Delay Tolerant Networking for Sensor Networks

Max Loubser
Swedish Institute of Computer Science
max@sics.se

January 12, 2006

SICS Technical Report T2006:01
ISSN 1100-3154
ISRN:SICS-T–2006/01-SE
Keywords: sensor networks, DTN, TCP

Abstract

The Delay Tolerant Networking Architecture (DTN) has been proposed for use in challenged networks that suffer from intermittent connectivity or high delay. The DTN architecture and the bundle protocol presents a standard method to interconnect heterogeneous challenged networks using asynchronous message switching. It provides a framework for dynamic routing, contact scheduling, naming, reliability and transmission status reports. Wireless Sensor Networks (WSNs) are often viewed as challenged networks as nodes operate at low power, often with weak or intermittent radio communication. WSNs are an important application area for DTN. In this work I present ContikiDTN, a TCP/IP based prototype implementation of the DTN architecture and bundle protocol. ContikiDTN aims to evaluate the suitability of the DTN bundle protocol as a solution for messaging inside a TCP/IP WSN and as a way of connecting the WSN to the Internet. I discuss the design and implementation of ContikiDTN using the Contiki operating system. I highlight the issues in implementing the bundle protocol with TCP and Contiki. I show that the event-driven Contiki kernel is very suitable for an asynchronous message forwarding application. I use ContikiDTN to communicate with a full PC platform implementation of DTN and show that it can be used as a gateway to the Internet. I present a simulation and experimental results showing the performance of multi-hop TCP based DTN as compared to only TCP. I show that the core propositions of the DTN architecture hold in a WSN and that it is feasible to implement DTN on resource constrained devices using TCP/IP and Contiki.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Problem statement

The model for communication over the Internet is built for end-to-end inter-process communication over potentially dissimilar networks. The TCP/IP protocol suite functions well on the Internet today, but there are an increasing number of internetworking environments where the Internet protocols do not perform well. In these environments the seldom stated assumptions built into the current Internet architecture need to be reconsidered. As these new challenged networks become more important a new architecture for communication over dissimilar networks is required.

An architecture for these challenged networks has been proposed. Delay Tolerant Networking (DTN) [Fal03] is designed for intermittently connected networks where network partitions are frequent and very high delays are associated with some links. The DTN architecture is accompanied by a messaging protocol called the bundle protocol. The DTN architecture addresses network issues such as high latency, frequent disconnection and long queueing time. These issues arise because nodes in challenged networks are sometimes located in extreme or difficult to reach locations, or because of the limitations of the nodes themselves. Node limitations can necessitate low duty cycle operation (eg battery powered nodes) or cause limited longevity. Extreme or difficult environments can cause low link quality, especially on wireless links. Extreme long distance links, for example between planets, can have considerable propagation delays.

Wireless Sensor Networks (WSNs) can in many cases by classified as challenged networks and is an important application area for the proposed DTN architecture and the accompanying bundle protocol. WSNs are characterised by sensor nodes that operate at extremely low power and with very limited resources. The areas that sensors are deployed in and the nature of the radio communication between nodes can cause high delay, unreliable links. Sensor nodes often operate with low duty cycles. Radio use for sending and listening for transmissions is expensive and radios are often turned off [RSPS02]. Sleeping nodes further complicates communication.

It is widely suggested ([Fal03], [DVA+04], [HF04]) that DTN can serve to alleviate the communication difficulties in sensor networks. DTN also provides the additional advantage that it can be used as a general way to communicate with nodes in other, possibly very different, networks. DTN is therefore a good way to connect a sensor network to the Internet, possibly through different sensor and other

networks, all implementing DTN.

The problem of communicating in a poorly connected network is not new and many solutions have been proposed. Early store and forward systems were employed on Fidonet and UUCP (Unix to Unix Copy) was used for transferring email over dial-up links. DTN, although also in essence a store and forward asynchronous messaging system, provides the important advantage of being a general solution that can be applied to networks built on very different lower level technologies.

It is not obvious that DTN is suitable for implementation on tiny sensors, or tiny sensors running a TCP/IP stack. Many network specific protocols are designed with the limitations of the platform in mind and this is not the case with DTN. A DTN prototype for a sensor network platform based on TCP/IP would provide insight into possible implementation difficulties.

The goal of this project is to implement the DTN architecture and the bundle protocol in a wireless sensor network in order to use the implementation to gain insight into the architecture and evaluate it as solution to the communication problems commonly encountered in such networks.

1.2 Motivation

A DTN implementation on a gateway between a sensor network and the Internet and on sensor nodes can be used to test the performance and usefulness of DTN for connecting sensor nodes to the Internet and communication between sensor nodes.

A working prototype DTN implementation can be employed and tested for normal sensor network tasks, such as code upload and data download. These tasks can also be done from anywhere on the Internet if the sensor network is connected to the Internet. Such an implementation can be used to test the core propositions behind the DTN design and highlight the issues in realising the bundle protocol on a very limited system.

The Contiki operating system for memory-constrained devices is gaining popularity for research and ”real world” use on sensors and limited devices. Contiki has been deployed on different sensor nodes and ported to many other platforms [Dun05]. Contiki provides TCP/IP networking and multi-tasking. It is a useful platform for implementation of the bundle protocol.

With a Contiki implementation of DTN, different scenarios for communication to and from a sensor network can be investigated. The interoperation of a PC (or other more powerful machine) implementation of DTN, based on TCP, and the Contiki TCP/IP stack can be tested. The performance of DTN as solution to sensor network communication difficulties can be evaluated. The feasibility of using DTN as a gateway to the Internet and implementing the full protocol on a sensor can be tested.

The motivation for this project is to acquire the tools needed to achieve a complete understanding of DTN related issues in general and in certain types of sensor networks. Further, the design of a suitable architecture for a DTN implementation on

the Contiki operating system would enable the testing, simulation and evaluation of DTN performance in solving common sensor network problems.

1.3 Evaluation

The success of the project can be assessed on the lessons learnt in the implementation stages and the insight gained by experiments using the prototype implementation.

The success of the architecture designed for the bundle protocol implementation on Contiki is a parameter for evaluation. The implementation must make use of the features of the Contiki operating system while negotiating its limitations and considering the platform that the code must eventually be used on. The solutions to problems of communications between parts of the program, memory management and asynchronous message handling must be appropriate and practical in this environment.

The system must achieve a balance between the full interoperability a complete implementation of the bundle protocol will provide and the compromises necessary for implementation on a limited system. These design choices must not compromise the other goals for the system.

Another criterion for evaluation is the success of the experiments using the prototype implementation. From previous work in the field, it is expected that DTN will perform well in a challenged sensor network and that the prototype will be successful in operating in a intermittently connected network. The experiments will be deemed successful if the results provide a strong case to prove or disprove this expectation.

1.4 Structure of the report

This report details the work done for the background research, completion of the design, development, testing and evaluation of a Contiki implementation of the DTN architecture and bundle protocol.

In Chapter 2 the basic concepts of DTN are introduced and the architecture and bundle protocol discussed. Background on the working environment, the Contiki operating system and the Contiki simulator is provided. Some alternatives to DTN are presented. A survey of related work on DTN in general, and in sensor networks specifically, is given. Relevant details of other implementations of DTN are highlighted.

Chapter 3 elaborates on the goals of the project and the research method undertaken to achieve them. The contribution of this project to the area of research is described.

In Chapter 4 the design and implementation process for the DTN implementation is described. The architecture is presented from a layered and function oriented perspective.The process of sending and receiving bundles in the implementation

is described. The chapter is concluded with discussion of the various architectural and programming issues that became important during the development process.

Chapter 5 presents the analysis and evaluation of the work. The extent to which the goals were achieved is discussed and the implementation is evaluated with consideration of the goals. The experiments done and experimental setup is explained and the results with possible explanations are presented.

The reports concludes in Chapter 6 with a final discussion of the conclusions made from the experiments and the development process. The lessons learned about DTN and a Contiki implementation are presented.

CHAPTER 2

Background

2.1 Delay-tolerant network architecture

The current specification of the Delay-Tolerant Network (DTN) architecture is a work in progress Internet-draft [CBH+05]. At the time of writing, the drafts are still in active development - here the versions in [CBH+05] and [SB05] are described.

The DTN architecture is designed to address the problems encountered in challenged networks, where the internetworking solutions used in the Internet are unsuitable. The DTN architecture introduces the bundle layer, comprising an asynchronous message oriented overlay network. The bundle layer operates above the transport layers of different networks. A network entity implementing the bundle layer is called a DTN node. The DTN architecture description mentions of anycast and multicast in DTN, but there are unresolved issues with regard to custody transfer. Only unicast DTN is discussed here. The separate issues regarding bundle security are also not included in this discussion.

2.1.1 Bundle layer

The bundle layer is the interface to applications for the DTN overlay network. Its position in the Internet-type protocol stack is shown in Figure 2-1. Its position in the protocol stack makes it possible to implement the Bundle layer in the application space of an operating system.

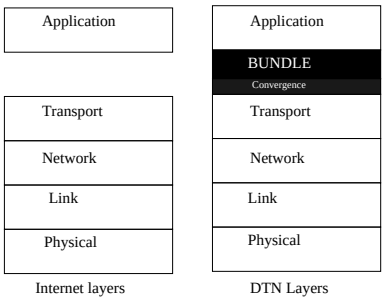

Figure 2-1: Internet and DTN stacks

DTN nodes can operate as gateways between different networks. In such a case the bundle layer is the common layer between the two different networks that the

DTN gateway interconnects. In Figure 2-2 the connection of a DTN sensor node to the TCP/IP Internet is shown. The location of potential delay and persistant storage requirements are also shown.

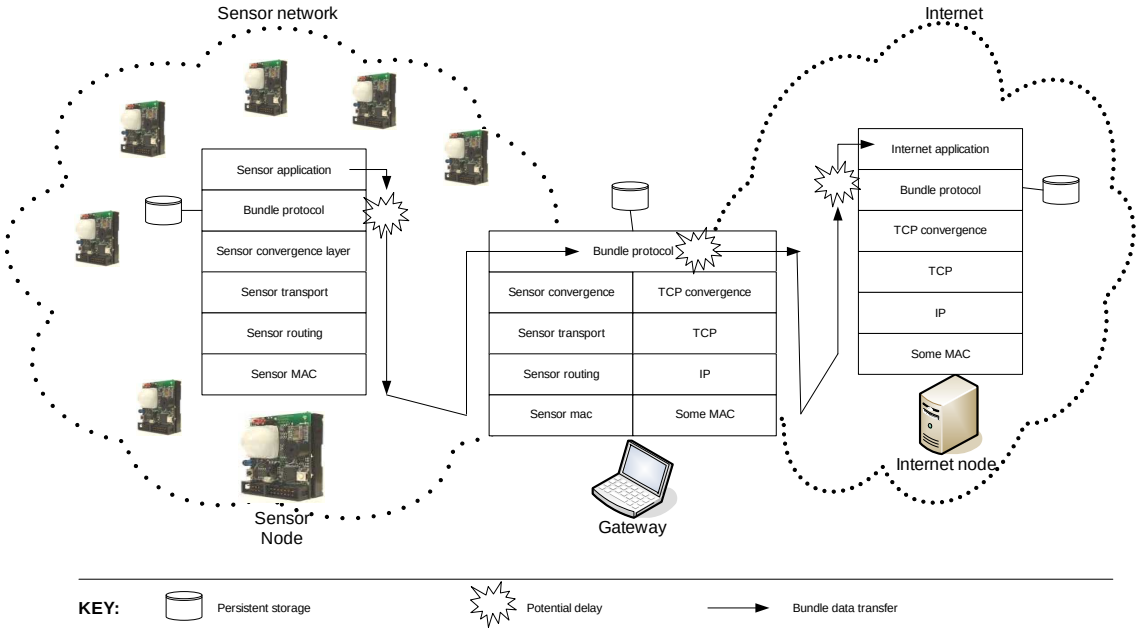

Figure 2-2: Using the bundle layer to connect a sensor network to the Internet

DTN nodes cooperate by means of the bundle layer in order to overcome communication disruptions. The bundle layer operates on messages as its basic unit of data and requires access to a persistant store for keeping messages until it is possible to forward them. DTN nodes can route messages and provide a message store-and-forward service with optional reliability across heterogeneous networks.

