RELIABLE BULK DATA DISSEMINATION IN SENSOR NETWORKS

by

Leijun Huang
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

Sanjeev Setia — Dr. Sanjeev Setia, Dissertation Director

Dr. Hakan Aydin

Brian L. Mark — Dr. Brian L. Mark

Dr. Daniel A. Menascé

Robert Simon — Dr. Robert Simon

Sanjeev Setia — Dr. Sanjeev Setia, Department Chair

Dr. Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering

Date: 12/6/07

Fall Semester 2007
George Mason University
Fairfax, VA

Reliable Bulk Data Dissemination in Sensor Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Leijun Huang
Master of Science
Beijing University of Posts and Telecommunications, 1999
Bachelor of Science
Xi'an Jiaotong University, 1996

Director: Dr. Sanjeev Setia, Professor
Department of Computer Science

Fall Semester 2007
George Mason University
Fairfax, VA

Acknowledgments

I would like to thank the following people who made this possible.

I would first like to thank my advisor, Dr. Sanjeev Setia, for his guidance and support.

I would also like to thank Dr. Hakan Aydin, Dr. Brian Mark, Dr. Daniel Menasce and Dr. Robert Simon for serving on my dissertation committee.

I would also like to thank the faculty and staff in Department of Computer Science at George Mason University for their teaching and service.

I would also like to thank my lab colleagues, Emerson Farrugia, Ghada Alnifie, Muhammad Abdulla and Bo Zhang, for their help and cooperation.

Table of Contents

List of Tables

List of Figures

Abstract

RELIABLE BULK DATA DISSEMINATION IN SENSOR NETWORKS

Leijun Huang, PhD

George Mason University, 2007

Dissertation Director: Dr. Sanjeev Setia

A wireless sensor network consists of a large number of resource-constrained sensor nodes that are self-organized into a multi-hop network and cooperate on a single task. In many situations, sensor networks need to run for a long time once deployed. When the environment changes during their lifetime, updating the code image or application data at the node for a new task becomes necessary, thus making data dissemination a critical issue where a large data object needs to be reliably propagated to all of the nodes in a network. While most of the current sensor nodes are equipped with a multiple-channel radio, the existing data dissemination approaches such as Deluge [1] do not take advantage of multiple channels. Moreover, these approaches mostly focus on the object delivery latency, while energy efficiency is also very important due to the resource constraints of the sensor nodes. This dissertation proposes three novel protocols for reliable bulk data dissemination, named McTorrent, CORD and McCORD, that focus on both object delivery latency and energy efficiency. These protocols use multiple channels, or a core-based two-phase approach, or both techniques to reduce object delivery latency and energy consumption at each node. The results from experiments on both indoor and outdoor testbeds and extensive simulations in various scenarios show that these protocols significantly reduce the latency and/or energy consumption, compared to the existing approaches.

Chapter 1: Introduction

A wireless sensor network consists of a large number of resource-constrained sensor nodes that are self-organized into a multi-hop network and cooperate on a single task. Examples of sensor network applications include environmental monitoring, perimeter surveillance and product inventory maintenance.

In many situations, sensor networks need to run for a long time once deployed. When the environment changes during their lifetime, updating their code image (the software that they are running) for a new task becomes necessary. Due to the large number of the deployed sensor nodes and the environment where physically collecting previously deployed nodes is either very difficult or infeasible, the code image needs to be updated "over-the-air", using the existing network itself.

In addition to the code image, application data may also require updating. For example, in a target detection application, the sensor nodes use the image of the target stored in their memory to detect the target. When the target changes, the image of the new target needs to be propagated to the network.

The critical service required for these purposes is a *reliable bulk data dissemination* protocol, i.e., a protocol for reliably distributing a large object to *all the nodes in the network.*

The reliable bulk data dissemination problem is a special case of the reliable downstream data transport problem for sensor networks [2–4]. The main challenge in bulk data dissemination is that the size of the object being distributed is two to three orders of magnitude larger than the size of a data packet, and is also much larger than the amount of RAM available on a typical sensor node. For example, the code image for a current generation sensor node such as a TelosB mote [5] could be as large as 48 KB, whereas the default size of packets transmitted by a TelosB is 36 bytes, and the amount of RAM available on a TelosB is 10 KB. Another challenge in reliable bulk data dissemination is that the object being disseminated has to be delivered to all the nodes in the network in as small a time as possible in order to minimize the service interruption to the deployed application. Moreover, the energy expense of the dissemination should be minimized in order to minimize the impact on the lifetime of the sensor nodes.

The objective of this dissertation is to propose energy-efficient, low latency protocols for reliably disseminating large data objects to every node in a sensor network.

Several protocols have been designed for reliable bulk data dissemination in sensor networks, such as MOAP [6], Deluge [1] and MNP [7]. A common characteristic of these protocols is that the data object is propagated from the base station to the rest of the network in a neighborhood by neighborhood fashion, i.e., nodes that receive the object (or part of the object) become sources for distributing the object to their neighbors who are further downstream in relation to the base station. In these protocols, the data propagation wave moves from one neighborhood to the next, after the object has been disseminated to *all the nodes* in the first neighborhood. While these protocols achieve reliable bulk data dissemination, there is still much space for improvement. First, in these approaches, retransmissions attributable to hidden terminal collisions are high, especially in sensor networks with high density. Second, the overhead of control packets during the dissemination takes a nontrivial portion of total transmissions. Third, most of these approaches require sensor nodes to keep their radios on at all times during the dissemination, consuming a large amount of energy on idle listening.

This dissertation proposes novel protocols for reliable bulk data dissemination that address these limitations of existing protocols. These protocols use the following techniques (separately or combined) to achieve this purpose.

- **Using multiple radio channels**

  Previous protocols [1, 6, 7] use a single channel for all packet transmissions, while current generation sensor nodes are usually equipped with multiple-channel radios. With a single channel, nodes have to compete with all of the other neighboring nodes for the channel before starting transmissions, otherwise collisions may occur if two or more of them are sending simultaneously. Moreover, hidden terminal problems occur where two nodes that are two hops away send packets at the same time, causing collisions at the nodes in between.

  With multiple channels enabled, nodes in a neighborhood can send packets simultaneously without causing collisions, as long as they use different frequencies and the gap between any two frequencies is large enough so that the interference between frequencies is negligible. Moreover, since more transmissions can proceed at a time by using multiple channels, the overall throughput is increased, leading to a reduced latency of data dissemination.

  On the other hand, using multiple channels also requires more complicated coordination between nodes. This dissertation addresses the challenges in using multiple channels, and presents a protocol that improves latency and energy efficiency over single channel approaches.

- **Using core-based multiple phase approach**

  In existing protocols, data are disseminated in a neighborhood by neighborhood fashion. Data have to reach every node in a neighborhood before being forwarded to the next neighborhood downstream. In such protocols, nodes with poor connectivity with neighboring nodes may delay the propagation of the data to the next neighborhood. Moreover, previous research [8] has reported that using a core-based approach where only a subset of nodes are selected as senders in the dissemination can significantly reduce collisions, especially in dense networks.

This dissertation proposes a core-based two-phase dissemination protocol, in which the data object is disseminated hop by hop to a subset of the nodes (approximately a connected dominating set of the network) in the first phase, and from those nodes to the remaining nodes in the second phase. One main feature of this two-phase approach is that it is incorporated with coordinated sleep scheduling so that nodes can turn radio off and sleep to save energy when they do not send or receive, which occurs at most time during the dissemination. One challenge for the core-based two-phase protocol is to achieve latency comparable to the previous protocols, which is addressed in this dissertation.

In summary, this dissertation proposes three novel protocols for reliably disseminating large data objects to large sensor networks in an energy efficient fashion. The three protocols, named McTorrent, CORD and McCORD respectively, adopt various techniques such as using multiple channels, multiple phases, pipelining and sleep scheduling, to minimize energy consumption and object delivery latency in data dissemination. These protocols are evaluated with previous approaches through experiments on indoor and outdoor sensor network testbeds and extensive simulations. The results show significant performance improvement in energy consumption and latency, compared to previous approaches.

The rest of the dissertation is organized as follows. Chapter 2 discusses related work. Chapter 3 presents the McTorrent protocol. Chapter 4 presents the CORD protocol. Chapter 5 presents the McCORD protocol. Chapter 6 evaluates the performance of the three protocols with existing approaches through experiments and simulations. Chapter 7 summarizes the results obtained in this dissertation and discusses open issues for future work.

Chapter 2: Background and Related Work

In this chapter, we review previous work related to reliable bulk data dissemination.

2.1 Reliable Multicast over Wired Networks

In the wired networks, reliable bulk multicast protocols are needed for two classes of applications. The first category includes the delivery of multimedia streams, in which total reliability is not required. In this category of applications, missing one or two packets occasionally reduces the quality of the multimedia service at the receivers, but does not render the service useless. On the other hand, time sensitiveness is a critical factor in such applications; packets received too late have no value.

The second category of applications includes the delivery of software packages, etc, in which total reliability is the primary requirement. The delivery time is not a critical consideration, but a small latency is desirable.

Many protocols have been proposed for reliable multicast in wired networks [9–15]. The difference among them is mainly the use of different flow and congestion control mechanisms and reliability mechanisms. The flow and congestion control mechanisms are used to achieve an optimal transmission rate from the sender such that congestion in any multicast paths is minimized and thus the number of retransmissions due to packet loss is also minimized.

The reliability mechanisms can be divided into three categories: sender-initiated (ACK based), receiver-initiated (NACK based) and hybrid techniques. In a sender-initiated mechanism, each receiver sends an ACK to the sender when the former receives a new packet. ACK implosion may occur at the sender when there are many receivers. Hop-by-hop ACK aggregation is imposed along the multicast path to mitigate the ACK implosion problem in some protocols [15]. Another disadvantage of a sender-initiated mechanism is the sender has to know the set of potential receivers or the immediate next hop nodes if ACK aggregation is used.

A receiver-initiated mechanism places the responsibility of reliability on the receivers. Each receiver detects packet losses and sends NACKs to the sender for retransmissions of the lost packets. NACKs are suppressed to avoid multiple NACKs requesting for the same packets and thus to mitigate NACK implosion at the sender [12,13]. One disadvantage of a receiver-initiated mechanism is the sender does not know if the receivers have received the data and thus has to keep the sent data in the buffer for a certain time.

Some protocols such as [14] use the combination of ACKs and NACKs to combine the advantages of each approach while avoiding their disadvantages.

As an error control technique to help achieve transmission reliability, Forward Error Correction (FEC) is used in some protocols [16]. To use FEC, packets are transmitted in blocks. For each block of k data packets, additional $n-k$ parity packets ($n > k$) are computed in such a way that given any distinguished k of the n packets, the original k data packets can be recovered. Therefore, for receivers that have lost different packets of a block, the sender may only need to send one parity packet to recover all of the receivers instead of sending each individual lost packet.

FEC can be applied in proactive mode or reactive mode. In the proactive mode, the sender sends the parity packets together with the original data packets, based on certain redundancy rate ρ, where $\rho = (n-k)/n$. A receiver reconstructs the original k data packets as long as it receives at least k distinguished packets in one block. In reactive mode, at the first round the sender sends the data packets only. On reception of the NACKs from the receivers that observe packet losses, the sender sends parity packets. FEC may also be applied in combination of both proactive mode and reactive mode, where the sender sends some parity packets with the original data packets at the first round and sends additional parity packets upon request from the receivers.

While some ideas and design components might be borrowed, these protocols are not suitable for data dissemination in sensor networks due to the many differences between wired networks and sensor networks:

First, sensor nodes are power constrained, which means all protocols used in sensor networks should be energy-efficient to maximize the lifetime. Energy efficiency is not an issue in wired networks.

Second, in wired networks, multiple applications coexist and compete for the bandwidth, and moreover, the bandwidth may be different at different parts of a network. Therefore rate control is an indispensable component of the protocol to avoid congestion and overloading while still achieving the optimal transmission rate. On the other hand, nodes in sensor networks share the broadcast media of homogeneous bandwidth. Moreover, a sensor network is usually deployed with one single application, each node in the network is designed to be cooperative in using the network resources. Therefore, rate control is usually not necessary, but collision avoidance becomes a primary consideration.

Third, in wired networks, packet losses are mainly caused by congestion. A packet loss should usually induce a reduction of transmission rate. In sensor networks, packet losses are caused by both the low quality of wireless links and packet collisions.

2.2 Reliable Multicast in Mobile Ad hoc Networks

Unlike wired networks, mobile ad hoc networks (MANETs) are self-organized wireless networks. Reliable multicast protocols for MANETs are usually ARQ-based, gossip-based or FEC-based [17]. ARQ-based protocols [18,19] use ACKs (for sender-initiated reliability) or NACKs (for receiver-initiated reliability) from receivers and retransmissions from senders to achieve reliability. In gossip-based protocols [20,21], packets are transmitted multiple times hop-by-hop so that the receivers can receive the packets with a high probability. FEC-based protocols [22] use similar mechanisms as those in wired networks.

While MANETs have similar characteristics as sensor networks such as wireless media and mobility, sensor networks are different in that sensor nodes have more limited computing capability, memory capacity and power budget, and the wireless links in sensor networks usually have higher loss rate. These constraints require that protocols for sensor networks should be more energy-efficient and more lightweight in sense of computation and communication complexity.

2.3 Reliable Data Dissemination in Sensor Networks

Protocols for data dissemination in sensor networks can be divided into two categories based on the size of the object being disseminated. Protocols in the first category are designed to exchange a relatively small amount of data among neighboring nodes. Protocols in the second category are designed for reliable bulk data dissemination.

2.3.1 Reliable Dissemination for Small Data Objects

One well-known protocol in this category is the SPIN family which uses a three-phase advertisement-request-data handshaking protocol to disseminate data [23]. SPIN takes advantage of the broadcast nature of the wireless medium, and use metadata negotiation to suppress redundant data messages. Firecracker [24] is a protocol to disseminate a small piece of data with help of a routing protocol. Trickle is another protocol that is designed to propagate small objects or configuration changes [25]. Trickle proposes mechanisms for dynamic broadcast rate adjustment with the goal of quickly propagating updates while reducing communication between nodes when there are no updates. Another protocol for disseminating a relatively small amount of data is GARUDA [4], which uses two phases to reliably deliver data objects downstream. In the first phase, data are reliably propagated along the core nodes, which are pre-selected subset of nodes that collectively approximates a dominating set of the network, while the non-core nodes receive data packets as best effort by passive listening. In the second phase, the non-core nodes recover the data object from the local core nodes. One advantage of this two-phase approach is that after the first phase, the nodes with the new data collectively cover the whole area due to the nature of the core nodes. Since the number of core nodes is usually much less than the total number of the nodes in the network, the first phase is usually much shorter than the total dissemination time of a corresponding one-phase approach. In the second phase, data object is disseminated from the core nodes to their neighboring non-core nodes in parallel. The duration of the second phase thus is independent of the network size but depends on the object size and network density only.

2.3.2 Reliable Dissemination for Large Data Objects

The second category includes protocols that are designed for reliable bulk data dissemination such as MOAP [6], Deluge [1], MNP [7]. All these protocols share some basic characteristics. First, these protocols were developed for supporting network reprogramming in multi-hop networks, and are used for entire code image delivery as opposed to difference-based application adjustment [26]. Second, these protocols extend the three-phase handshaking protocol used in SPIN-BL [23] for handling large data objects. Third, these protocols all borrow ideas such as the use of selective NACKs and hop-by-hop error recovery from prior work in reliable transport protocols [2] [3]. Deluge and MNP differ from MOAP in that a node does not need to receive the entire data object before it can start retransmitting it. By breaking the object up into pages, and allowing pipelined page delivery, Deluge and MNP take advantage of spatial multiplexing to reduce the latency of network reprogramming. Both protocols leverage the Trickle protocol [25] to limit transmissions of control messages. MNP differs from Deluge in its approach for sender selection and in allowing radios to be turned off to avoid unnecessary packet receptions.

Gappa [27] is another bulk data dissemination protocol. Gappa uses multiple channels and assigns one channel for each segment of the data object. A sender selection algorithm derived from MNP is used to determine the sender in a neighborhood and the segment to be sent. Unlike the previous protocols where the base station is the only source of the data object, Gappa assumes that every node in the network initially has a random segment of the data object.

Similar to [4], Sprinkler [8] is a two-phase approach, in which data are propagated to pre-select core nodes in the first phase and to the non-core nodes from the neighboring core nodes in the second phase. In selecting the core nodes, Sprinkler additionally assumes the existence of a localization mechanism at each node and uses the location information to set up the core structure. To minimize energy consumption, in addition to reduce the number of senders by selecting core nodes, Sprinkler uses TDMA transmission scheduling to mitigate collisions among senders. Moreover, the non-core nodes are put to sleep when their parent core nodes are not in a scheduled transmission slot.

One of our approaches, McTorrent, is similar to Deluge in the management of data objects and the use of pipelining. The main difference from Deluge is the use of multiple channels, which facilitates more aggressive pipelining in McTorrent, and packet collision mitigation in dense networks. Although Gappa [27] also uses multiple channels, Gappa assumes that every node in the network initially has a random segment of the data object and thus every node can be a sender from the beginning of the dissemination. McTorrent assumes that initially only the base station has the data object, and dissemination always starts from the base station.

Our another approach, CORD, is similar to the two-phase approaches such as GARUDA [4] and Sprinkler [8]. CORD differs from GARUDA in that CORD uses pipelining to facilitate propagation of large data objects, and adopts coordinated sleep scheduling to reduce energy consumption. While both CORD and Sprinkler adopt TDMA-like transmission scheduling, CORD differs from Sprinkler in that a slot in CORD is much longer than a slot in Sprinkler and thus is more robust to clock drifts. Moreover, with longer time slots, the overhead of switching the radio on and off is reduced. Another difference is that Sprinkler requires location information to construct the core, while CORD uses link quality statistics and thus eliminates the need of localization mechanism at each sensor node.

Our third protocol, McCORD, uses the techniques from both McTorrent and CORD to reduce energy consumption and object delivery latency.

2.4 Other Related Work

2.4.1 Network Reprogramming

While protocols in Section 2.3 can be used to disseminate the code image for network reprogramming, other research in network reprogramming focuses more on the code management. Maté [28] is a virtual machine running on sensor nodes. Instruction, each encapsulated in a single packet, are delivered to the sensor nodes and executed in the virtual machine. XNP [29] designs the mechanisms to receive and manage the code image and to boot from the new application. However, it does not support multiple-hop data transfer. [26], [30] and [31] deliver code difference instead of the whole new code image to reduce the amount of data being transferred and the energy expense as a result.

Security is one important consideration in network reprogramming. Some research has been done to authenticate the network reprogramming messages [32–34]. While the proposed approaches in this dissertation do not address security issues, they can be used in conjunction with the authentication schemes described in [32–34] to protect the sensor nodes from attacks.

2.4.2 Minimum Connected Dominating Set

One challenge of the core based protocols such as GARUDA is the selection of the core nodes. This problem, as known as *minimum connected dominating set (MCDS)* problem, is addressed by many researchers [35–38]. Most MCDS algorithms focus on achieving constant time distributed calculation based on local knowledge of one or two-hop neighbors. Since nodes are selected for MCDS in a distributed fashion, the construction time does not grow with the network size. This property is desirable for clustering and backbone-based routing in large ad hoc and sensor networks, and the information established from those algorithms is usually sufficient to set up cluster or to build routing table.

Data dissemination features a special one-to-many traffic pattern. While existence of MCDS is beneficial to the dissemination process if a core based approach is to be deployed, usually additional information needs to be established at each core node (for example, the band IDs in GARUDA, which are basically hop numbers away from the base station). Besides, the constant time property of MCDS becomes unnecessary for data dissemination, since the core construction is always followed by the actual data dissemination and it can be pipelined with the latter. Specifically, the core construction can be initiated by the base station and propagated hop by hop; the base station and the nodes closer to it can start disseminating data while those farther from the base station are still setting up the core. Therefore, although the whole setup time grows with the network size, the overall dissemination latency is not affected by the setup time.

2.4.3 Multichannel Wireless Systems and Sensor Networks

There are many papers that have dealt with the benefits of using multiple channels as a means to achieve higher throughput, often by also dealing with hidden terminal problems. One way of characterizing multichannel wireless networking research is by the number of transceivers and the assumed data link protocol. Examples of protocols that require multiple transceivers in a CSMA environment include the work of Nasipuri and Das [39][40]. The works of Wu et al. and Jain et al. also discuss protocols that dynamically assign channels, although they differ as to how the channels are assigned [41][42]. Examples of single transceiver approaches are presented in [43] and [44]. Both use a handshaking approach in a frequency-hopping spread spectrum system (FHSS) to improve throughput by avoiding hidden terminal problems. The drawback to these efforts is that they can only work in an FHSS environment. So and Vaidya describe a MAC layer protocol that also requires only one transceiver per host, and solves the hidden terminal problem using temporal synchronization [45]. There has also been some work done in data striping using multiple channels, which requires a separate radio interface for each channel [46].

All of the above research focuses on the 802.11 protocol, or suggests enhancements to that protocol. In contrast, this dissertation deals with multichannel sensor networking. The major issues in this environment are effective cross-layer and application-aware design, along with communication management that minimizes the packet collisions.

2.4.4 TinyOS

Part of our work involves implementation of the designed protocols in TinyOS [47]. TinyOS is a lightweight embedded operating system that is widely used in sensor networks. Applications developed for TinyOS platform are built on three kinds of components: configurations, modules and interfaces, all written in nesC language. A module encapsulates a set of functionalities of a hardware component or software component, and provides interfaces (C-like function calls) to other modules. A module can also use interfaces provided by other modules. Modules are therefore wired through interfaces. The configuration file of an application specifies the wiring of the modules that are used by the application. The use of modules enhances code reuse and maintenance.

The TinyOS distribution contains a simulation tool called TOSSIM, which simulates sensor network applications as a series of discrete events. One advantage of TOSSIM is that it simulates the communication between sensor nodes at bit level, capturing most features of real hardware platforms, at a price of a longer run time than packet level simulations. Another advantage is that the TinyOS code written for real hardware platforms can be run in TOSSIM with little change, which reduces the cost of software development and testing.

Chapter 3: McTorrent

This chapter describes the McTorrent protocol [48] for reliable bulk data distribution in sensor networks. This protocol is designed to use multiple communication channels while reliably distributing a large data object to all the nodes in the network.

3.1 Overview

The design of McTorrent shares many ideas with Deluge [1] and MNP [7], such as dividing a whole data object into multiple pages (segments as in MNP) and sending pages in a pipelined fashion. The key difference between McTorrent and its predecessors is in the use of multiple communication channels. The motivation for using multiple channels in McTorrent is twofold. First, by enabling different nodes to use different channels for transmitting data packets, a protocol like McTorrent can reduce the number of collisions during data propagation. Previous research [1] has shown that the rate at which a large data object can be propagated through a multihop network is greatly impacted by collisions, especially in dense networks. Second, Using multiple channels facilitates more aggressive pipelining and thus further reduces the latency of data propagation, as explained in 3.1.2.

On the other hand, using multiple channels during data dissemination reduces passive listening. In Deluge, all nodes transmit and receive on the same channel, thus a node may receive part of the data object just by passive listening without any packet transmissions. The reduction of passive listening might result in more overall packet transmissions.

3.1.1 Data Representation

A central focus of this chapter is to describe a multichannel protocol for reliable data dissemination of some data object OBJ. The size of this data object may be large, as in the case of new code for node reprogramming. The data object is of size S_{obj}, and is divided into fixed size packets of size S_{pkt}. In order to facilitate efficient communication control and data management, the size of the packets is the same as the size of the data link transfer unit.

By dividing OBJ into a series of packets, it is possible for each node to represent the complete object as a bit vector $BV\langle OBJ\rangle$ of the packets that make up the object. However, it is possible that S_{obj} might be so large that the size of $BV\langle OBJ\rangle$ might consume too much storage space within a resource constrained node. Under these circumstances, we adopt the approach taken by [1]. The data object OBJ is fragmented into P pages where each page has a fixed number of packets of size S_{pkt}. Because the pages are transferred in sequential order, each node only needs to maintain a bit vector for the current page being transferred.

3.1.2 Reliable Data Delivery and Pipelining

Like Deluge, McTorrent uses Trickle [25] to check whether a new data object is available to a node. Specifically, each node periodically broadcasts an advertisement of the data object it owns. The advertisement contains the ID of the data object (a higher ID value means a newer object) and the number of pages of the object it possesses. Every node maintains a bit vector of missing packets of the next incomplete page if it has not received the whole object. When a node R hears such an advertisement from a node S that indicates S has a newer object or more pages of the same object as R has, R sends a request for its next incomplete page to S. In the case that S has a newer object, R requests for the first page of the newer object. The request contains the ID of the page and a bit vector of requested packets within the page. Upon reception of the request, S broadcasts the requested packets indicated by

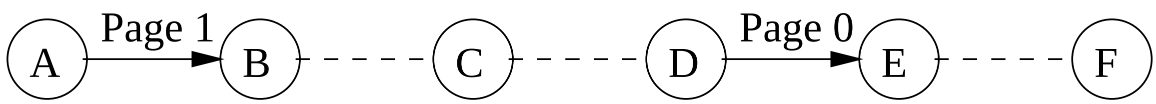


Figure 3.1: Pipelining in Deluge

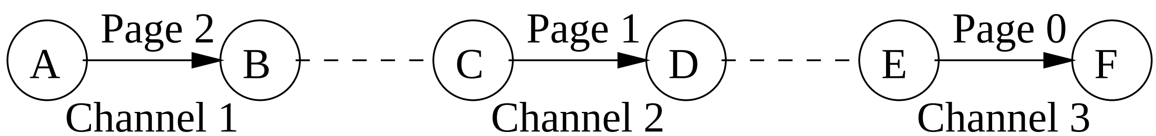


Figure 3.2: Pipelining in McTorrent

the bit vector. Eventually, all nodes in the network will have the same object. To reduce the overheard of control packets, the period of advertisements is gradually increased to an upper bound when nodes see the same object among them.

Unlike MOAP [6] where a node has to receive the entire object before it becomes a sender for the downstream nodes, in Deluge, MNP and McTorrent, a node can send a page to its downstream nodes right after it receives the page from its upstream nodes. Therefore, as illustrated in Figures 3.1 and 3.2 (a dotted line means the connected two nodes are within the radio range of each other, and an arrow indicates an ongoing data transfer), a node can send a page while at some place downstream, another node is sending a lower page at the same time. This kind of pipelining (also called spatial multiplexing) greatly reduces the latency to disseminate a large data object to a large sensor network, compared to non-pipelining approaches.

With pipelining, to ensure that transfers of different pages do not interfere with each other, Deluge takes special care to ensure that nodes sending simultaneously are at least three hops apart (see Figure 3.1). For instance, in Figure 3.1, after B receives Page 1 from A and broadcasts an advertisement, if D is still sending Page 0 to E (which can be overheard by C), C will not send a request for Page 1 to B since otherwise the data packets from B and D would collide at C.

However, by using different data channels, senders only two hops apart can send packets at the same time in McTorrent as shown in Figure 3.2, thus enabling more effective spatial multiplexing and pipelining. For instance, in the figure, when C is sending Page 1 to D, B can still request for Page 2 from A since data packets from A will not cause collisions at B as long as A uses a different channel than C.

3.1.3 Channel Coordination

Two issues need to be addressed in a data dissemination protocol that uses multiple channels. First, nodes need to advertise and request data from each other. Protocols such as Deluge exploit passive listening, whereby nodes suppress their own advertisements or requests on overhearing transmissions from neighboring nodes; this is also desirable for McTorrent. Second, the availability of multiple channels implies that nodes exchanging data need to agree on the channel used for a particular exchange, as well as the timing of the exchange.

We address these issues by extending the three-way advertisement-request-data handshake used by Deluge and other approaches to a four-way advertisement-request-*channel*-data handshake. The new *channel* message is a channel claim message, which identifies the channel on which the sender will transmit the following data packets. In order to receive the data packets from the sender, the receivers must switch to that same channel. To benefit from passive listening, all nodes send their control messages (i.e., advertisements, requests and channel claims) on a common channel. However, data packets are sent on different channels as described in detail in 3.2.

3.2 Protocol Description

In a multichannel sensor network, each node is able to switch to any of a fixed number of available channels, and send and receive packets on that channel. For convenience, we use Γ to denote the number of available channels including the shared common channel γ_0. McTorrent uses four types of messages: advertisement (ADV), request (REQ), channel claim(CHANNEL), and data (DATA). The first three types of messages (control messages) are always sent on γ_0, while DATA messages are sent on a pre-specified channel.

McTorrent adopts a four-way ADV-REQ-CHANNEL-DATA exchange to propagate bulk data:

1. A node, S, periodically broadcasts an ADV packet. The ADV packet contains the source address of S, summary information of the object S owns (such as the object ID and number of pages) and the channel (γ_j) on which it will send the data packets. γ_j is selected such that collisions of data packets are minimized. The channel selection algorithm is presented in 3.3.

