Kbps over a

maximum range of about 160m, while drawing 27mA. For practical use, the range is usually far less, and generally within several meters to tens of meters.

The radio stack uses carrier sensing to control access to the wireless medium (CSMA). However, it does not enforce any collision detection, except that it discards messages which appear to be corrupt. There is no automatic retransmission of corrupt messages. When a MICA2 mote wishes to send data over the radio, it first checks to see if any other traffic is already being broadcast by another mote. If no traffic is detected, then the mote broadcasts the message. If, however, it detects that the medium is in use, then it backs off for a random period before attempting again.

Operating System: TinyOS

The Crossbow motes run an operating system called "TinyOS", which was developed at the University of California at Berkeley ([40], [41]). TinyOS was designed for small low-power devices and is well suited to SensorNet applications, and is written in nesC. It thus provides the same simple architecture in which applications are composed of a graph of components, each of which is an independent computational entity ([42]) and which provides commands, event handlers and tasks.

Commands and events are used for inter-component communication. Commands are used to make calls to lower level components, and event handlers are used to signal information to higher level layers. Concurrency is handled using split-phase semantics, in which the request for a service (the command) is separated from the completion signal (an event).

Tasks are units of deferred computation, and represent the primary unit of work in TinyOS. They are atomic with respect to other tasks, and always run to completion, although they can be interrupted by events. Tasks are posted to a simple FIFO scheduler, which executes the tasks one at a time, and puts the processor into a sleep state when the queue is empty. The core of TinyOS occupies a very small memory footprint of about 400 bytes ([41], [38])

Hardware resources are also abstracted as components in TinyOS. These components exist on the lowest layer of the component graph, and have event handlers hooked to the low level hardware interrupts.

5.1.3 TOSSIM and PowerTOSSIM

TOSSIM is a simulator for running TinyOS applications on a PC, instead of on real motes. It allows developers to test applications on a simulated SensorNet mote, without having to install the application on a real mote. This makes development for the TinyOS platform far easier, and gives the developer far more insight into the operation of the application through various debugging and logging facilities. TOSSIM also supports thousands of motes in a single simulation, which makes it possible to test applications at much larger scales than would be possible with real motes. TOSSIM is described in detail in the TOSSIM Manual ([43]) and by Levis et al ([42]).

TOSSIM uses the component graph of nesC applications to produce compiled TOSSIM applications which can run on a PC. Such an application replaces the TinyOS hardware components with TOSSIM-specific components which simulate the underlying hardware. This allows TOSSIM to simulate network communications at bit level. TOSSIM also supports the use of radio models, which can simulate bit level errors in the radio stack. However, these models are simply approximations of real-world behaviour, and TOSSIM does not claim to produce authoritative results, especially since the accuracy of the results depend on the accuracy of the error model used.

TOSSIM simulations model hardware interrupts as discrete events. A consequence of these discrete events is that TOSSIM does not model preemption of events, which means that motes in the simulation will not exhibit concurrency problems which might cause a real mote to fail.

TOSSIM also allows external applications to inject messages into a simulation, or to read the messages being sent in the network. This is achieved by connecting to the simulation over a TCP socket. A tool called MIG is used to generate Java classes from the C header files used to define the message types for an application. These Java classes can then be instantiated in a Java application and injected into a TOSSIM simulation.

TOSSIM supports debug logging, which allows the developer to embed debug statements directly in the nesC code. This logging output can be filtered to include only areas of interest to the developer (for example only showing radio activity), and can be captured for later analysis. This allows the developer to gain an understanding of what the application is doing at a much deeper level than deployment on a real mote would allow.

One major drawback of TOSSIM is that it does not model power or energy consumption.

This is crucial in SensorNet research, as energy consumption is usually highly constrained in SensorNets and thus of particular interest to researchers. For this reason, PowerTOSSIM was developed ([44], [45]). PowerTOSSIM is an extension to TOSSIM which models transitions of energy states in the simulated hardware components. This information is then used, together with an appropriate power model of the mote hardware, to generate statistics of the energy consumed by each hardware component in each mote.

5.1.4 Azureus BitTorrent Software

Azureus is a BitTorrent client written entirely in Java and is available from the Azureus website for free ([46]). Although targeted mainly at end-users wanting to download content from the BitTorrent network, it has several features that make it ideal as a development platform for experimenting with BitTorrent:

- **Built-in Tracker:** The built-in tracker allows Azureus users to download as well as host content directly from within a single interface.
- **Support for Plugins:** Azureus' plugin architecture allows developers to write their own extensions to the basic Azureus application. Plugins are written in Java, like Azureus itself, and can interact with the Azureus client through a series of Java classes and interfaces designed for this task ([46]).
- **Web Plugin:** One of the core plugins for Azureus is the Web Plugin. This plugin adds a simple web server to Azureus, accessible from any web browser, which displays a list of files currently being hosted by the built-in tracker.

These features are useful for building additional functionality into the Azureus client, such as a plugin to interface with an external system. In this project, this facility was used to write a plugin to connect Azureus to a SensorNet, and to publish hosted content on the Web Plugin in order to expose the SensorNet data to external clients. The details of how this was achieved are discussed in Section **??**.

5.2 Design Decisions

Attempting to apply BitTorrent concepts to a SensorNet scale involves overcoming several important issues. These usually involve the limitations inherent in SensorNets (as discussed

in Section 2.3.2), and how to overcome them. In this section, the various important design decisions which had to be made during the development of TinyTorrent are discussed, and justifications for these decisions are made.

5.2.1 Persistent Connections

SensorNet motes have very limited processing power and memory. This means it is not feasible for motes to maintain significant state regarding other motes, such as the state required to maintain a persistent connection. Although it is possible to run stateful connection-oriented protocol such as TCP on SensorNet devices, there are significant problems which make such an approach infeasible, such as the large protocol overheads required, tolerance of high bit error levels, and the energy cost of maintaining persistent connections ([37]).

TinyTorrent was thus designed with a connectionless protocol (as far as possible), in which individual message and node failures are assumed to occur, but are not directly detected. Nodes are not required to maintain state about connections to other nodes (provided this does not relate to their own participation in the network), nor about the health of other nodes.