DTN nodes are named with a flexible scheme, based on Uniform Resource Identifiers (defined in [BLFM05]), making interoperability across different lower layer naming schemes possible. The DTN URI naming scheme can encapsulate the naming scheme of the underlying network. This is discussed further in Section 2.1.3.

DTN messages are of arbitrary length and are not guaranteed to be delivered in order. Messages are encapsulated in ”bundles”. Bundles and the bundle protocol are described in a work in progress Internet-draft, [SB05]. Bundle encapsulation is discussed further in Section 2.1.6. The message abstraction has the advantage that the bundling layer has complete knowledge of the size and performance requirements of a message transmission. It is assumed that storage is available and sufficiently persistent so that messages can be kept until the DTN node can forward them.

Bundles can be marked with three relative priority classes for prioritising a queue at a sender. The classes are similarly named to classes of service in a postal system.

- Bulk - Low priority, bundles of this type will only be sent if there are no higher priority bundles with the same source and destination.
- Normal - Bundles of this type are sent before Bulk bundles.

- Expedited - These bundles are sent before the other two classes, but are otherwise the same.

Bundles are also marked with a data lifetime. The bundle priority and the data lifetime, together with the forwarding policy and routing algorithms affect the progress of the bundle through the DTN overlay.

2.1.2 Convergence layers

In order to achieve interoperability between heterogeneous networks the bundle layer needs to make use of different interfaces to lower network layers. For this reason a *convergence layer* is introduced below the bundle layer (see Figure 2-1 and 2-2). It is necessary for the convergence layer to provide a standard interface to the bundle layer, but lower layers might provide different services to the convergence layer.

Lower layers can differ in their support of reliable delivery, connections, flow control, congestion control and message boundaries. The bundle protocol assumes that the lower layers can provide message boundaries and reliable delivery and it is up to the convergence layer to augment lower layers that do not provide this functionality. For example, a convergence layer for TCP would require the introduction of message boundaries into TCP streams.

2.1.3 Naming

A DTN node is identified by one or more Endpoint Identifiers (EIDs). EIDs use the general syntax of URIs. In Figure 2-3 an example EID is shown, using the *bp0* URI scheme (see [SB05]) and a DNS name. The EID in Figure 2-3 denotes a DTN node on the Internet. Different schemes can be defined for each different type of network that the bundle layer is implemented on top of. The URI naming allows for *late binding*, where the Scheme Specific Part (SSP) of the URI is only translated to a lower layer address or different EID late in the delivery process of a message. This is in contrast with DNS on the Internet, where such a mapping is completed before a message leaves its origin. Late binding is useful in a challenged network because the result of a early binding name lookup might no longer be valid when it becomes possible to start sending data into the network.

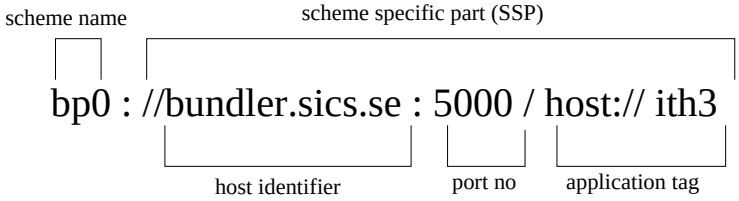

Figure 2-3: An example EID with the *bp0* URI scheme

A single EID can refer to more than one DTN node. Applications address messages to an EID. Applications can also request that the bundle layer deliver messages addressed to a certain EID to them. This request is called a *registration*.

2.1.4 Delivery options, status and signal bundles

The ICMP protocol, commonly used on the Internet, provides informational and diagnostic messages to Internet nodes. In the same way DTN nodes can request to receive information from other DTN nodes by using the delivery options. Bundles can be assigned different delivery options, depending on the relative importance of the bundle or information about network conditions. Delivery options are different from service classes, as they do not influence bundle forwarding decisions.

In the current DTN architecture specification, nine delivery options are defined, three of which are only available when DTN security is enabled. With combinations of the delivery options a DTN node can receive reports about *delivery*, *forwarding*, *receipt* and *deletion* of bundles.

Delivery options can also be used for hop-by-hop reliable delivery. A custody transfer mechanism is used to transfer responsibility for reliable delivery from one node to another. A *Custody Transfer Required* delivery option is used to indicate this and a report bundle about accepted custody can also be requested. This is explained more fully in Section 2.1.7.

DTN security, defined in [SFW05], allows for the specification of three more options if confidentiality, authentication or error detection is required for a bundle.

Bundle Status Reports (BSRs) are the diagnostic responses sent when an action (such as delivery, forwarding, deletion etc.) is performed on a bundle and the corresponding delivery option is set. BSRs are not sent to the source EID of a bundle, but to the report-to EID. This is a field in every bundle that is set to the DTN node EID that should receive status information for this bundle. The report-to EID can be the same as the source EID. For example, when a bundle with the "Report when Deleted" option set is discarded, a BSR will be sent to the report-to EID for that bundle. Expiration is one possible reason for discarding a bundle.

2.1.5 Contacts, routing and forwarding

A DTN network can be described as a directed, time-varying multigraph. More than one edge may exist between a pair of nodes. The reason for this abstraction is that it might be possible for a DTN node to select between two distinct physical connection when making a routing decision and that link capacities and propagation times in an intermittent network will vary with time [JFP04]. A simple DTN graph is shown in Figure 2-4.

The DTN architecture provides a framework for routing and forwarding at the bundle layer. Routing algorithms for DTNs is an active research area (see [LDS04], [WJMF05] and [JFP04]) and are not described as part of the architecture.

In an intermittently connected network the capacities of the edges fluctuate between zero and a positive capacity as links go up and down. The period of time that an edge is at positive capacity is called a *contact*. The *contact volume* is defined as the product of the contact time and the contact link capacity. Contact times and volumes are input for route calculation functions and predicted contacts can help optimise forwarding.

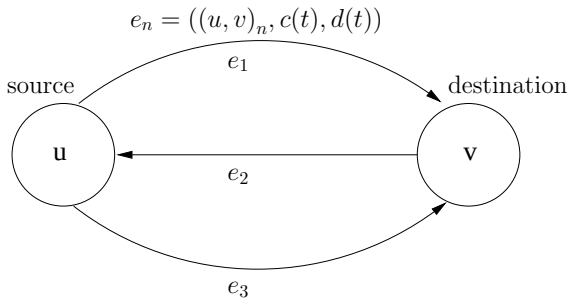

Figure 2-4: DTN graph, with an edge e_n described in terms of source, destination and a capacity ($c(t)$) and delay ($d(t)$) function.

Contacts are classified on their predictability and expected duration.

- Persistent - Always on contacts.
- On-demand - Can be turned on when required and will be persistant as long as required.
- Intermittent, scheduled - A scheduled contact is an agreement to establish contact at a particular time and for a particular duration. A satellite passing over a location at scheduled times is an example.
- Intermittent, opportunistic - This type of contact is not scheduled and must be made use of whenever it becomes available. A Bluetooth device moving into the transmission range of another such device is an example.
- Intermittent, predicted - These contacts are not based on a fixed schedule, but can be predicted with good probability based on earlier contact patterns.

2.1.6 Bundle contents and encapsulation

Application data is encapsulated into bundles that can be arbitrarily long. A bundle header is added to the application data, as shown in Figure 2-5. The bundle header makes use of optional extension headers, where the type, the length and the value of each extension header follows the previous header.

A bundle will typically be split into parts as it is processed by lower layer protocols, as shown in Figure 2-5. Fragmentation can also take place at the bundle layer, if the bundle application detects that it is necessary or beneficial. Fragmentation can be beneficial in an intermittently connected network when a bundle that has been partially forwarded is split into fragments when a break in communication occurs. Only the fragment that has not been forwarded can then be retransmitted when communication is reestablished.

DTN fragmentation and reassembly can be done proactively or reactively. When the volume of a contact is known or can be predicted it can be beneficial to proactively fragment a block of application data before the first transmission and send each fragment as an independent bundle. In this case it will be up to the final destination to reassemble the block of data. Reactive fragmentation takes place when

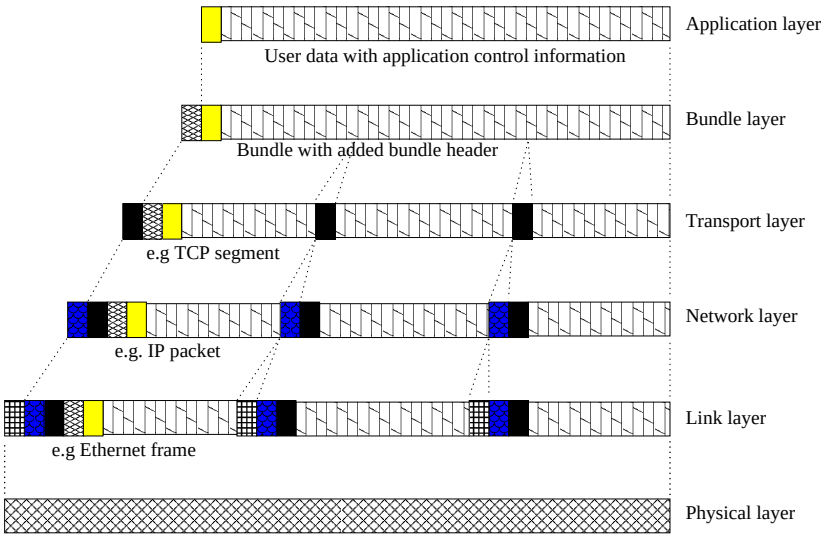

Figure 2-5: Encapsulation of a bundle with TCP/IP Internet lower layers

a message is only partially forwarded because of a communication interruption. A DTN node that receives a partially forwarded bundle (identified by using header length fields) will mark it as a bundle fragment and then forward it in the same way as a regular bundle. The sender of the partial bundle will have to be notified that fragmentation took place and will forward the rest of the bundle when a contact is available.

All DTN nodes need to be able to reassemble fragments, but reactive fragmentation need not be implemented on every node. Reactive fragmentation complicates bundle security (see [CBH+05] and [SFW05]).

1 byte			
Version	Processing Flags	Class of Service flags	Header length (variable size)
Destination scheme offset		Destination SSP offset	
Source scheme offset		Source SSP offset	
Report-to scheme offset		Report-to SSP offset	
Custodian scheme offset		Custodian SSP offset	
Creation Timestamp (8 bytes)			
Lifespan (4 bytes)			
Dictionary length			
Dictionary byte array (variable)			

Figure 2-6: Bundle header

Every bundle and bundle fragment contains the fields shown in Figure 2-6. The header includes a dictionary that consists of an array of bytes formed by concatenating the null-terminated scheme names and Scheme Specific Parts (SSPs) of all EIDs referenced in the primary header and other headers. The address fields in the header are each divided into a scheme offset and a SSP offset. The scheme offset indicates the offset in the dictionary of the scheme name for that EID and the SSP offset indicates the offset in the dictionary for the SSP. The use of such a dictionary

enables some header compression. For example, if the source and report-to EID are the same, the EID string can be included in the dictionary once, but referenced more than once.

Some other fields require additional clarification:

- Creation Timestamp - The time the application sent the bundle. DTN nodes are assumed to have the ability to synchronise their clocks (see Section 2.1.8)
- Lifespan - Indicates the expiration time for the bundle, expressed as an offset relative to the creation time.
- Priority Class - bulk, normal or expedited.
- Report-to EID - The EID where BSRs should will be sent to.