2. On reception of the ADV packet, a node, R, replies with a REQ packet if R needs the pages that S has, that is, either S has a newer object than R or S has the same object as R but has more pages of the object. The REQ packet contains the destination address (S), the requested page number, and a bit vector of requested packets of that page. The channel number received from the ADV packet, γ_j, is echoed in the REQ packet.

3. After receiving REQ packets (there might be multiple requesting nodes), S broadcasts a CHANNEL packet containing the number of the requested page, the bit vector of requested packets of that page and the channel number γ_j. In case that multiple nodes request for different pages, S selects the page with the lowest ID. After sending

the CHANNEL packet, S tunes its radio to γ_j. On the other hand, R also tunes its radio to γ_j on reception of the CHANNEL packet from S if R needs the page that S is going to send. Moreover, based on the number of data packets S is going to send indicated by the CHANNEL packet, R computes the time period to stay on γ_j in order to receive the data packets, T_r. R starts a timer, RxTimer, which will fire after a duration of T_r.

4. S then broadcasts all of the requested data packets on γ_j and tunes back to γ_0. R receives the data packets and tunes back to γ_0 either when RxTimer fires or when R receives all packets for the current page.

The McTorrent protocol can be described in terms of a state machine, as shown in Figure 3.3. Each node can be in one of the five states at a time, namely, S_IDLE, S_ADV, S_REQ, S_DATA_TX and S_DATA_RX. In the figure, a transition from one state to another is annotated with the event that triggers the transition and the corresponding action the node takes. The states and transitions are described in detail as follows. For convenience of description, nodes in the S_ADV state and the S_DATA_TX state are called senders, while nodes in the S_REQ state and the S_DATA_RX state are called receivers.

3.2.1 State S_IDLE

In McTorrent, all nodes are initially in the S_IDLE state and listen on the common channel, γ_0. Each node maintains a timer, AdvTimer, to trigger the transmission of an advertisement. The duration of the timer follows the same dynamic adjustment from Trickle [25] and Deluge [1]. Specifically, the duration is set to a lower bound, τ_l, when the node needs to transfer data from/to the neighboring nodes, and is increased exponentially when there is no such need until it reaches an upper bound, τ_h. A node determines whether such need exists based on the control messages it receives from the neighboring nodes.

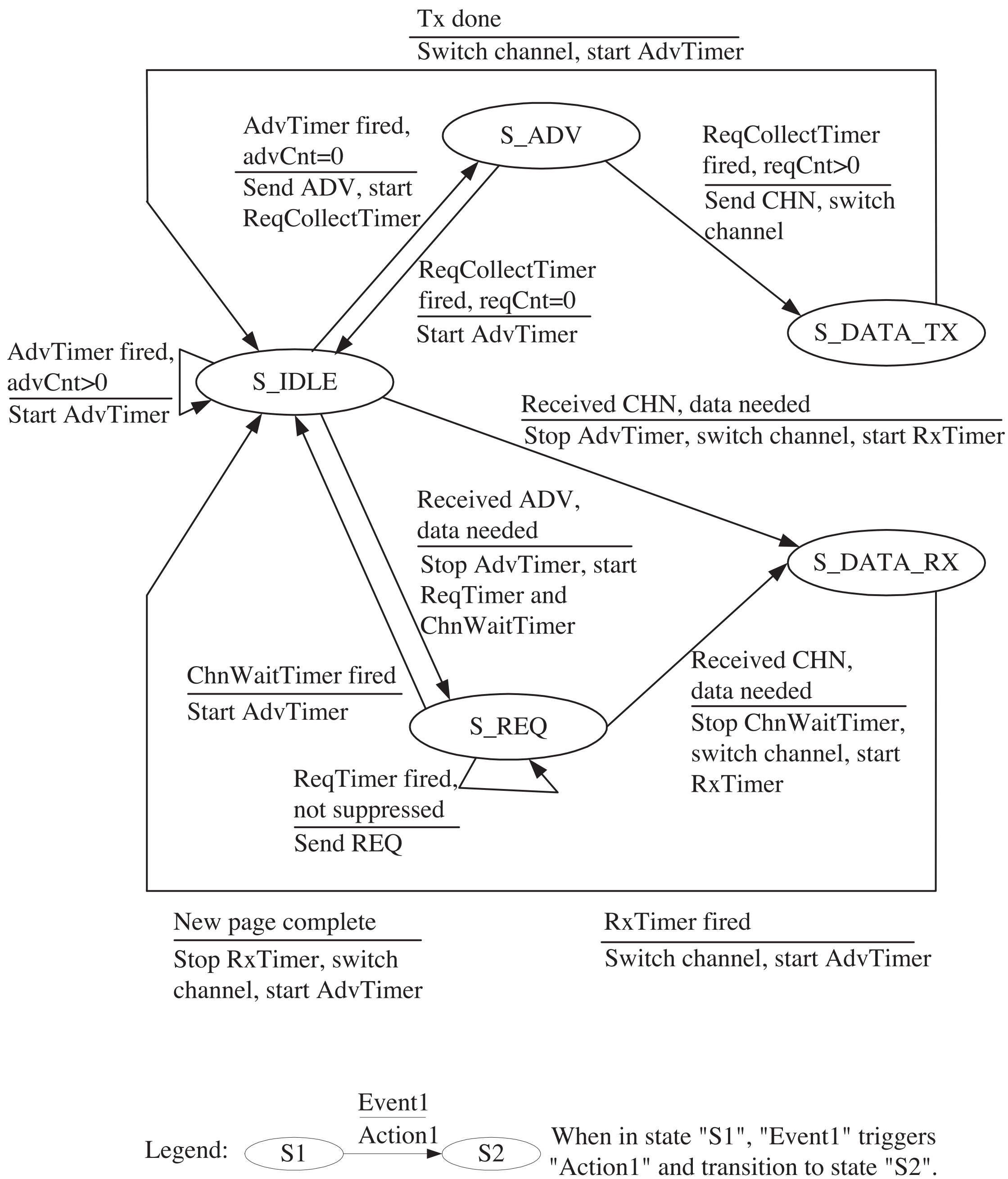


Figure 3.3: McTorrent state machine

McTorrent also adopts the same advertisement suppression scheme from Trickle [25] and Deluge [1] to reduce redundant advertisements. When AdvTimer fires, the node broadcasts its advertisement only when it has not overheard similar advertisements in the neighborhood (the number of such advertisements it has overheard since its last advertisement, advCnt, is 0).

After broadcasting the advertisement, the nodes starts ReqCollectTimer and enters the S_ADV state. The duration of ReqCollectTimer is set to the maximum delay before a node sends a request in response to the advertisement.

If a node, R, receives an advertisement indicating the data it needs in the S_IDLE state, the node stops its AdvTimer, starts ReqTimer and ChnWaitTimer, and enters the S_REQ state. ReqTimer adds a random delay before sending the request to the advertising node, S. ChnWaitTimer is set to the time when R expects to receive the channel claim message from S (the maximum request delay), plus a small grace period.

If R receives a channel claim message indicating the data it needs in the S_IDLE state, the node stops its AdvTimer. It switches to the channel specified in the received channel claim message, γ_j, and starts RxTimer. RxTimer is set to the period that the transmission from the sender will last, which is calculated from the number of data packets to be sent indicated in the channel claim message. The node then enters the S_DATA_RX state to receive the data packets on Channel γ_j.

3.2.2 State S_ADV

A node in the S_ADV state, S, waits for requests from nodes that need the data it advertised. Requests for a page with a lower page ID have higher priority to be fulfilled. If multiple requests for the same page are received, S takes the union of the bit vectors of requested packets to construct a bit vector of packets to be sent. When ReqCollectTimer fires, the ID of the page and the bit vector of packets to be sent, along with the number of the channel selected at the time of advertisement, are broadcast in a channel claim message.

The successful reception of the channel claim message is critical for the receivers to receive the data packets that follow. If a receiver fails to receive the message, it will return to the S_IDLE state without switching to the data channel and listening for the impending data packets. To increase the probability that nodes in need of the data receive the channel claim message, the sender, S, may broadcast the channel claim message for multiple times.

After broadcasting the channel claim message, S switches to the data channel and enters the S_DATA_TX state.

On the other hand, if S receives no request when ReqCollectTimer fires, it returns to the S_IDLE state.

3.2.3 State S_REQ

Nodes in the S_REQ state send requests for the needed data packets to the nodes that advertised and wait for the channel claim message from the latter. The requests are sent as broadcast while including the address of the data source node in the message, so that every node in the neighborhood that is listening on the common channel can overhear the message. If a node in this state, R, overhears a request destined to the same data source, S, R suppresses its own request if the overheard request is for a page with a lower ID or contains a bit vector of data packets that is a superset of R's requested data packets from the same page. If the request is not suppressed, it is sent when ReqTimer fires.

When R receives the channel claim message before ChnWaitTimer fires and the message indicates the right page and data packets that R needs, it stops ChnWaitTimer, switches to the channel specified in the message and enters the S_DATA_RX state. RxTimer is started at the same time and will fire when the estimated time to transmit all of the data packets indicated in the channel claim message elapses.

If R has received no channel claim messages when ChnWaitTimer fires, it returns to the

S_IDLE state.

3.2.4 State S_DATA_TX

Nodes in the S_DATA_TX broadcast the data packets specified by the preceding channel claim messages on the channel also specified by the messages. After transmitting all of the data packets, the nodes switch back to the common channel and return to the S_IDLE state.

3.2.5 State S_DATA_RX

Nodes in the S_DATA_RX listen on the specific data channel for the data packets. When RxTimer fires, which indicates the sender has finished transmitting all of the data packets by this time, or when the receiving node has received all data packets of the currently receiving page, the node switches back to the common channel and returns to the S_IDLE state.

3.3 Channel Monitoring and Selection

In McTorrent, a node needs to select a channel before it can send any data packets. Ideally, the sender should select a channel that does not conflict with the channels being used by other senders one or two hops away. We use a contention-based approach in which a node monitors the control messages that contain the specified channel number and are broadcast by nodes in its neighborhood, so that the node can keep track of the channels in use. It then attempts to select an unused channel for its own data transmissions, or if there is no such channel, to postpone its own data transmissions to a future point. Our channel monitoring and selection scheme does not introduce new messages. The related information is piggybacked on the existing control messages.

In our scheme, each node locally maintains a timeToFree value, initially 0, for each possible data channel $[1..\Gamma - 1]$, which indicates the estimated time when the channel will become free. A channel is free if its timeToFree value is smaller than or equal to the current time. The current time is read from the local node's system, thus no synchronization between nodes is required.

A node selects a data channel before it broadcasts an advertisement. The channel number is included in the advertisement. The same channel number is echoed in the corresponding requests and the following channel claim messages. The reason that the requests contain the channel number is to inform nodes two hops away from the sender that the channel is in use. These nodes are not able to receive the advertisements and channel claim messages from the sender. Each request also contains the time difference, t_{req}, between the time when the advertisement is received and the time when this request is sent (i.e., the duration of ReqTimer).

When a node overhears a request which has a channel number γ_j, it updates γ_j's timeToFree value to the current time plus the time needed for the remaining duration of ReqCollectTimer (calculated from t_{req} in the request) and for the transmissions of the channel claim messages and a whole page of data packets. The actual data transmissions may only involve part of a page, however, the time to transmit a whole page is used for the estimation of timeToFree since the node does know how many data packets will actually be sent.

Nodes that are one-hop away from the sender of the advertisement will suppress their advertisements as specified by the protocol. However, they also update their timeToFree value of the channel specified by the overheard advertisement and channel claim messages, in case that the data transfer takes a longer time than the interval of the advertisements and the nodes at the time of the next advertisement thus need to select different channels than the one still being used in the ongoing data transfer.

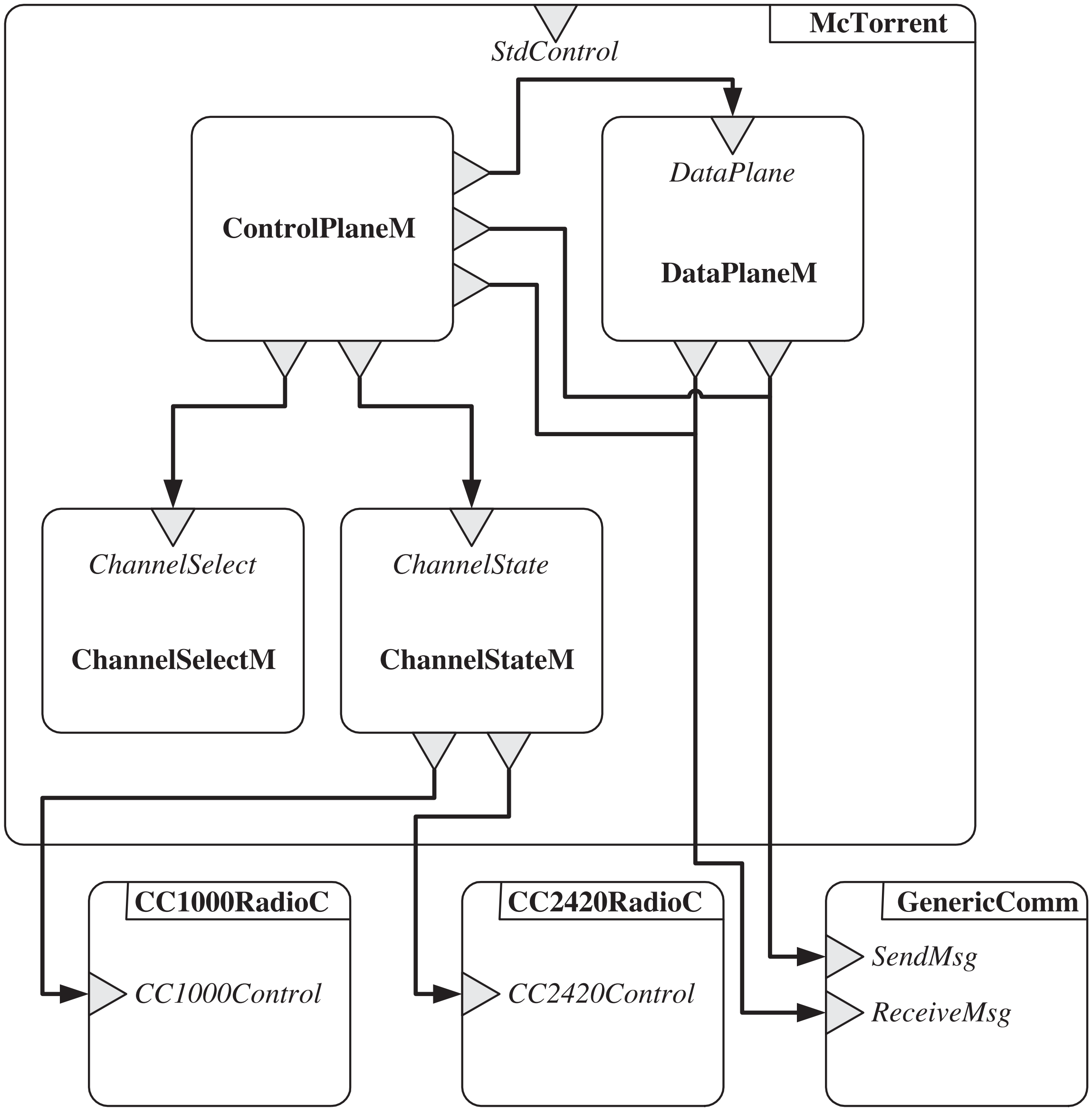
Selecting a free channel for imminent data transmissions in this case is simply picking up randomly a channel whose timeToFree value is smaller than or equal to the current time. However, when the total number of channels (Γ) is small, a node may find that all channels are occupied when it needs to send data. If this happens, the node postpones its advertisements to a future time which is a random delay after the earliest timeToFree among all data channels.

While simple and lightweight, our channel selection scheme does not consider link quality variation on different channels. That is, the scheme assumes all available channels have the same quality, which is not true according to our experiments (see Chapter 5). The link quality variation on different channels does not cause McTorrent to fail, since each node has a probability to become a sender, and the channel for each data transfer is randomly selected. However, using a channel with bad quality for data packets results in energy waste at the sender and delay of the delivery, since the receivers would have a high loss rate in receiving the data packets on the channel.

3.4 Software Structure

We have implemented McTorrent in TinyOS [47] for MICA2 [49] and TelosB [5] motes. Figure 3.4 shows the software structure of our implementation of McTorrent, following the same style used in [50]. In TinyOS, a component is either a module which implements a set of functions in C code, or a configuration which wires other components together. Components are wired through interfaces, which means a component, A, uses the interfaces provided by another component, B, to invoke the functions of B (using the *call* primitives) and to receive events from B (issued by B with the *signal* primitives). In our implementation, we focus on modular design and reusable components.

In the implementation, the main body of the McTorrent protocol is implemented as two modules, the control plane module (ControlPlaneM) and the data plane module (DataPlaneM). Module ControlPlaneM handles the exchange of control messages including advertisements, requests and channel claim messages, following the state machine depicted in



Note: Rectangles are modules and configurations (names of configurations are at the top right corner of the rectangles, and names of modules are in the center of the rectangles). Triangles pointing into a rectangle are provided interfaces with names of the interfaces shown aside; triangles pointing out are used interfaces. Lines are wires between modules or configurations.

Figure 3.4: McTorrent software structure

Figure 3.3. It also uses the functionalities of other modules for sending and receiving data packets, monitoring channel usage and selecting a free channel, and switching the radio channel when necessary. These modules and their interfaces exposed to ControlPlaneM are discussed as follows.

- Module DataPlaneM and Interface DataPlane. DataPlaneM sends the requested data packets when the node is in the S_DATA_TX state and receives data packets when in the S_DATA_RX state. The main interface that DataPlaneM provides is DataPlane:

```
interface DataPlane {
    command result_t startTx(uint16_t pageId, uint8_t * pktsToSend);
    command result_t startRx(uint8_t * pktsToReceive, uint16_t rxPeriod);
    event result_t txDone();
    event result_t rxDone(bool receivedNewPage);
}
```

  ControlPlaneM calls the `startTx()` command of Interface DataPlane provided by DataPlaneM when the node enters the S_DATA_TX state, where the parameters `pageId` and `pktsToSend` denote the page and the bit vector of requested packets to be sent in that page, respectively. DataPlaneM transmits each packet denoted by the bit vector continuously, and signals a `txDone()` event after transmitting all of the packets.

  On the other hand, ControlPlaneM calls the `startRx()` command when the node enters the S_DATA_RX state. The parameter `pktsToReceive` is a bit vector of packets needed by the receiving node to complete the current page (not necessary identical to `pktsToSend` at the sender). The second parameter, `rxPeriod` is an estimate of the total transmission time of data packets from the sender. The estimation is based on the channel claim message received prior to the S_DATA_RX state, which contains `pktsToSend` from the sender. DataPlaneM starts a timer for an interval of `rxPeriod`. When the timer fires or the node receives all data packets denoted by `pktsToReceive` (before the timer fires if this happens), DataPlaneM exits from the receiving state by signaling an `rxDone()` event to ControlPlaneM, with the boolean parameter `receivedNewPage` set to `FALSE` or `TRUE`, respectively.

In ControlPlaneM, the channel is switched to the data channel before the node enters the S_DATA_TX or S_DATA_RX state, and is switched back to the common channel when ControlM receives the `txDone()` or `rxDone()` event.

- Module ChannelSelectM and Interface ChannelSelect. ChannelSelectM maintains the usage information of all available channels, and provides the ChannelSelect interface:

```
interface ChannelSelect {
    command result_t setChannelTaken(uint8_t channel, uint32_t period);
    command uint8_t getFreeChannel();
}
```

In ControlPlaneM, when a node overhears a request or channel claim message, it estimates the period during which the channel will be taken (see Section 3.3). ControlPlaneM then calls `setChannelTaken()` to update the channel usage maintained in Module ChannelSelectM. Before broadcasting an advertisement, ControlPlaneM calls `getFreeChannel()` to find a free channel. If no free channel is currently available, the advertisement is postponed.

- Module ChannelStateM and Interface ChannelState. ChannelStateM interacts with the radio component to actually switch the channel. It provides the ChannelState interface, which is used by ControlPlaneM:

```
interface ChannelState {
```

```
    command result_t setChannel(uint8_t newChannel);
    command uint8_t getChannel();
}
```

Since different sensor nodes may use different radios (for example, MICA2 uses CC1000 while TelosB uses CC2420), the implementation of ChannelStateM is platform dependent. However, the ChannelState interface is generic, hiding the implementation details from the users of the interface.

The modules in McTorrent are also wired to other components, such as GenericComm for sending and receiving messages, and BlockStorageC for reading from and writing to the persistent storage. Those are standard TinyOS components and are provided by the TinyOS distribution [47].

Chapter 6 presents various empirical and simulation results of McTorrent, along with other existing approaches. As shown by the results, by using multiple channels, McTorrent greatly reduces the object delivery latency, compared with single channel approaches such as Deluge.

Chapter 4: CORD

This chapter describes CORD (COre based Reliable Dissemination) [51], an energy-efficient reliable bulk data dissemination protocol for large scale sensor networks.

CORD adopts a different scheme than Deluge and McTorrent. In Deluge and McTorrent, each page of the object is propagated from one neighborhood to the next, after the page has been disseminated to all the nodes in the first neighborhood. While in CORD, the object dissemination is divided into two distinct phases. Before the data dissemination commences, a subset of nodes in the network that have reliable links and that form an approximate minimum dominating set [52] are selected as *core nodes*. After this core construction step, in the first phase of the data dissemination protocol, the object is reliably propagated from the base station to the core nodes in a pipelined page by page fashion. After the entire object has been propagated to the core nodes, the second phase commences in which the core nodes disseminate the object to their neighboring non-core nodes in parallel.

The core-based two-phase approach used by CORD is motivated by the goal of reducing the energy consumption for disseminating the object within the network. By constructing a core for data dissemination, the protocol implicitly selects the set of nodes that are responsible for disseminating the object to their neighbors. This reduces the number of control messages that need to be exchanged between neighboring nodes in comparison to protocols such as Deluge and MNP which use a three-message (advertisement-request-data) handshake between nodes to select senders and disseminate data. Reducing the number of control messages and the number of nodes within a neighborhood competing to transmit the object also results in a reduction in the number of message collisions, which is an important consideration, especially in dense sensor networks.

Another important reason why CORD uses a core-based two-phase approach is to support sleep scheduling in order to further reduce energy consumption for large object dissemination. Protocols such as Deluge rely on the use of passive listening by nodes for dynamically adjusting the rate of advertisements and for request suppression; these techniques only work well when nodes are awake at all time. To reduce energy consumption incurred in idle listening, CORD aggressively uses sleep scheduling. At the time of core construction, CORD installs a coordinated sleep schedule on the nodes in the network whereby nodes that are not involved in receiving or transmitting data can turn off their radios to reduce their energy consumption. Both the core and non-core nodes adhere to this sleep schedule during the data dissemination. However, only the core nodes actively participate in the first phase of the protocol, whereas the non-core nodes participate passively by listening to the transmissions of core nodes.

The two-phase core-based approach is also suitable for heterogeneous networks, where a subset of nodes in a network are more powerful than the others. Such networks are likely to be increasingly common in the near future. Selecting the more powerful nodes as core nodes and using CORD to disseminate large data objects saves energy at the more power-constrained non-core nodes, prolonging the life time of the whole network.

4.1 Protocol Overview and Design Tradeoffs

CORD first selects a subset of the nodes in the network as core nodes before data dissemination. Following the core construction step, each node in the network is either a core node, or is an immediate neighbor of a core node.

After the core is set up, the data object is propagated from the base station (the only node that initially possesses the data object to be distributed) to the core nodes in the first phase of dissemination by reliable hop-by-hop forwarding. In this phase, the non-core nodes passively participate in the protocol by listening to communications between core nodes. Consequently, at the end of the first phase, a non-core node may already possess a significant fraction of the object being disseminated. After the core nodes have received the entire object, the protocol enters the second phase of the protocol. In this phase, each non-core node requests and receives the missing portions of the object from its neighboring core node.

4.1.1 Design Tradeoffs

The core-based approach used in CORD reflects a different design choice than protocols such as Deluge and MNP. By establishing a core, we are implicitly selecting the nodes that will be responsible for transmitting packets to their neighbors *for all the pages* of an object. In contrast, Deluge selects a sender separately *for each page* using a three-message handshake, and thus requires more control messages. The designers of Deluge motivate their approach by arguing that core-based approaches are relatively inflexible to variations in connectivity between nodes [1].

Our approach is motivated by the fact that most sensor networks are stationary, and by empirical evidence from extensive indoor and outdoor experiments that suggests that while the links between nodes are highly variable in their quality, it is possible to identify links that have low loss rates, and that the quality of such links is relatively stable *at the time scale of an object propagation.*

More importantly, a core-based approach makes it possible to use coordinated sleep schedules to reduce the energy consumption of the protocol. In contrast, single-phase protocols such as Deluge rely on the use of passive listening by nodes for dynamically adjusting the rate of advertisements and for request suppression; these techniques only work well when nodes are awake most of the time, thus precluding the use of aggressive sleep scheduling. The results in Chapter 6 demonstrate that sleep scheduling is the most effective technique in reducing energy consumption. On the other hand, an open question that needs to be addressed is whether it is possible to use sleep scheduling to reduce energy consumption while at the same time achieving an object dissemination latency that is comparable to that of protocols such as Deluge.

4.2 Protocol Description

In this section, we describe the main components of the CORD protocol.

4.2.1 Core Construction

We used Cheng's degree-aware single leader algorithm [53] as the basis for CORD's core construction approach, due to the similarity between the traffic pattern of the dissemination and the CDS structure rooted at the base station (the leader). In Cheng's algorithm, initially all nodes are labeled as white and non-active. During the core construction, nodes are marked either black or gray. Black nodes become dominators (core nodes), while the gray ones become dominatees (non-core nodes). The leader initiates the core construction by marking itself black (and thus becoming a dominator). A white node marks itself gray and becomes a dominatee if one of its neighbors becomes a dominator. A non-active white node changes to active if one of its neighbors becomes a dominatee. Active nodes compete to be a dominator. The node with the maximum effective degree (number of white neighbors) wins the competition, and its gray parent also becomes a dominator to make the CDS connected.

Link Quality We extend Cheng's algorithm in several ways for bulk data dissemination. First, we extend the algorithm to take link quality into account while forming the core. Two nodes are considered connected only when the link quality between them is above a threshold, Q_{th}. We assume that a sensor node maintains statistical information for the packet loss rate of its links to its neighbors. If the sensor network has been deployed for sufficient time, a node can estimate the quality of the links to its neighbors [54] based on packet loss rates. Alternatively, for motes using the more recent CC2420 radio such as MicaZ and TelosB, we can use the CC2420's Link Quality Indicator (LQI) as a metric of the link quality [55]. This requires only a single packet reception instead of relatively long-term link quality estimation. In our protocol, the link quality between u and v, Q_{uv}, is a metric that combines packet reception rates in both directions, $Q_{u \to v}$ and $Q_{v \to u}$, by $Q_{uv} = min(Q_{u \to v}, Q_{v \to u})$. Consequently, asymmetric links are not included in the core.

Establishing Coordinated Schedules We further extend Cheng's algorithm to facilitate the establishment of coordinated schedules at each node (described in more detail in 4.2.2). The schedule consists of a repeating series of fixed-length slots in which a node either acts as a parent node in the data dissemination, or as a child node, or is quiescent. The schedules of neighboring nodes are coordinated to reflect their role in the data dissemination. The role of each slot is assigned after the dissemination begins; however, the schedule is installed at the time of core construction, as discussed below.

We modified Cheng's algorithm to integrate core construction with the establishment of coordinated node schedules. Specifically, the base station initiates core construction by starting the schedule and sending a CLAIM message, announcing itself as a core node. The message contains the time offset from the beginning of its first time slot to when the message was sent, as well as a list of neighbors to which the base station has good links. Nodes at level one that receive the CLAIM message (i.e., nodes that are one hop away from the base station) update their effective degrees by counting their neighbors that are not included in the base station's neighbor list. If a node that receives the CLAIM message has a good link with the base station, it selects the base station as its parent in the core, and initiates its own repeating schedule (based on the information in the message), such that the start time of the slots in its schedule coincides with the slots of the base station.

The nodes that have selected the base station as their parent then broadcast their effective degrees in COMPETE messages. Nodes at level 2 that receive one or more COMPETE messages respond with a SUBSCRIBE message to the competitor with the maximum effective degrees among the competitors it hears (the node ID is used to break the tie). A node that receives SUBSCRIBE messages become a core node, otherwise it becomes a non-core node. Each core node at level one then broadcasts a CLAIM message announcing that it is a core node. This process continues recursively until every node in the network becomes either a core node or a non-core node.