5.2.2 Number of Messages

In Section 2.3.2, the power constraints under which SensorNets must operate were described. Every bit transferred across the network in a SensorNet costs energy, and thus the number and size of messages should be kept to a minimum. TinyTorrent thus abandons the periodic state updates (both to the tracker and to other peers) which BitTorrent employs to keep participants in the network informed of key information. This state information consists of peers keeping other peers informed about which pieces of a file they have, so that all peers can make accurate "rarest first" piece selection decisions. This information was replaced in TinyTorrent with a bit vector (represented as a 16 bit integer), in which each bit represents a boolean flag indicating whether or not a given peer has the indexed piece. This is a compact way to represent the progress of a peer regarding a file, and can easily be appended to any other peer to peer message, which then has the side effect of allowing peers to keep themselves updated. Also, 16 bits is compact enough that it does not add significantly to the protocol overhead, while removing the need for dedicated update messages. Thus, this approach achieves a reduction in the number of messages required for the protocol to work.

5.2.3 Message Size

TinyOS uses a style of messaging called "active messages" ([47]). This is a message paradigm in which messages contain the identifier of an application-level handler which will be responsible for dealing with the message. This scheme reduces the buffering difficulties that other message scheme have, and is ideally suited to embedded applications such as SensorNets. However, the messages supported by TinyOS only allow a 29 byte user-definable payload. This limitation has a big impact on the design of any protocol running on TinyOS. All messages must be kept within this limit. This led to a number of design decisions.

Firstly, the largest unit of transfer for content pieces was chosen to be 16 bytes. The structure of the piece message (described in Section 5.3.3) allows for more than 16 bytes to be sent at once without the total payload exceeding the 29 byte limit. However, 16 bytes is also the size of a Flash memory page on the MICA2 hardware platform (which was used for this project). Since it is envisaged that the mote will store data (from sensor readings) to this Flash memory before sending it across the network, or store data in the Flash memory temporarily while acquiring data from other motes, it seemed clear that using the same sized chunks of data would be the simplest to implement. However, it is possible to handle pieces of more than 16 bytes, and to implement a more complex storage scheme to map the data to the underlying Flash memory correctly.

In its current implementation, TinyTorrent allows nodes to publish data up to 255 bytes in length. This limitation is a consequence of the 16 byte piece size (described above), and the use of a 16 bit vector to record the availability of pieces (described in Section 5.2.2). Thus the maximum length for a TinyTorrent file is 256 bytes. However, the size was restricted to 255 bytes, so that the file length could be represented by a single unsigned byte.

5.2.4 Data Naming

In SensorNet applications, the node address at which data originated is often not as important as an indication of the type, severity or physical location at which data was recorded. It is thus often preferable to identify data using metadata which describes the attributes of the data (such as type, priority, or location). This concept of naming data using name-attribute pairs is called "data-centric storage" ([25]). The use of metadata to describe data is an important part of the SPIN protocol ([48]), in which data is exchanged between nodes in a SensorNet using a negotiation protocol. The SPIN protocol leaves the exact format of the data name

to the application, rather than specifying it as part of SPIN itself. This allows the naming to be as flexible as possible for a given application. Issues identified are the uniqueness of data names, and the ability to distinguish different pieces of data by the data name. These are issues which must be addressed by the application. In some ways, BitTorrent uses a similar concept in the form of the torrent file, which is simply metadata describing the content.

TinyTorrent makes use of this metadata concept by associating an application-defined name to each piece of data published in TinyTorrent. This data name needs to be small in size, due to the limitations of the allowable message size of 29 bytes, yet be large enough to allow the application adequate expressive power. The node ID for TinyOS motes consists of 2 bytes. It may be useful for an application to use the node ID as part of the data name, and thus the TinyTorrent data name should be at least 2 bytes. The final choice of 4 bytes for the data name is believed to be short enough that it does not add significantly to the protocol overhead, yet still gives the application enough expressive power in the metadata for most purposes.

5.2.5 Integrity Checking

A key part of BitTorrent is the breaking up of a file into pieces, which are treated independently of each other. In BitTorrent, integrity checking of pieces is done using the SHA-1 hash of each piece. SHA-1 produces a 20 byte hash, which is not feasible, considering the 29 byte limit on payload length. Thus, an alternative integrity checking mechanism had to be found. CRC checksums were considered, but are considered to be to complex for pure software implementations (especially in embedded systems). Thus the Adler-32 checksum algorithm, a variation of the Fletcher checksum algorithm ([49], [50]), was chosen. This algorithm can be implemented in software more efficiently, and relies on addition and modular arithmetic, making it ideally suited to embedded systems. Although the Adler-32 checksum shows some weaknesses when operating on short messages, it is able to detect certain bit errors which the Fletcher algorithm cannot detect.

The Adler-32 algorithm produces 32 bit checksums. This is still to large to fit more than 7 such checksums into a single 29 byte message. Since the piece bit vector is 16 bits, this implies that the checksum must be small enough so that 16 checksums can fit into a single 29 byte message. The simplest way to achieve this was to sacrifice some of the error detection properties of the full 32 bit Adler-32 checksum by compressing the 32 bits into 8 bits. This

was achieved using a simple exclusive OR operation on each of the 4 bytes making up the 32 bit checksum. The resulting 8 bits were used as a simple and concise checksum.

5.3 TinyTorrent Implementation

5.3.1 Overview of TinyTorrent

TinyTorrent is the protocol by which motes in a SensorNet can publish data to a tracker mote, and download data cooperatively from each other using similar principles to BitTorrent. Motes are expected to gather information (or generate information based on sensor observations) over a period of time. This data may be processed by motes, either individually or cooperatively, and it may be necessary to make this information available to systems outside the SensorNet. TinyTorrent provides a mechanism whereby motes can achieve this functionality.

In TinyTorrent, the flow of information starts with motes, which gather sensor input, process it locally, and then assign an application-defined name to the data. This named data can then be published to the BitTorrent tracker via the base mote (see Figure 5.2). The SensorNet Plugin records this newly published SensorNet data, and makes it visible to external clients via the Azureus built-in tracker and the Web Plugin (see Section 5.1.4). The data can then be retrieved by other motes (using TinyTorrent within the SensorNet) or by external BitTorrent clients using the standard BitTorrent protocol. The design of the Azureus SensorNet plugin is an example of the proxy design outlined by Dunkels et al ([37]).