2.1.7 Reliability and custody transfer

Reliable message delivery is a major problem in disconnected networks and DTN can be used as a basis for a reliable message delivery system. The bundle layer provides unordered, prioritised and unacknowledged message delivery. Applications can make use of the bundle delivery and other BSRs (see Section 2.1.4) to build their own end-to-end reliability mechanisms, but custody transfer can provide a hop-by-hop solution that amounts to a different form of reliability. Custody transfer can relieve end nodes of keeping state about the delivery of their bundles.

Custody transfer involves the acknowledged movement of responsibility for reliable delivery from one node to another. The use of the custody transfer mechanism is indicated by using the Custody Transfer Required delivery option. A custody transfer will normally take place from a DTN node to another DTN node that is closer to the destination node according to some routing metric. A node that accepts custody of a bundle is called a *custodian*. Not all nodes have to be custodians and some nodes can potentially be custodians only when resources allow them to accept custody.

A custody transfer relies on the underlying network's reliable delivery protocols for reliability over a single hop. The bundle layer then provides a coarse-grained timeout and retransmission mechanism for each custody transfer, making use of custodian-to-custodian acknowledgements. The acknowledgement is sent to the current custodian EID found in an arriving bundle and the field is updated to the new custodian's EID before it is forwarded.

In a challenged network end nodes will not always remain active for the duration of a transfer. The custody transfer hop-by-hop delegation of responsibility increases the chances that the bundle will be successfully delivered relative to the chance that an end-to-end reliability mechanism would have in a challenged network [Fal03].

2.1.8 Time synchronisation

DTN nodes are required to have the ability to synchronise their clocks. This is needed for four reasons:

- Bundle and fragment identification
- Routing with scheduled or predicted contacts
- Calculating bundle expiration times
- Calculating application registration expiration

The time stamp in the primary bundle header is made available to applications, enabling identification of expired bundles. The concatenation of the time stamp and the Source EID is a unique identifier for each bundle. When an application registers to receive bundles for a particular EID, this registration is only valid for a limited time specified by the registering application. The bundle layer needs timing to expire application registrations.

2.2 Alternatives to DTN

DTN is not the only way of organising communication in an intermittently connected network. DTN has certain key advantages. Investigating other solutions aids in understanding these advantages.

The DTN architecture, as described in this chapter, is essentially an asynchronous store and forward messaging system. The most commonly used such messaging system today is e-mail. E-mail does have many properties in common with DTN, but there are important differences. E-mail provides only very primitive routing in the form of static redirections with MX records in the Domain Name System and mostly relies on IP routing to deliver messages. DTN makes it possible to implement sophisticated multipath DTN routing mechanisms. (This is an active research area, see Section 2.5.2). E-mail also does not make provision for the handling of opportunistic contacts and will normally assume that a connection is available. E-mail is designed to only work with TCP as the underlying transport, while DTN makes provision for any type of lower layer.

An early asynchronous message system that was designed to work in a frequently disconnected network was the Usenet NNTP protocol. As with e-mail, NNTP routing is usually not dynamic [Fal04], but NNTP can operate over different underlying transport protocols. NNTP provides only limited feedback about the delivery of messages, while DTN provides a range of status reports. In theory, NNTP could operate using DTN as transport and therefore make use of dynamic routing, fragmentation and more feedback from the network.

From these two examples it is clear that DTN shares important characteristics with earlier systems. DTN's multi path routing, use of predicted, anticipated and opportunistic contacts, feedback from the network, fragmentation, flexible naming and compatibility with any lower layer, all designed specifically for today's challenged networks, make it a major improvement over earlier protocols.

2.3 The Contiki operating system

Contiki is a lightweight, portable, multi-tasking operating system with an event driven kernel [DGV04]. Contiki also includes the μIP TCP/IP stack. Contiki is designed to be used on tiny devices that have severe memory and other resource constraints.

Operating system that are designed to work in memory constrained environments are often designed to be event-driven in order to better deal with the memory constraints [DSV05]. The problem with this solution is that it is difficult to design programs as explicit state machines, as usually required by event driven systems. The Contiki event-driven kernel makes use of a programming abstraction called protothreads that helps to simplify implementation in the Contiki environment [DSV05]. Protothreads are extremely lightweight stackless threads that provide conditional blocking in the event driven system. The protothread abstraction greatly simplifies the μIP TCP/IP implementation.

Protothreads also make it possible for the protosocket interface to exist. Protosockets provide an interface to μIP similar to the traditional BSD socket interface. Using protothreads makes it possible to write network programs that are structured in a linear way, built around conditional blocking calls to the protosocket interface.

The protosocket library works only with TCP. It provides functions for sending and receiving data without having to use lower level μIP functions to deal with retransmissions or acknowledgements. The interface also provides functions for reading data as a stream from a buffer, irrespective of the data being split into multiple TCP segments during transmission.

Contiki includes modules for managing blocks of memory, list handling event timers and access to the system clock.

2.4 The Contiki Network simulator

Uploading code to a actual sensor node and testing a network of these sensors is a very time consuming way of testing in sensor networks. In order to test sensor network applications and protocols a simulation is often the best starting point.

The Contiki Network Simulator, Contiki Netsim, is a simulator for sensor nodes running the Contiki operating system. Contiki's portability makes it possible to run Contiki as a user-level process on a PC operating system such as Linux [DFGV04]. In Contiki Netsim each sensor node is simulated as such a user level process that can communicate with other nodes in the simulation via a simulated wireless network layer. The simulated network layer makes it possible to easily control radio strength, interference levels and measure data volume and the number of packets sent. The network layer is also designed to simulate some of the communication difficulties that may be encountered between wireless sensors.

The simulator runs using the same Contiki code as would be used on certain sensor hardware, making it very easy to move from simulation prototype to real implementation. The simulator also provides a graphical view of the nodes being simulated and the packet communication between them.

2.5 Survey of related work

2.5.1 Challenged networks

Challenged networks are defined as networks that suffer from intermittent connectivity or long delays. A general solution to the problem of challenged networking was first addressed in work on interplanetary or deep-space communications. This project later led to the specific study of DTN for use in deep-space communications. In [BHT+03] the particular difficulties of communicating over massive distances are discussed and compared to the model of communication used on the Internet. Signal propagation delays, low data rates and non-permanent connectivity are examples of these difficulties. In this work an overview of why the current Internet protocols do not behave well under these conditions is given. The way that TCP handles reliable transport and the assumptions under which routes are calculated make the Internet protocols unsuitable for interplanetary communication. The reasons for not using TCP/IP are discussed in more detail in [DFS02].

A general architecture for challenged internets, Delay Tolerant Networking (DTN) is proposed in [Fal03]. The author argues that interoperability is essential in challenged networks and, as mentioned in [BHT+03], a "least common denominator" protocol is needed, making the overlay suitable for operation in any networking environment. The idea being that the networking stacks under the DTN layer would be the best available for whatever challenged network the DTN is operating in. DTN would be the common interface, offering end-to-end data transmission and optional reliability. In [Fal03] the characteristics of challenged networks are elaborated on and the suggestion that the Internet protocols can be "fixed" to operate in challenged networks is explored. Performance Enhancing Proxies (PEPs), protocol boosters and different proxies are possible ways to increase the performance of the Internet protocols in difficult environments. It is concluded that the major weakness of this type of solution is the specificity to a certain environment. Email has some of the properties required to communicate in challenged networks, but the lack of dynamic routing and weakly defined delivery semantics is said to make it unsuitable as solution in challenged networks.

The principle of *fate sharing* is introduced in [Cla88]. It suggests that conversation state should only be kept at the end-nodes in a conversation, because the conversation would be useless if one of these nodes fail. This is one of the basic principles in the Internet architecture. In [Fal03] the design of a new architecture (DTN) is motivated by the fact that the principle of fate sharing does not hold in challenged networks. In challenged networks it can be advantageous for nodes to transfer connection state to other nodes along the path from end to end. Limited power or connectivity are possible reasons for transferring state.

In [BHT+03] and [Fal03] other problems with Internet protocols in challenged networks are also mentioned: In IP routing the forwarding function will normally drop a packet if a next-hop route is not available. This is a problem for links that are frequently disconnected. The connection negotiation, retransmission and network congestion handling mechanisms of TCP also make it unsuitable for use on intermittent links.

In [Fal03] an early version of the DTN architecture is described. At the time of

writing the latest version is a work in progress Internet-draft [CBH+05]. The issues of Convergence Layers, Time Synchronisation, Security and Congestion and Flow Control are all discussed in the first work.

In [Fal04] the use of DTN as a generalised messaging service is described. Electronic messaging is increasing in popularity and a wide variety of hardware platforms operating on different networks have to support messaging applications. The general concept of asynchronous messaging is helpful in tolerating delays in networks and the DTN architecture with routing and fragmentation can offer reliability and generalised naming across different types of networks. DTN has the additional benefit of handling network disruptions. DTN is compared to existing messaging systems - E-mail, NNTP, SMS/MMS and Instant Messaging. It is concluded that the most appropriate basic abstraction for a messaging service is asynchronous, eventual message delivery and that DTN can be the basis architecture for such a service.

2.5.2 DTN routing

The problem of routing in a DTN is addressed in [JFP04]. The DTN routing problem is defined and the important differences from traditional routing are described. The DTN network is viewed as a time-varying multigraph where contemporaneous end-to-end paths may never exist. Algorithms that require different levels of information about the entire network are described and evaluated. It is concluded that focus on predictable communication opportunities in frequently disconnected networks is important and that there is a need for smart routing algorithms in network situations where network resources are limited. It was also found that global knowledge of a challenged network may not be required for good routing performance.

A different approach to routing in a DTN is presented in [WJMF05]. Erasure-coding based replication of message blocks is compared to different replication strategies. Routing in a DTN using these methods is simulated with trace data extrapolated from a wildlife monitoring project. The authors find that the erasure-coding based approach significantly improves the worst case delay, but has no ”very small delay” cases. It is noted that the erasure-coding was evaluated in a setting where all nodes are equally good relays and that in a more complex network a more sophisticated approach might be needed.

2.5.3 DTN and sensor networks

It is widely suggested ([Fal03], [DVA+04], [HF04]) that sensor networks are an important application area for DTN. In [DVA+04] different methods for connecting sensor networks to TCP/IP networks are discussed. Three different methods are suggested - proxy architectures, delay tolerant networking and running TCP/IP on the sensor nodes. The first involves a sensor-network specific proxy that is used to translate the sensor network communication protocols to TCP/IP. The disadvantage of this is that the proxy is specific to one sensor network and there exists no general mechanism to route messages between proxies. DTN is viewed as a generalisation of the proxy architecture and provides the possibility of multiple

proxies for one sensor network, with routing between them. Multiple proxies also solves the problem of a single point of failure between the TCP/IP network and the sensor network. The third option discussed is that of using TCP/IP in the sensor network. This will be convenient, but presents some difficult problems like header overhead and energy inefficient TCP retransmission strategies. It is concluded that a combination of the three methods would likely be the best solution.

The application of the DTN architecture specifically to sensor networks is explored with some real world application examples in [HF04]. The DTN architecture is presented as an appropriate framework for the diverse network conditions in sensor networks. Absence of network infrastructure, network interruption and heterogeneity in networks are identified as the main challenges in sensor networks. The ZebraNet project for wildlife monitoring is cited as an example. In ZebraNet epidemic routing and in-network storage is used to distribute data to a mobile base station. DTN would be suitable for configuring the base station as a data mule, reducing the need for human presence. In a heterogeneous sensor network deployed by UCLA researchers in Mexico, multi-hop routing and a data mule is employed. Stationary nodes pass on messages according to buffer availability and proximity to the roaming data mule. The authors remark that DTN also supports buffer management and multi-hop delivery of data and that knowledge of the data mule schedule could make low-power node operation possible. It is concluded that DTN can contribute to sensor networks by connecting remote such networks to the Internet, as well as facilitating communication within the sensor network. It is suggested that an *interoperable subset* of DTN will be likely be implemented on sensor nodes, because of limited memory and communications resources.