While Cheng's algorithm first selects the core nodes at even-numbered levels and then chooses the core nodes at odd-numbered levels to make the CDS connected, in our modified algorithm, the core nodes are selected sequentially at each level. The modification guarantees that every node that wins the competition and becomes a core node already knows its parent and is synchronized with the parent.

Figure 4.1 shows the state transition at each node during core setup in CORD. Initially, nodes are in the S_READY state, except the base station, which enters the S_CORE_CLAIM_TX immediately.

- S_READY: In this initial state, if a node, u, receives a COMPETE message from one of its good link neighbors, it enters the S_CORE_COMPETE_RX state. Meanwhile, the node starts the schedule that consists of repeating fixed-length slots.

- S_CORE_COMPETE_RX: In this state, u receives COMPETE messages from its good link neighbors till the next slot, when it enters the S_CORE_SUBSCRIBE_TX state. However, if u receives any CLAIM messages from its good link neighbors during the slot, it enters the S_CORE_COMPETE_TX state at the next slot.

- S_CORE_SUBSCRIBE_TX: u sends a SUBSCRIBE message at a random time in

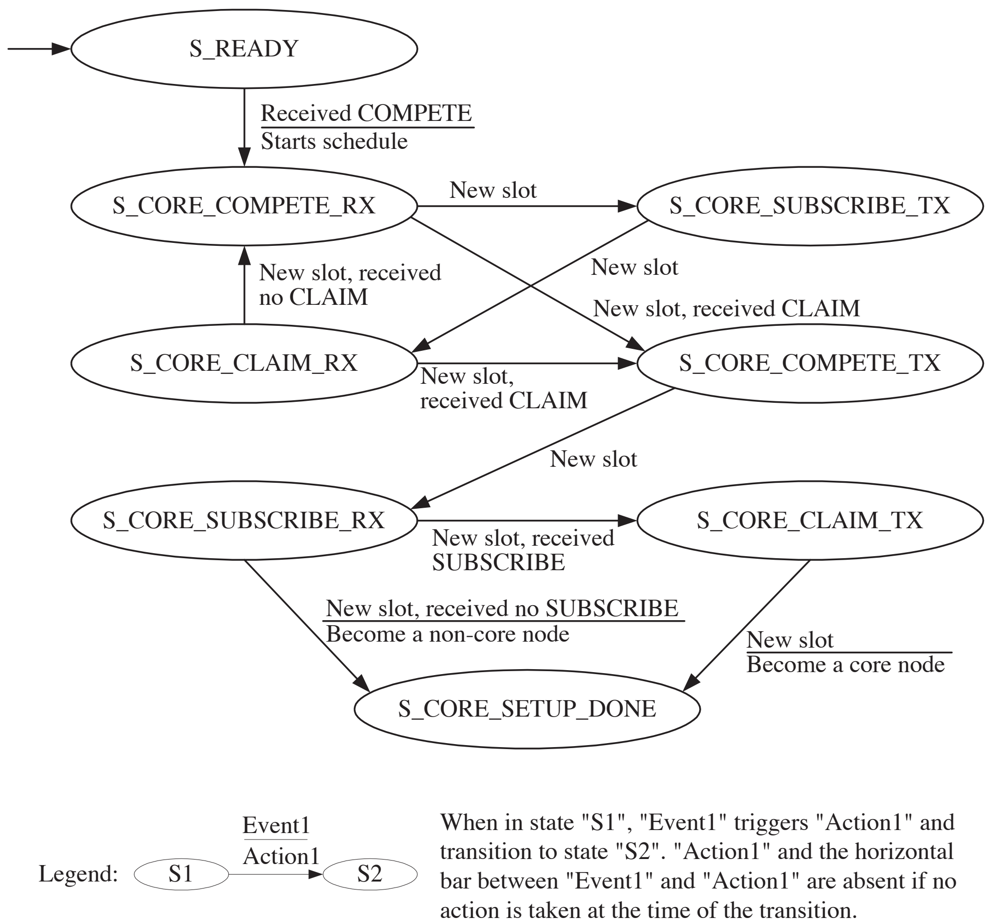


Figure 4.1: CORD state transition diagram for core setup

the slot to the node, v, from whom u has received the COMPETE message in the previous slot that indicates the maximum effective degrees. At the next slot, it enters the S_CORE_CLAIM_RX state.

- S_CORE_CLAIM_RX: During the slot, if u receives CLAIM messages from its good link neighbors, it enters the S_CORE_COMPETE_TX state at the next slot. u picks up a sender of the CLAIM messages as its parent for the impending data dissemination. Otherwise it returns to the S_CORE_COMPETE_RX state at the next slot.

- S_CORE_COMPETE_TX: u broadcasts a COMPETE message at a random time during the slot. The message contains the node's effective degree. At the next slot, u enters the S_CORE_SUBSCRIBE_RX state.

- S_CORE_SUBSCRIBE_RX: During the slot, if u receives any SUBSCRIBE messages, it enters the S_CORE_CLAIM_TX state at the next slot. Otherwise, u becomes a non-core node and the core setup at u finishes.

- S_CORE_CLAIM_TX: u broadcasts a CLAIM message at a random time during the slot. The message contains a list of neighbors that have not been covered by other core nodes. u becomes a core node and the core setup at u finishes.

4.2.2 Coordinated Node Sleep Scheduling

The coordinated schedule adopted by nodes in CORD is motivated by the observation that in protocols that use a pipelined data dissemination approach, nodes that transmit data simultaneously should be at least three hops apart to ensure that transfers of different pages do not interfere with each other. For example, consider the linear network shown in Figure 3.1. In this network, nodes A and D can simultaneously send different pages to their downstream neighbors without interference. Thus, after delivering page 0 to node B, node A has to wait until page 0 has been delivered by B to C and by C to D, before it starts transmitting page 1 to B. Consequently, protocols such as Deluge are designed to ensure the three hop separation between senders [1].

CORD exploits this enforced delay between transmissions of consecutive pages by allowing nodes that are idle to go to sleep to conserve energy. For example, still referring to Figure 3.1, when node A is transmitting page 0 to node B, node C is idle and can go to sleep. When node B is transmitting page 0 to node C, node A can go to sleep. Finally, when node C is transmitting page 0 to node D, node B can go to sleep.

Based upon this simple idea, a coordinated schedule is established at each node (during core construction) that will be used during data dissemination in the first phase of the protocol. For core nodes, the schedule consists of repeating fixed-length slots of time L in which a node takes turns acting as a child in the data dissemination, acting as a parent in the dissemination, and sleeping. We refer to these consecutive slots in a node's schedule as a C-slot, P-slot, and Q-slot respectively (where the C, P, and Q denote child, parent and quiescent respectively). For non-core nodes, the schedule consists of a C-slot followed by two Q-slots.

The selection of the number of Q-slots in one round of core nodes' schedule, N_Q, affects the degree of pipelining during dissemination (the number of Q-slots in one round of non-core nodes' schedule is $N_Q + 1$ accordingly). In an ideal network (such as the linear network in Figure 3.1) where nodes are either connected with link quality $Q > Q_{th}$ or out of range of each other, $N_Q = 1$ would lead to perfect pipelining with maximized channel usage. However, in a real sensor deployment, the topology is usually irregular. A node may have strong connections with some nodes, and weak connections with others. Although our approach only considers high quality links while forming the core, nodes with weak connections can still interfere with each other's transmissions even if they are separated by three or more hops in the core. Thus, a larger value of N_Q adds a " time buffer" between these nodes and therefore reduces collisions. On the other hand, a larger value of N_Q results in increased propagation latency. Our experiments show that the savings in energy consumption due to reduction of collisions by adopting a larger value of N_Q are trivial, compared to the increase in latency. Therefore, the discussion hereafter is based on $N_Q = 1$, and the results of CORD in Chapter 6 were obtained from experiments and simulations with $N_Q = 1$.

As discussed in Section 4.2.1, each node synchronizes the boundaries of its time slots with its parent in the core at the time of core construction. However, it is also necessary to coordinate the schedules of neighboring nodes so that a node's C-slot coincides with the P-slot of its parent in the core. This coordination is performed at the time of data dissemination as follows. The base station marks its first slot as a P-slot and initiates the data dissemination by broadcasting advertisement packets. Nodes that receive advertisements or data from their parents assign the current slot to be a C-slot. Subsequently, each core node follows the repeating C-P-Q schedule, whereas each non-core node follows the C-Q-Q schedule. Figure 4.2 illustrates the coordinated schedule of three core nodes u, v, and w at adjacent levels of the core.

We note that since CORD only requires slot coordination between neighboring nodes, it does not require an explicit network-wide time synchronization service. Further, as discussed below, these slots are relatively long, so even neighboring nodes do not need tight clock synchronization. The time interval corresponding to a slot, L, is set to be long enough so that a core node will be able to reliably deliver all the K packets in a page to its downstream children in the core. Thus, it should be larger than the time taken to deliver K packets plus the expected time for retransmitting packets that are not received in the previous round of transmissions. The value of L is determined based on the following analysis, which we extended from the analysis in [56] to adapt to sending and requesting for multiple packets in each transmission round in CORD.

In our analytical model, a parent core node (sender) has R child core nodes (receivers). We analyze the time for the sender to reliably deliver a page of K packets to all of the R

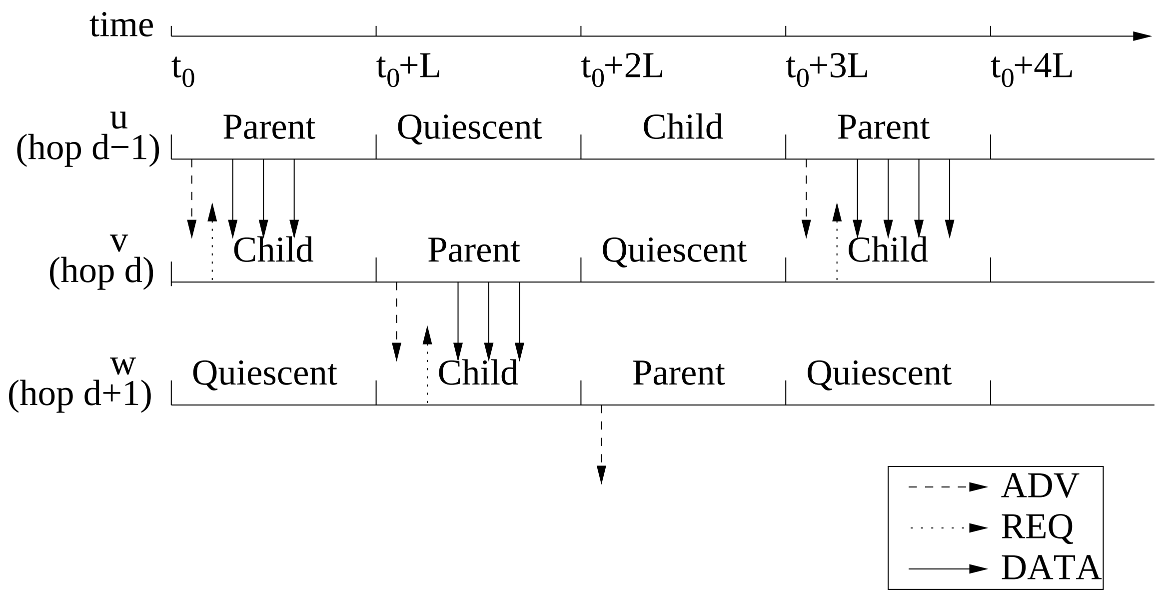


Figure 4.2: Coordinated schedules of adjacent core nodes in the first phase of CORD

receivers. Initially no receiver has any packet of the page. The sender transmits packets only upon requests from the receivers. We call the burst of transmissions of the requested packets a transmission round. After each transmission round, a receiver sends another request after a random delay if it has not received all packets in the page. For simplicity, we made the following assumptions:

1. The packet loss rate between the sender and each of the receivers is equal and is denoted by p.

2. The events of packet losses at different receivers are independent.

3. The events of losses of different packets are also independent.

4. While the receivers send requests at random time in the range (0, t_{req}), the sender always waits for the whole length of t_{req} before transmitting all requested packets.

5. Requests are never lost.

Let $X_{i,r}$ be the random variable representing the number of transmissions needed for packet i to be correctly received by receiver r. Then we have

$$P[X_{i,r} \leq x] = 1 - p^x, x \geq 0 \tag{4.1}$$

Let X_i be the random variable representing the number of transmissions needed for packet i to be correctly received by all of the R receivers. Then we have

$$\begin{aligned} P[X_i \leq x] &= \prod_{r=1}^{R} P[X_{i,r} \leq x] \\ &= (1 - p^x)^R, x \geq 0 \end{aligned} \tag{4.2}$$

Let Y be the random variable representing the maximum number of transmissions among all of the K packets in a page that are needed for a packet to be correctly received by all receivers, that is, $Y = \max_{i=1}^{K}(X_i)$. Then we have

$$\begin{aligned} P[Y \leq y] &= \prod_{i=1}^{K} P[X_i \leq y] \\ &= (1 - p^y)^{R \times K}, y \geq 0 \end{aligned} \tag{4.3}$$

In fact, Y also denotes the number of transmission rounds that the sender needs in order to reliably deliver all of the K packets to all of the R receivers.

Now let Z be the random variable that represents the total number of transmissions that the sender needs to reliably deliver all of the K packets to all of the R receivers. Then we have

$$E[Z] = K \times E[X_i] \tag{4.4}$$

The expected time taken to reliably deliver a page from the sender to the receivers is computed as

$$\begin{aligned} E[T] &= E[Z] \times t_{pkt} + E[Y] \times t_{req} \\ &= K \times E[X_i] \times t_{pkt} + E[Y] \times t_{req}, \end{aligned} \tag{4.5}$$

where t_{pkt} is the transmission time of one packet, and t_{req} is the maximum delay of requests.

$E[X_i]$ and $E[Y]$ can be computed as

$$E[X_i] = \sum_{x=1}^{\infty} (x \times (P[X_i \leq x] - P[X_i \leq x-1])), \tag{4.6}$$

and

$$E[Y] = \sum_{y=1}^{\infty} (y \times (P[Y \leq y] - P[Y \leq y-1])) \tag{4.7}$$

Thus we can derive $E[T]$ by combining the equations 4.2, 4.3, 4.5, 4.6 and 4.7. In our experiments on the TelosB testbeds, we set $K = 48$ and $t_{req} = 256$ ms. t_{pkt} is about 16 ms according to our measurement. As an estimate based on our measurement, we used $R = 4$ and $p = 10\%$ in the above equations, and the resulting $E[T]$ is about 2 seconds, which is the slot length we used in the experiments.

Unlike most TDMA approaches where a node is assigned a time slot only long enough to send one packet, in CORD, we use much longer time slots due to two reasons. First, by having longer time slots, the schedule coordination in CORD is more robust to clock drifts. Second, for most sensor nodes, switching the radio on and off consumes extra energy and time. With longer time slots, the overhead of switching the radio on and off is reduced.

4.2.3 Data Representation

CORD divides the data object to be disseminated into *pages*, each page consisting of a fixed number, K, of *packets*. Each packet contains a fixed number of bytes. Each node maintains an *Availability Bit Matrix* (ABM), M_A, in its internal EEPROM. Bit $M_A(i, j)$ in the ABM indicates whether the node has already received packet j of page i. For a 48 KB data object, the ABM will occupy 268 bytes in the internal EEPROM (assuming each packet has a payload of 23 bytes). This storage requirement is not a problem on a TelosB mote, which has a 16 KB internal EEPROM.

4.2.4 Two-phase Data Dissemination

In CORD, the two-phase data dissemination follows the core construction step. In the first phase, pages of the object are propagated through the core in a pipelined fashion. A core node that has received one or more pages of the object broadcasts an advertisement with this information. Its child core node sends a request to the parent for the desired packets. The parent then broadcasts the requested data packets. Requests are suppressed if another request is being sent for the same or lower numbered page. This process is repeated until the entire object has been received by the downstream nodes in the core. In the second phase, non-core nodes make requests to their local core nodes for any missing data packets.

Unlike Deluge where all nodes broadcast advertisements, and each of them has a possibility to become a data sender, in CORD only core nodes are allowed to send advertisements and data packets. Moreover, all nodes stick to a fixed schedule. The schedules are synchronized such that when a parent node enters a P-slot, its child nodes enter a C-slot, and when the child nodes enter the following P-slot, the parent node enters a Q-slot and turns its radio off. Generally, nodes send advertisements and data packets in P-slots, and send requests and receive data packets in C-slots. However, a node may sleep in P-slots or C-slots if there is no need to send or receive packets, e.g., if it (and its children) has already received the entire object.

In both phases, the protocol uses three message types: ADV, REQ and DATA. An ADV packet contains the sender's address, the number of consecutive complete pages the sender has received (i.e., the pages available to its children) and a field *PhaseFlag* that indicates the current phase of the protocol. A REQ packet contains the destination address, the page ID and a bit vector corresponding to the requested packets. A DATA packet contains the page ID, the packet ID, the payload data, the number of available pages at the sender, and a field *MorePackets* which indicates the number of packets that follows this packet. The last field allows the receiving nodes to know when the current transmission "burst" will be over. Note that all messages are broadcast and can be overheard by any neighboring nodes that are awake.

Each node maintains the following state information:

- n_p: the number of consecutive complete pages at the parent. This information is learned from both ADV packets and DATA packets sent by the parent.
- p_s: the ID of the page whose packets are to be sent.
- V_s: a bit vector of outstanding packets to be sent.
- n_l: the number of consecutive complete pages at the local node.
- M_A: the ABM at the local node as discussed in Section 4.2.3.

First Phase: Propagation over Core Structure

After the core construction, every node has established a schedule, comprising of repeating slots of fixed length L. At the beginning of each slot, a node wakes up for a short time, τ, and goes to sleep if no packets are received from the parent node during this interval. The short wake-up window provides the node an opportunity to detect an ongoing dissemination, while saving energy if the dissemination has not started. If the node receives ADV or DATA packets from its parent during the wake-up window, it assigns the current slot as a C-slot, and starts the C-P-Q schedule if it is a core node or C-Q-Q schedule if it is a non-core node.

As shown in Figure 4.2, a core node v acts as follows in a C-slot (assume that the time when the C-slot begins is t_0):

1. If v has all pages, it goes to sleep for the whole C-slot.

2. if v has fewer pages than its parent u ($n_l < n_p$), it sends a REQ for Page n_l to u after a random delay between t_0 and $t_0 + \tau$, and schedules another request in case that the current request is lost. However, the request is suppressed if v overhears a request to u for Page p_i during the delay, where $p_i \leq n_l$. This rule gives higher priority to the lower numbered pages. The request is also suppressed if v receives a DATA packet from u.

3. On reception of ADV packets and DATA packets from u, v updates its n_p.

4. On reception of each DATA packet from u, v reschedules the next request for a time approximately equal to the current time plus the total transmission time of *MorePackets* packets, where *MorePackets* is retrieved from the DATA packet. v sets the corresponding bit of M_A if v has not received the DATA packet before.

5. The request process is repeated at the scheduled time. At the end of the C-slot, any scheduled requests are canceled, and the node enters a P-slot.

A core node v in a P-slot acts as follows (assume the time when the P-slot begins is t_1, which is equal to $t_0 + L$ in Figure 4.2, where t_0 is the start time of the previous C-slot):

1. If v has a non-zero V_s (i.e., v did not have enough time to send all requested DATA packets by the end of the last P-slot), v starts sending the DATA packets indicated by V_s. If V_s is equal to zero, i.e., there are no pending DATA packets that need to be

sent, v broadcasts an ADV after a random delay between t_1 and $t_1 + \tau$. However, the advertisement is suppressed if v receives a request addressed to v during the delay.

2. On reception of a REQ packet from a child core node w for Page p_r with a bit vector V_r, if v is not already sending a page, then it sets $p_s = p_r$, $V_s = V_r$, and starts sending DATA packets of Page p_s corresponding to the bit vector V_s. However, if v is already in the middle of transmission, and

   (a) if w has requested the same page as the one v is sending (i.e., $p_r == p_s$), then v sets the union of the two bit vectors, V_r and V_s, to be the new V_s.

   (b) if w has requested a lower page than the one v is sending (i.e., $p_r < p_s$), then v terminates the transmission of packets of Page p_s, and starts sending those of Page p_r by setting p_s to p_r and V_s to V_r. This rule means the lower numbered pages have a higher priority and can preempt the transmission of higher numbered pages.

   (c) if w has requested a higher numbered page than the one v is sending (i.e., $p_r > p_s$), then the request is discarded.

   For all three cases, v resumes its ongoing transmission.

3. At the end of the P-slot, v enters a Q-slot.

4. If v has not received any requests in the P-slots of the last consecutive k rounds ($k = 3$, for example) since it received the entire data object, v sets the field *PhaseFlag* (which indicates that it has entered the second phase of the protocol) in the ADV packets sent in future P-slots. A non-core node receiving an ADV packet with this field set will in turn enter the second phase of the protocol and begin to actively make requests for any missing DATA packets.

In the first phase, a non-core node stays awake in its C-slots and passively listens to communications between core nodes unless it has already received all the pages of the object. It turns its radio off and sleeps at all other times.

Second Phase: Local Recovery

A non-core node determines the first phase is over if it receives an ADV packet from its parent core node indicating that the parent has entered the second phase. In the second phase, a non-core node replaces passive listening in the C-slots with the rules followed by a core node in its C-slots in the first phase, i.e., it actively sends requests to its parent core node for the portions of the object which it does not possess. However, unlike a core node in the first phase, it enters a Q-slot instead of a P-slot after a C-slot.

Since the recovery process is local to each network neighborhood, the time used for the second phase largely depends on the size of the data object, the network density and quality of the link between a non-core node and its parent core node. In other words, it is not affected by the network size. While the latency of the first phase increases with the network size, we expect the latency of the second phase to remain relatively independent of the network size.

After receiving the entire object, a non-core node resumes the duty cycles of the old application, or reboots and starts the duty cycles of the new application if the object just disseminated is a code update. A core node takes the same action after a pre-set number of consecutive rounds without receiving any REQ packets in the second phase. However, a core node infrequently broadcasts advertisements of the current object during wake-up time of the duty cycles, in case that a new node joins the network.

The above two-phase data dissemination can be described in terms of state transition diagrams, as shown in Figures 4.3 and 4.4.

Figure 4.3 shows the state transition of a core node in CORD. The states in the diagram are described in brief as follows.

- S_SHORT_LISTENING: this is the initial state of each node in which it listens for any ADV or DATA message from its parent. An exception is the base station, who enters the S_TX_1 state at the beginning of the data dissemination.
- S_SLEEP_0: this is the state in which a node goes to sleep after not receiving any ADV or DATA message from its parent in the short listening window of the current slot.
- S_ACTIVE_RX: a node in this state actively makes requests for the missing packets of the object to its parent, and receives corresponding DATA packets from its parent.
- S_TX_1: a node in this state broadcasts ADV, receives REQ from its child core nodes and broadcasts requested DATA till the next slot (Q-slot).
- S_SLEEP_1: a node in this state sleeps till the next non-Q slot.
- S_TX_2: this is the state for a core node in a P-slot in the second phase. This state differs from S_TX_1 in that $PhaseFlag$ in the ADV packets broadcast by a core node in this state is set to 1, which indicates the phase transition. Thus a core node in this state responds to the requests sent by its non-core child nodes.
- S_SLEEP_2: this is a similar state as S_SLEEP_1. However, a core node in this state sleeps till the next P-slot, since it has received the entire object at this time and sleeps in all C-slots as well as Q-slots.
- S_MAINTAIN: this is the state in which nodes run the normal application while occasionally exchange information of the local data object by means of ADV packets to discover the discrepancy of the objects at nodes. In case such discrepancy exists, the nodes with a newer object transfers the entire object to the one-hop away nodes

that possess an older object. How this one-hop transfer approaches is beyond the CORD protocol.

Figure 4.4 shows the state transition of a non-core node in CORD. While not having the S_TX_1 and S_TX_2 states, the diagram contains one new state, S_PASSIVE_RX. A non-core node, u, enters the S_PASSIVE_RX state after receiving the first ADV or DATA from its parent in the S_SHORT_LISTENING state. In the S_PASSIVE_RX state, u listens to the channel and receives DATA packets. If u receives an ADV from its parent that indicates phase transition, it switches to Phase 2. In Phase 2, the S_ACTIVE_RX state (in C-slots) alternates with the S_SLEEP_2 state (in Q-slots). Non-core nodes in the S_ACTIVE_RX state act same as core nodes in the same state.

4.3 Software Structure

We have implemented CORD in TinyOS [47] for TelosB [5] motes. Figure 4.5 shows the software structure of our implementation of CORD, following the same style used in [50].

The main body of the CORD protocol is implemented as the following modules: Control Plane (ControlPlaneM), Core Setup (CoreM), Core Node (CoreNodeM), Non-core Node (NoncoreNodeM) and Data Plane (DataPlaneM). The Control Plane is the main control module that starts the Core Setup module, and the Core Node module or Non-core Node module depending on the result of Core Setup. The Data Plane module handles the transmission, reception and management of data objects. The modules in CORD are also wired to other components, as shown in the figure, to send and receive TinyOS messages.