5.3.2 Publishing Data to Tracker

The exchange of messages in the TinyTorrent protocol begins with a mote attempting to publish data to the tracker via the Base Mote (see Fig 5.2). To do this, the mote constructs a `PublishDataMessage`, which is defined as follows:

```
typedef struct PublishDataMessage {
    uint16_t fromid;
    uint8_t key[KEY_SIZE];
    uint8_t length;
    uint8_t checksum[MAX_PIECES_PER_FILE];
} PublishDataMessage;
```

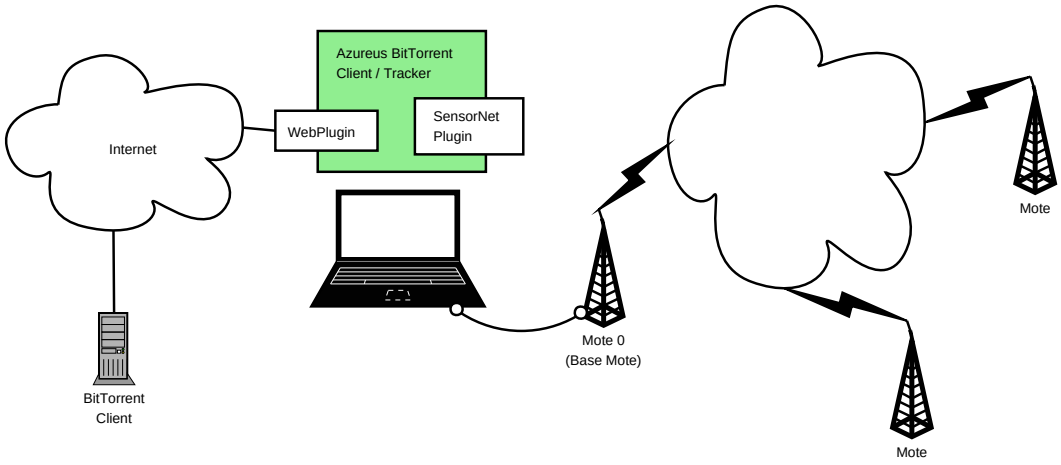

Figure 5.2: Architecture of TinyTorrent system

The message contains the identity of the originating node, the key (or data name, defined by the application), the length of the content, and an array of modified Adler-32 checksums (one for each 16 byte piece of the content). This message is sent by radio to the Base Mote, which is always mote 0.

When the Base Mote receives this message, it passes it on to the SensorNet Plugin, which records the metadata and publishes this to the Web Plugin. The content which is externally visible at this point is merely a "trigger" file, whose name ends with the suffix "seed". This acts as a trigger for the SensorNet Plugin, as described in Section 5.3.4.

This process of publishing data is very similar to the BitTorrent process, described in Section 3.4.1.

5.3.3 Downloading Data by Motes

Once the data is published to the tracker, it is possible for other motes to download the data. To do this, the mote must know the data name. How motes determine the data names is outside the scope of the TinyTorrent protocol, since data naming is left entirely to the application level. The mote must construct a `PeerListRequestMessage`, which is defined as follows:

```
typedef struct PeerListRequestMessage {
    uint16_t fromid;
```

```
    uint8_t key[KEY_SIZE];
    uint16_t pieceBitVector;
} PeerListRequestMessage;
```

This message contains the node identity (so that responses can be routed back to this node), the unique data name, and the current state of its piece bit vector for this content. This is required, since this message may be sent to the tracker at any point after starting the download, for example if the node runs out of peers and needs to request a new peer set. The tracker then needs the piece bit vector so that it can chose a peer set which contains the missing pieces that the node still requires.

In response to the `PeerListRequestMessage`, the SensorNet Plugin determines a suitable peer set, and constructs a `PeerListMessage` to send back to the node. This message is defined as follows:

```
typedef struct PeerListMessage {
    uint16_t fromid;
    uint8_t key[KEY_SIZE];
    uint8_t numPeers;
    uint16_t peerList[MAX_PEERS];
} PeerListMessage;
```

The maximum number of peers supported currently is four, although this number can be increased provided the 29 byte TinyOS message limit is taken into account.

Once the node has a peer set, it begins a series of handshakes with each member of the peer set. The handshake messages contain the following information:

```
typedef struct HandshakeMessage {
    uint16_t fromid;
    uint8_t key[KEY_SIZE];
    uint16_t pieceBitVector;
    bool initiator;
} HandshakeMessage;
```

Once again, the node identity and the data name are included so that nodes can verify that this is not a spurious message. Handshakes are also used to exchange the piece bit vectors, so that nodes can maintain an accurate view of the progress of their peers, and thus make better "rarest first" piece selection decisions. The `initiator` variable determines if this message is an initiated handshake, or a response to a previous handshake, and is required to prevent the nodes from entering into an infinite handshake loop.

Once handshakes are complete, the node will start to request pieces of the file which it is missing. To do this, it determines the rarest piece using the bit vectors which it acquired from its peers during the handshake phase. This is the same approach used by BitTorrent (see Section 3.4.3). The rarest first approach here has a similar effect as it does in BitTorrent: the pieces of data are replicated in the network as quickly as possible. This rapidly spreads the cost of publication across the network, and ensures that redundant copies of all pieces are produced as quickly as possible, thus improving data availability. This phase of interaction consist of a series of `RequestPieceMessage` and `PieceMessage`, which are defined as follows:

```
typedef struct RequestPieceMessage {
    uint16_t fromid;
    uint8_t key[KEY_SIZE];
    uint8_t pieceIdx;
} RequestPieceMessage;

PieceMessage {
    uint8_t key[KEY_SIZE];
    uint8_t pieceIdx;
    uint8_t piece[16];
} PieceMessage;
```

An important point to note is that the node identity is not included in the PieceMessage. This is because the source of a particular piece is not important, only that it passes the integrity check. Since the checksums are acquired from the tracker directly and not from one of the motes, there is little chance that a malicious mote can inject false messages into the TinyTorrent network, since these would be discovered when the checksum is applied. If a piece fails the checksum check, then it is discarded.

If the mote finds that it is missing pieces, and that none of its peers seem to have those missing pieces, then the mote contacts the tracker again, and requests a new set of peers using the `PeerListRequestMessage`. The algorithm then repeats until the mote has all the pieces, and can verify them all.

5.3.4 Downloading Data Externally

When data is published by a mote to the Base Mote, the metadata is passed to the SensorNet Plugin, which is responsible for publishing the metadata to the Web Plugin in Azureus.