2.5.4 DTN Implementations

The DTN Research group is active in developing a reference implementation of the current DTN Internet-draft [CBH+05]. An important goal of the reference implementation is that it be easily extensible and modular enough to enable experimentation. For example, it is intended to be relatively simple to ”plug in” different routing algorithms for experimentation.

In [DBF+04] the team that created the DTN2 reference implementation describes their experience in creating and evaluating the implementation. The difficulties encountered in exporting a consistent interface from the convergence layer up to the bundle layer from radically different lower layers are described. The authors remark that an earlier assumption that a TCP convergence layer is easy to implement turned out to be false. They conclude from this that the implication of intermittent network links span multiple system layers. The program structure of the implementation is presented and its designed thoroughly motivated. The implementation is evaluated to test the core value propositions about DTN's ability to operate in intermittently connected networks. DTN performance in compared to a simple file transfer program and the SMTP email protocol. The authors eventually describe the reference implementation as successful as validation of the DTN architecture and conclude that a store-and-forward message overlay network performs significantly better than existing approaches in challenging networks.

Students at the University of California at Berkeley implemented DTN on the

TinyOS platform for sensor nodes and describe their work in [PN03]. They present a simulation of custody transfer in sensor nodes, based on their TinyOS implementation. Querying and selection of the next best custody hop is investigated. It is concluded that it is possible to optimise this selection for one particular objective (energy or delay) based only on information local to a sensor node. The need for better selection policies that can optimise for both energy and delay is acknowledged.

CHAPTER 3

Research method

3.1 Goals: Towards DTN in a TCP/IP sensor network

The purpose of a Wireless Sensor Network is for nodes to collaborate to gather data from their sensors. These data will not normally be processed on the sensors themselves, but will be extracted from the network to a more powerful machine. The sensor network will often be in a different location from the more powerful processing machine or the people analysing the data and there might also be the need to upload updates or new code to the sensors. The natural way to form the connection between the two locations is to use the Internet.

For this solution to work it must be possible to connect the WSN to the Internet. The authors of [DVA+04] suggest three main ways to make this connection:

- Proxy Architectures
- DTN
- TCP/IP in the sensor network

If it is assumed that the sensor network does not use TCP/IP then there must be some translation between the sensor network protocols and TCP/IP on the Internet. This can be done with various proxy architectures, normally designed for a specific network. The DTN architecture described in Chapter 2 can be viewed as a more general case of a proxy architecture. DTN can operate over any lower network layer and serves as the bridge between different protocols, as shown in Figure 2-2. In the third case, when TCP/IP is used in the sensor network, sensor nodes can communicate directly with Internet hosts.

A combination of some of the three solutions is likely to be the best [DVA+04]. The Contiki operating system (Section 2.3) and μIP is a working solution to employing TCP/IP in a WSN. There are some known problems with TCP in challenged networks [DFS02] and we assume the WSN to be a challenged network. DTN is a proposed solution for connecting WSNs to the Internet and for mitigating the problems of challenged networks. The combination of DTN and TCP/IP in a WSN is the main goal of this work.

The main contribution of this project is the production of a prototype implementation of DTN for use in a TCP/IP enabled sensor network. There exists earlier, limited implementations of DTN for sensor nodes, but none implement a TCP convergence layer or focus on connectivity with the Internet.

The existence of a prototype implementation would serve two general goals:

1. Connect a TCP/IP WSN to the Internet through a DTN gateway
2. Test operation of DTN under the challenging WSN conditions

The first goal is significant because of the general problem described above. It is desirable to connect a sensor network to the Internet, even more so to do it in a general way. The existence of a DTN gateway to the Internet means there can be multiple gateways to the same WSN and they can perform DTN routing among themselves. If one gateway fails data from the WSN can be routed (with DTN routing) to another gateway and to any other DTN node on the Internet.

There exists a full implementation of DTN with a TCP convergence layer for use on PCs. This implementation can be used as a DTN gateway to the Internet. The existence of a compatible DTN with TCP in a sensor networks will make it immediately possible to effect DTN communication between the sensor network and the Internet. The sensor network DTN application will implement a *interoperable subset* of DTN functions.

The second goal is intended to test if DTN is a good way of organising communication in the presence of the specific challenges that a sensor network presents. Mathematical analysis of TCP and DTN to model their behaviour in a sensor network environment could be done to do these tests. An actual implementation of DTN makes the same tests possible, but is also useful for the first goal and for practical use. The DTN prototype can be used to measure real delays and resource use for typical sensor network tasks.

3.2 Method

To achieve the goals stated in the previous section the project is divided into three distinct phases. The first involved a literature study phase where related work was studied.

The second phase involved the design and implementation of the bundle protocol and DTN architecture on the Contiki operating system. The design stages involved the study of other DTN implementations, mainly [PN03] and [DBF+04]. The design needed to stay true to the principles of DTN while making compromises because of a constrained resource environment. The prototype was written in C using the "Minimal" (Linux) and Netsim (Contiki simulation) ports of the Contiki operating system. During development the limitations of the eventual destination platform (sensor boards) had to be kept in mind. Only Contiki and basic C library functions could be used. Even though functions like `malloc` can normally be used when compiling the C code for a sensor board platform, fragmentation of the limited (sometimes only 2048 bytes) RAM could have severe negative effects. The Contiki dynamic memory module is used instead.

The third phase was the experimentation stage. During this phase the prototype was used for setting up experiments to establish whether or not DTN functions as expected.

CHAPTER 4

Implementation

4.1 ContikiDTN architecture overview

The design of a suitable architecture for Contiki DTN implementation was a major part of this work. A suitable architecture would follow the bundle protocol and DTN architecture specifications while accommodating the restrictions of the Contiki operating system environment. I introduce ContikiDTN, a prototype DTN implementation for the Contiki operating system and sensor nodes. A layered component view of ContikiDTN is described in this section.

An overview of the ContikiDTN architecture, showing the different DTN components is shown in Figure 4-1.

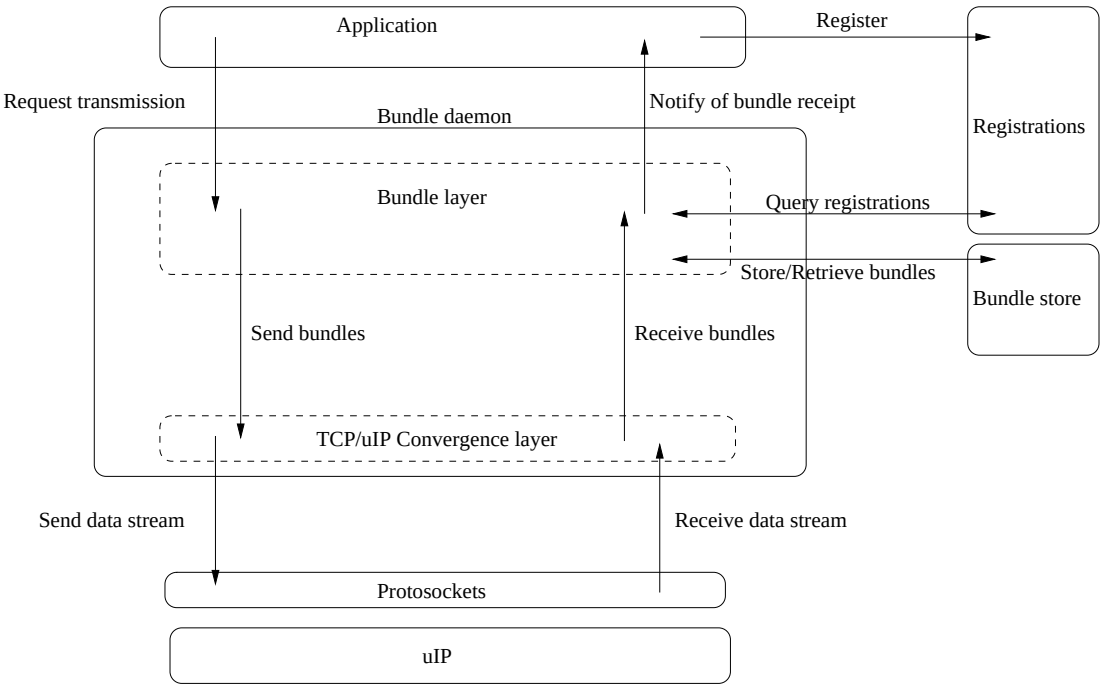

Figure 4-1: Architectural overview of ContikiDTN

The layer closest to the physical network is the Contiki μIP component. Above that is the protosockets interface. As described in Section 2.3, protosockets provide a programming interface to μIP that is similar to BSD sockets. This greatly simplifies

the implementation of the virtual layer above, the TCP convergence layer.

The bundle daemon makes up the overlay routing layer. In the daemon bundles are sent and received and routing and forwarding decisions are made. The ContikiDTN daemon contains a basic routing mechanism for static routing. Each DTN daemon assumes a unique local EID and is statically configured with routing information about other DTN nodes. The daemon further consists of the bundle layer and the convergence layer.

In the DTN architecture, described in Chapter 2 convergence layers are described as a general interface between the bundle daemon and the transport layer of the underlying network. One of the first design decisions in ContikiDTN was to integrate a single TCP convergence layer into the bundle daemon. In ContikiDTN convergence layers can not be changed or plugged in. Since protosockets only work with TCP, this was a logical design choice. The convergence layer is very much part of the bundle daemon and therefore depicted as a ”virtual layer” in Figure 4-1.

The protosocket interface provides the convergence layer with a reliable send and receive TCP byte stream. Between the convergence layer and the bundle layer, bundles are serialised and deserialised from byte streams to logical bundles to be stored in the bundle store.

The bundle store provides persistant storage for bundles. This is a vital part of the store-and-forward architecture. In ContikiDTN the bundle store makes use of the primary memory available to Contiki and no secondary store is implemented.

Above the bundle daemon are DTN applications. These application make use of the services that the daemon exports. In ContikiDTN the daemon provides a bundle transmission service and a bundle delivery service. A DTN application does not have direct access to bundles in memory. Before an application can receive bundles it must register its intent with the bundle daemon. The daemon checks the destination of incoming bundles for registered applications. Since the bundle daemon provides a general exported service it is easy to create different application that make use of the the bundle sending and receiving functions. Multiple applications can be registered with the daemon at the same time.

4.2 Implementation details

The layered architecture described above is the conceptual framework for the ContikiDTN implementation. The details of realising the architecture in the event-driven Contiki environment are described in this section.

The bundle daemon is primarily concerned with dealing with multiple TCP connections for receiving, sending or forwarding bundles. These bundles are kept in the bundle store and must be updated and forwarded as TCP/IP and system events arrive. The interaction between the Contiki event kernel, the bundle daemon and the bundle store is shown in Figure 4-2.

The global list of bundles in the bundle store is periodically processed and appropriate actions are taken based on the current status of the bundle. In reaction

to different bundle states events are posted to the system to cause the processing required.

Much of the processing will require the use of Contiki protothreads. Protothreads and protosockets are "driven" by repeated calls to the function that represents the specific thread or socket - each time the function is called the thread or socket has the opportunity to resume execution. During ContikiDTN operation events arriving at the DTN daemon must be identified and organised by the bundle, connection, timer or application that they are related to. This is done by an event handler function, as shown in Figure 4-2. The event handler function will make the function call needed to drive the protothreads and sockets.