The main interfaces in the software structure are listed as follows:

```
interface Core {
    command void setup(uint8_t testId);
    event result_t setupDone(result_t result);
```

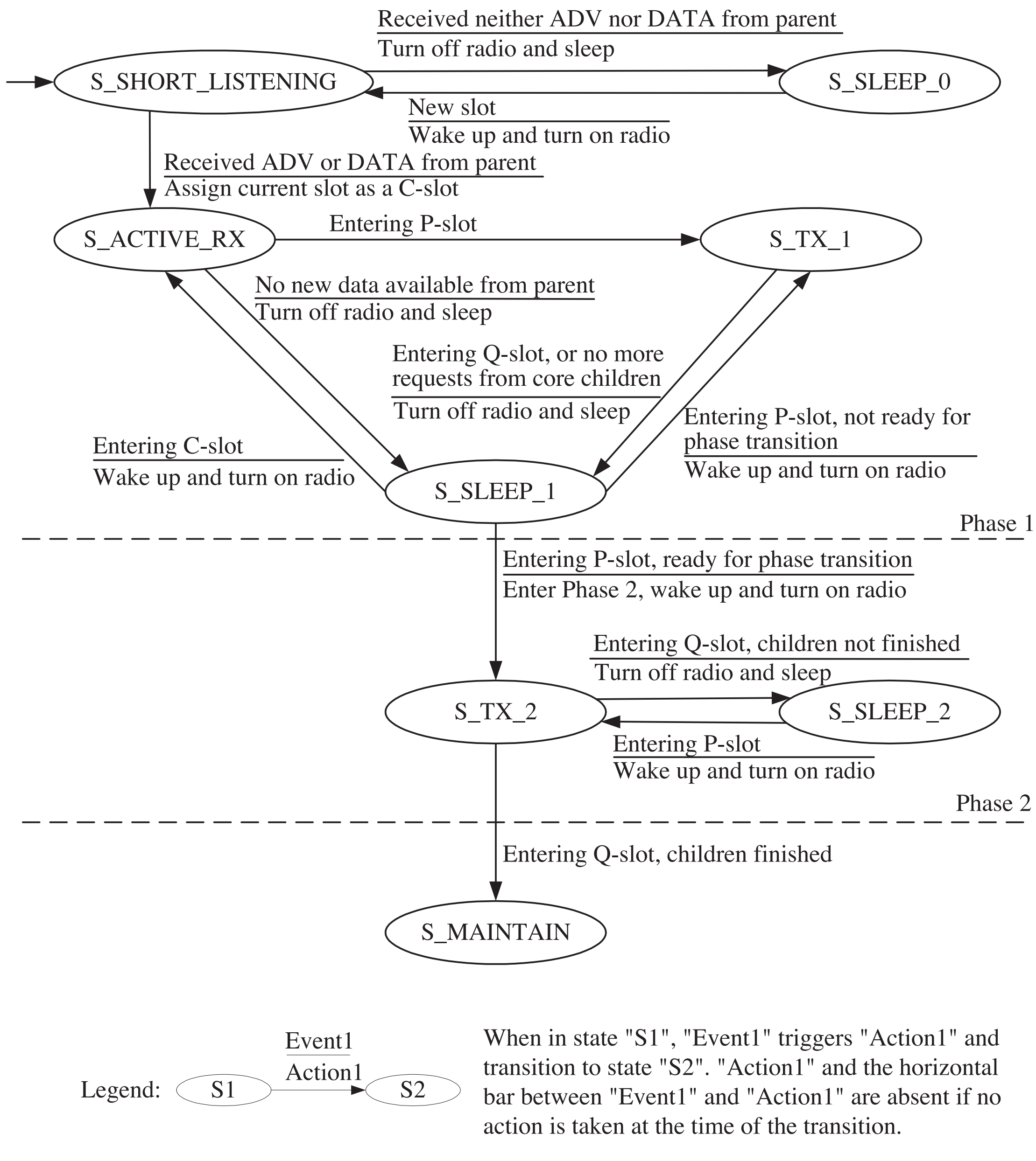


Figure 4.3: CORD state transition diagram for core nodes

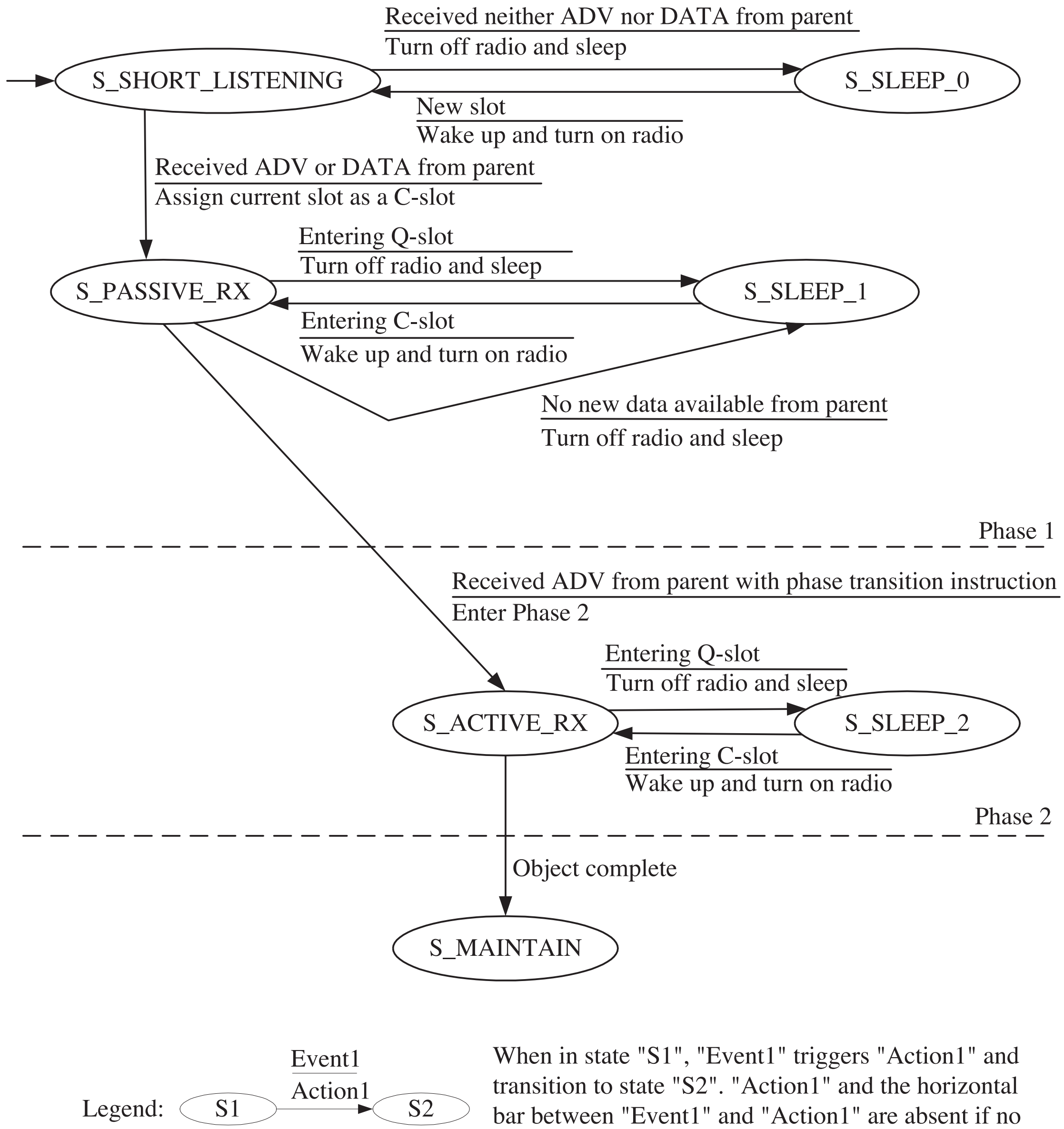
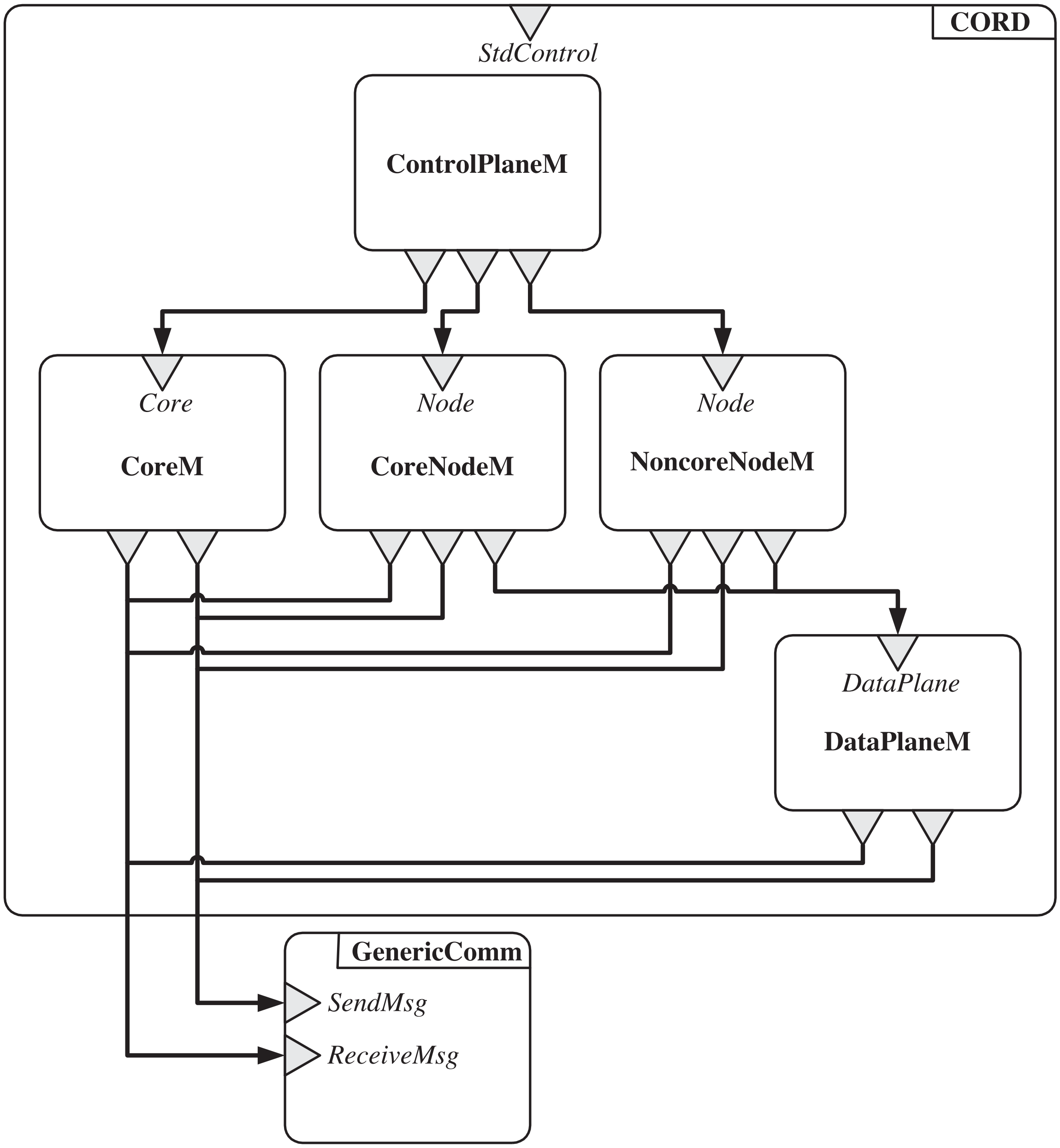


Figure 4.4: CORD state transition diagram for non-core nodes

```
    command bool isCoreNode();
    command uint8_t getDepth();
    command uint16_t getParent();
    command uint8_t getDataChannel();
    command uint8_t getParentDataChannel();
}
interface Node {
    command result_t start();
}
interface DataPlane {
    command result_t startTx(uint8_t objId, uint8_t pageId,
            uint8_t * pktsToSend);
    command result_t startRx(uint16_t parent);
    command result_t stop();
    event void txDone();
    event void receivedDataFromParent(uint8_t pageId, uint8_t morePackets,
            uint8_t pages, uint8_t phase);
}
```

Chapter 6 presents various empirical and simulation results of CORD, along with other existing approaches. As shown in the results, by adopting a core-based two-phase approach and coordinated sleep scheduling, CORD greatly reduces the energy consumption during data dissemination, while maintaining a comparable latency as existing approaches such as Deluge.



Note: Rectangles are modules and configurations (names of configurations are at the top right corner of the rectangles, and names of modules are in the center of the rectangles). Triangles pointing into a rectangle are provided interfaces with names of the interfaces shown aside; triangles pointing out are used interfaces. Lines are wires between modules or configurations.

Figure 4.5: CORD software structure

Chapter 5: McCORD

As we have seen in Chapters 3 and 4, generally McTorrent is designed to reduce the object delivery latency by using multiple channels, while CORD is designed to reduce the energy consumption during data dissemination by using a two-phase approach and aggressive sleep scheduling. Naturally, one would think of a protocol that combines the techniques from the two protocols to have both a reduced latency and reduced energy consumption for data dissemination in sensor networks.

The design of CORD is based on the assumption that the network link quality in a stationary network is relatively stable in the time span of disseminating a data object, which is true as observed from our experiments. Usually at the time of data dissemination, the network has already been deployed for a long time so that the link quality between any two nodes is known based on the previous packet transmissions. If such knowledge is not available, CORD conducts neighbor probing before core construction. The result of probing is used to select the core nodes.

In a multichannel two-phase approach that is extended from CORD, the core nodes may use different channels to transmit data packets in order to reduce latency as the nodes do in McTorrent. According to our experiments (see Section 5.1), nodes may observe different link quality on different channels. Since the core nodes are selected based on link quality on the default channel (and the only channel used) in CORD, the core construction method used in CORD cannot be used in the multichannel approach without change. In this chapter, we present McCORD, which extends CORD by using multiple channels in data dissemination to reduce object delivery latency while taking multichannel link quality variation into consideration. Performance comparison between McCORD and CORD in various empirical and simulation scenarios is presented in Chapter 6, which shows McCORD greatly reduces the object delivery latency while in average consuming a similar amount of energy as CORD.

Before presenting the McCORD protocol, Section 5.1 discusses the multichannel link quality variation observed from our experiments.

5.1 Multichannel Link Quality Variation

In order to learn the lossy characteristics of sensor networks on different channels, we have done various experiments using TelosB sensors in different circumstances, with time periods spanning from half hour to several days. In this paper, we discuss what we observed from two deployments, one in indoor office environment and the other on top of a spacious garage (see Figures 5.1 and 5.2). On the indoor testbed, the sensors were put on desks or bookshelves. On the outdoor testbed, the sensors were put on the floor. There were 802.11 signals in both environments. In the indoor environment, there were also human activities, while in the outdoor environment, the garage was almost empty and there were few human activities nearby. On both testbeds, the experimenter synchronized the sensors' time initially by manually moving an additional sensor that broadcast its time periodically.

The TelosB motes each have 16 preset channels, with frequencies at $(2405 + i \times 5)$ MHz, $(i = 0, 1, 2, ..., 15)$ [57], that is, they are also in the 2.4G band as 802.11 networks. Therefore, the sensor network may have interference with the 802.11 networks if they are deployed in the same place.

In the indoor experiment, all sensors were programmed such that they broadcast a HELLO message every four seconds. At the end of each hour, all sensors switched to a new channel simultaneously. The events of packet transmission and reception were recorded in the sensors' EEPROM. After the experiment, those events were downloaded to a PC

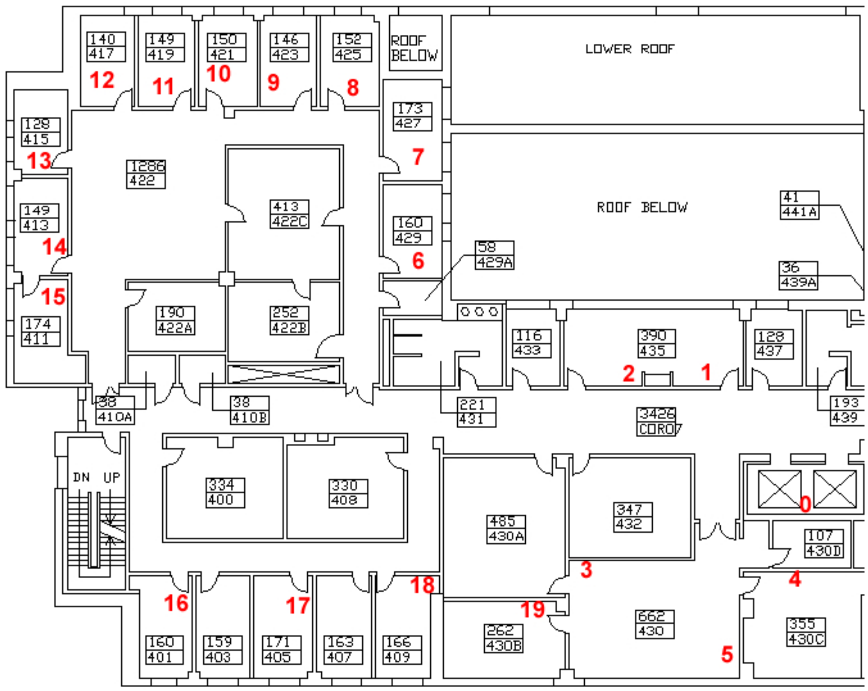


Figure 5.1: Indoor 20-node TelosB testbed

Figure 5.2: Outdoor 3x11 TelosB testbed

for further processing. More specifically, for each pair of sensors, we computed the packet reception rate in each 60-second interval for each direction.

Figure 5.3 shows the computed packet reception rate on the link from Node 16 to Node 17 (see Figure 5.1 for their locations). The X axis of the figure is the time of the experiment in hours. The sensors use a different channel in each hour, that is, Channel 1 in the first hour, Channel 2 in the second hour, and so on. From the figure, we see that the link has a high quality consistently on all of the six channels. In most 60-second intervals, the packets sent by Node 16 were all received by Node 17. Only in a few occasions, one or two of the total 15 packets sent in a 60-second interval were lost.

Figure 5.4 shows the computed packet reception rate on the link from Node 11 to Node 17, which is much different from the previous figure. This link has a longer distance than the previous one, and also worse quality. Moreover, we see the packet reception rate is significantly different on different channels: on Channels 1, 2 and 4, Node 17 received more than 80% of the packets from Node 11 at most times, while on Channels 3, 5 and 6, it received no packets from Node 11. Since Node 17 did receive packets on all of the six channels as Figure 5.3 shows, and Node 11 in fact sent packets on all channels that were received by others such as Node 12 (not shown in the dissertation), we can rule out the hardware problems on the sensors. It also seems the interference of 802.11 does not play in this scenario, since in both cases, the receiver was the same one, Node 17.

Figure 5.5 shows the confidence interval length of mean value of packet reception rate at 95% confidence level ($\alpha = 5\%$). Each data point in the figure stands for the mean value of packet reception rate for a link on the specified channel and its associated confidence interval length. For most links on each channel, especially for those with high mean packet reception rate, the confidence interval length is smaller than 0.05, which indicates stable link quality on each channel.

Similar results are obtained from the outdoor experiments. Based on these results, we

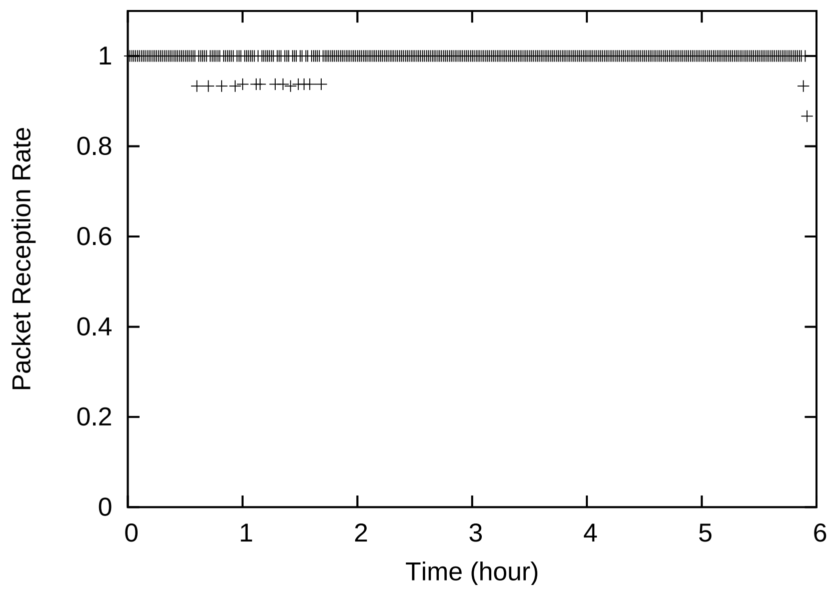


Figure 5.3: A link with consistent high packet reception rate on the indoor TelosB testbed

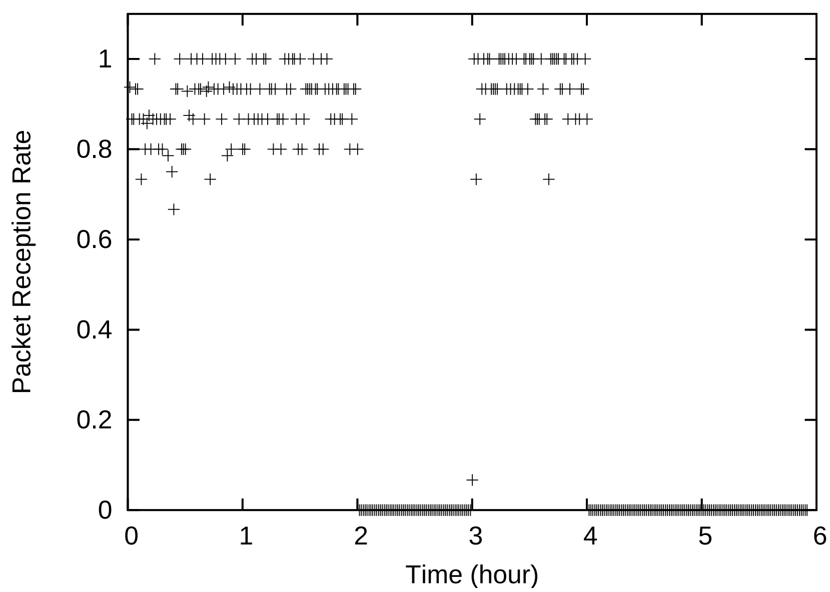


Figure 5.4: A link with significantly different packet reception rate on different channels on the indoor TelosB testbed

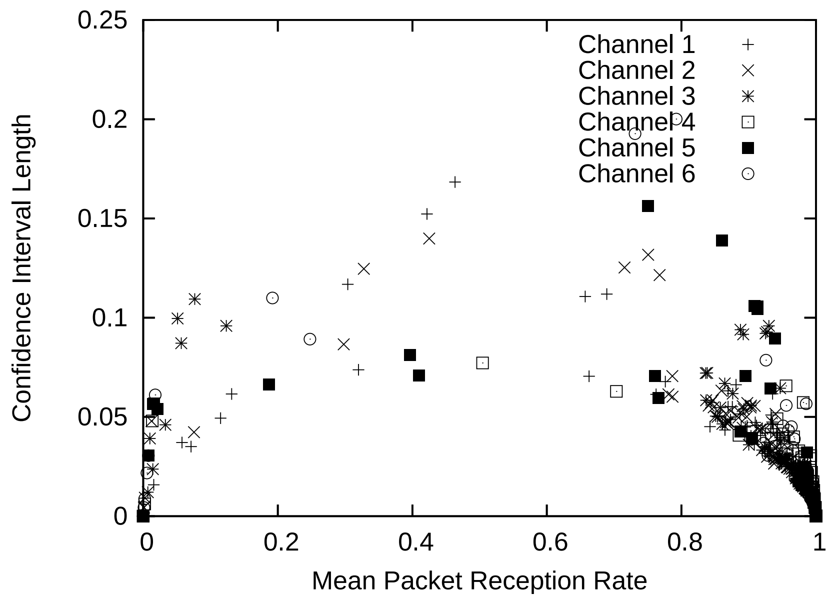


Figure 5.5: Confidence interval length ($\alpha = 5\%$) of mean of packet reception rate on each channel for each link on the indoor TelosB testbed

believe that the stationary TelosB sensors have relatively stable link quality when they stay on one channel. However, when they switch to a different channel, the link quality may also change.

5.2 McCORD

In McCORD, we assume that the sensor network is time synchronized. Prior to disseminating a new object, nodes in the network probe the quality of links between them on all channels that will be used in dissemination, and set up a core structure based on the result of probing. To schedule the probing and core setup at all nodes, the base station broadcasts a SCHEDULE message to the network. The message contains two time values in the future, t_{probe} and t_{core}, which indicates the times when probing and core setup will be started at each node, respectively. The SCHEDULE message is flooded hop by hop to all nodes in the

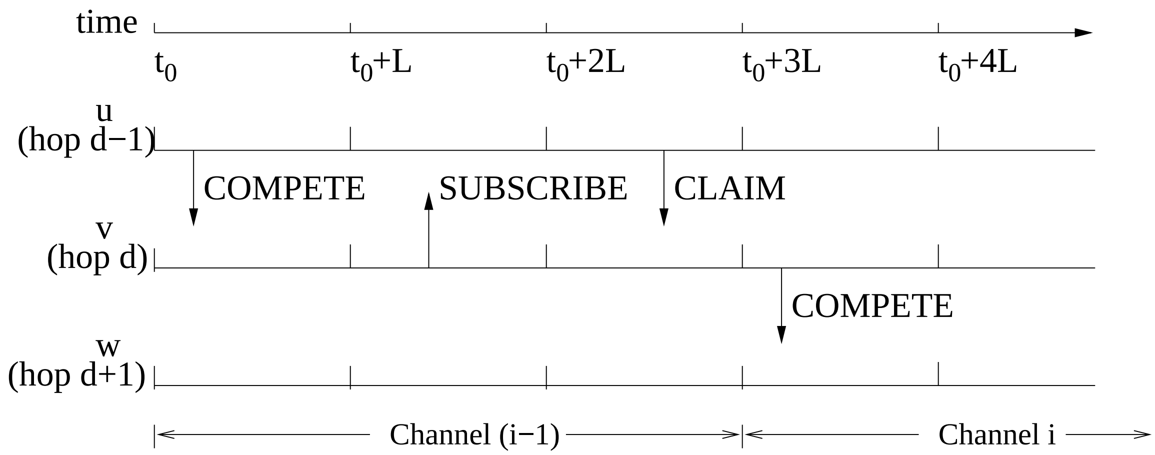


Figure 5.6: Core setup in McCORD

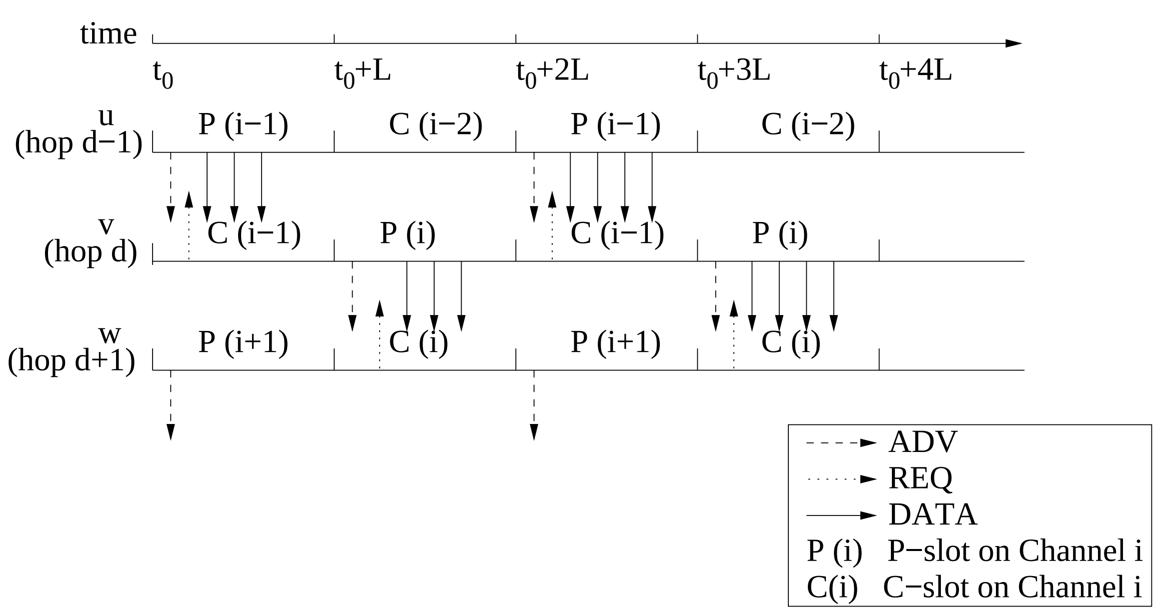


Figure 5.7: Coordinated schedules of adjacent core nodes in the first phase of McCORD

network.

At t_{probe}, all nodes start probing by broadcasting HELLO messages. Assume the data dissemination uses Γ channels ($\Gamma \geq 3$ to reduce collisions among nodes of different hops): c_0, c_1, ..., $c_{\Gamma-1}$. At this probing stage, on each of the Γ channels, every node broadcasts n_{hello} HELLO messages. Every node that receives a HELLO message records the sender's ID and the Link Quality Indicator (LQI) value from the message. After sending n_{hello} HELLO messages, every node computes the quality of the link to each of its neighbors based on the number of the received HELLO messages and the average value of LQI's. Then the nodes exchange the neighbor information by broadcasting NEIGHBORS messages, which contains a list of neighbors, each of which the sender has a good bidirectional link to on the current channel. This process repeats for each of the Γ channels.

At t_{core}, all nodes start a schedule that consists of repeating fixed-length slots of time L. The core setup algorithm is same as the one in CORD, except that nodes start the schedule at the same time and switch channel every three slots, iterating each of the Γ channels and wrapping around, until it becomes a core node or a non-core node. More specifically, at slot 0, nodes all tune to channel c_0. During slots 0, 1 and 2, core setup messages are sent and received on channel c_0. At slot 3, nodes all tune to channel c_1 and so on. Nodes that become core nodes on c_i during core setup will use c_i to send data packets during dissemination. And their child nodes will use c_i to receive data packets. Moreover, what neighbor information that nodes use in core setup depends on the channel that nodes currently are. More specifically, if nodes are on channel c_i, then the neighbor information that was obtained from neighbor probing on channel c_i is used when nodes compete to be core nodes. Figure 5.6 illustrates the core setup process. According to this scheme, core nodes at hop i are assigned channel $c_{(i \mod \Gamma)}$ to send data during data dissemination. Figure 5.8 shows an example core structure of a 25-node network. In the figure, each core node is annotated with a channel number that the node will use to send data packets,

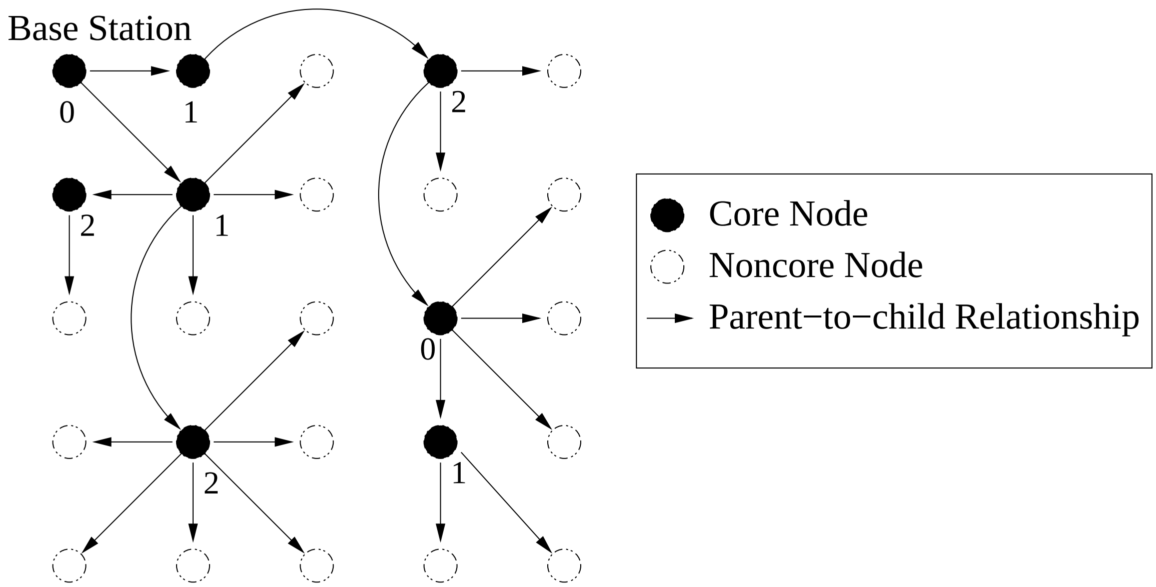


Figure 5.8: An example core structure set up by McCORD

assuming the total number of channels is 3.