The SensorNet Plugin enables the SensorNet data to be accessible to systems outside the SensorNet itself. The SensorNet Plugin acts as a proxy ([37]) between the TinyTorrent protocol used in the SensorNet and the BitTorrent protocol used beyond the SensorNet.

To do this, the SensorNet Plugin first constructs a "seed" file, containing the metadata describing the published data. The filename used for this file is the data name followed by the suffix "seed", for example "34-55-12-5-seed" (assuming the data name consist of the 4 byte values indicated). This file is then published as a ".torrent" file to the built-in Azureus tracker using the Azureus API defined for this purpose. Once this file is published to the tracker, it is visible via the Web Plugin to any client browser. At this point, the data itself still resides on the mote: only the metadata is published at this point.

When an external client wishes to download published data from the SensorNet, it must first download the corresponding "seed" torrent file. The action of requesting this file is captured in the Web Plugin, and used to trigger the SensorNet Plugin to begin downloading the actual SensorNet data. This is done using the TinyTorrent protocol between the Base Mote and the motes which hold the data, as described in Section 5.3.3.

Once the SensorNet Plugin has downloaded the actual data, it saves it in a file named similarly to the "seed" file, except with the suffix "data". This file is also published to the Azureus built-in tracker as a ".torrent" file using the Azureus API, which again makes it visible via the Web Plugin.

External clients can then download this "data" torrent file, which can be used by any BitTorrent client to download the actual content from the Azureus tracker.

In this way, the data in a SensorNet can be published to the Azureus tracker, made visible to external entities and is accessible from outside the SensorNet.

Chapter 6

Experiments and Results

6.1 TinyTorrent Protocol Overhead

6.1.1 Goals

This experiment aims to determine the overhead of the TinyTorrent protocol. This was done by simulating 10 motes using the TOSSIM simulator, and running a simple application using TinyTorrent, in order to compare the total data transferred with the total size of the payloads.

6.1.2 Method

The TinyOS application used in this simulation performed the following steps after booting up the mote:

- Generate a fixed size file
- Assign a name to the content. The name was simply the first byte of the 16 bit node ID, followed by the sequence "1,2,3".
- Publish the data to the SensorNet Plugin via mote 0 (the Base mote)

The SensorNet Plugin was configured to immediately download any published content, instead of waiting for the Web Plugin trigger (see Section 5.3.4). By combining the logs generated by the TOSSIM simulator and the Azureus application log, it was possible to determine the number and types of all messages sent, and thus to determine the total number

of bytes transferred. It was also possible to determine the total bytes of application payload transferred.

This experiment was repeated for different file sizes from 16 bytes to 255 bytes (in steps of 64 bytes), and the ratio of total bytes to payload bytes was determined.

6.1.3 Results

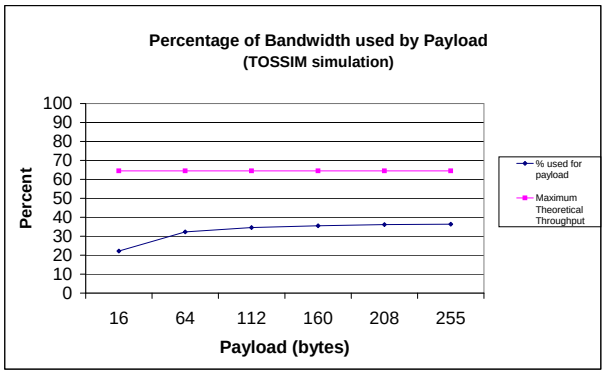

Figure 6.1: Results of TinyTorrent Protocol Overhead Simulation

Figure 6.1 shows the percentage of the total payload as a proportion of the total bandwidth used by TinyTorrent, and how this varies with the individual payloads used by the motes. For 16 byte payloads, only 22% of the total bytes sent represent that actual data. As the payload sizes increase, the proportion of the protocol devoted to transferring this payload increases as well, and appears to reach an asymptote slightly above 36

Also included in the figure is an indication of the maximum possible throughput that can be achieved in TinyOS for an ideal request/response protocol. An ideal request/response protocol is one in which data transfer involves a request for data, followed by a response containing the data (which is how TinyTorrent and BitTorrent work). The lowest possible overhead for such an ideal protocol would require that the request message is the smallest possible size and that the response message contains the largest possible payload. In TinyOS, the structure of active messages is given below (taken from tos/types/AM.h, [51]):

```
typedef struct TOS_Msg
{
/* The following fields are transmitted/received on the radio. */
```

```
    uint16_t addr;
    uint8_t type;
    uint8_t group;
    uint8_t length;
    int8_t data[TOSH_DATA_LENGTH];
    uint16_t crc;
    uint16_t strength;
    uint8_t ack;
    uint16_t time;
    uint8_t sendSecurityMode;
    uint8_t receiveSecurityMode;
}
TOS_Msg;
```

From this it can be seen that TinyOS Active Messages (described in Section 5.2.3) allow a maximum usable payload of 29 bytes per message (since `TOSH_DATA_LENGTH` is defined as 29). Additionally, these messages incur a further 7 byte overhead consisting of the fields: `addr`, `type`, `group`, `length` and `crc`. These fields are used internally by the TinyOS radio stack, and cannot be removed from the message. The other fields in the `TOS_Msg struct` are not sent over the radio. Thus, the minimum possible request message that can be constructed with TinyOS active messages is one in which the payload contains only the identity of the node requesting the data, which takes up 2 bytes (16 bits). Together with the 7 byte overhead that TinyOS incurs, the length of the smallest request message in TinyOS is 9 bytes.

The response with the greatest possible payload would be one which transfers a full 29 bytes of application data. Together with the 7 byte overhead, this would amount to 36 bytes in total.

Thus, the total number of bytes sent in a single request/response interaction is 45 bytes, of which 29 bytes are the application payload, the rest being protocol overhead. This means that at most 64.4% of all bytes (ie: 29 out of every 45 bytes) transferred in TinyOS can contain application payloads. This is shown in Figure 6.1 as the Maximum Theoretical Throughput.