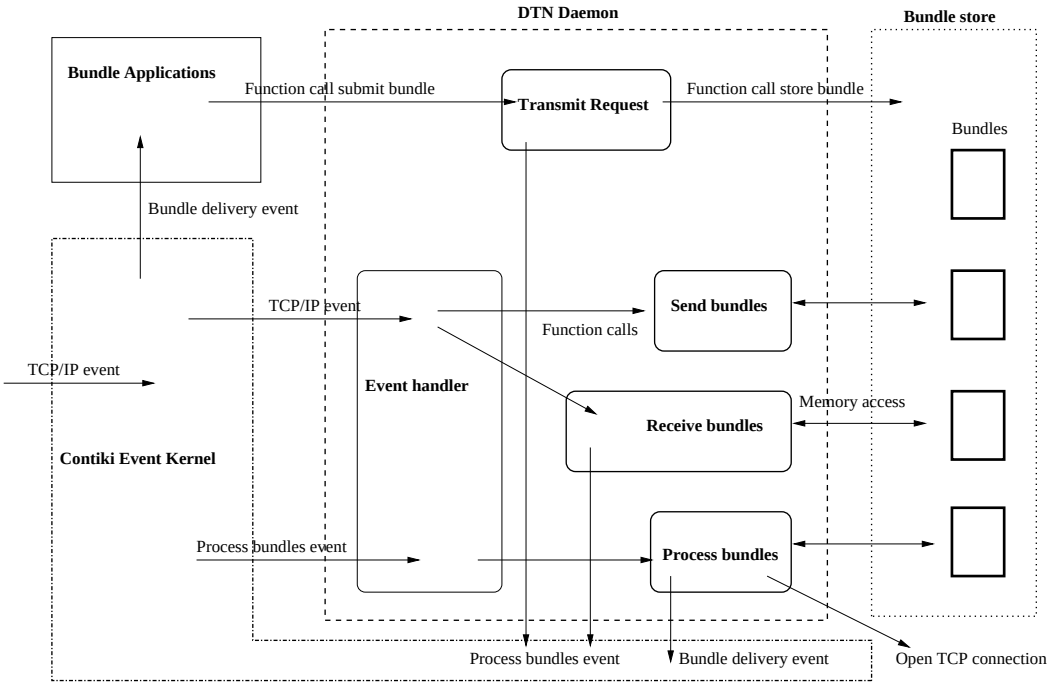

Figure 4-2: Events and communication flow in ContikiDTN. Functions within the DTN daemon are shown and arrows indicate event posting or function calls.

The most important form of bundle processing relates to sending and receiving bundles using the TCP convergence layer. When a TCP connection is made in ContikiDTN a memory structure of connection state is associated with the connection. A protosocket function is then started to handle the processing of this connection. When a blocking protosocket call is made in the protosocket function the protosocket's execution is interrupted and the system returns to a state where it can receive events. The term blocking call is used to refer to this interruption of execution and return to an idle state. When a TCP/IP event then arrives it is again associated with the appropriate connection state and the previously interrupted protosocket function resumes execution at the blocking call. One example of this is when reading data from a protosocket. A blocking call is made to the protosocket read function and execution leaves the protosocket function. When data is received on the socket μIP notifies the application with an event passed through the kernel. The application event handler recognises the event as being

associated with a connection receiving bundles and calls the receiving protosocket again where it resumes execution at the blocking call.

This protothreaded handling of TCP/IP events is ideal for the DTN TCP convergence layer. The function of the convergence layer is to receive bundles from the bundle layer and take care of their transmission, as described in Section 4.1. The destination of a bundle transmission might not be available, causing a TCP connection to time out. If normal blocking system call were used this would mean that the system waits at the blocking call until the connection eventually times out. This is undersirable behaviour, as other bundles might be arriving and require processing. In Contiki this problem is elegantly solved by the immediate return of the blocking call, causing the system to be able to respond to any events.

To further clarify the working of the convergence layer with Contiki, the processes of sending, receiving and forwarding a bundle are described with the aid of Figure 4-2.

Sending a bundle

The sending application makes a function call to the DTN daemon, requesting the transmission of a bundle and providing the necessary parameters that should be sent in the bundle. The bundle daemon then creates a bundle using the bundle store and places it in the global list of bundles, marked as requiring dispatch. The Transmit Request function then posts a Process bundles event. This event is received by the daemon event handler, which will call the Process bundles function. This function will act on the global list of bundles in the bundle store and step through each bundle to determine if its status requires action to be taken. The submitted bundle will be marked for sending and process bundles will then open a TCP connection to destination acquired from a route lookup. Process bundles will then return as μIP connects the TCP connection. When the connection is established a TCP/IP event will come up through the kernel to the Event handler. This event will be associated with a connection that is associated with a bundle that has to be sent. The Send bundles protosocket function will be called and will reach the protosocket send function that will block as described in the previous section. The sending protosocket will finish when the data has been serialised, sent and acknowledged. It is important to note that the sending process can take as long a required by the challenged network, the system is tolerant to delay throughout the process.

If the connection could not be established the application will initiate an exponential backoff retry mechanism.

Receiving or forwarding bundle

A receiving application will be registered with the DTN daemon. Incoming data on the listening port will cause a TCP/IP event being passed up to the DTN daemon. The Event handler will respond by creating a connection and calling the receive bundles protosocket function. The receive bundles function will block while waiting for the data to arrive. Again delays are acceptable at this stage. When a TCP/IP event for this connection indicates that data has arrived the Event handler

will again call the receiving protosocket which will read the data. If the data contains a valid bundle it will be deserialised and stored in the global list of bundles. The receiving protosocket function will then post a Process bundles event. When the Event handler receives the Process bundles event the Process bundles function is called and works through the list of bundles. If a bundle is found with a destination EID that matches a locally registered EID a Bundle delivery event will be posted to all processes. The registered receiving application will receive this event and read the bundle from the pointer provided.

If the Process bundles function finds a bundle that can after consultation of the routing table, be forwarded a TCP connection is opened and the same procedure as for sending a bundle is followed from that point.

This process may be interrupted and continued at any point since the events can happend asynchronously and independent of each other.

4.3 Design discussions

In this section some of the difficulties and issues encountered in implementing the system are discussed.

4.3.1 TCP and retransmissions

As is evident from the Implementation Details, the Contiki event driven kernel is well suited to handling multiple TCP connections that are all subject to challenging network connections. The combination of the protosocket programming interface and the event driven kernel make the Contiki environment ideal for the asynchronous nature of DTN. This architectural match does, however, not solve all of the issues that have to be considered when dealing with TCP connections in an intermittently connected environment.

When a destination is unreachable TCP will continue to try and connect until a maximum exponential backoff is reached. Only after this maximum has been reached will the kernel, and then the DTN daemon, be notified that the destination is unreachable. This presents two problems. The first is that it is likely to be desirable to be notified of an unreachable node before the connection times out entirely. The second is what to do once it has been established that a node is unreachable. The first problem is alleviated by the fact that the system can continue with other tasks while the connection is attempted, and μIP will take care of successive attempts. This is not such a simple matter when using Unix sockets, as described in [DBF+04]. Other than making changes to the μIP TCP parameters, the application level DTN daemon cannot do much more than wait for TCP to time out. For the second problem, what to do once a connection has failed, the DTN daemon can try to be more intelligent when attempting successive new TCP connections. For this an application level retry timer is used in ContikiDTN. This timer operates on a larger timescale than the TCP retry. ContikiDTN will exponentially increase the retry timer every time a TCP connection fails to complete. A possible different strategy could be to start an application level timer at the start of every connection

and the explicitly terminate one TCP connection attempt and start another when this timer expires.

4.3.2 DTN and IP routing

As mentioned in Section 2.1.5, DTN routing is not described as part of the architecture and is an active area of research. The problem of routing in a challenged network is far from trivial, especially since the dissemination of routing information is also subject to the delays and disconnections in the network.

In the ContikiDTN implementation the goal was to get a basic working prototype of a DTN network in simulation. To send messages in this network basic routing was required. It was decided to implement static routing for use in the Contiki simulation environment (see 2.4). The first problem was to assign each instance of the DTN daemon (ie. each simulated node) a unique identifier from which to determine its own EID. A small modification to the Contiki Network Simulator provided an ID for this use. In the absence of a dynamic routing algorithm and without prior knowledge of the EID of nodes in the simulated network, static routing tables could not simply be entered before compilation. A neighbour discovery mechanism was introduced, but this was only useful for nodes directly next to each other. Eventually another modification to the assignment of EIDs to simulated nodes made them predictable for a fixed topology and static DTN routing could be set up at compile time.

IP routing was another issue, the Contiki Netsim provides a basic IP forwarding service that will forward any packet not recently seen. The list of recently seen packets was kept in a cache that had to be enlarged in order to cope with the load of packets in the simulation and the high probability of a node seeing a packet again in the experimental topologies.

4.3.3 Dynamic memory allocation

DTN is based on message switching. Messages, and therefore bundles, can vary greatly in size. The headers in die Bundle protocol are all of variable length. These facts make the memory requirements of a bundle protocol implementation variable.

The primary DTN header, shown in Figure 2-6 contains variable length fields. Self-Delimiting Numeric Values (SDNVs) is an encoding scheme adapted from the Abstract Syntax Notation and makes it possible to encode and delimit values of variable size. SDNVs are used for the header length and the dictionary length. The dictionary itself is also a variable length part of the header. The dictionary is intended to make some basic header compression possible (see 2.1.6). The length of the dictionary and the SDNV fields are only known when a bundle is created or processed because these fields depend on the EIDs and other properties of the bundle. EIDs are also strings of variable length.

When locally creating and processing incoming bundles the bundle daemon needs to allocate memory for the header and the payload in order to store them in the bundle store, this must be done using the Contiki memory management functions.

Contiki offers a simple memory allocation system that works with blocks of fixed size, the size being declared before compilation. Blocks can be dynamically allocated and freed, but the size remains fixed. There can also only be one declaration of one size of memory block per C module, because of namespace clashes.

The fixed size of Contiki memory blocks and the variable size of the DTN headers and messages present a serious problem on systems where memory is a scarce resource. In designing ContikiDTN fixed blocks of memory had to be allocated for variable sized elements. This means that some maximum value had to be decided for each variable length item. The easiest way to do this would be to choose a maximum that is more than is likely to ever be needed, but in a memory constrained system this is not a feasible option. The compromise was to build a slightly less flexible system that uses fixed size memory blocks of average size. Average here meaning roughly halfway between the minimum and maximum size a field is likely to need. For example, for EIDs string length was fixed at 30 bytes, bundle payloads at 1024 bytes and SDNV fields were allocated the number of bytes the locally stored value would take (the SDNV representation of the locally stored value might take up fewer bytes - a `long int` of 4 bytes containing a small value could produce a SDNV of a single byte).

From the need for this compromise it can be concluded that any DTN implementations will benefit from the availability of fully dynamic memory allocation that could lead to more efficient use of the available memory.

4.3.4 Code size

The memory issues mentioned in the previous section have a direct effect on the size of the compiled code and its memory requirements. Contiki's memory management makes it necessary to declare all memory usage at compile time.

A sensor in use at the Swedish Institute of Computer Science is the TI MSP430 / FU Berlin sensor board. This sensor has 2048 bytes of RAM and 60 kilobytes of flash ROM. It can measure vibration, temperature, IR-light, sound, tilt and motion [Dun05].

ContikiDTN has a compiled code size of approximately 14 KB and, a data segment size of 64 bytes and a bss segment size of about 11 KB. Even though the ContikiDTN prototype is larger than can be used on the MSP430 sensor, better memory management and tweaking of static memory declaration can dramatically reduce its size. The number of connections that can be handled, bundles that can be stored and size of the incoming data buffer are all statically declared and can easily be changed. If the ContikiDTN daemon need only handle 1 bundle and 2 connections at a time (changed from 5 and 5 above), the bss segment size falls to 7 KB.