The reason of probing link quality on different channels and switching channels during core setup is based on the observation that the quality of the link between two nodes may vary on different channels (see Section 5.1). According to our empirical results, the link quality could change drastically when the two nodes switch to a new channel, especially when they are not close enough. Therefore, the set of neighbors that a node has good links may vary depending on the channel. For the same reason, the core setup messages are also sent and received on the same channel that nodes will use to disseminate the data later.

After core setup, each node still maintains the schedule that consists of repeating fixed-length slots of time L. Similar to CORD, for the core nodes in the first phase of data dissemination, the schedule consists of such slots in which a node takes turns acting as a child (C-slot) and a parent (P-slot). However, in McCORD, changing slots involves channel switch. More specifically, in a C-slot, a core node tunes to its parent's assigned data channel to receive data packets. In a P-slot, a core node tunes to its assigned data channel and sends data packets upon requests from its child nodes. The schedules at a parent and its child nodes are coordinated such that when the parent is in P-slot, the child nodes are in C-slot (Figure 5.7). This coordination is achieved by having all nodes tune to their parents' channel initially except the base station, who starts broadcasting the advertisements on its own channel. A node that receives any advertisements or data packets from its parent determines its current slot as a C-slot, and its next slot becomes a P-slot, and so on (C-P scheduling). For non-core nodes in the first phase, the schedule consists of a C-slot followed by a quiescent slot (Q-slot) (C-Q scheduling). The difference of a non-core node's C-slot and a core node's C-slot in the first phase is that the non-core node passively listens to its parent without making any requests. The nodes sleep during a Q-slot.

A core node enters the second phase when all of its child nodes have received the entire data object. In the second phase, all slots of a core node's schedule are P-slots. And all slots of a non-core node's are C-slots, where they actively make requests to their parents to recover the missing portion of the data object. This process continues until all nodes receive the entire data object.

While similar to CORD, by using multiple channels, McCORD adopts more aggressive pipelining in data dissemination. More specifically, in the first phase, the core nodes in CORD adopt C-P-Q scheduling that contains a Q-slot in which nodes are put to sleep. In McCORD, the core nodes adopt C-P scheduling. The use of multiple channels makes the Q-slot unnecessary. The effect is a reduced latency of data dissemination, while consuming similar amount of energy in average (nodes consume little energy in Q-slots, in which they sleep). The main additional cost is the probing of neighbors on multiple channels. In a single channel approach, nodes may have learned their neighbors in the normal application as a free ride by the time of disseminating a new data object. Due to the fact that most applications currently use one channel only, the neighbor information on multiple channels is usually not available from the normal application. This gives rise to the need of ad hoc neighbor probing. On the other hand, as observed from the experiments, the quality of a link between two stationary nodes is relatively stable over time on a given channel. Thus the information collected from neighbor probing may also be useful to other applications if needed.

5.3 State Transition Diagrams

The McCORD protocol can be described in terms of state transition diagrams, as shown in Figures 5.9, 5.10 and 5.11. Figure 5.9 shows the state transition for each node during core setup in McCORD. Figure 5.10 shows the state transition for each core node in the two-phase dissemination in McCORD. Figure 5.11 shows the state transition for each non-core node in the two-phase dissemination in McCORD. These diagrams are very similar to those for CORD (Figures 4.1, 4.3 and 4.4).

5.4 Software Structure

We have implemented McCORD in TinyOS [47] for TelosB [5] motes. Figure 5.12 shows the software structure of our implementation of McCORD, following the same style used in [50]. In comparison to the software structure of CORD (Figure 4.5), McCORD contains a new module, NeighborProbeM, that probes the link quality and learns neighbor information on each channel that is to be used in data dissemination. Moreover, each of NeighborProbeM, CoreM, CoreNodeM and NoncoreNodeM is wired to interface ChannelState in the ChannelStateM module in order to set the node's channel. The ChannelStateM module is same as the one in McTorrent (Figure 3.4), which interacts with the platform dependent radio component and provides the platform independent interface (ChannelState) to the users of the interface.

Chapter 6 presents various empirical and simulation results of McCORD, along with

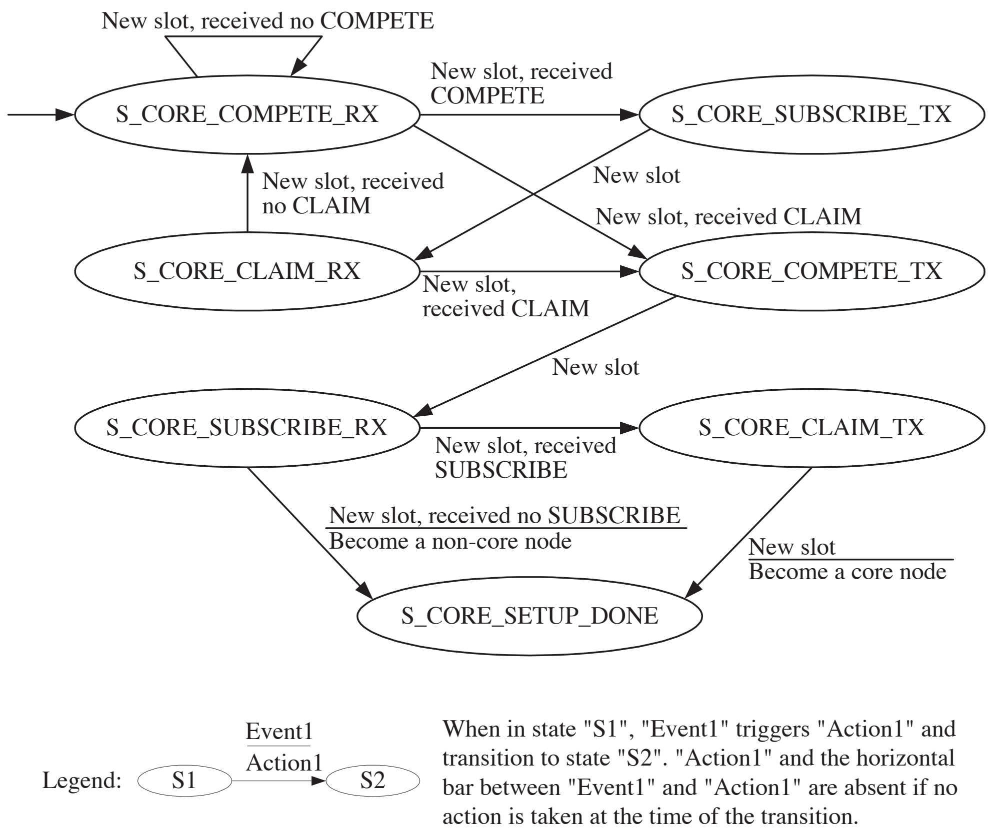


Figure 5.9: McCORD state transition diagram for core setup

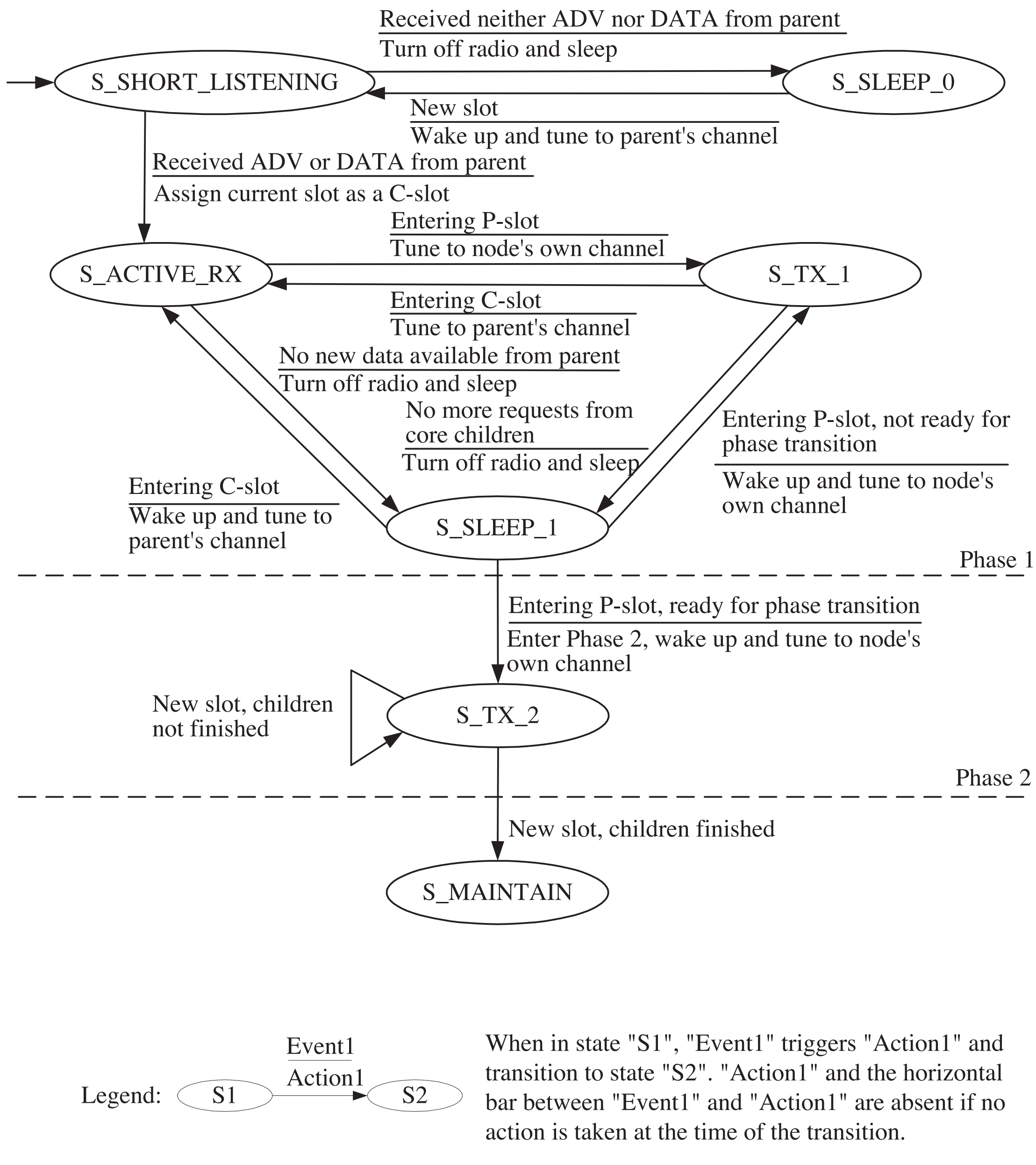


Figure 5.10: McCORD state transition diagram for core nodes

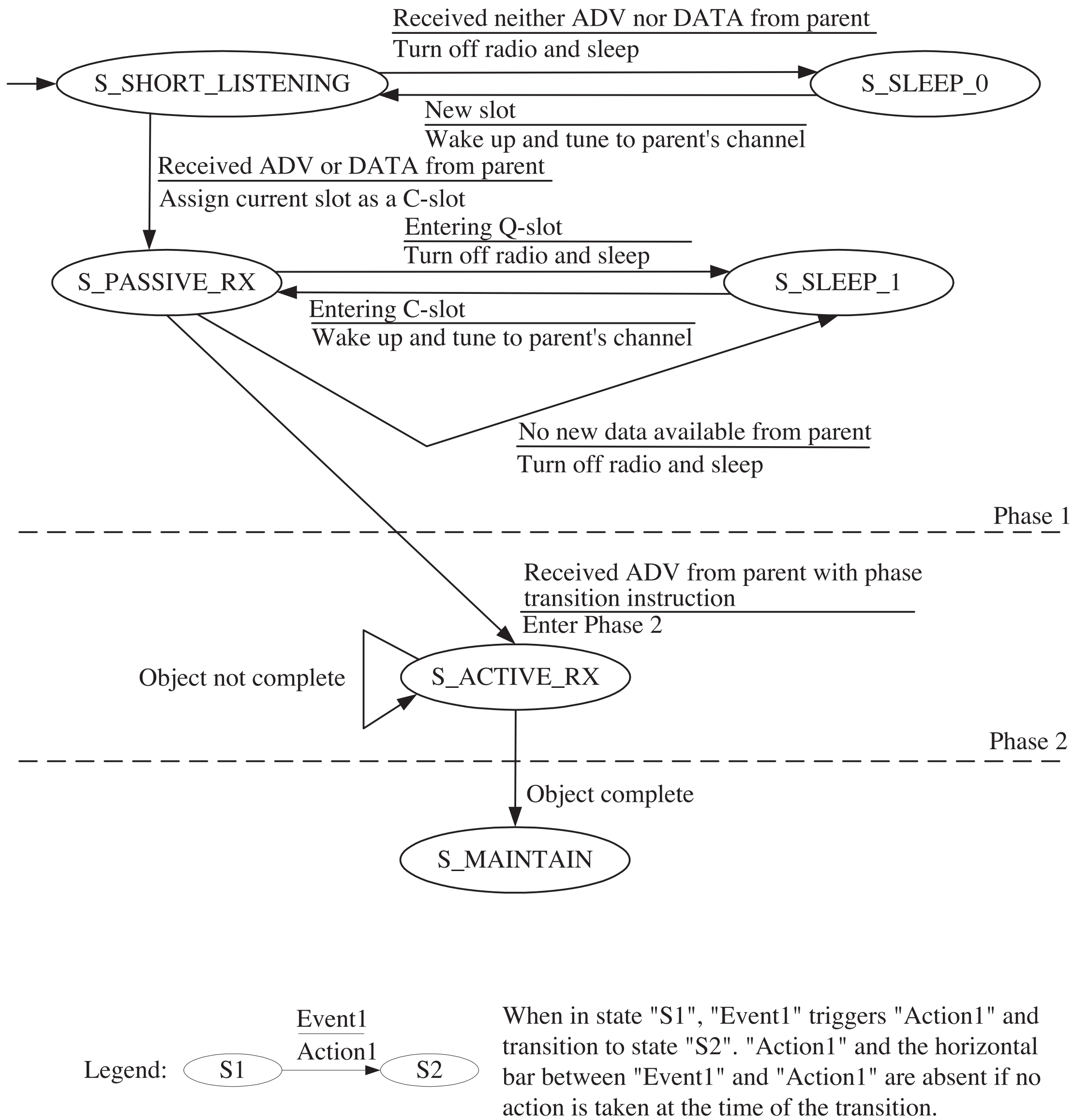
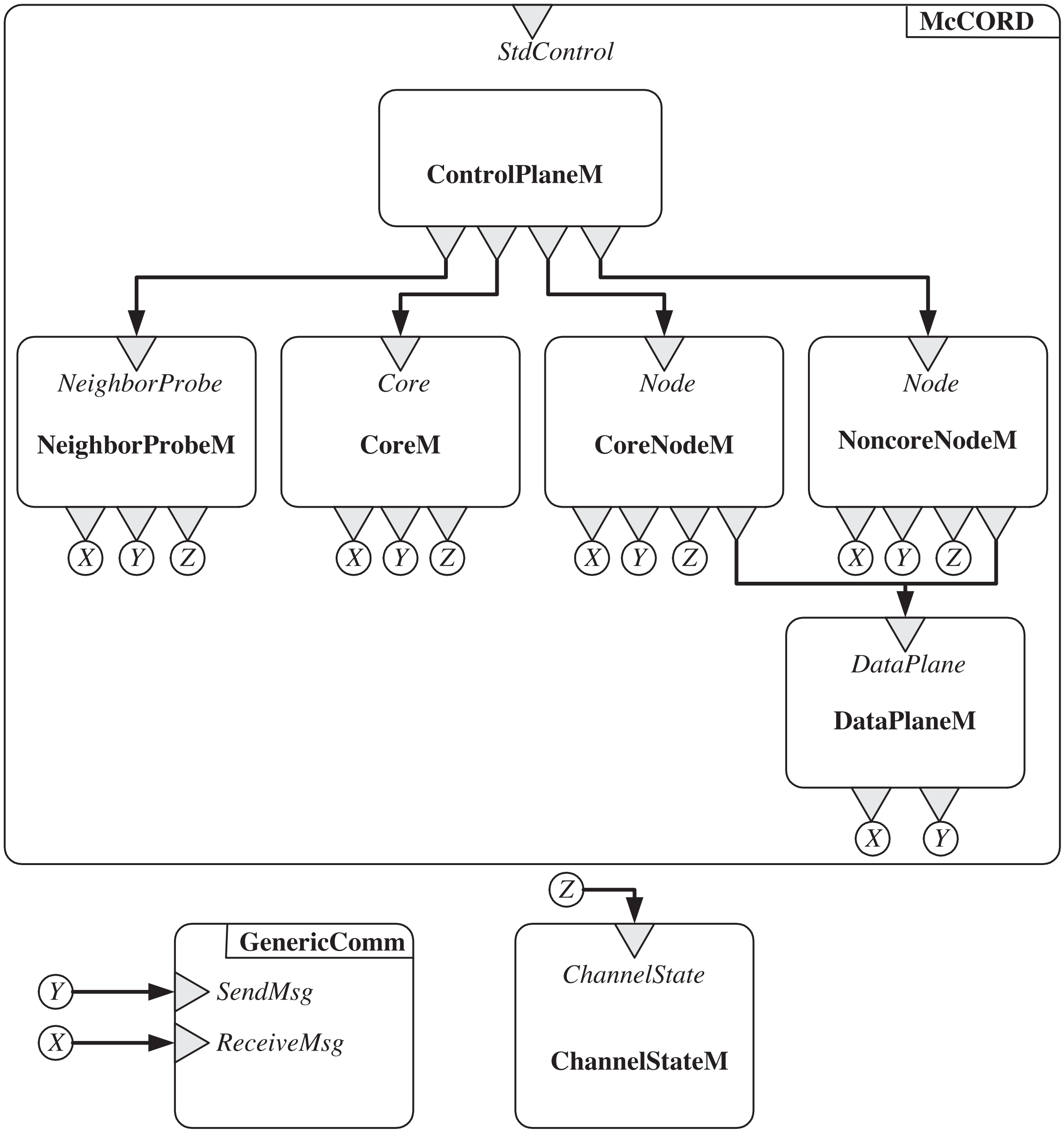


Figure 5.11: McCORD state transition diagram for non-core nodes



Note: Rectangles are modules and configurations (names of configurations are at the top right corner of the rectangles, and names of modules are in the center of the rectangles). Triangles pointing into a rectangle are provided interfaces with names of the interfaces shown aside; triangles pointing out are used interfaces. Lines are wires between modules or configurations.

X Wire to the *ReceiveMsg* interface in **GenericComm**

Y Wire to the *SendMsg* interface in **GenericComm**

Z Wire to the *ChannelState* interface in **ChannelStateM**

Figure 5.12: McCORD software structure

other existing approaches. As shown in the results, by using multiple channels, McCORD greatly reduces the latency of data dissemination, while consuming comparable amount of energy as its single channel counterpart, CORD.

Chapter 6: Performance Evaluation

We implemented McTorrent, CORD and McCORD using the nesC programming language on the TinyOS platform, and evaluated the relative performance of McTorrent, CORD and McCORD via extensive simulations and experiments. We compared the performance of these protocols with existing protocols such as Deluge and MNP. The simulation and experiment code for Deluge and MNP was based on the version included in the TinyOS 1.x distribution [47].

6.1 Performance Metrics

The two important metrics to measure the performance of a data dissemination protocol are the object delivery latency and the energy consumption. The object delivery latency is the time used to disseminate the entire data object to all the nodes in the network. It is the duration from the time when the base station initiates the data dissemination to the time when the last node in the network receives the entire data object. This time denotes the latency until when the sensor network starts functioning fully with the new data (or the new code image in a network reprogramming case), and is critical to time-sensitive applications. On the other hand, most sensor networks are power constrained with sensor nodes running on batteries. The dissemination of a large data object involves a high volume of radio activity and a long duration of system uptime. Given the fact that sensor nodes usually run with low duty cycles in order to prolong the lifetime of the sensor network, dissemination of one large object may consume an amount of energy that can sustain the sensor nodes for days in normal operating state.

In the analysis of simulation and experiment results, we also used other metrics such as number of packet transmissions and individual node uptime. These metrics are described in the following individual sections.

6.2 Empirical Results

We evaluated McTorrent and Deluge on an indoor MICA2 testbed, CORD and Deluge on an indoor TelosB testbed, and all of the four protocols, McTorrent, CORD, McCORD and Deluge on an outdoor Telosb testbed.

6.2.1 Methodology

In our experiments, we logged the events of packet transmissions and receptions, radio on/off operations, and EEPROM reads and writes in the nodes' external EEPROM (not including the writes for logging). After each experiment, the log entries were downloaded to a PC for post-processing. For McTorrent and Deluge experiments on the indoor MICA2 testbed, we report the object delivery latency and number of packet transmissions. For the experiments on the indoor and outdoor TelosB testbeds, we report the object delivery latency and energy consumption for each experimented protocol.

To compare the energy consumption of the protocols experimented on the indoor and outdoor TelosB testbeds, we determined the average energy consumption per node during the time the data object is being disseminated. Due to the lack of a mechanism for directly measuring the residual energy level of the battery in a mote, we used an indirect mechanism for determining the energy consumption of the protocols. We used the log entries recorded in the nodes' external EEPROM and downloaded to the PC to compute the total energy expenditure of each node as well as the contributions of different operations – specifically radio transmissions and receptions, external EEPROM reads and writes, and CPU – to the total energy consumed by a node. The energy consumption of an operation was based on the current specification on the data sheet for the type of nodes. Table 6.1 [5] lists the current specification for the TelosB motes.

Table 6.1: TelosB current specification

CPU current in active state	1.8mA
CPU current in sleep state	5.1uA
Radio current in receive state	23mA
Radio current in transmit state	21mA
Radio current in sleep state	1uA
External EEPROM current in write state	20mA
External EEPROM current in read state	4mA
External EEPROM current in sleep state	2uA

In all experiments, the data object being disseminated consists of 960 packets, divided by 20 pages (each page contains 48 packets). Each packet has a 23 byte payload. Thus the size of the data object is 22,080 bytes. In all experiments, initially, only Node 0, located at one corner of the deployed area, has the entire object.

6.2.2 Indoor MICA2 Testbed

We conducted McTorrent and Deluge experiments on an indoor 4x7 MICA2 testbed (Figure 6.1). We set the radio transmission power of the MICA2 motes to a small value such that the diameter of the 4x7 network was 4-5 hops even in the small lab environment. To show the effectiveness of channel monitoring and selection algorithm in this relatively small network, we set the number of channels available to McTorrent (Γ) to a smaller value, 8, although a MICA2 mote supports more channels.

Figure 6.2 shows the distribution of individual object delivery latencies. While most nodes receive the entire object in 500-660 seconds in Deluge, in McTorrent, all nodes receive the object within 350 seconds, a 47% reduction in latency by using multiple channels. This

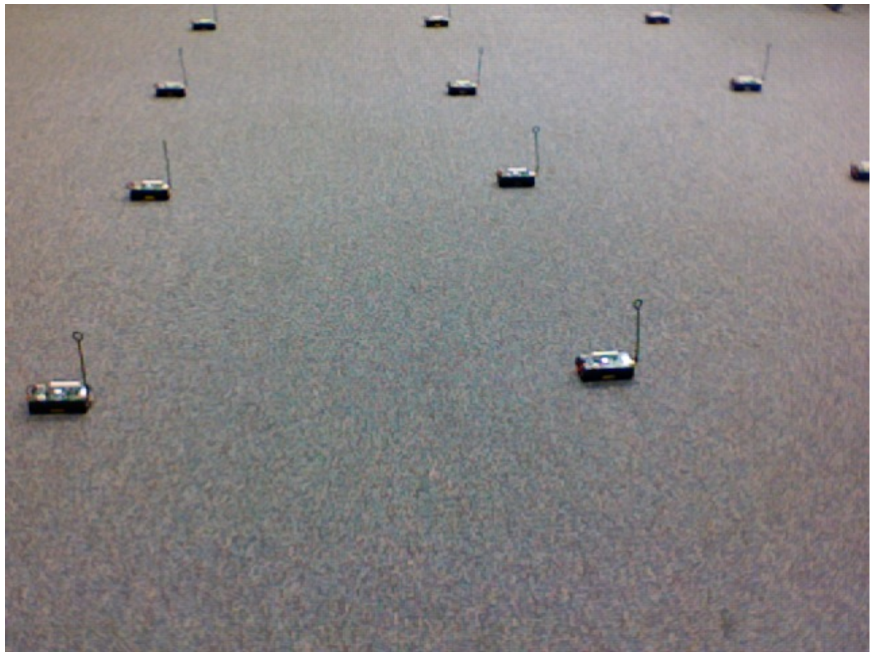


Figure 6.1: Indoor 4x7 MICA2 testbed

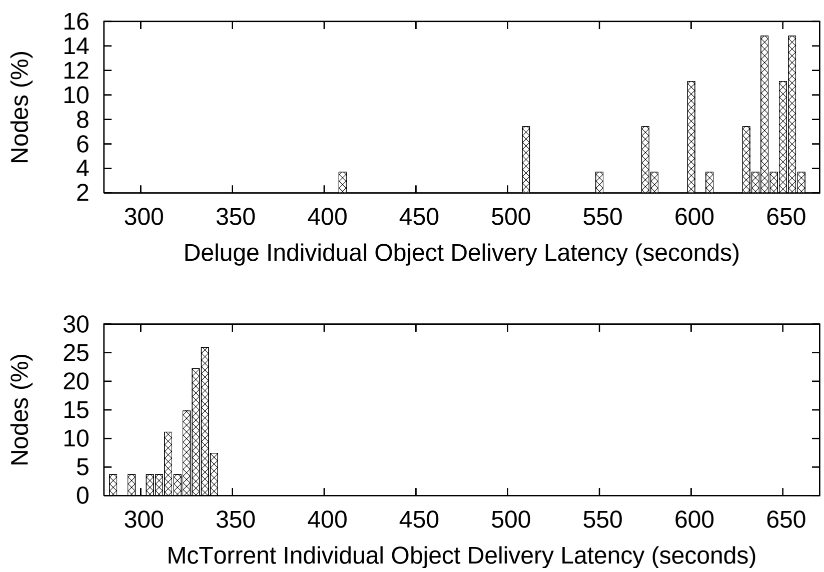


Figure 6.2: Distribution of individual object delivery latency on the indoor MICA2 testbed for McTorrent and Deluge

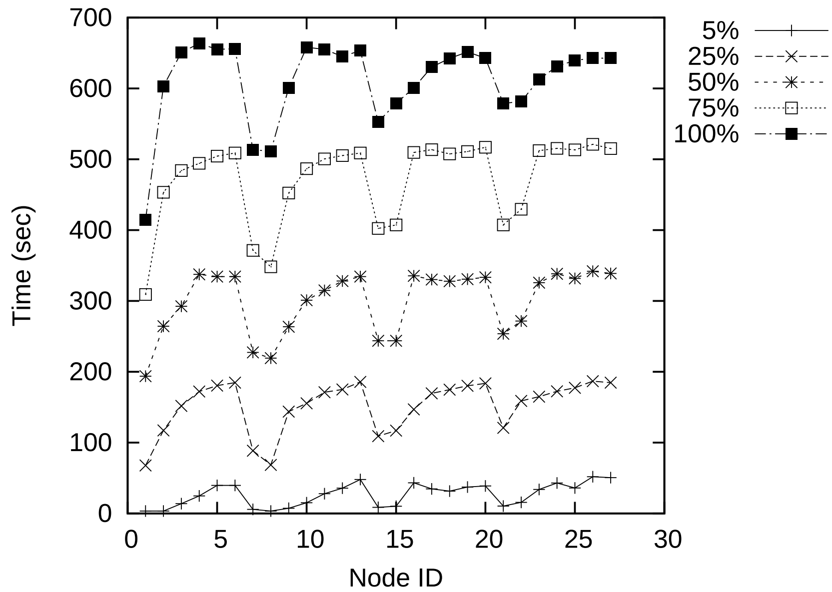


Figure 6.3: Time to receive certain percentages of the data object at individual nodes on the indoor MICA2 testbed for Deluge

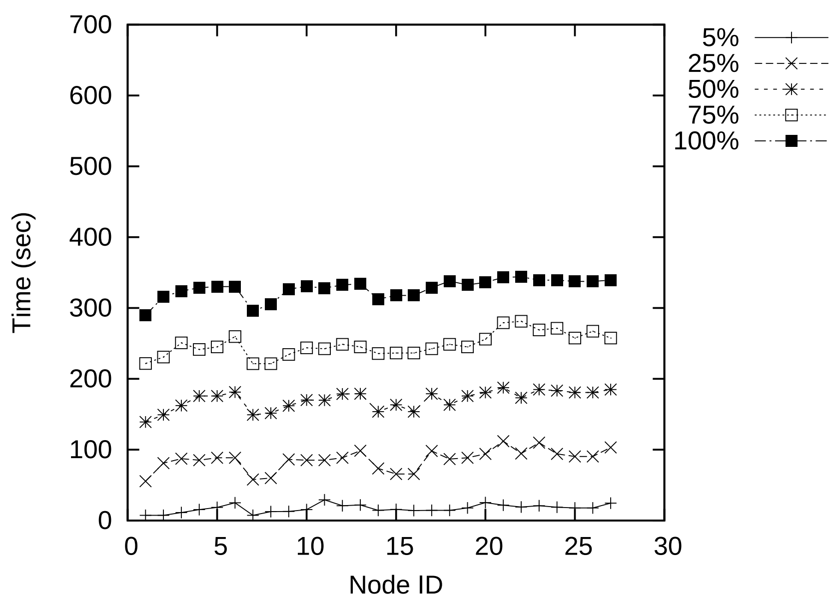


Figure 6.4: Time to receive certain percentages of the data object at individual nodes on the indoor MICA2 testbed for McTorrent

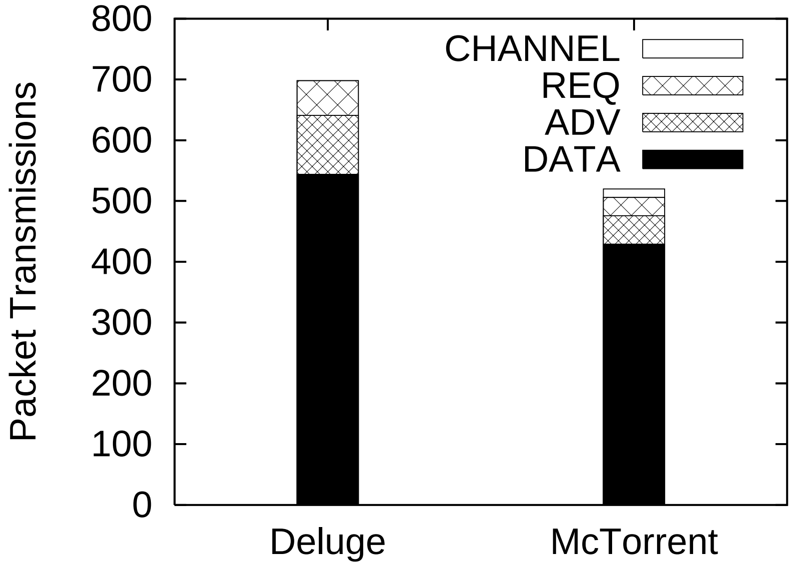


Figure 6.5: Average number of packet transmissions per node on the indoor MICA2 testbed for McTorrent and Deluge

is further illustrated in Figures 6.3 and 6.4, which show the time to receive 5%, 25%, 50%, 75% and all of the data object at individual nodes in Deluge and McTorrent, respectively. We see that every node receives each reported percentage of the object earlier in McTorrent than in Deluge, and the time gap between two adjacent percentages (25% and above) is almost constant at each node in both Deluge and McTorrent. This observation suggests that the latency improvement brought by McTorrent should be consistent when the object size increases.