This experiment shows that extreme care must be taken when designing a protocol for SensorNets to avoid unnecessary overhead. With such small message sizes (eg: 29 bytes in TinyOS) it is hard to develop low-overhead protocols, or conversely to develop information-rich representations of data. This is made more difficult by the 7 byte overhead incurred by every TinyOS message. This analysis also shows that there may be some benefit to be gained

by allowing more than 16 bytes to be transferred in each `PieceMessage`, despite the fact that this may incur additional processing overhead at each mote due to the more complex data manipulation (bearing in mind that the Flash memory uses 16 byte pages). Another possibility is to examine the use of compression techniques to improve the throughput, and to lower the ratio of total bytes to payload bytes. The additional processing load on the motes of using such techniques should be investigated, especially in terms of the implications for power consumption. Since transmitting data over the radio has a higher energy cost than local processing, it may be worthwhile to use such techniques if it means that the protocol overhead can be reduced.

6.2 Energy Consumption

6.2.1 Goals

SensorNets operate under particularly tight energy constraints. Thus it is crucial to investigate the implications on energy use of any protocol or algorithm used in SensorNets. This experiment was aimed at determining the energy usage characteristics of TinyTorrent using the PowerTOSSIM extension to the TOSSIM simulator (described in Section 5.1.3).

6.2.2 Method

A TinyOS application was developed and compiled for TOSSIM using nesC. This application caused Mote 1 to generate a file of a fixed size, apply a suitable name to the data, and publish it to the SensorNet Plugin via Mote 0. The SensorNet Plugin was configured to use this publish action as a trigger to prompt the next mote (ie: Mote 2) to start downloading the data, after passing it an appropriate peer set. This process was allowed to continue until all motes had acquired the data. In this way, the full peer-to-peer aspect of the TinyTorrent protocol was used to determine the energy consumption.

The log files from the PowerTOSSIM simulator were captured and analysed with the `postprocess.py` utility, which is part of the PowerTOSSIM package. This produced details of the energy usage by each hardware component within the motes, as well as total energy usage for all motes.

The simulation was repeated for various numbers of motes, from 20 motes to 50 motes

in steps of 10. The size of application payload was also varied from 64 bytes to 255 bytes in steps of 64 bytes, in order to determine the energy per bit expended with different payloads.

6.2.3 Results

The first result that became clear from the log file analysis was that the radio component was using the bulk of the energy. The proportion of energy used by the radio remained fairly consistent across all test runs at about 63% of total energy. The remaining 37% of energy was consumed by the processor. This shows the high energy cost of transmitting data across the radio, and implies that more complex local processing would be justified (from the perspective of energy use) if it meant that the number or size of messages could be reduced.

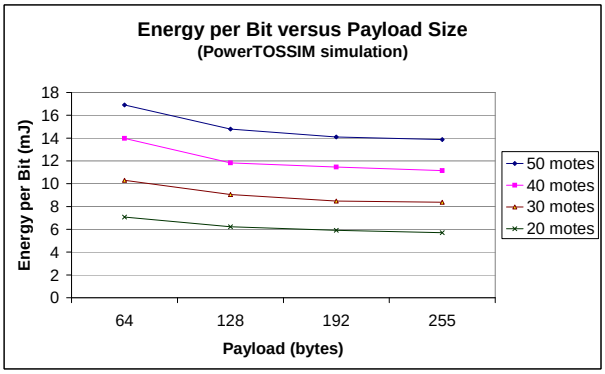

Figure 6.2: Energy Simulation showing Energy per Bit

The results in Figure 6.2 show that the energy per bit decreases as payload size increases. For example, when using 20 simulated motes, the energy to transfer 64 bytes to each mote is 7.08mJ/bit, which drops to 5.70mJ/bit for 255 bytes. This means that the energy overhead follows the same pattern as the protocol overhead: larger payloads are more efficient.

Figure 6.2 also shows that the energy per bit seems to reach an asymptote in all cases. This implies that there is a lower bound on the energy required to transfer a certain payload to all nodes in a SensorNet. However, since TinyTorrent does not currently support more than 255 byte payloads, this cannot be verified.

Another interesting observation is that for a given payload, the energy per bit grows as the number of motes increases. For example, when transferring 255 bytes to 50 motes, the energy cost is 13.87mJ/bit, while for 20 motes it is only 5.7mJ/bit. This is explained by the

fact that in a radio-based medium, sending data and receiving data both incur a significant energy cost (see Section 2.3.2 and [15]). When more motes are involved in the SensorNet, then each mote spends significantly more of its energy listening to the radio spectrum and discarding messages which were not destined for that mote. For less motes (eg: for 20 motes), this problem is not as severe.

The limitations of this experiment are that it does not take into account the bit errors which could occur in the real world. Thus, this shows the energy usage under ideal conditions.

6.3 Throughput

6.3.1 Goals

The purpose of this experiment was to determine the real world throughput that TinyTorrent could achieve. This involved implementing a simple TinyOS application which used TinyTorrent, and deploying this onto Crossbow MICA2 motes. The log files of the Azureus BitTorrent software was used to determine the real throughput of TinyTorrent under conditions of high contention and interference.

6.3.2 Method

A TinyOS application was developed using nesC, which was compiled and deployed on the Crossbow MICA2 motes. This application performed the following functions:

1. The motes initially wait for a `GenerateMessage`. This message is injected into the SensorNet via Mote 0 using a small command line application designed for this task.

2. On receiving this `GenerateMessage`, each mote generates a file in memory of 255 bytes. This is the longest file length supported by TinyTorrent, and was chosen in order to extend the length of the experiments to increase the effect of congestion and interference in the wireless medium.

3. The mote then assigned a name to the generated content. The name included the first byte of the 16 bit node ID, so that the file from each mote was named uniquely and could be treated individually.

4. The node then published its generated data via Mote 0 using the TinyTorrent protocol.

The SensorNet Plugin was configured to immediately download any published content, instead of waiting for the Web Plugin trigger. This means that all the data from each mote will be downloaded as soon as possible by the SensorNet Plugin. The Azureus application logs are then used to determine the number and types of messages sent, as well as recording the time of each message to millisecond accuracy. This makes it possible to determine the actual throughput that each mote individually achieves, as well as a throughput for the whole system. The logs also indicate which `PieceMessages` had to be retransmitted, as these appear in the logs as duplicated requests. The design of this experiment ensures that *all* motes will simultaneously try to transfer their payloads to mote 0 at the same time, and thus there is high contention for the wireless medium, and heavy interference should be experienced by the motes, especially as the size of the SensorNet grows.

The experiment was repeated four times for each of various network sizes, ranging from 1 to 6 motes, 8 motes and 10 motes. The results were recorded in the Azureus application log. This information was then used to determine the real-world throughput of TinyTorrent, as well as the effect of interference and congestion.