4.3.5 Time synchronisation

As discussed in 2.1.8 DTN nodes need synchronised clocks. One of the goals of DTN is that it must be able to run over different lower network layers on potentially very different hardware platforms. This goal is important in this project because the use of DTN for connecting a sensor node to a PC and the Internet is

investigated.

If bundles are transmitted from a ContikiDTN node to a PC running another DTN implementation the clocks on these two platforms must agree so that, for example, bundles can be expired at the correct time. ContikiDTN makes use of the Contiki function `clock_time()` to set timestamps in created bundles. In the Contiki simulation and on a Linux platform compatibility was not a problem, since the clock time was derived from the C library function `gettimeofday` in all cases. Care must be taken when communicating with other platforms that might have different clocks and this highlights the importance of this part of the DTN architecture.

4.3.6 Compatibility with DTN2

Time synchronisation was not the only compatibility issue when connecting ContikiDTN with other versions of DTN. ContikiDTN was designed to be compatible with DTN2, the DTN reference implementation [DTN05, DBF+04]. DTN2's TCP convergence layer makes use of a DTN handshake before bundle transmission. ContikiDTN also implements this handshake at its TCP convergence layer. The handshake involves the exchange of contact headers containing information about the type, unique ID, length and processing parameters that apply to the bundle to be sent. It also serves to establish a two way connection for the exchange of a bundle. The DTN messages exchanged are shown in Figure 4-3.

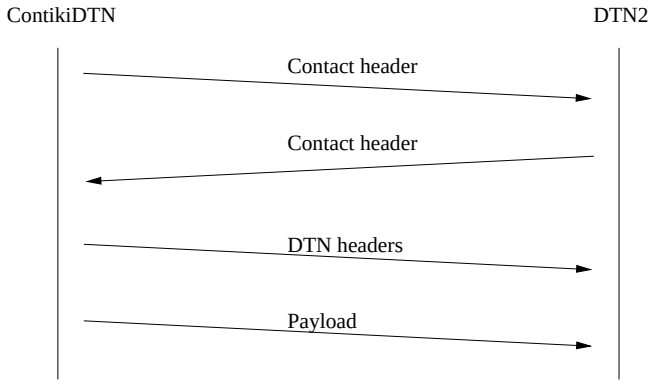

Figure 4-3: DTN level message exchange using TCP between two DTN2 compatible DTN nodes

It is not clear why the authors of DTN2 decided to add this handshake to the TCP convergence layer. It made additional work to ContikiDTN necessary in order to test using DTN2 as a gateway to the Internet. Even though the details of convergence layers are not specified in the architecture or the bundle specification this modification limits immediate interoperation between different DTN/TCP implementation.

CHAPTER 5

Analysis and Evaluation

5.1 Implementation analysis

The ContikiDTN implementation was successful as both a proof of the feasibility of implementing the Bundle protocol on Contiki sensor nodes and as a tool for evaluating DTN as a sensor network solution. In this section the implementation is analysed and some experiments and results in simulation presented.

The implementation was successful in its goal to enable testing of DTN in a sensor network simulation. The implementation can asynchronously send and receive messages with TCP connections and store messages while a application level retry timer affects retransmissions - this represents the core functions of the DTN architecture. The ContikiDTN architecture design, discussed in Chapter 4, was found to be successful.

The event driven Contiki kernel provided a "ready made" solution to some potentially serious implementation difficulties. It is considered easier [vBCB03, DSV05] to reason about and design systems using threads and blocking system calls, as opposed to designing programs as explicit state machines for event-driven systems. The asynchronous nature of the DTN network and the possible intermittency in it make this an important consideration. Event-driven systems provide a natural way of dealing with concurrent tasks of unpredictable duration, such as transmitting a bundle using TCP in a intermittently connected network. The Contiki environment was found to be ideal for the TCP DTN convergence layer because of a combination of the advantages of an event-driven environment and the Protothread abstraction described in [DSV05]. Protothreads and Protosockets made it easier to write the program, while the event driven kernel simplified concurrent connection handling and retransmissions. This is a significant result considering the difficulties encountered by the authors of [DBF+04] in implementing a TCP convergence layer using Unix sockets.

The implementation was also successfully tested to be compatible with a contemporary version of the DTN reference implementation, DTN2 [DTN05, DBF+04]. The compatibility was verified for simple bundle transmission with and without link disconnections. This compatibility makes it possible to easily connect ContikiDTN sensor nodes to any DTN node on the Internet (see also Figure 2-2).

The compromises needed in realising the Bundle protocol in the Contiki environment are described in more detail in Chapter 4. The absence of fragmentation and full support for custody transfer did not make ContikiDTN less suited to the test-

ing goals, but removed the possibility of investigating questions surrounding these two relatively uninvestigated parameters. The implementation of an application level acknowledgement using Bundle Status Reports made experimentation with reliability possible.

The Contiki NetSim simulation environment was used to test bundle transmission with multiple nodes. It was necessary to tweak some of the memory parameters for experiments. Since a lot of statically allocated blocks of memory are used, increasing the capacity of the system also increased the memory footprint of ContikiDTN. This was acceptable for experimentation, but in a real sensor network the current system will only be able to handle small loads of bundles. For experimental purposes there could be up to 100 bundles in the system at a time, while 5 would be more realistic for ContikiDTN running on an actual sensor.

The ContikiDTN prototype is a solid starting point to using DTN as a general solution for message passing in a sensor network. The experimental goals in creating the prototype were achieved and insight into the architectural requirements for a Contiki and TCP DTN implementation was gained.

5.2 Experimental evaluation

5.2.1 Experimental setup

The goal of the experiments was to test the core value proposition of the DTN architecture - to verify that in the presence of disconnection and node failure, DTN will improve message delivery in a sensor network environment.

It is first necessary to define what is considered as improved message delivery. A commonly used measure is throughput. In these experiments a constant bundle payload size was used. Since throughput equals message size divided by time and the message size is constant, the transmission time is considered an adequate measure of throughput. A decrease in delivery time is considered an improvement.

In wireless sensor networks communication expends the largest amount of energy [ASSC02]. For this reason throughput is sometimes considered to be a lower priority than power consumption. A second measure used in these experiments is two rough measures of power consumption - the number of packets sent in the system and the number of bytes transmitted in the system. It is assumed that more packets and more bytes sent means higher power consumption by the nodes and that a lower number for this metric is an improvement in message delivery.

All experiments were designed for comparison of two different configurations for reliable delivery of a single bundle across multiple hops in a sensor network. The bundle loads (hundreds) used in the simulation are not realistic for normal use of a sensor network and therefore make the numbers produced for the various metrics not meaningful on their own, but meaningful when compared to the other experimental configuration.

The two configurations are depicted in Figure 5-1. Both use a linear topology. The first, called End to End, simulates the use of TCP and DTN to transmit a message across multiple IP hops to an endpoint. The intermediate nodes do not pass the

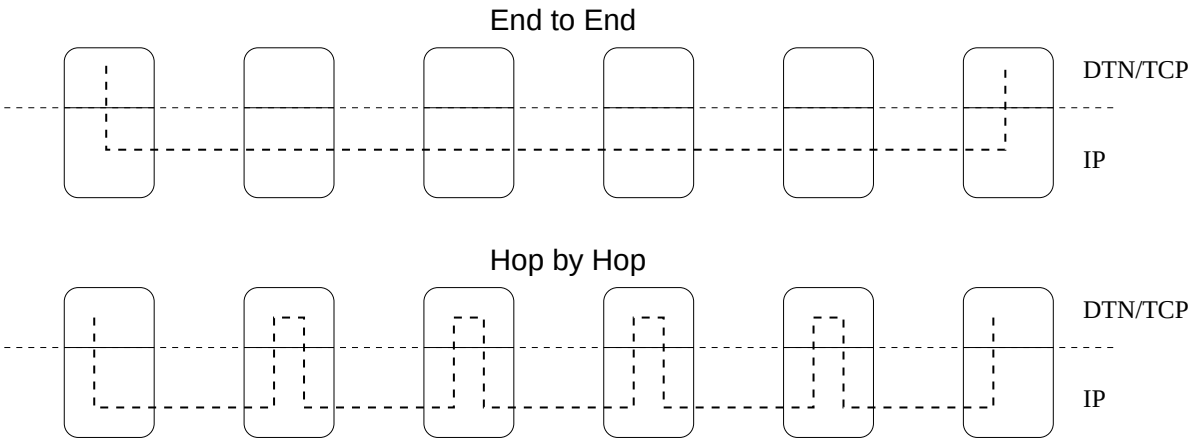

Figure 5-1: Experimental setup with End to End and Hop by Hop configurations

message up to the application layer and do not store it, the message is merely forwarded at the IP level. The end nodes run DTN daemons. This is the typical situation for a conversation on the Internet. This is also similar to the case if only TCP were used in the presence of multi-hop IP routing in a Contiki sensor network. In the End to End scenario the sending node keeps state about the TCP connection until the transmission is complete and all data has been acknowledged from the destination node.

In the second configuration, called Hop by Hop, DTN daemons are run on all nodes. The message is passed up to the DTN application at every hop and DTN messages are exchanged on a hop by hop basis. The TCP convergence layer is used at every hop, but messages are stored in the DTN message store before being forwarded on the next TCP connection. The original sender does not need to keep TCP state for the bundle after transmitting it to the next hop.

The difference between the two configurations focuses the experiment on the hypothesis that DTN will perform better and more cheaply in the presence of node failure by making use of hop by hop, rather than end to end TCP.

In all experiments it is assumed that multi-hop IP routing is available to all nodes in the system and that DTN routing and mapping to IP addresses is also available. For the experiments static routing tables were used.

Three categories of experiments are performed. The first is the control and compares End to End with Hop by Hop when there is no node failure. The second category of experiments introduces periodic node failure of one node in the linear chain of nodes. The performance of DTN is again measured and compared in the End to End and Hop by Hop configurations. Scheduled failure is introduced in a third set of experiments. In these experiments sending nodes are aware of the sleep schedule of the node they are sending to.

The experimental setup is shown in the Contiki simulation environment in Figure 5-2

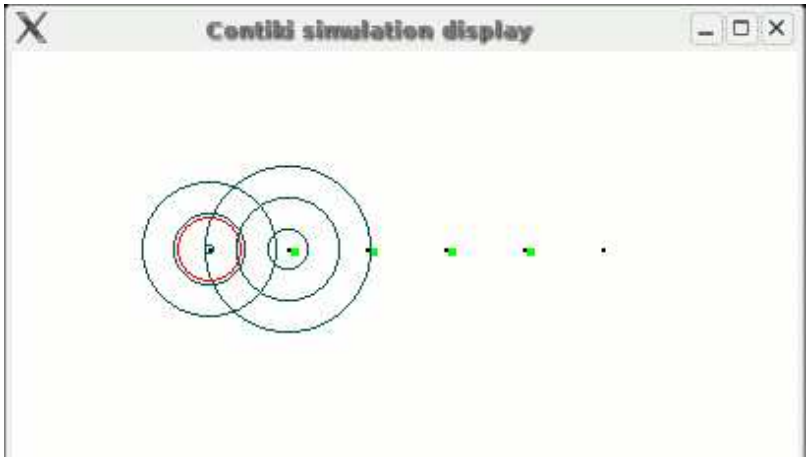

Figure 5-2: The Contiki simulation environment, showing the experimental setup

5.2.2 Experimental results

No node failure

The control experiment compared the transmission time, number of packets and bytes sent for the acknowledged delivery of 100 bundles. The results showed no significant difference in the number of packets sent or bytes transmitted, but as seen in Figure 5-3, the End to End setup had a faster transmission time. End to End took, on average, 42% of the time needed by Hop by Hop to deliver and acknowledge a bundle. This can be explained by the fact that each intermediate DTN node has to do the processing involved with storing and forwarding a bundle and also initiate and wait for a new TCP connection, while the intermediate nodes in End to End can only forward packets without passing them up to the application.