Figure 6.5 shows a breakdown of packet transmissions during the dissemination (including the packets transmitted by the base station, Node 0) in Deluge and McTorrent. Overall, nodes send 25% fewer packets in average in McTorrent than in Deluge, while the data packets are 21% fewer. 18% of the transmitted packets are control packets in McTorrent, compared to 22% in Deluge.

The average latencies of multiple runs on this testbed for McTorrent and Deluge are 336 ± 11 seconds and 641 ± 23 seconds, respectively (at 90% confidence level). And the average numbers of per-node packet transmissions of multiple runs for McTorrent and Deluge are 505 ± 21 and 654 ± 34, respectively (at 90% confidence level). The empirical results show that when properly designed, using multiple channels reduces both the dissemination latency and packet transmissions, and thus as well reduces the energy expense during the dissemination.

6.2.3 Indoor TelosB Testbed

We also conducted CORD and Deluge experiments on an indoor 20-node TelosB testbed in office environment (Figure 5.1), where the motes were put on desks or bookshelves in the faculty's offices. We ran multiple experiments on the testbed over a period of 32 hours commencing on a Sunday afternoon, covering times when there was little human activity in the building as well as times with both heavy human activity and wireless LAN traffic in the building. An additional node was programmed such that it issued a command to the network once every four hours to initiate the dissemination of an object of the same size. For the CORD experiments, the packet reception rate between a node and each of its neighbors was estimated by exchanging HELLO messages before each run. This information was used by CORD during the core construction.

Although the indoor experiments spanned periods with both light and heavy human activity, we found that the performance of both CORD and Deluge was relatively insensitive to the time at which the experiment was conducted. For example, Figure 6.6 plots the object delivery latency and average node uptime for each experiment in which CORD was used to disseminate the object. We observe that the object delivery latency for CORD varies between 200 seconds to 270 seconds depending on the time of day.

Figure 6.7 plots the object delivery latency for each experiment in which Deluge was used to disseminate the object.

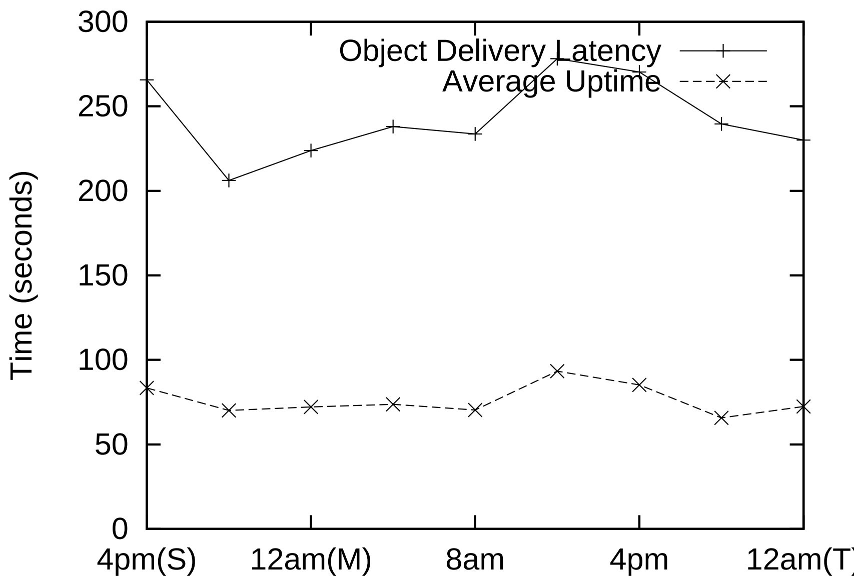


Figure 6.6: Object delivery latency and average node uptime at different times of day for CORD on the indoor TelosB testbed.

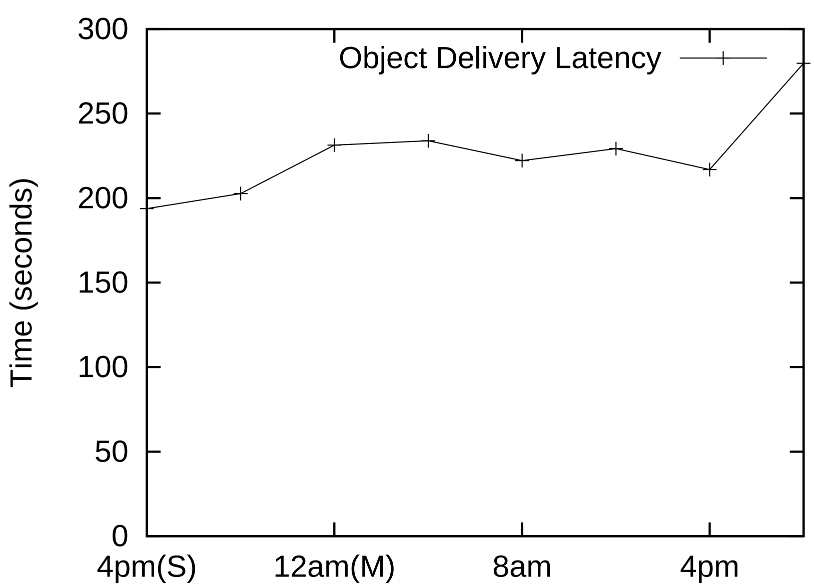


Figure 6.7: Object delivery latency at different times of day for Deluge on the indoor TelosB testbed.

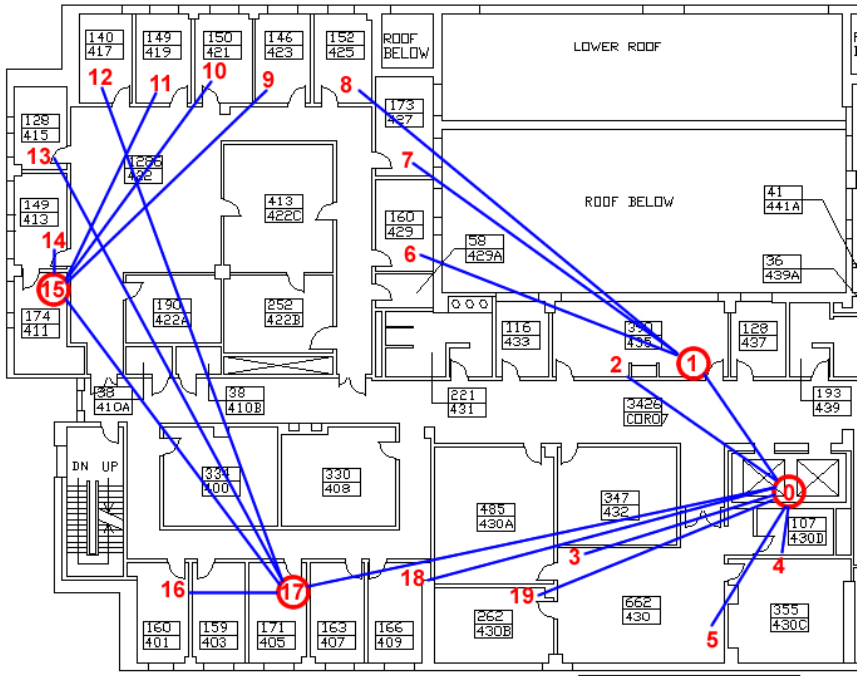


Figure 6.8: The core structure set up by CORD on the indoor TelosB testbed from one experiment (nodes in circles are core nodes)

For CORD, the size of the core structure remains almost constant, consisting of four to six core nodes, although the core nodes may be different in different experiments. Figure 6.8 shows the core structure in one experiment. Nodes with circles are core nodes, and lines show the parent-child relationship between two nodes.

Table 6.2 shows the object delivery latency, average node uptime, average number of packet transmissions per node and average energy consumption per node averaged over the experiments. Note that the average uptime is equal to the object delivery latency for Deluge since nodes do not sleep in Deluge. The latency and uptime for CORD includes the time for core construction, and the number of packet transmissions for CORD includes the messages exchanged during core construction. The energy expenditure per node is computed based on TelosB current specification as shown in Table 6.1 and the measurement results in [58]. The result shows that on average CORD consumes 34% of energy that Deluge consumes. This is because for CORD, on average, a node sleeps for 69% of the time during the dissemination. Further, a node transmits 20% fewer packets in CORD than in Deluge. The object delivery latency for CORD is slightly higher than for Deluge. Note, however, that the network has only three hops, therefore pipelining in Deluge or CORD is not effective in these indoor experiments.

Table 6.2: Average object delivery latency and energy expenditure per node for CORD and Deluge on the indoor TelosB testbed (confidence intervals are shown with 90% confidence level)

	Latency (sec)	Node Uptime (sec)	Number Packet Transmissions	Node Energy(Joules)
CORD	243 ± 14.7	76 ± 5.6	261 ± 32.6	5.66
Deluge	226 ± 17.3	226 ± 17.3	331 ± 21.5	16.81

6.2.4 Outdoor TelosB Testbed

The outdoor testbed we used to experiment all of McTorrent, CORD, McCORD and Deluge consisted of 33 TelosB motes, placed in a 3x11 grid on the floor on the top of the Sandy Creek Garage at George Mason University (Figure 5.2). The spacing between two adjacent nodes in the same row was 2 meters, whereas the spacing between adjacent nodes in the same column was 2.6 meters. Figure 6.9 shows the statistical packet reception rate on a channel as a function of distance between nodes observed in our experiments. We noted that the TelosB motes usually have a larger radio range than shown in the figure when they are placed at an elevation. However, in our experiments, the nodes were placed on the floor in order to form a multi-hop network in a reasonable area. In fact, we found the diameter of the network was 4-5 hops, thus making pipelining more useful than on the indoor TelosB

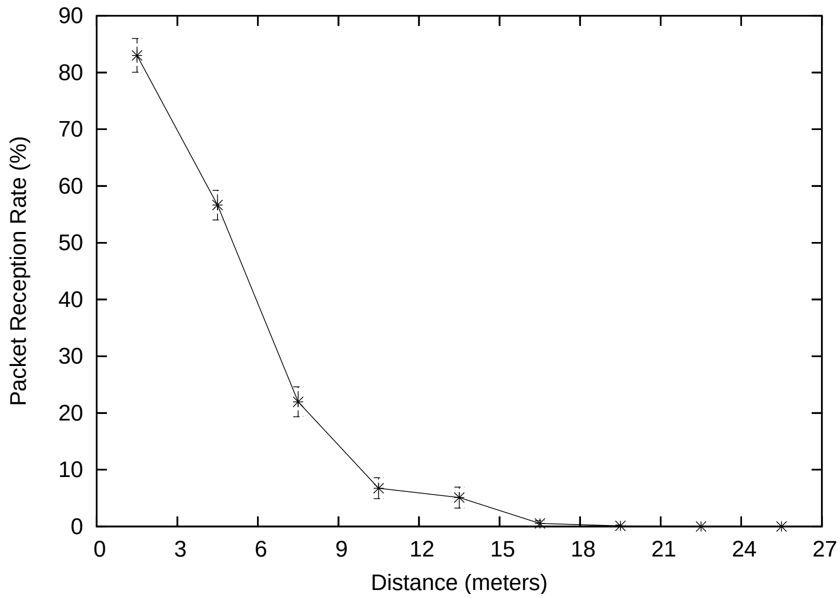


Figure 6.9: Mean packet reception rate as a function of distance observed from outdoor 3x11 TelosB testbed (confidence intervals shown at 90% confidence level)

testbed.

In the experiments on this testbed, McTorrent used 16 channels, while McCORD used 4 channels to have a short neighbor probing time. In the CORD and McCORD experiments, we found that 9 to 11 of the 33 nodes acted as core nodes while the remaining were non-core nodes. The outdoor experiments were conducted when there was little to no human activity near the testbed.

Figure 6.10 shows the distribution of individual object delivery latencies observed in one experiment for each of the four protocols. To deliver the 960 packet object, divided into 20 pages, to all the nodes in the network, Deluge needs 320 seconds, while CORD needs similarly 300 seconds. On the other hand, McTorrent and McCORD use only about 180 seconds to deliver the same object, which is only 56% of Deluge's latency and 60% of CORD's latency. The result shows using multiple channels in dissemination can significantly

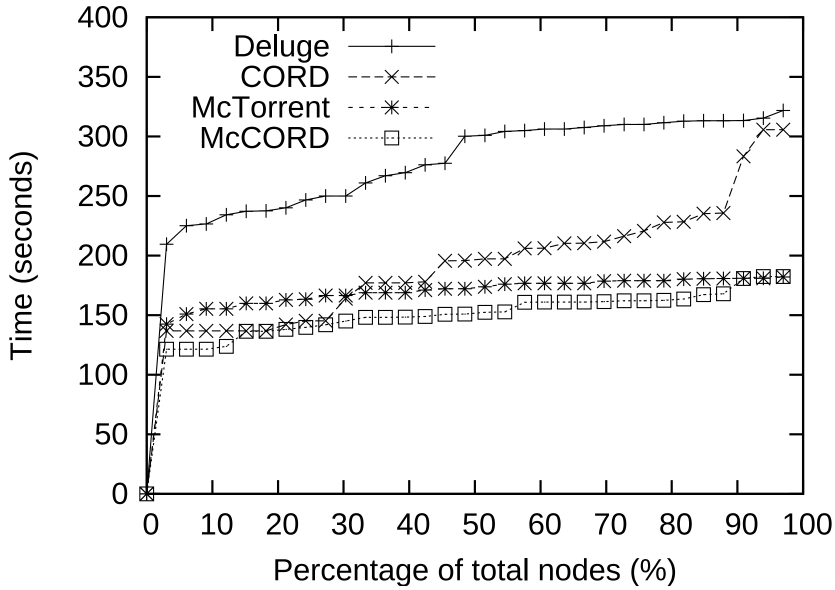


Figure 6.10: Individual object delivery latency in one experiment on the outdoor TelosB testbed

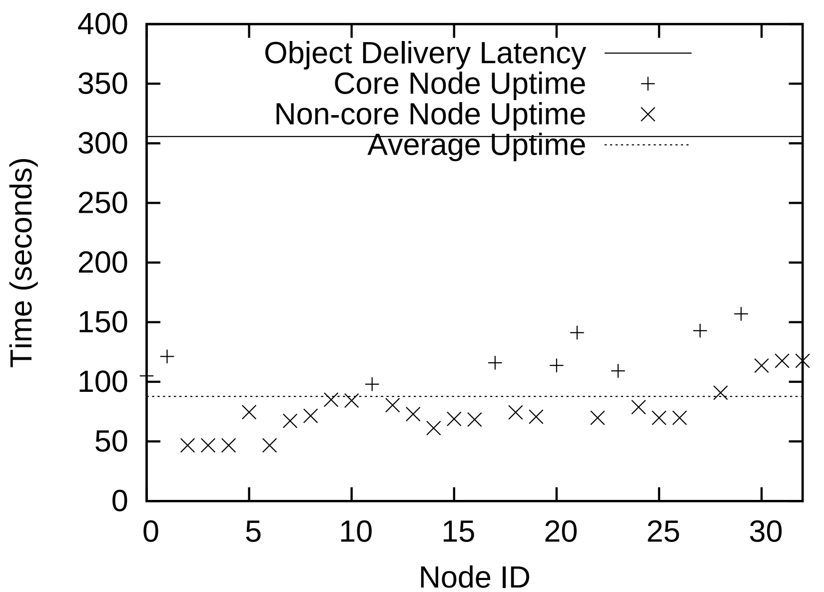


Figure 6.11: Individual node uptime for CORD in one experiment on the outdoor TelosB testbed

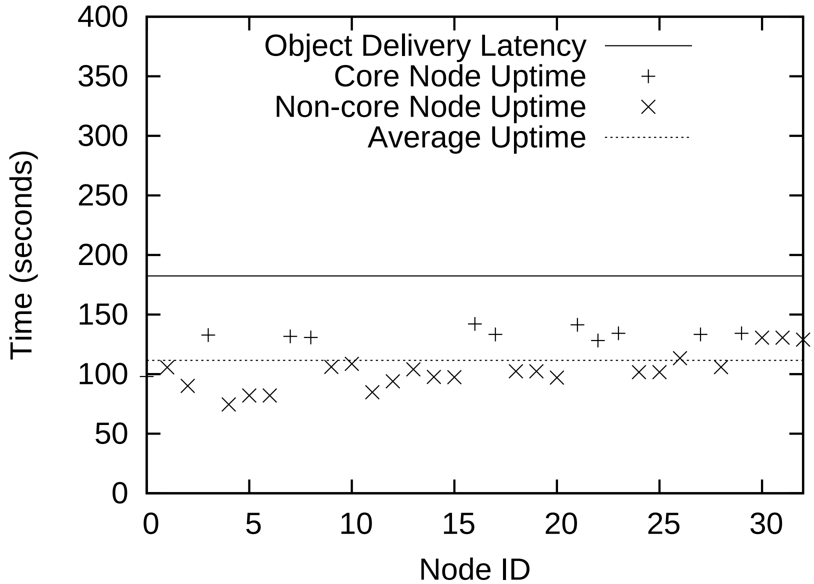


Figure 6.12: Individual node uptime for McCORD in one experiment on the outdoor TelosB testbed

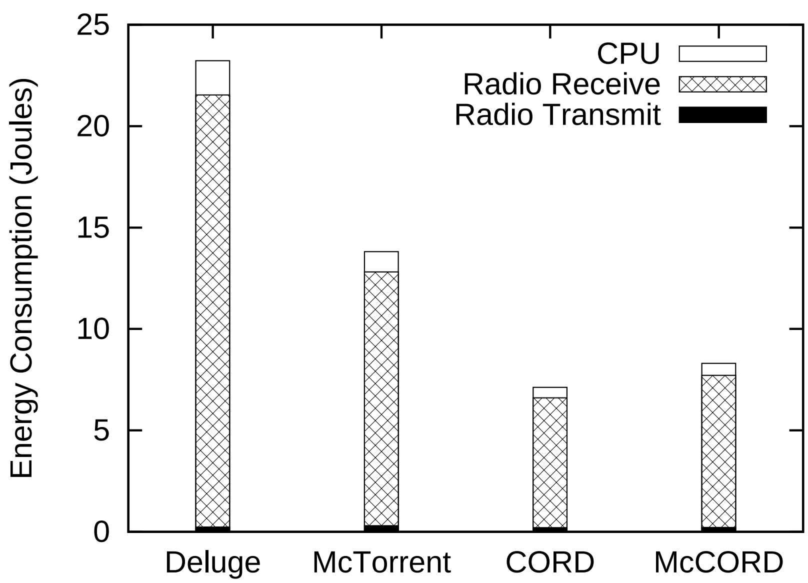


Figure 6.13: Average energy consumption per node on the outdoor TelosB testbed

reduce the object delivery latency.

In Figures 6.11 and 6.12, we show the individual node uptime for CORD and McCORD, respectively. For CORD, a node's uptime is the sum of times when the node is awake from the time when the base station initiates core construction to the time when the last node receives the object. For McCORD, a node's uptime also includes the time for neighbor probing. While the core nodes generally have larger uptimes than non-core nodes in both protocols, on average, a node is up for less than one-third of the time during the dissemination in CORD, and two-third in McCORD. From the figures, we also see that although the latency of McCORD is only 60% of the latency of CORD, the average node uptime in McCORD is slightly more than that in CORD. This is because McCORD removes the quiescent slots in the schedules, which makes the pipelining more aggressive, but does not reduce the node uptime in overall. Moreover, McCORD needs extra time for neighbor probing.

Figure 6.13 shows the average energy consumption per node for each protocol with a breakdown for CPU and radio (energy consumed by other components takes less than 0.5% of the total energy and is not shown in the figure). We see that the vast majority of energy is consumed in radio receive state when a node is listening to the channel for incoming packets. Thus, the most effective way to reduce energy consumption is to keep radio off as much as possible. This is also illustrated by the close correlation between average node energy consumption and average node uptime in the previous figures (Figures 6.10, 6.11, 6.12 and 6.13) (for Deluge and McTorrent where nodes don't sleep, the average node uptime is same as the object delivery latency). More specifically, McTorrent uses 56% of time that Deluge uses to deliver the object, and also similar percentage of the energy that Deluge uses. On the other hand, McCORD uses a slightly larger amount of energy than CORD, since the average node uptime of McCORD is slightly more than that of CORD.

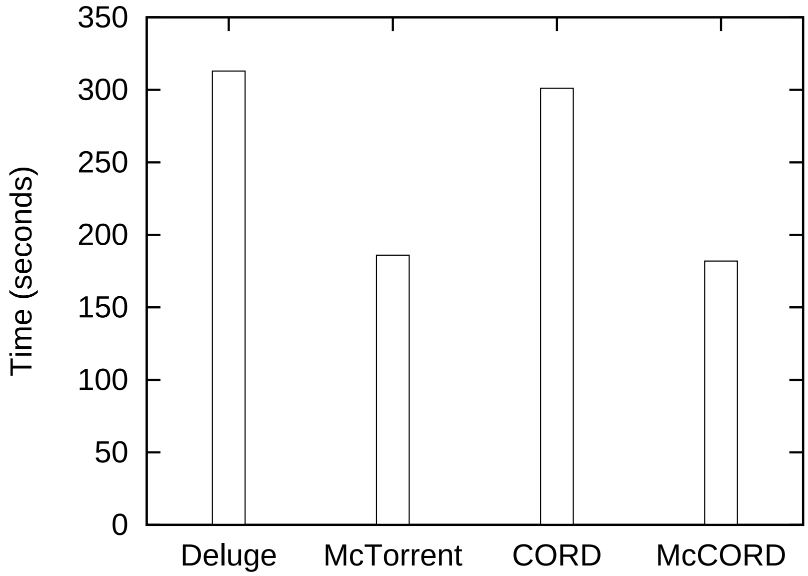


Figure 6.14: Mean values of object delivery latency of multiple runs on the outdoor TelosB testbed

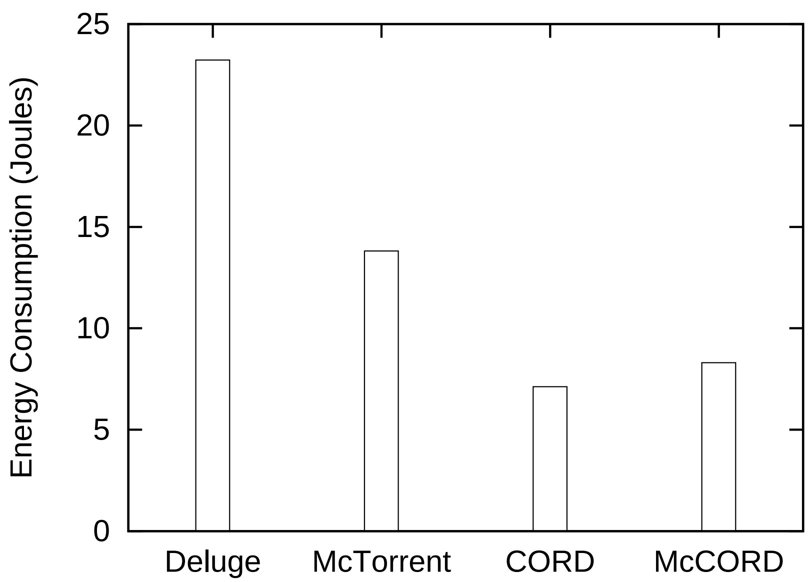


Figure 6.15: Mean values of energy consumption of multiple runs on the outdoor TelosB testbed

Figures 6.14 and 6.15 show the mean object delivery latency and per-node energy consumption of multiple runs on the outdoor TelosB testbed for each protocol. All results have 90% confidence intervals with width less than 10% of the mean (not shown). The results are consistent with the aforementioned results from an individual run.

6.3 Simulation Results

We used the TOSSIM simulator [59] to evaluate the performance of the protocols in more extensive scenarios. TOSSIM is a discrete event simulator for TinyOS wireless sensor networks that simulates communication between nodes at the bit level, and captures the behavior of the entire TinyOS network stack. For our simulations, we modified the radio models (for MICA and MICA2 motes) in TOSSIM to support multichannel functionality. Due to limitation of TOSSIM, we assume all channels are equal in simulations of the multichannel protocols (McTorrent and McCORD).

In TOSSIM, the network is modeled as a directed graph with each vertex corresponding to a node. Each edge has a bit error rate modeling the probability that a bit can be corrupted if sent along that link. The bit error rate is independently chosen based on the distance between the communicating nodes and a random distribution derived from empirical measurements of a certain platform (such as MICA or MICA2). A file that contains the bit error rates of all possible links of a network is called a loss model, and is passed to TOSSIM as an input.

We did two simulation studies. One focused on performance comparison of McTorrent and Deluge on MICA loss models. The reason that we used MICA loss models is that MICA loss models were also used in previous simulation studies [1], and for comparison purpose, we used some similar scenarios in our study as in the previous studies. The second study focused on performance comparison and energy consumption comparison of protocols including McTorrent, CORD, McCORD, Deluge and MNP, using MICA2 loss models.

6.3.1 Performance Comparison of McTorrent and Deluge

In this section, we report the simulation results of McTorrent and Deluge on MICA loss models.

Methodology

In this simulation study, we compared the performance of McTorrent and Deluge using MICA loss models (the code of McTorrent and Deluge was compiled with MICA radio model accordingly). Figure 6.16 shows the MICA motes' means and standard deviations of packet reception rate at various distances, which are derived from empirical measurements and used in TOSSIM to generate the loss models. The data and tools to generate the loss model from a given network model are provided by the TinyOS distribution. From the figure, we can see that basically, the packet reception rate decreases with the distance. Packets are hardly received at a distance longer than 40 feet.