6.3.3 Results

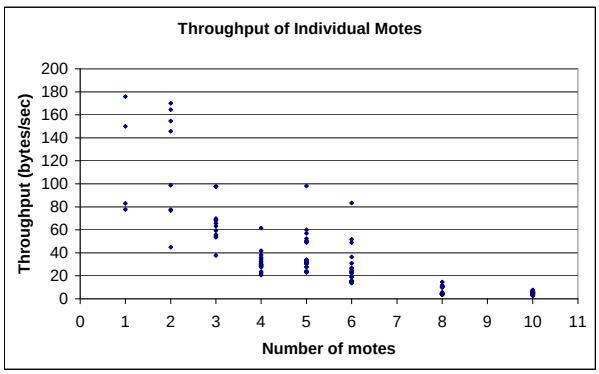

Figure 6.3: Individual Throughput for Each Mote

The results in Figure 6.3 represent the throughput calculated for individual motes during the course of this experiment. The most noticeable trend is that as the number of motes

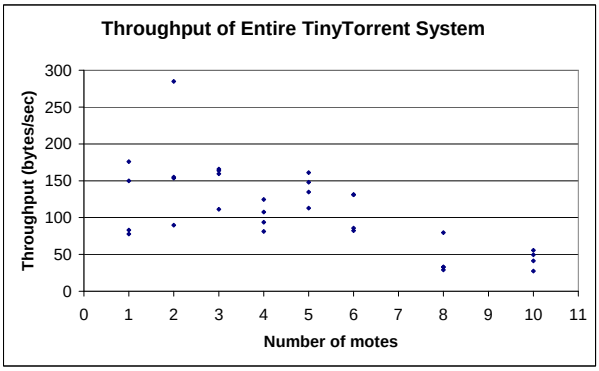

Figure 6.4: Total Throughput for Entire SensorNet using TinyTorrent

in the SensorNet increases, each mote's throughput drops. With only a single mote in the SensorNet, TinyTorrent achieves its maximum individual throughput of 175.86 bytes/sec. When the SensorNet grows to 10 motes, the average individual throughput drops to only 5.03 bytes/sec. It should be taken into account that this experiment was conducted under high contention and interference conditions, since all motes were simultaneously trying to transmit their individual payloads to Mote 0.

The total throughput for the whole SensorNet was also determined by measuring the time taken for the *entire* SensorNet to complete all transfers successfully using TinyTorrent. This result is shown in Figure 6.4. It follows a similar pattern to the individual throughput, showing the same gradual decrease in performance of the SensorNet as a whole as more motes are added. This is again due to the effects of interference in the network, which cause messages to be lost, and thus makes TinyTorrent less efficient. The highest total throughput measured was 284.92 bytes/sec with a SensorNet size of 2 motes. However, this appears to be an outlier. The next highest total throughput measured was for a SensorNet of 1 mote with a total throughput of 175.86 bytes/sec. This is more likely to be a real indication of the maximum real-world total throughput, since the maximum should be achieved in a SensorNet containing only a single mote (and thus one without any interference). When the SensorNet grows to 10 motes, the total throughput drops to between 27.44 bytes/sec and 55.56 bytes/sec. The drop in total performance is thus not as extreme as the drop in performance experienced by each individual mote (where the average throughput drops to 5.03 bytes/sec). This shows that even under conditions of high contention and interference, TinyTorrent can still sustain a reasonably high total throughput, compared to the throughput of individual motes.

A further observation is that in both the individual and the total throughput measurements, it appears that the throughput tends towards an asymptote. This would place a lower bound on the performance of TinyTorrent. A possible explanation for this is that as more motes are added to the SensorNet, the collision detection built into the MICA2 radio stack of each mote will spend greater amounts of time detecting radio traffic and entering the random back-off period. This might place a lower bound on the effects of packet collisions. However, this cannot be confirmed without extending this experiment beyond 10 motes.

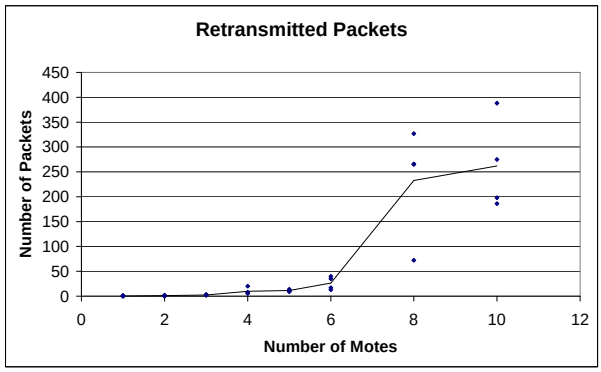

Figure 6.5: Retransmitted Packets recorded by Azureus Plugin

It was also possible to measure the number of packet retransmissions that took place during this experiment. This was achieved by observing the number of times the SensorNet Plugin had to re-request a piece from a mote. Packets which become corrupt due to interference are not automatically retransmitted by the MICA2 radio stack (see Section 5.1.2). This means that the application is responsible for detecting and retransmitting data. The number of retransmissions for the various SensorNet sizes were recorded, and are shown in Figure 6.5. This graph shows the number of retransmissions for each of the four experimental runs for each network size, as well as the average retransmissions (shown as the solid line). This shows that as motes are added to the SensorNet, the effects of interference become more pronounced, with an exponential growth in the number of packet retransmissions. An interesting observation is that the average retransmission rate seems to level off at a network size between 8 and 10 motes. It is not clear if this trend will continue beyond 10 motes. As mentioned above, it is possible that as more motes are added to the network, the overall effect is to reduce the number of corrupt packets, since more motes will spend more time in the random back-off phase of the medium access algorithm (see Section5.1.2). However,

this cannot be confirmed with the data available.

Chapter 7

Future Work

This project has raised some interesting questions for future research. The first one is to extend the throughput experiments to determine what effect interference and the carrier sensing algorithm in the MICA2 mote has on throughput. Currently, the data implies that throughput drops as the network size increases, but approaches an asymptote. This could be confirmed by further experiments.