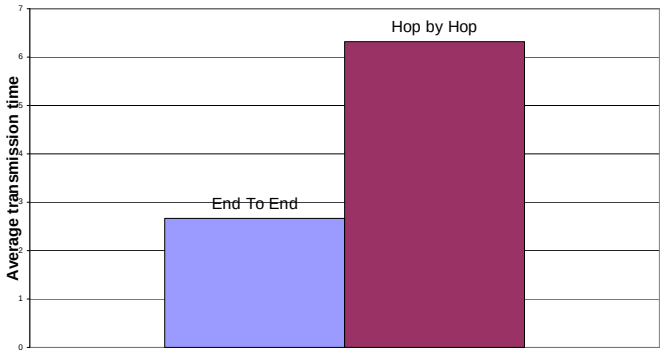

Figure 5-3: Relative transmission time without node failure

Periodic node failure

In order to compare End to End with Hop by Hop performance in a challenged network, periodic failure of the fourth node in the chain (the fourth node from the

left in Figure 5-1 and 5-2) was introduced. This meant that the fourth node in the chain would periodically shut down - for example, when the failure period is 4 the node would be active for 4 seconds and the dead for 4 seconds and repeat the cycle for the duration of the experiment. Bundles are sent at random intervals to make exposure to the failing node random. The hypothesis was that hop by hop DTN with store and forward messaging would be more efficient for reliable delivery of bundles across the network. The expectation was that retransmissions would take place from the end of the network in End to End while retransmissions could take place at the point of failure in Hop by Hop, resulting in improved overall efficiency.

Different periods of failure were used and each tested with 100 bundles of an equal size of 50 bytes. The same parameters as in the control experiment (average transmission time, packets sent and bytes sent) were measured. Since there is a processing delay in each node, the period of the failure can make a significant difference in system performance - if the failure period is very small and the failing node is only alive for a very short while at a time there might never be enough time to complete a bundle transmission. This was seen at a failure period of 1 system second - no bundles were acknowledged in the End to End configuration, while very few bundles were acknowledged in the Hop by Hop configuration.

The experiment was repeated for periodic failures of 2, 4, 8 and 16 seconds. The different failure periods introduced different levels of exposure to TCP timeouts and DTN retransmissions. The transmission times for acknowledged bundles are compared in Figure 5-4. It is clear that with a failure period of 2 seconds Hop by Hop achieves a significantly better throughput, while at the other failure periods End to End achieves a better throughput. A possible explanation here is related to the result in the control experiment - the storing and forwarding of Hop by Hop slows down the movement of the bundles compared to end to end, when there is no failure. When the failure period is longer this effect is more pronounced: When bundles do get through they get through quickly with End to End, but when failure is encountered in End to End it takes a longer for the bundle to eventually get through than in Hop by Hop. In order to further explore this possibility the standard deviation in the sample of bundle transmission times was calculated. With a node failure period of 2, the standard deviation for End to End was more than 6 times the standard deviation in the Hop by Hop data. For a period of 4 the End to End standard deviation was double that of Hop by Hop and by period 8 it was only slightly under double. At period 16 the ratio of standard deviation in End to End to standard deviation in Hop by Hop was about 1:1.2. This supports the theory that End to End will take much longer to recover from a disconnected link and therefore show greater variance in transmission time when failure is frequent. Hop by hop, on the other hand, maintains a much more even distribution of transmission times through node failure. This fits with the DTN notion that retransmission ”closer” to the destination improves performance in a challenged network.

As mentioned above, throughput and transmission time is not always important in a sensor network. The amount of packets and data sent is a rough indication of power use and therefore an important measure of efficiency. These metrics are compared under different node failure cycles between End to End and Hop by Hop in Figures 5-5 and 5-6.

A first observation is that for failure period 2, where there was a lower average

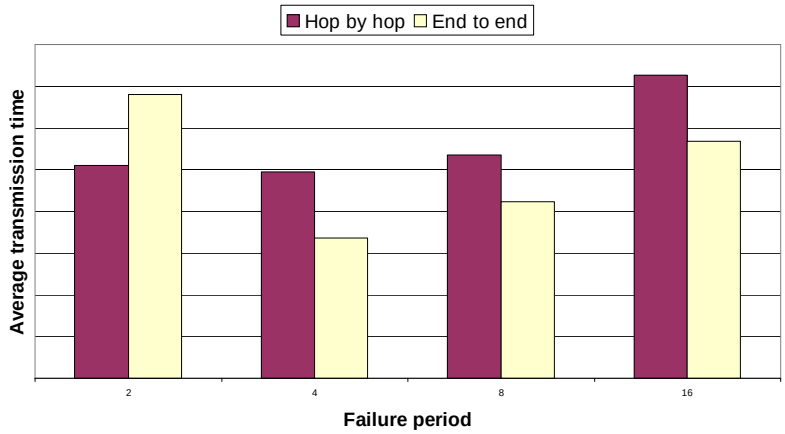

Figure 5-4: Relative transmission time with different node failure periods

transmission time, the packets sent and bytes transmitted is significantly higher for End to End. Even though there is a higher throughput, the transmission is less efficient. The same result is visible for all the other failure periods, the DTN hop by hop consistently uses a smaller number of bytes and packets to deliver and acknowledge the same number of bundles.

The number of packets and bytes remains more or less constant when the failure period increases in Hop by Hop, while the End to End numbers drop as the failure period increases. This could be another indication that Hop by Hop will provide more reliable, constant performance in a disconnected network, with no major fluctuations in bandwidth use when the failure pattern changes.

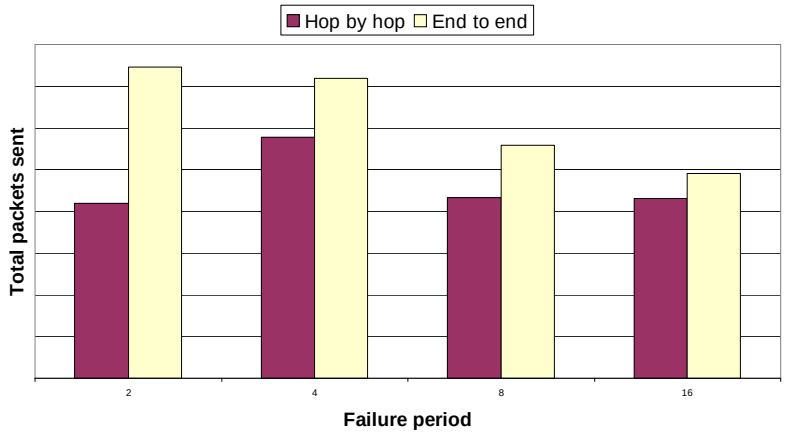

Figure 5-5: Relative total number of packets sent with different node failure periods

Scheduled node sleep times

In the previous experiment periodic node failure was simulated. In some WSNs it may be more likely that nodes will switch off their communication devices periodically to conserve power, rather than fail. It is reasonable to assume that other nodes

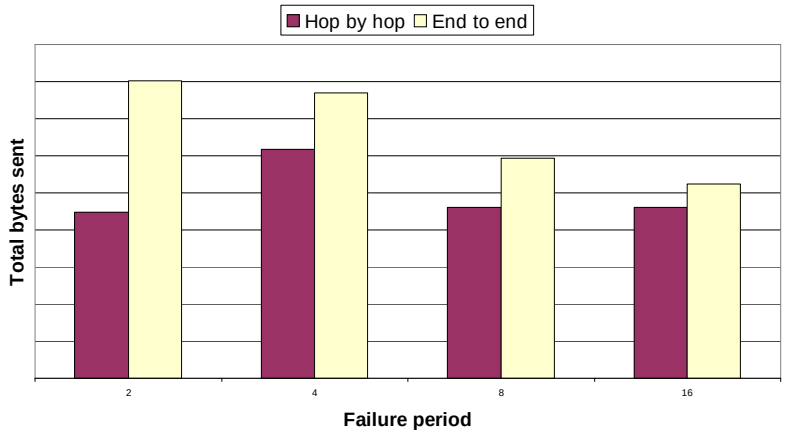

Figure 5-6: Relative total bytes sent with different node failure periods

in the network will be aware of this sleep schedule by using a clock synchronised with the sleeping node's clock. In this experiment the fourth node in the chain will periodically go to sleep, as in the previous experiment, but the neighbouring nodes will not attempt a bundle transmission during the sleep periods. The sleep period is 16 seconds and all other parameters are the same as in the previous experiments.

The hypothesis is that it will be more efficient to store the bundle at the neighbour of a sleeping node and continue transmission from there as soon as the sleep cycle ends. This will happen when using Hop by Hop DTN. In End to End the sending node will start a transmission regardless of the sleep schedule, but a node with a sleeping neighbour will not forward a packet to the sleeping neighbour. This type of scenario is considered to be likely in a WSN and it is expected that the the advantages of DTN will be more pronounced under these conditions.

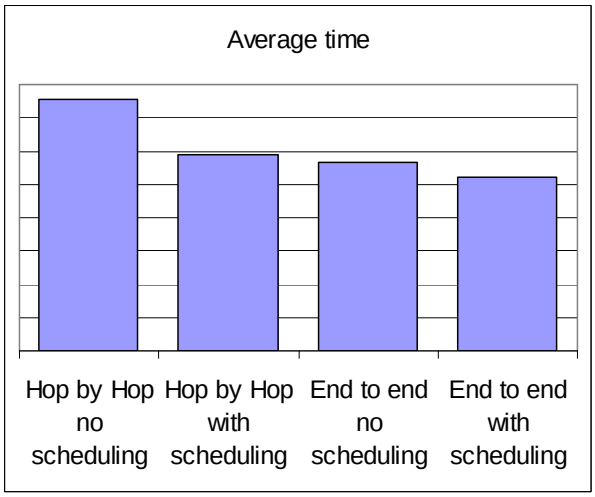

Figure 5-7: Throughput (average time) compared for the two different configurations, with and without scheduled sleep times

In Figure 5-7 the average transmission time for a bundle is compared in the different scenarios. In the previous experiments it was found that the Hop by Hop

throughput is lower, but more efficient in terms of packets and bytes sent. Here there is a clear reduction in the average transmission time for Hop by Hop - Hop by Hop with scheduling competes with the End to End configuration in terms of throughput.