For the network models used in the simulations, we considered both grid topologies and random topologies. In grid topologies, nodes are organized in a rectangular grid, while the base station is always at one corner of the grid. In random topologies, nodes are randomly put in a rectangular area based on a uniform distribution, while the base station is always fixed at one corner of the area. In this section (6.3.1), a grid network is denoted by mxn-s, where m and n are the dimensions of the rectangular grid topology, and s is the distance between a node and its closest neighbor measured in feet.

In our simulations, we also varied network parameters such as network size, object size and network density. Note that some of the network scenarios and parameters considered in our simulations are identical to those used for some of the simulations reported in [1]. Each simulation was run multiple times with different random number seeds. The values reported and shown in the figures are the average values from multiple runs unless stated otherwise. A confidence interval with 95% confidence level is drawn for each of the average

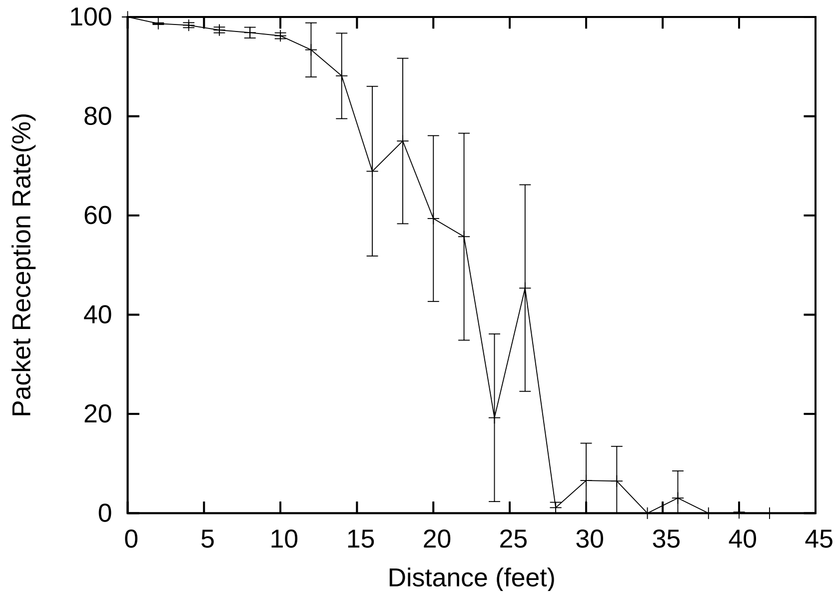


Figure 6.16: MICA's mean and standard deviation of packet reception rate as a function of distance, provided by TOSSIM

values.

In the simulations, the data objects to be disseminated are divided into pages, where each page consists of 24 packets and each data packet has a 23 byte data payload. If not stated otherwise, McTorrent used 16 channels.

Effect of Number of Channels

To evaluate the impact of multiple communication channels, we considered various configurations of McTorrent with different numbers of available channels. Specifically, we considered configurations with number of available channels Γ equal to 1, 4, and 16 (denoted as McTorrent-Γ). We also considered a channel configuration (denoted by $\Gamma = x$) in which each node was statically assigned a unique data channel. This configuration represents an ideal case in which there is no contention between nodes for channels.

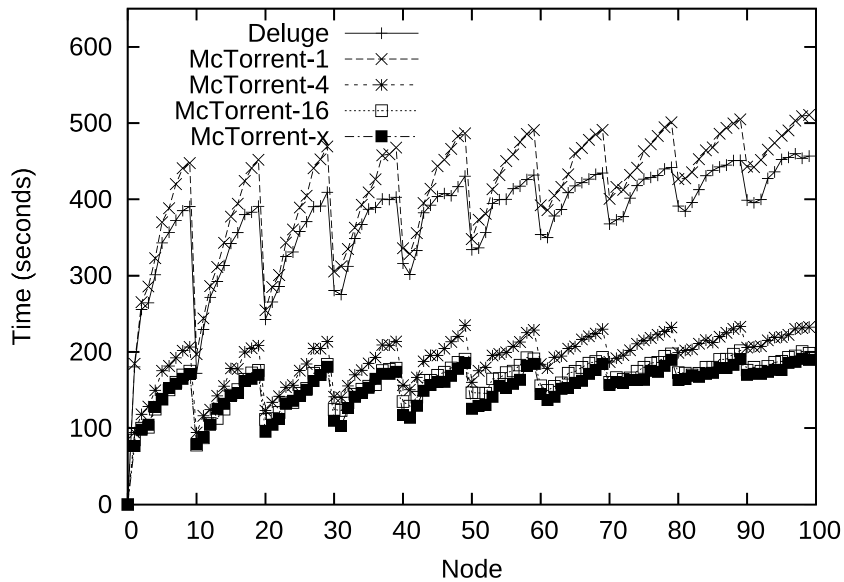


Figure 6.17: Object delivery latency to disseminate 5 pages to a 10x10-10 MICA network for McTorrent using various numbers of channels

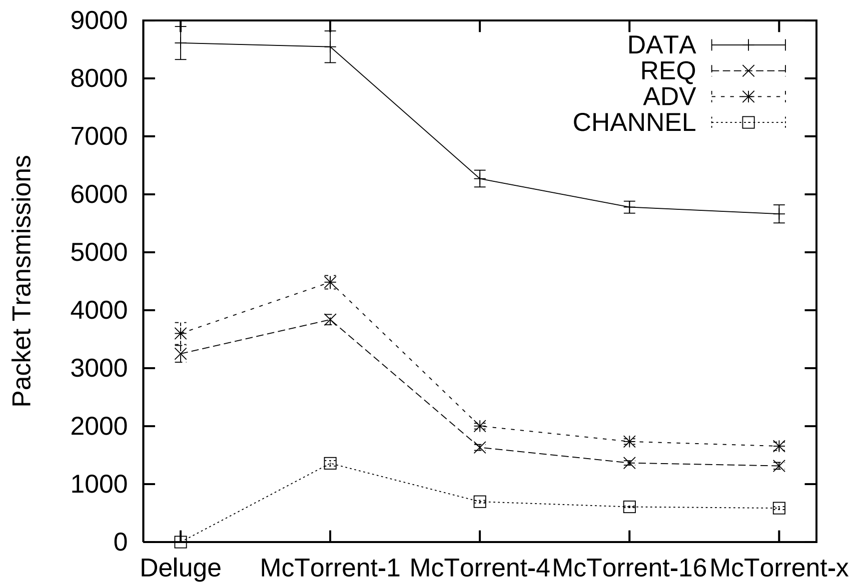


Figure 6.18: Total number of packet transmissions to disseminate 5 pages to a 10x10-10 MICA network for McTorrent using varous numbers of channels

We simulated these McTorrent variations and Deluge to disseminate five pages over a 10x10-10 grid network. Figure 6.17 shows the time for each node to receive all of the five page (as known as individual object delivery latency). We see that the object delivery latency at each node for McTorrent (with four or more channels) is lower than that for Deluge and McTorrent with a single channel. The latencies for Deluge and McTorrent-1 are comparable, reflecting the fact that they are essentially the same protocol. McTorrent-4 and McTorrent-16 have larger latencies than McTorrent-x, showing the impact of contention between nodes for communication channels.

Figure 6.18 shows the total number of packet transmissions. In comparison to Deluge, although McTorrent introduces about 1000 channel claim packets, for configurations with more than 4 channels, the number of advertisements, requests and data packets is reduced by more than 1000 packets each.

Effect of Network Size

The savings in transmissions become more significant when the network size increases, as shown in Figure 6.19. In this Figure, we plot the number of control and data packets for Deluge and McTorrent (with 16 channels) for different network sizes. The control packets in the figure include all of the advertisements, requests, and channel claim packets. For a network of 400 nodes, the total number of data packets transmitted is 60% lower for McTorrent than Deluge, and this trend continues for larger networks.

Next, in Figure 6.20, we plot the object delivery latency for propagating five pages to the entire network for McTorrent and Deluge as a function of the network size. We observe that the latency of McTorrent is 2 to 4 times smaller than the latency of Deluge, depending on the size of the network.

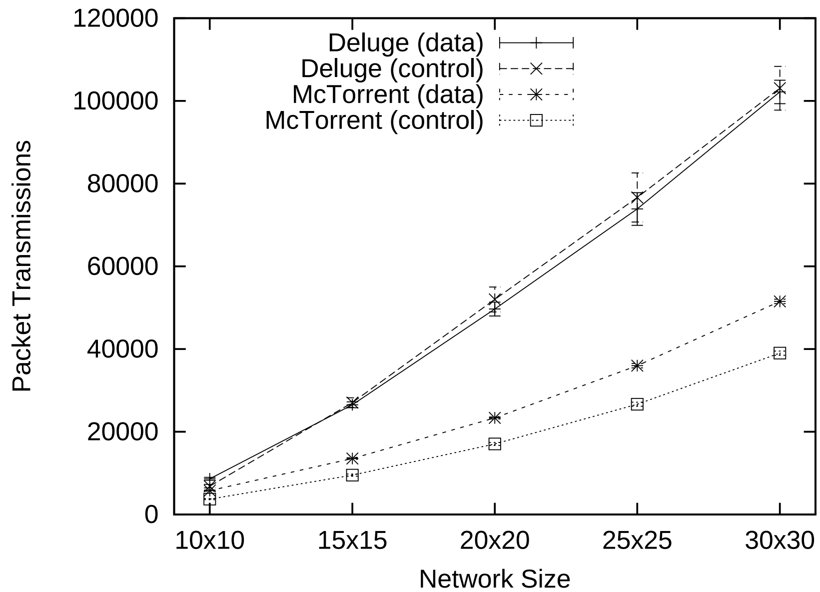


Figure 6.19: Total number of packet transmissions from MICA simulations with different network sizes for McTorrent and Deluge

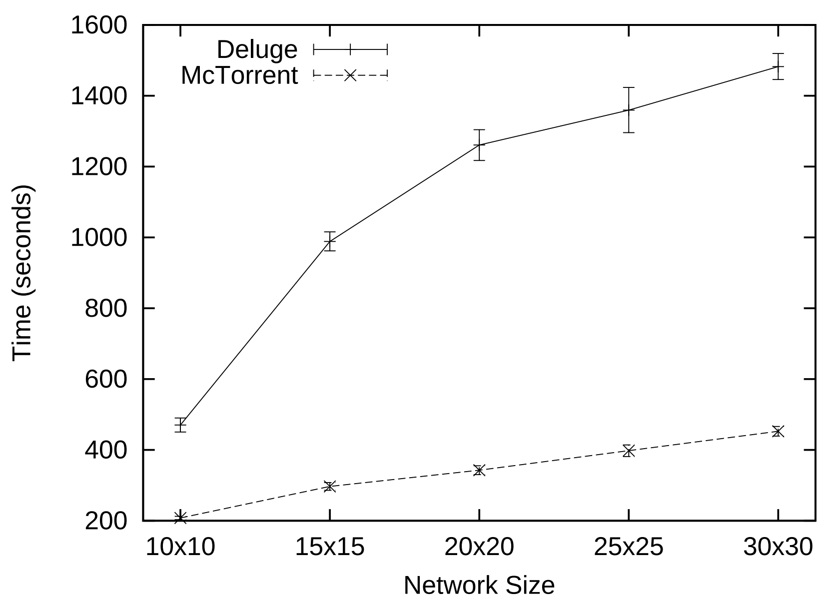


Figure 6.20: Object delivery latency from MICA simulations with different network sizes for McTorrent and Deluge

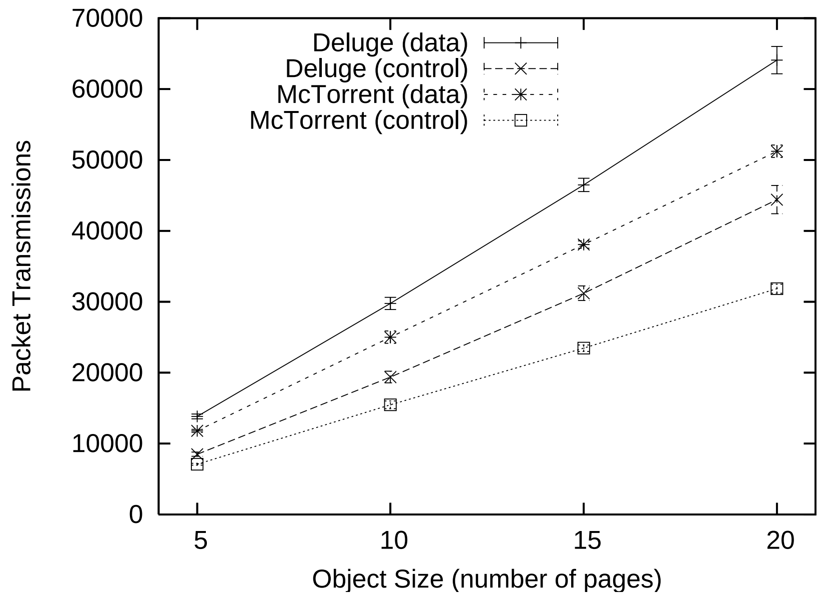


Figure 6.21: Total number of packet transmissions from MICA simulations with different object sizes in a 5x40-10 network for McTorrent and Deluge

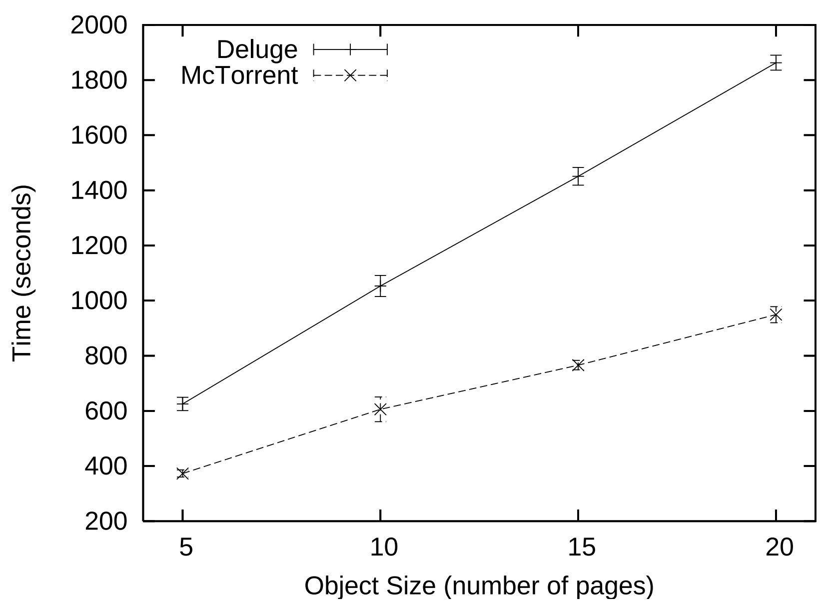


Figure 6.22: Object delivery latency from MICA simulations with different object sizes in a 5x40-10 network for McTorrent and Deluge

Effect of Object Size

Figure 6.21 and 6.22 show the total number of packet transmissions and object delivery latency for Deluge and McTorrent to disseminate data objects of variable sizes. In order to have a sufficient large network without increasing simulation time by too much, we used a 5x40-10 network. We see that McTorrent uses 15%-24% fewer packets than Deluge to propagate each object, and the saving is more significant for larger objects. The object delivery latency of McTorrent is consistently half of that of Deluge for disseminating the objects ranging from 2,760 bytes (5 pages) to 11,040 bytes (20 pages).

Effect of Network Topology and Network Density

We also ran simulations on random topologies where nodes are randomly put in a 200ft x 200ft space based on a uniform distribution. We varied the network density by using different numbers of nodes. The loss rate between any two nodes is derived from the empirical model in TinyOS distribution based on the distance of the two nodes (Figure 6.16). The size of the data object is fixed to five pages for this set of simulations.

Figure 6.23 shows the object delivery latency to disseminate five pages to networks with different densities. A network density is denoted by the average number of nodes per square feet. For comparison, a grid network with 10 feet spacing corresponds to a network density of 0.01. As we see from the figure, in Deluge, the latency changes little when the network density increases from 0.005 nodes/square feet (200 nodes in total) to 0.01 (400 nodes in total). The increase of the density in this range improves the quality of links between nodes, leading to the almost constant object delivery latency when the number of nodes in the network increases from 200 to 400. However, when the network density continues to increase, the latency increases dramatically, due to the increasing number of packet collisions. This trend does not occur in McTorrent results. In McTorrent, the object delivery latency decreases when the network density increases from 0.005 to 0.01 and

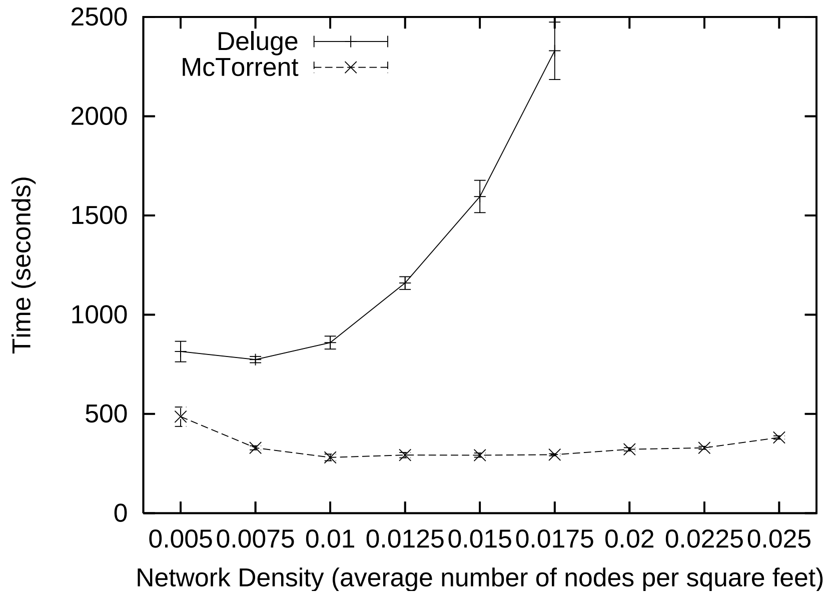


Figure 6.23: Object delivery latency from MICA simulations with different network densities in a 200ft x 200ft space for McTorrent and Deluge

remains almost constant thereafter, indicating the effective mitigation of collisions by using multiple channels.

We also show the average number of packet transmissions per node for the dissemination under different network densities in Figure 6.24. We see that in Deluge, nodes averagely transmit more packets when the density is either too low or too high because of fewer good connections between nodes at the low end and more intense collisions at the high end. The increase of the control packets caused by increasing collisions is especially drastic when the network density exceeds 0.015. On the other hand, in McTorrent, the average number of packet transmissions changes little when the network density increases from 0.01 to 0.025. As we expect, the results show that using multiple channels has more benefits in dense networks than in sparse networks.

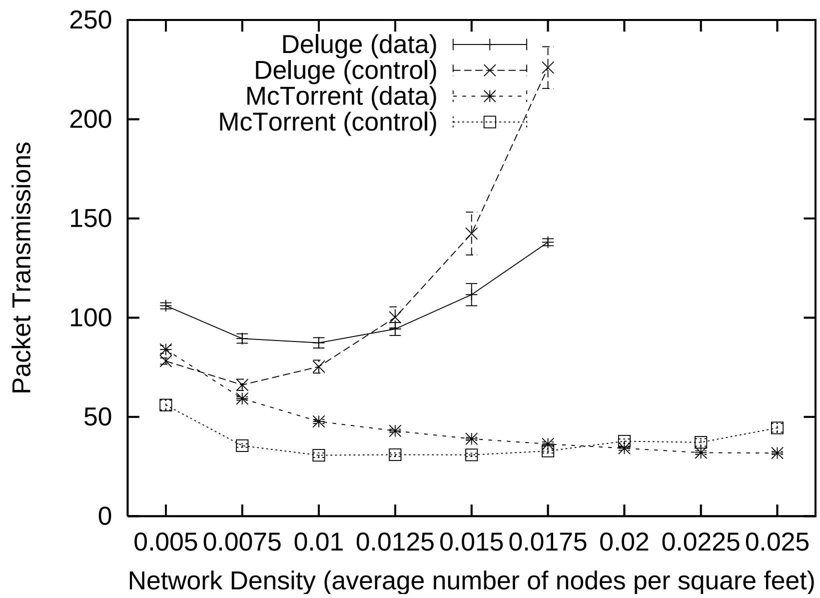


Figure 6.24: Average number of packet transmission per node from MICA simulations with different network densities in a 200ft x 200ft space for McTorrent and Deluge

Summary

Overall, our simulation results show that the use of multiple channels can lead to a significant reduction in the time taken to reliably propagate a large data object through a multihop network, and a reduction in packet transmissions. There are two reasons for this improvement in performance. First, the number of collisions is reduced. Second, the multichannel capability leads to more effective spatial multiplexing and pipelining as shown in Figures 3.1 and 3.2.

To illustrate the more effective pipelining enabled by McTorrent, we conducted a simulation in which we simulated McTorrent and Deluge for an 11 node network with a linear topology in which each node can only communicate with the nodes on its left and right. For this simulation, we ran TOSSIM to simulate communications at the packet level. We also set the page size to 128 packets in order to have longer transmission bursts.

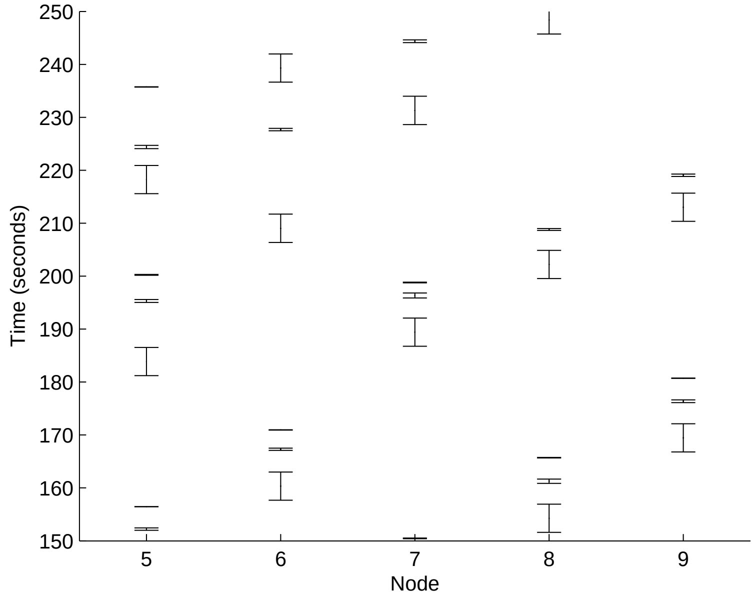


Figure 6.25: Pipelining in Deluge in a linear topology

We show the data transmission bursts in the simulation for nodes 5 – 9 as line segments in Figures 6.25 and 6.26, for Deluge and McTorrent respectively. The figures show that, unlike Deluge, nodes that are two hops apart can transmit simultaneously in McTorrent as long as they use different channels. For example, nodes 6 and 8 have a simultaneous transmission at time 200.

6.3.2 Performance and Energy Consumption Comparison of Protocols

In this section, we report the simulation results of McTorrent, CORD, McCORD, Deluge and MNP on MICA2 loss models.

Methodology

In this simulation study, we compared the performance and energy consumption of McTorrent, CORD, McCORD, Deluge and MNP using MICA2 loss models. To compare the energy

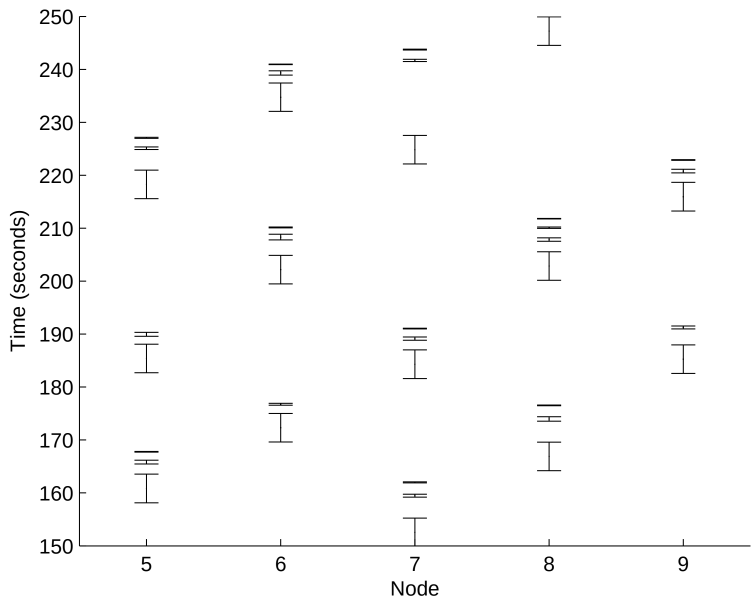


Figure 6.26: Pipelining in McTorrent in a linear topology

consumption of the protocols, we determined the average energy consumption per node during the time the data object is being disseminated. The energy consumption was collected with the PowerTOSSIM extension of TOSSIM and post processing scripts. Table 6.3 lists the MICA2 specification used to derive the energy consumption in the simulations.

The MICA2 loss models we used in the simulations were derived from the results of packet reception experiments using MICA2 motes we conducted on the second floor of ST2 Building at George Mason University. These models capture the radio irregularities and link asymmetry that have been reported by several researchers [54, 60, 61]. Figure 6.27 plots the mean packet reception rate as a function of the distance between the nodes and the transmission power level of the nodes, based on the results of the packet reception experiments. The figure shows that increasing the transmission power from -10dBm to 0dBm doubles the effective radio range. In our simulations, we used the 0dBm power level only.

Table 6.3: MICA2 current specification

CPU current in full operation	8mA
CPU current in idle state	4mA
CPU current in sleep state	8uA
External EEPROM current in write state	15mA
External EEPROM current in read state	4mA
Radio current in receive state	8mA
Radio current in transmit state (-10dBm)	10.1mA
Radio current in transmit state (0dBm)	16.8mA
Radio current in transmit state (5dBm)	25.4mA
Radio current in sleep state	2uA

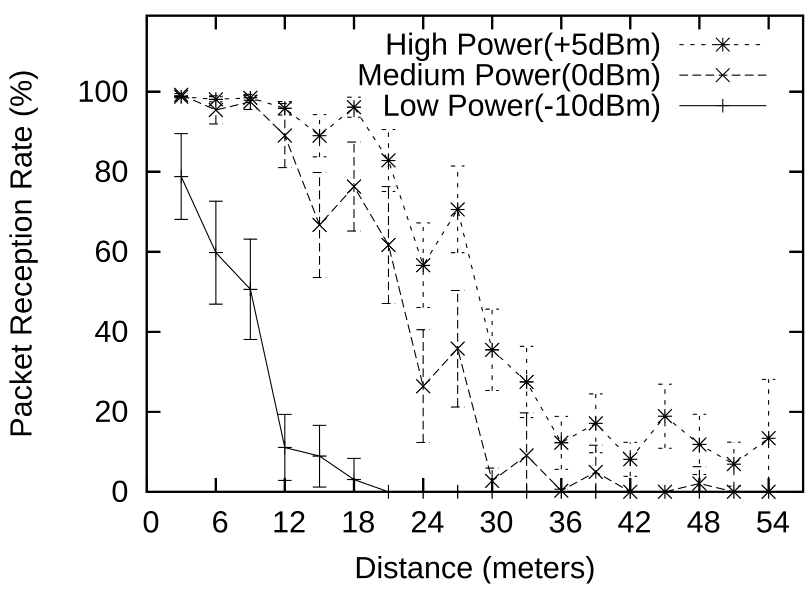


Figure 6.27: MICA2's mean packet reception rate as a function of distance (confidence intervals shown at 90% confidence level)

We considered both grid topologies and random topologies. In this section (6.3.2), a grid network is denoted by mxn-s, where m and n are the dimensions of the rectangular grid topology, and s is the distance between a node and its closest neighbor measured in meters. Starting from a default scenario with the parameters listed in Table 6.4, we varied the network size and object size one at a time, and evaluated the effect of these factors on the performance and energy consumption of the protocols.

Table 6.4: Default parameter settings for MICA2 simulations.

Network topology	5x20 Grid
Spacing	9 meters
Transmission power level	medium (0dBm)
Object size	10 pages
Page size (K)	128 packets
Packet payload size	23 bytes
Slot length (L)	6 seconds

In the simulations discussed in this section, McTorrent used 16 channels, while McCORD used 4 channels. Unless stated otherwise, all of the results reported in this section are mean values obtained from multiple simulation runs. All results have 90% confidence intervals with width less than 5% of the mean.

Effect of Network Size

Simulations of CORD and McCORD on 5xn-9 (n=10,20 and 30) MICA2 networks show the diameter of the network increases when n increases from 10 to 30. The diameter of the network is 6 hops for the 5x10-9 network and 17 hops for the 5x30-9 network. However, the percentage of core nodes in the network is relatively constant, at 34%-39%, reflecting a constant network density.