The use of the Adler-32 checksum algorithm could also be investigated. Specifically, the process by which a 32 bit checksum is compressed into 8 bits should be investigated to determine to what extent this algorithm impairs the bit error detection properties of Adler-32. It would be expected that the error detection would be worse with the compressed Adler-32, since there is a far greater chance of collisions of checksums. It might also become trivial to construct two pieces of data that generate the same Adler-32 checksum, which would have serious security implications. Considering that the primary purpose of the Adler-32 checksum is for integrity checking, this may not be an issue. It may be possible to further compress the checksum into 4 bits for example. This would have several implications: The bit error detection ability of the checksum would be further impaired, however the checksums would each take up less space. This means that more checksums can be packed into a single `PublishDataMessage`, and thus the number of pieces can be increased from 16. This means that TinyTorrent could be made to support files longer than 255 bytes. Bit error detection should also be evaluated in terms of real-world experiments or simulations (using an appropriate error model) to determine the actual effect of these checksum variations.

The protocol overhead experiments raised some interesting points. The TinyTorrent pro-

tocol incurs a significant overhead, with only 36% of all bytes transferred being used for the actual payload (at a total payload of 255 bytes). This overhead could be reduced using several techniques. It may be feasible to allow more than 16 bytes per piece, thus increasing the amount of data transferred with each `PieceMessage`. This would reduce the relative overhead. However, such extension must take into account that the TinyOS messages can only contain at most 29 bytes of application defined payload. This places a hard limit on the effectiveness of this approach. Other approaches might be to use some form of compression to increase the effective payload transferred by TinyTorrent. The energy consumption when using TinyTorrent is weighted heavily to the radio hardware (which uses about 63% of the total energy). This means that is should be possible to employ more processor-intensive algorithms such as compression, if these will result in less messages or more useful data per message.

Security is an area that was not investigated during the course of this research. However, this is a very important aspect of any wireless system, where data is transmitted in a broadcast medium. Security protocols pose a serious challenge to researchers in the SensorNet field, since they are often to complex to be easily implemented in software running on microprocessors. Memory constraints also make security a difficult challenge for SensorNets. Security protocols could be built into TinyTorrent, and the impact of these evaluated in terms of the increased energy consumption, impact on payload sizes, and their impact on the throughput that TinyTorrent can achieve.

Unfortunately it was beyond the scope of this project to perform a detailed comparison of other data dissemination algorithms. An interesting future direction would be to implement some of the competing protocols (such as SPIN) and to repeat the experiments outlined in Section 6 in order to be able to make a side-by-side comparison of these protocols.

Chapter 8

Conclusion

The goals of this research were:

- to investigate the BitTorrent protocol, and attempt to apply the concepts that make BitTorrent successful on an Internet scale to a SensorNet environment
- to investigate a means of exposing SensorNet data to external systems
- to evaluate the implementation through experiments and simulations

The first goal was achieved by implementing the TinyTorrent protocol. This protocol was an implementation of several BitTorrent concepts adapted for use on SensorNet devices. The key components of the protocol are the breaking down of files into independent pieces and rapid propagation of data through a BitTorrent-like piece selection strategy. This allows TinyTorrent to rapidly replicate data in the SensorNet, and spreads the cost of publication across the whole network. It also ensures multiple redundant copies of data exist in the network. The TinyTorrent implementation had to make several changes to the BitTorrent protocol in order to adapt it successfully to the limited resources available on the Crossbow MICA2 motes, which were the target hardware platform.

The second goal was exposing SensorNet data to external systems. This is a key element in SensorNet design. The information contained in a SensorNet must reach external systems in some way in order for the SensorNet to have any use. This can be achieved by allowing the SensorNet devices to control the environment using actuators (such as automatic control of air-conditioning), or by making the data in the SensorNet available to external systems.

The approach chosen in this project was to make the data available to BitTorrent clients by writing a SensorNet Plugin for the popular Azureus BitTorrent client. This SensorNet Plugin acted as an interface between the SensorNet and the BitTorrent networks, allowing SensorNet motes to publish data to the BitTorrent network. The SensorNet Plugin also acted as a tracker for the SensorNet motes, which were running the TinyTorrent protocol. The SensorNet Plugin was used in all experiments and simulations. Hence, this aspect of the research was also a success.

The experimental evaluation showed that the percentage of bytes used by the payload ranged from 22% to about 37% of the total bytes transferred by TinyTorrent. The theoretical maximum was found to be about 64.4%. Various techniques for improving this situation were proposed. The energy consumption showed that about 2/3rds of the energy was consumed by the radio hardware. This means that more complex algorithms can be employed in TinyTorrent, provided these lead to smaller messages or fewer messages. The Flash memory unit of the MICA2 motes works in 16 byte pages, and hence the piece size was chosen to be 16 bytes in order to simplify the storage scheme. However, given the energy consumed by the radio, it may be feasible to design a more complex Flash memory storage scheme to store pieces of more than 16 bytes, and hence allow more than 16 bytes per piece. The added complexity and energy cost of a such a storage scheme may be offset by the gains in terms of reduced message numbers and higher throughput.

The effects of interference were also studied, by running a TinyTorrent application under conditions of high contention and interference. This showed that packet corruption increases exponentially with the size of the network. At the same time, the throughput of the TinyTorrent system is negatively affected. The effects of interference on energy consumption were also observed as a marked increase in energy per bit required for larger networks. A network of 20 motes (transferring 255 bytes each) incurred an energy cost of 5.7mJ/bit. For 50 motes, the cost jumped to 13.87mJ/bit.

In conclusion, the TinyTorrent implementation was successful, showing that it is possible to adapt BitTorrent concepts to a SensorNet. TinyTorrent also exhibits interesting properties in terms of throughput and energy use. This research shows that more complex algorithms can be employed in SensorNets, provided they reduce the number and size of messages, as this will lead to better energy efficiency. The development of the SensorNet Plugin was also successful, and makes it possible for a SensorNet running TinyTorrent to publish data to the built-in Azureus Web Plugin, which an external client can access. This means that

SensorNet data is available outside the SensorNet itself using the standard BitTorrent protocol. Opportunities for future investigations were also made clear during the course of this research.

Bibliography

[1] J. M. Hellerstein, W. Hong, and S. R. Madden, "The sensor spectrum: Technology, trends, and requirements," *SIGMOD Record*, vol. 32, no. 4, 2003.

[2] A. Hać, *Wireless Sensor Network Design*. John Wiley & Sons Ltd, 2003.

[3] IEEE, "Ieee 802 standard." Online, September 2005. http://www.ieee.org/portal/index.jsp?pageID=corp_level1&path=about/802std&file=index.xml&xsl=generic.xsl.