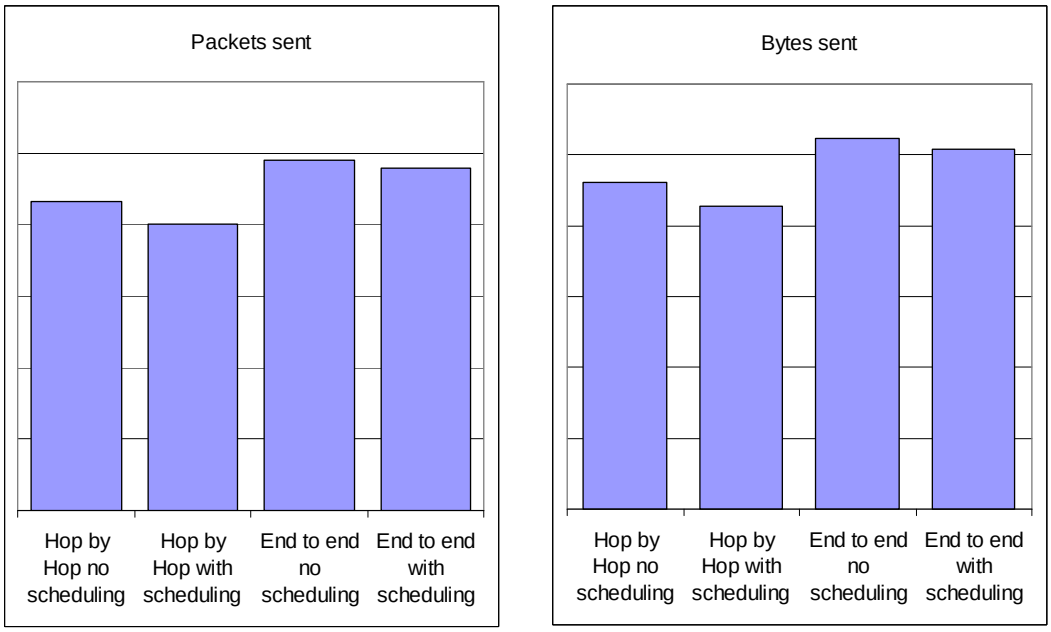

Figure 5-8: Packets and bytes sent

In Figure 5-8 the packets and bytes sent are compared for the different configurations. As in the node failure experiment, Hop by Hop DTN is in all cases more efficient overall in delivering the bundles. From Figure 5-8 it is also clear that there is a further slight reduction in the number of packets and bytes sent when nodes are made aware of the sleep schedule. This is in line with the expectation that more intelligent neighbour nodes (who know each other's sleep schedules) will increase the benefit of DTN. Hop by Hop with scheduling is therefore significantly more efficient than End to End without scheduling and offers comparable throughput.

CHAPTER 6

Conclusions and Future work

6.1 Conclusions

The implementation process, described in Chapter 4 and the experiments with Contiki DTN (Chapter 5) made led to several conclusions about DTN, Contiki and ContikiDTN.

The general design and debugging of an event-driven asynchronous system was found to be a challenging task. ContikiDTN went through several redesigns and the eventual success was due to a clear design of the flow of events and the crucial decision to handle all bundles in a single global list that is processed when certain events arrive.

The event-driven Contiki kernel was shown to be well suited to the asynchronous TCP messaging required in DTN. Contiki is an excellent environment for this type of architecture. It is not trivial to create a TCP convergence layer for DTN, but the Contiki protosocket interface made the design of the convergence layer much easier. Protosockets and protothreads were found to be a very valuable development tool. Protosockets are better suited for a TCP convergence layer than traditional, blocking BSD/Unix sockets.

The code size and memory use of ContikiDTN proved that it is feasible to use the DTN architecture on tiny sensors. This proof of concept is important as an argument for the advantages of a general (as opposed to network specific) messaging solution such as DTN. Communication problems in a WSN and to the Internet can be solved on a case by case basis with solutions designed for a specific environment, but DTN can be implemented on any network. DTN in a WSN will make the WSN compatible with potentially any network on the Internet through the DTN overlay.

ContikiDTN was successfully used to exchange bundles with the DTN2 implementation of DTN, both using TCP convergence layers. This served as proof that already existing DTN implementations (DTN2 and ContikiDTN) can be used to connect a sensor network to the Internet as described earlier.

Valuable lessons were learned from the Contiki implementation process. Fully dynamic memory allocation would be an extremely useful addition to the Contiki operating system, but the overall memory needed for a basic DTN implementation is not excessive, even with dynamic memory blocks of fixed size. The implementation confirmed the fact that DTN routing is a complex problem. The static routing used in ContikiDTN experimentation would need to be substantially expanded

to provide a framework for dynamic DTN routing. The Contiki event timers and clock was crucial in almost all aspects of the implementation. The bundle protocol relies heavily on timing and the clock and Contiki's easily portable interface to these function proved invaluable.

The Contiki network simulator was critical for the experiments and testing during development. The difficulty in debugging ContikiDTN with a process for each simulated node showed the general difficulty of debugging network code, but also highlighted some weaknesses in the Netsim. A debugging interface and an easier way to assign and retrieve information about nodes would be helpful. At the time of writing, a new Contiki simulator is in development.

The experimental results validated the core propositions of the DTN architecture. DTN outperforms TCP in an intermittently connected sensor network and the DTN architecture's use of hop by hop TCP produces better bundle delivery results than end to end TCP. The cost of retransmission is consistently lower using DTN. Using DTN will result in an overall power saving in a sensor network where communication uses a lot of power. In the presence of scheduled sleep cycles, using DTN causes a further reduction in the use of the transmission device, and therefore even lower power use compared to end to end TCP. Making sending nodes aware of the receiver's sleep cycle also improves throughput for DTN.

TCP has an important influence on DTN and the TCP convergence layer cannot easily be separated from the application. Experiments showed that TCP timeouts and their importance to decisions at the application level make delay tolerance a cross layer issue.

The bundle protocol is still an Internet-draft and being updated regularly. The major areas of change relate to routing and fragmentation and the simplification of naming. The bundle protocol implemented in ContikiDTN is mature enough to enable the main advantages of using the a delay tolerant overlay.

Even though fragmentation was not implemented in ContikiDTN it was clear that it is an important part of a DTN messaging system. ContikiDTN will retransmit a whole bundle if the transmission is interrupted in the middle and this can be made more efficient if fragmentation is done. Only the untransmitted part of the bundle can then be sent with retransmission.

6.2 Future work

There are many opportunities to build on and expand the results presented in this work. The obvious next step is to implement the full bundle protocol on Contiki. ContikiDTN currently does not support full custody transfer or fragmentation and reassembly.

The experiments done in simulation can be extended. The same experiments can be done with more elaborate node failure, different sized bundles, different topologies and different TCP behaviour. The DTN daemon's handling of TCP timeouts can be adjusted and experimented with. ContikiDTN can be used for these experiments with little modification, but time constraints prevented them being done in this work. The possible effect of other simulation parameters like the bundle pro-

cessing time and the rate of bundle transmission can be investigated. ContikiDTN can also be tested in a new Contiki simulation that is currently under development at the Swedish Institute for Computer Science.

ContikiDTN currently does not make use of secondary storage for storing bundles on nodes while they await processing or forwarding. Adding the use of a secondary store like EEPROM for long term bundle storage could make the network tolerant to extreme delays.

ContikiDTN has not yet been tested in a real sensor network. Uploading the code to several nodes and repeating the experiments under real world conditions would be useful as further validation of the conclusions drawn from simulation results and as tool for normal use of the sensor network.

A lot of research can be done on DTN routing. ContikiDTN used static routing in the network simulator and when testing with DTN2. Adding a dynamic routing protocol to ContikiDTN and investigating DTN routing in a sensor network would be very interesting. Applying existing DTN routing techniques in a sensor network is a logical next step.

As the bundle protocol specification matures it would be interesting to test the solutions proposed for fragmentation in a sensor network environment.

Security issues can also be experimented with in a sensor network. The additional computational and memory requirements of implementing the bundle security protocol might not be feasible, or even required, in a sensor network environment.

Bibliography

[ASSC02] AKYILDIZ I, SU W, SANKARASUBRAMANIAM Y AND CAYIRCI E. Wireless sensor networks: a survey. *Computer Networks*, vol. 38:pages 393–422, 2002.

[BHT+03] BURLEIGH S, HOOKE A, TORGERSON L, FALL K, CERF V, DURST B, SCOTT K AND WEISS H. Delay-tolerant networking: An approach to interplanetary internet. *IEEE Communication Magazine*, pages 128–136, June 2003.

[BLFM05] BERNERS-LEE T, FIELDING R AND MASINTER L. Uniform resource identifier (uri): Generic syntax. Request for Comments: 3986, STD: 66, January 2005. URL `ftp://ftp.rfc-editor.org/in-notes/std/std66.txt`.

[CBH+05] CERF V, BURLEIGH S, HOOKE A, TORGERSON L, DURST R, SCOTT K, FALL K AND WEISS H. Delay-tolerant network architecture. Internet-draft draft-irtf-dtnrg-arch-03 (work in progress), Jul. 2005. URL `http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-arch-03.txt`.

[Cla88] CLARK DD. The design philosophy of the DARPA internet protocols. In *SIGCOMM*, pages 106–114. ACM, Stanford, CA, Aug. 1988.

[DBF+04] DEMMER M, BREWER E, FALL K, JAIN S, HO M AND PATRA R. Implementing delay tolerant networking. Tech. Rep. IRBTR-04-020, Intel Research Berkeley, 2004.

[DFGV04] DUNKELS A, FEENEY LM, GRNVALL B AND VOIGT T. An integrated approach to developing sensor network solutions. In *Proceedings of the Second International Workshop on Sensor and Actor Network Protocols and Applications*. Boston, Massachusetts, USA, Aug. 2004. URL `http://www.sics.se/~adam/sanpa2004.pdf`. Invited paper.

[DFS02] DURST R, FEIGHERY P AND SCOTT K. Why not use the standard internet suite for the interplanetary internet? MITRE White Paper, 2002. URL `http://www.ipnsig.org/reports/TCP_IP.pdf`.

[DGV04] DUNKELS A, GRÖNVALL B AND VOIGT" T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*. Tampa, Florida, USA, Nov. 2004. URL `http://www.sics.se/~adam/dunkels04contiki.pdf`.

[DSV05] DUNKELS A, SCHMIDT O AND VOIGT T. Using Protothreads for Sensor Node Programming. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks*. Stockholm, Sweden, Jun. 2005. URL `http://www.sics.se/~adam/dunkels05using.pdf`.

[DTN05] Website: Code - delay tolerant networking research group, October 2005. URL `http://www.dtnrg.org/wiki/Code`.

[Dun05] DUNKELS A. Website: The contiki operating system, August 2005. URL `http://www.sics.se/~adam/contiki`.

[DVA+04] DUNKELS A, VOIGT T, ALONSO J, RITTER H AND SCHILLER J. Connecting Wireless Sensornets with TCP/IP Networks. In *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)*. Frankfurt (Oder), Germany, Feb. 2004. URL `http://www.sics.se/~adam/wwic2004.pdf`. (C) Copyright 2004 Springer Verlag. http://www.springer.de/comp/lncs/index.html.

[Fal03] FALL K. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34. ACM Press, New York, NY, USA, 2003.

[Fal04] FALL K. Messaging in difficult environments. Tech. Rep. IRB-TR-04-019, Intel Research Berkeley, December 2004.

[HF04] HO M AND FALL K. Poster: Delay tolerant networking for sensor networks. In *In the First IEEE Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004)*. October 2004.

[JFP04] JAIN S, FALL K AND PATRA R. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 34, pages 145–158. ACM Press, October 2004.

[LDS04] LINDGREN A, DORIA A AND SCHELN O. Probabilistic routing in intermittently connected networks. *Lecture Notes in Computer Science*, pages 239–254, Jan 2004.

[PN03] PATRA R AND NEDEVSCHI S. Dtnlite: A reliable data transfer architecture for sensor networks, 2003. Project report for CS294-1, University of California, Berkeley.

[RSPS02] RAGHUNATHAN V, SCHURGERS C, PARK S AND SRIVASTAVA M. Energy aware wireless microsensor networks. *IEEE Signal Processing Magazine*, vol. 19(2):pages 40–50, Mar. 2002.

[SB05] SCOTT K AND BURLEIGH S. Bundle protocol specification. Internet-draft draft-irtf-dtnrg-bundle-spec-03 (work in progress), July 2005. URL `http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-spec-%03.txt`.

[SFW05] SYMINGTON S, FARRELL S AND WEISS H. Bundle security protocol specification. Internet-draft draft-irtf-dtnrg-bundle-security-00 (work in progress), June 2005. URL `http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-secur%ity-00.txt`.

[vBCB03] VON BEHREN R, CONDIT J AND BREWER E. Why events are a bad idea. In *HotOS IX*. May 2003.

[WJMF05] WANG Y, JAIN S, MARTONOSI M AND FALL K. Erasure-coding based routing for opportunistic networks. In *SIGCOMM '05 Workshops*. ACM Press, August 2005.