Figure 6.28 shows the latency of delivering a 10 page object for Deluge, MNP, McTorrent,

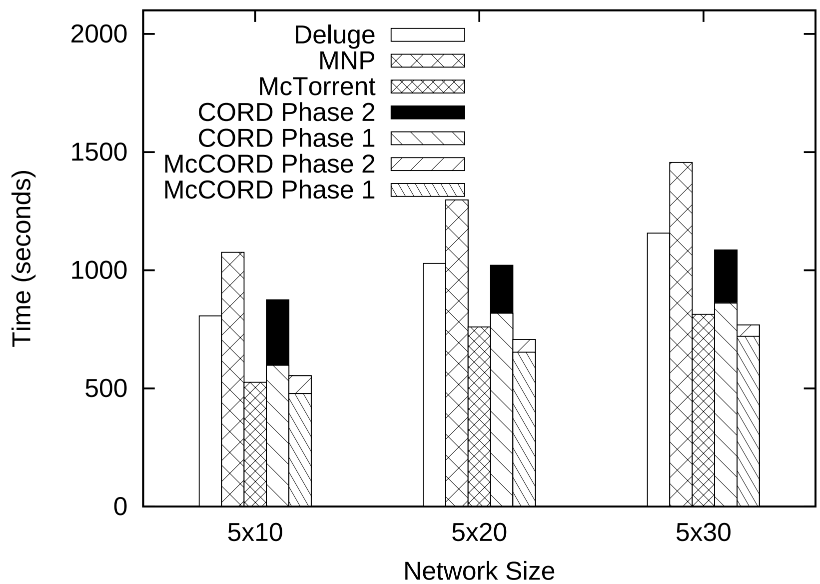


Figure 6.28: Object delivery latency from MICA2 simulations with various network sizes

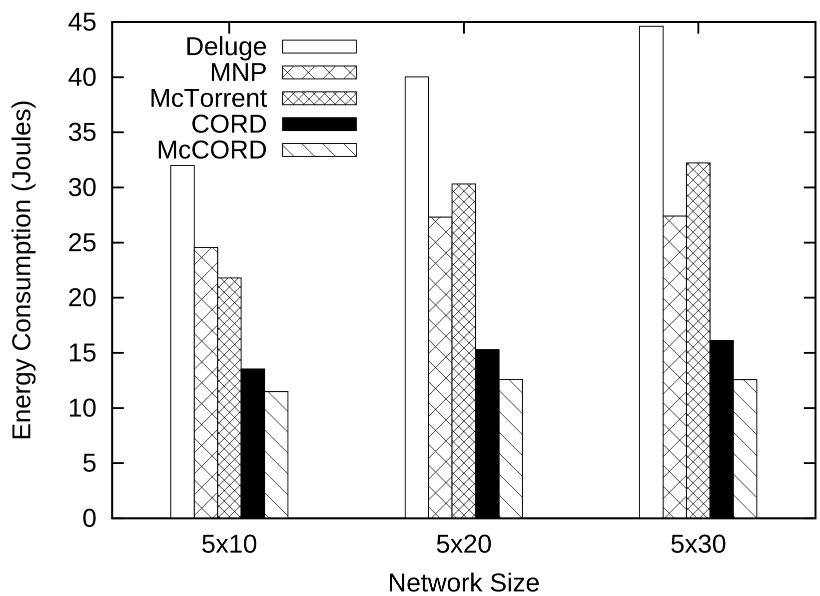


Figure 6.29: Energy consumption from MICA2 simulations with various network sizes

CORD and McCORD in networks with different sizes. From the figure, we can see that for all of the three networks, CORD has a similar latency as Deluge, and McCORD has a similar latency as McTorrent. Compared to the single channel counterparts (Deluge and CORD), the multichannel protocols (McTorrent and McCORD) reduce the latency by 25%-40%. We also see that while the object delivery latency increases with the network size, the duration of the second phase of CORD and McCORD is relatively constant, due to the approximately constant density of the given networks. CORD has a slightly lower latency than Deluge for the larger networks, while McCORD has a slightly lower latency than McTorrent for larger networks. MNP has the highest latency among the five protocols.

Figure 6.29 shows the average energy consumption at each node in the networks. For all three networks, McTorrent uses less energy than Deluge. The percentage of savings in energy consumption is similar to the percentage of reduction in latency, due to the fact that neither Deluge nor McTorrent put nodes to sleep during dissemination, and the majority of energy is consumed in radio listening. MNP consumes less energy than Deluge due to its energy saving optimizations such as turning the radio off when a node is neither receiving nor transmitting data. Compared to MNP, McTorrent uses slightly more energy in the larger two networks. Compared to Deluge, MNP and McTorrent, CORD significantly reduces energy consumption mainly due to the aggressive sleep scheduling at each node. McCORD uses slightly less energy than CORD. Moreover, when the network size increases, the energy consumption of the core-based approaches (CORD and McCORD) increases slower than the other approaches. This is because in the core-based approaches, the actual time used to forward the object by a node (time on receiving the pages from the parent and sending the pages to downstream nodes) is theoretically constant regardless of network size, and theoretically nodes are awake only when receiving and sending the pages (in reality, the energy consumption of the core-based approaches slightly increases when the network size increases due to the overhead such as periodically waking up at the beginning of the

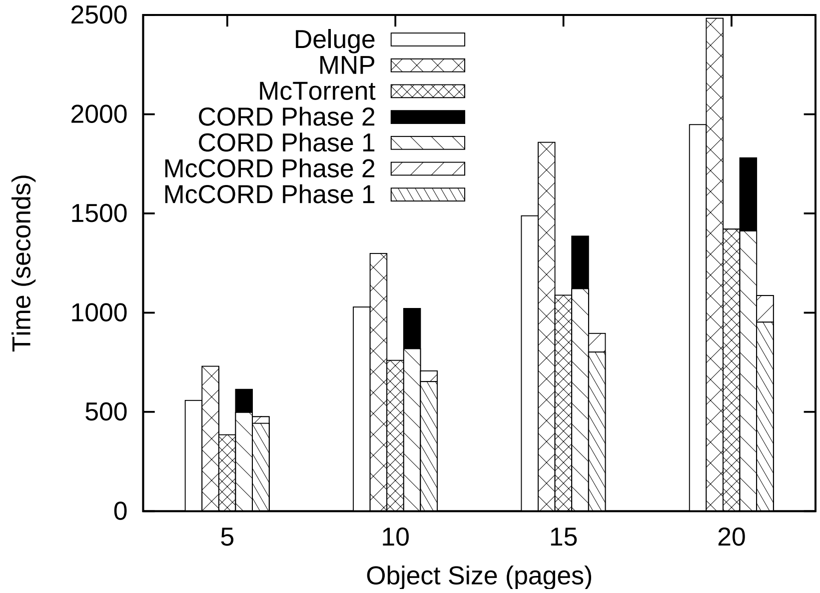


Figure 6.30: Object delivery latency from MICA2 simulations with various object sizes

dissemination for slot coordination with the node's parent).

Effect of Data Object Size

Figures 6.30 and 6.31 show the results for simulations in which 5, 10, 15 and 20 pages were disseminated over the default 5x20-9 MICA2 network. When the object size increases, the second phase of CORD and McCORD becomes longer because there is a corresponding increase in the number of packets that a non-core node needs to obtain from its parent core node. The latency of CORD is lower than that of Deluge when the object is larger than 10 pages, and so is McCORD compared to McTorrent. This is because CORD and McCORD use pre-selected senders and thus avoids overhead of sender selection for each page during data dissemination, which is more beneficial when disseminating larger data objects. MNP has the highest latency among the five protocols, however, it consumes less energy than Deluge and McTorrent due to its energy saving optimizations. The energy consumption of

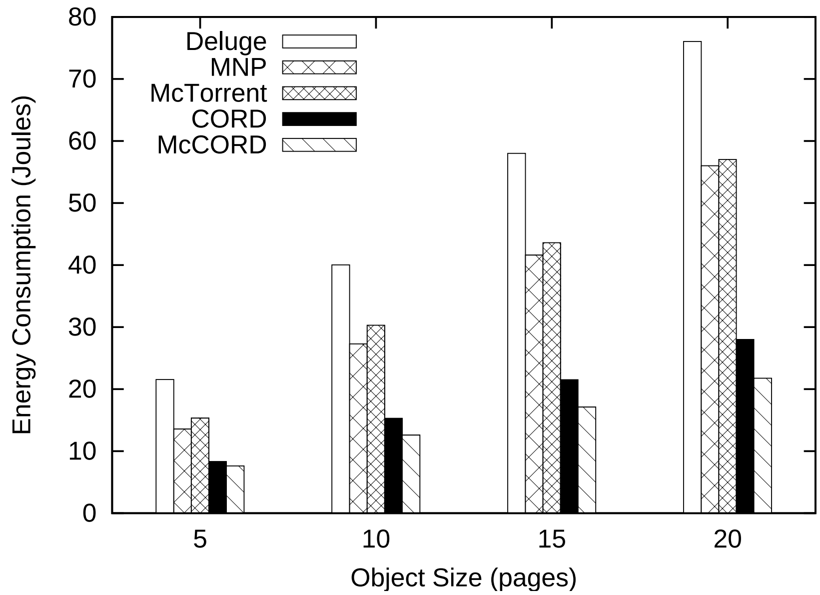


Figure 6.31: Energy consumption from MICA2 simulations with various object sizes

CORD and McCORD is 30%-40% of that of Deluge for all of the four object sizes, and the savings become larger when the object size increases.

Effect of Network Topology and Network Density

We also examined the relative performance and energy consumption of the protocols for networks in which nodes were randomly deployed over a given area based on a uniform distribution (the base station was always fixed in one corner of the area). We varied the number of nodes in a 50mx200m area in the simulations. We ran simulations for a network of 100, 150 and 200 nodes, respectively (accordingly, the network density is 0.01, 0.015 and 0.02, respectively), disseminating a 10 page object to the network using Deluge, McTorrent, CORD and McCORD. The results are shown in Figures 6.32 and 6.33. We found that the relative performance and energy consumption of the protocols for each random topology

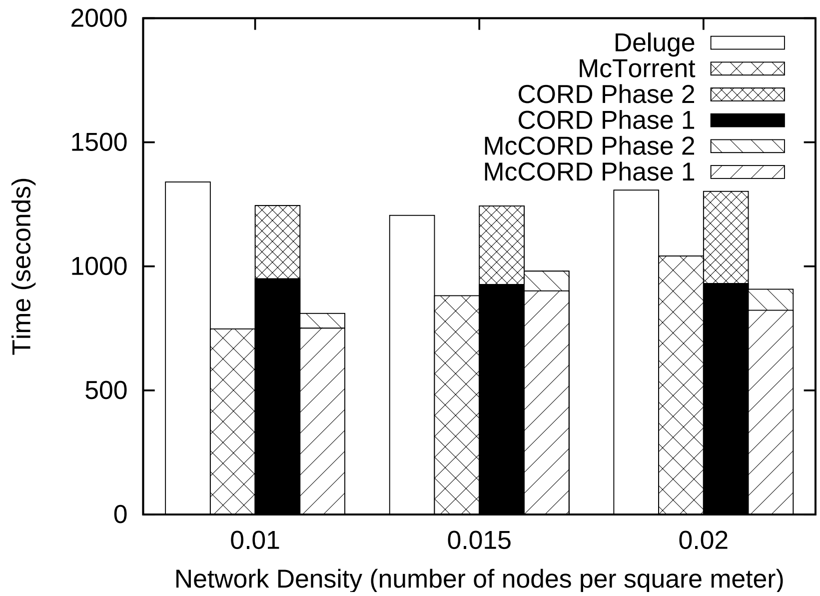


Figure 6.32: Object delivery latency from MICA2 simulations with various network densities in a random topology

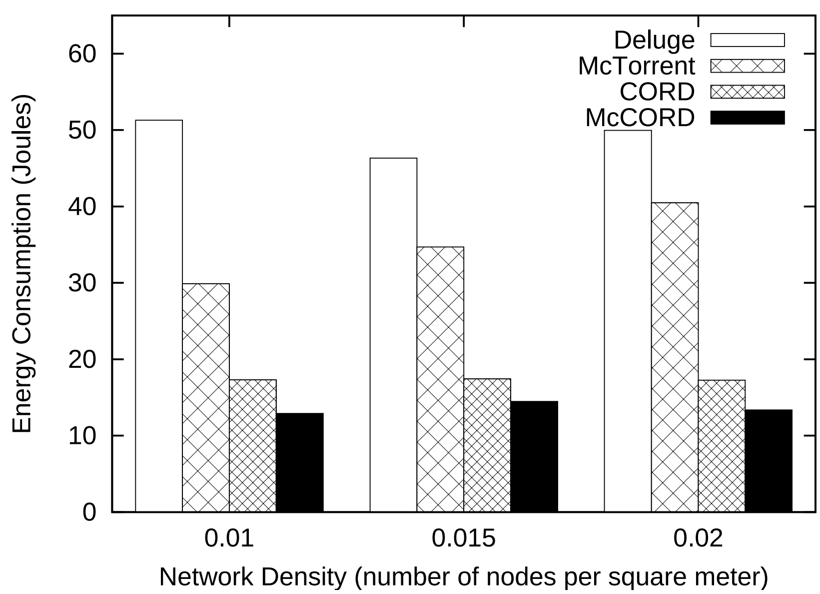


Figure 6.33: Energy consumption from MICA2 simulations with various network densities in a random topology

are similar to those for the grid topologies. On the other hand, unlike the results in Section 6.3.1, Deluge with the MICA2 radio model is less sensitive to network density change given the fixed deployment area, thus making McTorrent less beneficial than in simulations on MICA loss models where the improvement of latency using multiple channels becomes more significant when network density increases. For CORD and McCORD, the size of the core structure remains almost constant given the same area regardless of the network density. The object delivery latency and average energy consumption per node are thus almost constant when the network density increases from 0.01 to 0.02.

6.4 Summary

In this chapter, we evaluated the performance of McTorrent, CORD and McCORD with previous approaches, Deluge and MNP, through experiments on indoor and outdoor testbeds and extensive simulations.

The empirical results from the testbeds show that McTorrent reduces the object delivery latency and energy consumption by about 40%, compared to Deluge. CORD and McCORD reduces the energy consumption by 60%-70%, compared to Deluge, while CORD maintains a similar object delivery latency as Deluge and McCORD maintains a similar object delivery latency as McTorrent.

The results from simulations on MICA2 loss models for various network sizes and object sizes show that McTorrent reduces the object delivery latency and energy consumption by 25%-40% in most scenarios, compared to Deluge. CORD consistently reduces the energy consumption by 60%-70%, compared to Deluge, while maintaining a comparable object delivery latency. McCORD consumes a similar amount of energy as CORD while maintaining a similar object delivery latency as McTorrent in most scenarios. Moreover, CORD and McCORD are more beneficial for larger networks and larger objects.

The results from simulations on MICA loss models show that McTorrent has more benefits over Deluge than in the simulations on MICA2 loss models, especially when network density increases. Due to the improvement of the MICA2 radio model over the MICA radio model, unlike in the simulations on MICA loss models, the increase of network density does not cause significantly more collisions in Deluge in the simulations on MICA2 loss models. This makes McTorrent less beneficial in reducing packet collisions by using multiple channels.

Overall, our results show that:

- By using multiple channels, a data dissemination protocol can adopt more aggressive pipelining as well as reducing packet collisions. The object delivery latency is thus reduced. For protocols that require nodes to be awake all the time during data dissemination, since the energy consumption incurred during data dissemination is proportional to the time when the nodes are awake, the energy consumption is also reduced due to the reduced object delivery latency.

- During data dissemination, a major portion of energy is consumed by the radio on idle listening. Thus a protocol such as CORD achieves significant energy savings due to its aggressive sleep scheduling.

Chapter 7: Conclusions

7.1 Summary

In this dissertation, we presented three data dissemination protocols, named McTorrent, CORD and McCORD, that focus on reducing the object delivery latency and energy consumption incurred during data dissemination.

McTorrent is similar to the previous approach Deluge, in that they both adopt pipelining and data are propagated in a neighborhood by neighborhood fashion. The main difference of McTorrent from Deluge is the use of multiple channels, which are available on most of current generation sensor nodes.

CORD uses a different core-based two-phase approach in conjunction with coordinated sleep scheduling. In CORD, data are propagated to a set of pre-selected core nodes in the first phase, and the non-core nodes request for the missing portion of the data from their neighboring core nodes in the second phase. A main feature of CORD is the aggressive use of sleep scheduling at each node, which helps reduce the energy consumption.

McCORD adopts the same core-based two-phase approach as CORD while using multiple channels. McCORD uses a simple yet effective channel assignment method during core construction. The assigned channels are used during data dissemination, which facilitates more aggressive pipelining than CORD.

We conducted various indoor and outdoor experiments and extensive simulations to evaluate the presented protocols with previous approaches, and reported the results in the dissertation. On the experimental testbeds, McTorrent reduces the object delivery latency by about 40%, compared to the de-facto protocol Deluge. In most of the simulation scenarios, McTorrent reduces the object delivery latency by 25%-40%, compared to Deluge. The energy savings in data dissemination are in the same range, respectively.

The empirical and simulation results show that in comparison to Deluge, CORD reduces the average energy consumption by 60%-70% while maintaining a comparable object delivery latency. The majority of energy savings comes from the coordinated sleep scheduling that CORD adopts.

In comparison to CORD, McCORD reduces the object delivery latency by 25%-40% in most scenarios by using multiple channels, while consuming a comparable amount of energy.

Overall, by using multiple channels, a data dissemination protocol can significantly reduce object delivery latency, compared to its single channel counterpart. By using a core-based two-phase approach in conjunction with coordinated sleep scheduling, a protocol is more energy efficient in disseminating a large data object to a large sensor network. A hybrid protocol that takes advantage of both multiple channels and the core-based two-phase approach inherits the merits of low latency and energy efficiency in disseminating large data objects to large sensor networks.

In our experiments, we also found that while the stationary TelosB sensors had relatively stable link quality on one channel, the link quality could differ drastically on different channels. This occurred more often when the sensors were not close enough. This observation has not been reported in previous research. The variation of link quality on different channels makes channel assignment in multichannel protocols such as McCORD more challenging. In the design of McCORD, we took this issue into consideration by using a simple heuristic in assigning channels to core nodes during core setup.

While in our experiments and simulation scenarios, McCORD performs best among all protocols under evaluation in the sense of both energy efficiency and low latency, it may not work well in some special cases, such as in a hybrid network. With the increase of sensor network deployment, there may be cases where legacy sensors with only one channel are deployed with modern sensors equipped with multiple channels in the same network. In such a hybrid network, it may be infeasible for McCORD, at network wide, to set up a core structure while using multiple channels. On the other hand, McTorrent will work without any modification in such cases. In McTorrent, nodes select channels dynamically using a contention-based scheme. At the time of advertisement, a legacy sensor will always choose the only channel if the channel is not being used in the neighborhood, otherwise it will postpone its advertisement. If all sensors in the network are legacy sensors, McTorrent will perform same as Deluge. CORD will also work in such hybrid networks since CORD uses one channel only.

7.2 Future Work

There are some open issues related to this dissertation that may be addressed in future research.

Quality-aware channel selection and assignment: The channel selection scheme in McTorrent does not consider link quality variation on different channels. While the link quality variation does not cause McTorrent to fail, using a channel with bad quality for data transfer results in energy waste at the sender and delay of the delivery, since the receivers would have a high loss rate in receiving the data packets on the channel. Therefore, it would be desirable for a sender to select a channel for the next data transfer that not only avoids conflicts with other senders but also brings high packet reception rate at most of the potential receivers.

In McCORD, while the simple channel assignment scheme we adopted guarantees that nodes assigned to the same channel are at least three hops apart, the collisions among senders at the same hop can still occur due to the irregularity of radio ranges of the sensor nodes. Better channel assignment heuristics may be able to reduce this kind of collisions while still taking into consideration the link quality variation on multiple channels.

Area-oriented data dissemination: While this dissertation focuses on delivering large data objects to *all the nodes in the network*, there are also applications where only nodes in a certain target area need to be updated with the new data objects. Using the protocols addressed in the dissertation, including McTorrent, CORD, McCORD and previous approaches, would cause unnecessary energy consumption and object delivery delay, since nodes not in the target area would also receive the data objects. Further research is necessary to design protocols to reliably disseminate a large data object to the nodes in a target area, while involving as few nodes out of the target area as possible to avoid unnecessary energy consumption and object delivery delay.

Bibliography

[1] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[2] F. Stann and J. Heidemann, "RMST: Reliable data transport in sensor networks," in *Proceedings of the 1st International Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003.

[3] C. Wan, A. Campbell, and L. Krishnamurthy, "PSFQ: A reliable transport protocol for wireless sensor networks," in *Proceedings of the 1st ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[4] S. Park, R. Vedantham, R. Sivakumar, and I. Akyildiz, "A scalable approach for reliable downstream data delivery in wireless sensor networks," in *Proceedings of the 5th ACM International Symposium on Mobile Ad hoc Networking and Computing (MobiHoc)*, 2004.

[5] Crossbow, "Telosb datasheet." [Online]. Available: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf

[6] T. Stathopoulos, J. Heidemann, and D. Estrin, "A remote code update mechanism for wireless sensor networks," CENS, Tech. Rep., 2003.

[7] S. Kulkarni and L. Wang, "MNP: Multihop network reprogramming service for sensor networks," in *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS)*, 2005.

[8] V. Naik, A. Arora, P. Sinha, and H. Zhang, "Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices," in *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[9] J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Trans. Comput. Syst.*, vol. 2, no. 3, pp. 251–273, 1984.

[10] S. Armstrong, A. Freier, and K. Marzullo, "Multicast transport protocol (internet rfc 1301)," 1992.

[11] B. Whetten, T. Montgomery, and S. M. Kaplan, "A high performance totally ordered multicast protocol," in *Proceedings of Dagstuhl Seminar on Distributed Systems*, 1994, pp. 33–57. [Online]. Available: citeseer.ist.psu.edu/whetten94high.html

[12] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 784–803, 1997.

[13] R. Morris, "Bulk multicast transport protocol," in *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1997.

[14] R. Yavatkar, J. Griffoen, and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," in *Proceedings of ACM Multimedia*, 1995, pp. 333–344.

[15] J. C. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," in *Proceedings of the 15th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1996.

[16] J. Byers, M. Luby, and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast," 2002. [Online]. Available: citeseer.ist.psu.edu/byers02digital.html

[17] B. Ouyang and X. Hong, "A comparison of reliable multicast protocols for mobile ad hoc networks," in *Proceedings of IEEE SoutheastCon*, 2005.

[18] T. Gopalsamy, M. Singhal, D. Panda, and P. Sadayappan, "A reliable multicast algorithm for mobile ad hoc networks," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[19] K. Tang, K. Obraczka, S. Lee, and M. Gerla, "A reliable, congestion-controlled multicast transport protocol in multimedia multi-hop networks," in *Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2002.

[20] R. Chandra, V. Ramasubramanian, and K. Birman, "Anonymous gossip: improving multicast reliability in mobile ad hoc networks," in *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2001.

[21] J. Luo, P. Eugster, and J. Hubaux, "Route driven gossip: probabilistic reliable multicast in ad hoc networks," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[22] L. Rizzo and L. Vicisano, "RMDP: an FEC-based reliable multicast protocol for wireless environments," *ACM Mobile Computing and Communications Review*, vol. 2, no. 2, pp. 23–31, 1998.

[23] J. Kulik, W. Heinzelman, and H. Balakrishnan, "Negotiation-based protocols for disseminating information in wireless sensor networks," *Wireless Networks*, vol. 8, no. 2/3, pp. 169–185, 2002.

[24] P. Levis and D. Culler, "The firecracker protocol," in *Proceedings of the 11th ACM SIGOPS European Workshop*, 2004.

[25] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[26] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, 2003.

[27] L. Wang and S. Kulkarni, "Gappa: Gossip based multi-channel reprogramming for sensor networks," in *Proceedings of the 2nd International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2006.

[28] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002. [Online]. Available: citeseer.ist.psu.edu/levis02mate.html

[29] Crossbow, "Mote in network programming user reference." [Online]. Available: http://www.tinyos.net/tinyos-1.x/doc/Xnp.pdf

[30] P. Marrón, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Rothermel, "Flexcup: A flexible and efficient code update mechanism for sensor networks," in *Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN)*, 2006. [Online]. Available: citeseer.ist.psu.edu/745172.html

[31] J. Jong and D. Culler, "Incremental network programming for wireless sensors," in *Proceedings of the International Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[32] J. Deng, R. Han, and S. Mishra, "Secure code distribution in dynamically programmable wireless sensor networks," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, 2006.

[33] P. Dutta, J. Hui, D. Chu, and D. Culler, "Securing the deluge network programming system," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, 2006.

[34] P. Lanigan, R. Gandhi, and P. Narasimhan, "Sluice: Secure dissemination of code updates in sensor networks," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[35] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *Journal of Computer and System Sciences*, 1974.

[36] B. Das and V. Bharghavan, "Routing in ad hoc networks using a spine," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 1997.

[37] J. Wu and H. Li, "On calculating connected dominating sets for efficient routing in ad hoc wireless networks," *Telecommunication Systems, Special issue on Moible Computing and Wireless Networks*, vol. 18, no. 1/3, pp. 13–36, 2001.

[38] F. Dai and J. Wu, "Distributed dominant pruning in ad hoc networks," in *Proceedings of IEEE International Conference on Communications (ICC)*, 2003.

[39] A. Nasipuri and S. Das, "A multichannel CSMA MAC protocol for multihop wireless networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 1999.

[40] ——, "Multichannel CSMA with signal power-based channel selection for multihop wireless networks," in *Proceedings of the IEEE Fall Vehicular Technology Conference (VTC)*, 2000.

[41] S. Wu *et al.*, "A new multi-channel MAC protocol with on-demand channel assignment for multi-hop mobile ad hoc networks," in *Proceedings of the 2000 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, 2000.

[42] N. Jain *et al.*, "Multichannel CSMA MAC protocol with receiver-based channel selection for multihop wireless networks," in *Proceedings of the 9th International Conference on Computer Communications and Networks (IC3N)*, 2001.

[43] Z. Tang and J. Garcia-Luna-Aceves, "Hop reservation multiple access (HRMA) for ad-hoc networks," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1999.

[44] J. Garcia-Luna-Aceves and A. Tzamaloukas, "Receiver-initiated collision avoidance in wireless networks," *Wireless Networks*, vol. 8, no. 2/3, pp. 249–263, 2002.

[45] J. So and N. Vaidya, "Multi-channel mac for ad hoc networks: Handling multi-channel hidden terminals using a single transceiver," in *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2004.

[46] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, "A multi-radio unification protocol for IEEE 802.11 wireless networks," in *Proceedings of the International Conference on Broadband Networks (Broadnets)*, 2004.

[47] "Tinyos community forum." [Online]. Available: http://www.tinyos.net

[48] R. Simon, L. Huang, E. Farrugia, and S. Setia, "Using multiple communication channels for efficient data dissemination in wireless sensor networks," in *Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2005.

[49] Crossbow, "MPR MIB user's manual." [Online]. Available: http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_User_Manual_7430-0021-05_A.pdf

[50] D. Gay, P. Levis, and D. Culler, "Software design patterns for tinyos," *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 4, 2007.

[51] L. Huang and S. Setia, "CORD: Energy-efficient reliable bulk data dissemination in sensor networks," in *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM)*, 2008, to appear.

[52] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," in *European Symposium on Algorithms*, 1996, pp. 179–193.

[53] X. Cheng, M. Ding, D. Du, and X. Jia, "Virtual backbone construction in multihop ad hoc wireless networks," *Wireless Communications and Mobile Computing*, vol. 6, pp. 183–190, 2006.

[54] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[55] B. Chen, K. Reddy, and M. Welsh, "Ad-hoc multicast routing on resource-limited sensor nodes," in *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, 2006.

[56] D. Towsley, J. Kurose, and S. Pingali, "A comparison of sender-initiated and receiver-initiated reliable multicast protocols," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, 1997.

[57] "CC2420." [Online]. Available: http://focus.ti.com/docs/prod/folders/print/cc2420.html

[58] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-low power data storage for sensor networks," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, 2006.

[59] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[60] G. Zhou, T. He, S. Krishnamurthy, and J. Stankovic, "Impact of radio irregularity on wireless sensor networks," in *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2004.

[61] A. Cerpa, J. Wong, L. Kuang, M. Potkonjak, and D. Estrin, "Statistical model of lossy links in wireless sensor networks," in *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.

Curriculum Vitae

Leijun Huang received his Bachelor of Science from Xi'an Jiaotong University, China in 1996. He received his Master of Science from Beijing University of Posts and Telecommunications, China in 1999.