[4] E. Ferro and F. Potortì, "Bluetooth and wi-fi wireless protocols: A survey and a comparison," *IEEE Wireless Communications*, 2004.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, March 2002.

[6] M. A. M. Vieira, C. N. Coelho, Jr, D. C. da Silva, Jr, and J. M. da Mata, "Survey on wireless sensor network devices," vol. 1, pp. 537–544, September 2003.

[7] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister, "Smart dust: Communicating with a cubic-millimeter computer," *Computer*, vol. 34, pp. 44–51, January 2001.

[8] V. Kumar, "Sensor: The atomic computing particle," *SIGMOD Record*, vol. 32, December 2003.

[9] Wikipedia, "Wikipedia entry for field-programmable gate array." Online, September 2005. Available at http://en.wikipedia.org/wiki/FPGA.

[10] V. Tsiatsis, S. A. Zimbeck, and M. B. Srivastava, "Architecture strategies for energy-efficient packet forwarding in wireless sensor networks," in *Proceedings of ISLPED*, pp. 92–95, ACM, August 2001.

[11] C. T. Inc, *MICA2 Datasheet*. Crossbow Technology Inc., Crossbow Technology Inc., 4145 N. First Street, San Jose, CA 95134, August 2005.

[12] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *In Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[13] C. I. Merzbacher, A. D. Kersey, and E. J. Friebele, "Fiber optic sensors in concrete structures: A review," *Smart Materials and Structures*, vol. 5, pp. 196–208, April 1996.

[14] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors,"

[15] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE SIGNAL PROCESSING MAGAZINE*, pp. 40–50, March 2002.

[16] R. Gummadi, X. Li, R. Govindan, C. Shahabi, and W. Hong, "Energy-efficient data organization and query processing in sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 273–274, ACM, 2004. ISBN:1-58113-879-2.

[17] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks," in *ACM MOBICOM*, 2002.

[18] C.-Y. Chong and S. P. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, 2003.

[19] N. Noury, T. Herve, V. Rialle, G. Virone, E. Mercier, G. Morey, A. Moro, and T. Porcheron, "Monitoring behavior in home using a smart fall sensor and position-sensors," *Microtechnologies in Medicine and Biology, 1st Annual International, Conference On*, pp. 607–610, 2000.

[20] N. Noury, P. Barralon, G. Virone, P. Boissy, M. Hamel, and P. Rumeau, "A smart sensor based on rules and its evaluation in daily routines," in *Microtechnologies in Medicine & Biology*, pp. 3286–3289, IEEE-EMBS, September 2003.

[21] R. Beckwith, D. Teibel, and P. Bowen, "Report from the field: Results from an agricultural wireless sensor network," *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, 2004.

[22] A. Deshpande, C. Guestrin, and S. R. Madden, "Using probabilistic models for data management in acquisitional environments," *Proceedings of the 2005 CIDR Conference*, 2005.

[23] California Department of Transportation, "Caltrans realtime freeway speed map." Available at http://www.dot.ca.gov/traffic/.

[24] V. P. Sisiopiku, "Variable speed control: Technologies and practice," *Proceedings of the 11th Annual Meeting of ITS America*, 2001.

[25] F. Silva, J. Heidemann, R. Govindan, and D. Estrin, "Directed diffusion," Tech. Rep. ISI-TR-2004-586, USC/Information Sciences Institute, January 2004. To appear in *Frontiers in Distributed Sensor Networks*, S. S. Iyengar and R. R. Brooks, eds.

[26] R. Kannan, L. Ray, and A. Durresi, "Security-performance tradeoffs of inheritance based key predistribution for wireless sensor networks," *CoRR*, vol. cs.NI/0405035, 2004.

[27] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications*, pp. 41–47, ACM, 2002.

[28] R. Krishnan, M. D. Smith, and R. Telang, "The economics of peer-to-peer networks." Working Paper, Carnegie Mellon University, Pittsburgh, PA 15213, September 2003.

[29] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing," tech. rep., HP Laboratories, Palo Alto, March 2002.

[30] B. Cohen, "Incentives build robustness in bittorrent," *Workshop on Economics of Peer-to-Peer Systems, 2003*, May 2003.

[31] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," *Lecture Notes in Computer Science*, vol. 3015, pp. 1–11, January 2004.

[32] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "A measurement study of the bittorrent peer-to-peer file-sharing system," May 2004. Preprint submitted to Elsevier Science 15 May 2004.

[33] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, "To share or not to share: An analysis of incentives to contribute in collaborative file sharing environments," *Workshop on Economics of P2P Systems (P2P Econ)*, June 2003. Berkeley, CA. (a refined version in proceedings of CCGrid'04).

[34] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes,"

[35] Wikipedia, "Wikipedia entry for bittorrent." Online, September 2005. Available at http://en.wikipedia.org/wiki/Bittorrent.

[36] B. Cohen, "Bittorrent website," August 2005.

[37] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, and J. Schiller, "Connecting wireless sensornets with tcp/ip networks," in *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)*, (Frankfurt (Oder), Germany), Springer Verlag, February 2004.

[38] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003.

[39] N. Lee, P. Levis, and J. Hill, *Mica High Speed Radio Stack*.

[40] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 93–104, November 2000.

[41] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," *Ambient Intelligence*, 2005.

[42] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003. Los Angeles, California, USA.

[43] P. Levis and N. Lee, *TOSSIM: A Simulator for TinyOS*. University of California, Berkeley, September 2003. Available from http://www.cs.berkeley.edu/ pal/pubs/nido.pdf.

[44] V. Shnayder, M. Hempstead, B. rong Chen, and M. Welsh, "Powertossim: Efficient power simulation for tinyos applications," August 2005.

[45] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 188–200, 2004.

[46] "Azureus: Java bittorrent client." Available online, September 2005. http://azureus.sourceforge.net/.

[47] P. Buonadonna, J. Hill, and D. Culler, "Active message communication for tiny networked sensors,"

[48] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of MOBICOM*, ACM, 1999.

[49] J. Fletcher, "An arithmetic checksum for serial transmissions," *IEEE Transactions on Communications*, vol. 30, pp. 247–252, January 1982.

[50] Wikipedia, "Wikipedia entry for adler-32." Online, September 2005. Available at http://en.wikipedia.org/wiki/Adler_checksum.

[51] "Tinyos cvs repository at sourceforge.net." http://sourceforge.net/projects/tinyos.