Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Newlyn S. Erratt

Entitled
A Compressed Data Collection System for use in Wireless Sensor Networks

For the degree of Master of Science

Is approved by the final examining committee:

Yao Liang
Chair

Rajeev Raje

Mihran Tuceryan

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Yao Liang

Approved by: Shiaofen Fang 11/1/2012
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

A Compressed Data Collection System for use in Wireless Sensor Networks

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Newlyn S. Erratt

Printed Name and Signature of Candidate

11/1/2012

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

A COMPRESSED DATA COLLECTION SYSTEM FOR USE IN WIRELESS SENSOR NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Newlyn S. Erratt

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2012

Purdue University

Indianapolis, Indiana

For my wife, my parents, and my brother.

ACKNOWLEDGMENTS

I would like to begin by thanking my advisor Dr. Yao Liang for his encouragement, passion, and guidance in undertaking this degree. Additionally, I would like to thank Dr. Rajeev Raje and Dr. Mihran Tuceryan for their help in completing this thesis. In addition to my professors, I must say thanks to the members of my research group for their support, encouragement and advice during the duration of my work.

I would like to thank Senior Lecturer Andy Harris for his constant encouragement in my position as Teacher's Assistant for his classes. His encouragement and unique lecturing style have vastly improved my ability to convey new concepts in a way that students understand and enjoy. Additionally, I thank all of my fellow Teacher's Assistants and all of our students for ensuring that my life is full of excitement and new ideas on a daily basis.

I would also like to thank my family for their encouragement throughout my entire academic career so far. My parents and brother, specifically, for always being supportive. Without their encouragement I never would have been able to achieve this. My wife, Carrie, deserves recognition for always being supportive, understanding, and loving especially when my work cut into our personal time. She has never wavered in her support of this endeavor. Without all of you, I would not be who I am today.

Finally, I would like to thank the entire Department of Computer and Information Sciences. The students, staff, and faculty have all helped provide a wonderful learning experience.

This work is supported in part by the National Science Foundation under grant $CNS - 0758372$. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author*s* and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

ABSTRACT viii

1 INTRODUCTION 1
1.1 Use Cases 1
1.1.1 WSN Deployment 2
1.1.2 WSN Research 2
1.2 A Compressed Data Collection System 3
1.2.1 Challenges 3
1.2.2 System Organization 4
1.2.2.1 CDP 5
1.2.2.2 Gateway 5

2 BACKGROUND 6
2.1 Wireless Sensor Network Background 6
2.1.1 WSN Transport Layer Background 7
2.2 WSN Gateway Background 7
2.3 Data Collection System Background 8

3 COMPRESSED DATA-STREAM PROTOCOL (CDP) 9
3.1 Introduction 9
3.2 System Model and GPC 11
3.2.1 System Model 11
3.2.2 GPC Framework 12
3.2.3 GPC Realisation 14
3.3 CDP Protocol Design 15
3.3.1 Data Streams 17
3.3.1.1 Overhead Reduction 18
3.3.1.2 Stream Setup and Control Packet 20
3.3.1.3 Data Packets 20
3.3.2 Modular Design 21
3.3.2.1 Network Access 22
3.3.2.2 Utility Modules 23
3.3.2.3 Compression Modules 23
3.4 Protocol Implementation 24

Page
3.4.1 Collection Tree Protocol . . . 25
3.4.2 Packet Buffers . . . 26
3.4.3 Decoding Implementation . . . 27
3.5 Simulations and Analyses . . . 28
3.5.1 TOSSIM and PowerTOSSIM-z . . . 28
3.5.2 Simulation Setup . . . 29
3.5.3 Performance Evaluation . . . 31
3.5.4 Energy Evaluation . . . 34
3.6 Conclusions . . . 36

4 GENERAL GATEWAY . . . 38
4.1 Introduction . . . 38
4.2 Criteria of the Gateway . . . 40
4.3 Gateway Design . . . 41
4.3.1 Gateway Design Goals . . . 41
4.3.2 Top-down Design of Gateway . . . 42
4.3.2.1 Gateway Architecture . . . 42
4.3.2.2 User and Core Gateway Spaces . . . 43
4.3.2.3 User and Core Modules . . . 45
4.4 Implementation . . . 47
4.5 Testing . . . 49
4.5.1 Load tests . . . 49
4.5.2 Real-World Test-Bed Tests . . . 49
4.6 Conclusion and Future Work . . . 50

5 SYSTEM . . . 52
5.1 System Design . . . 52
5.1.1 Motes . . . 53
5.1.2 Gateway . . . 54
5.2 Challenges . . . 54
5.3 Testing . . . 54

6 FUTURE WORK . . . 56
6.1 CDP Future Work . . . 56
6.2 Gateway Future Work . . . 57
6.3 System Future Work . . . 58

7 CONCLUDING REMARKS . . . 60

LIST OF REFERENCES . . . 62

LIST OF TABLES

Table Page

3.1 Coding table ($K = 14$) in GPC of CDP 16

3.2 State table for the codes 00 and 01 28

3.3 Node assignment to sensorscope data and CDP compression ratios . . . 32

LIST OF FIGURES

Figure		Page
3.1	GPC framework	13
3.2	Example of a node with two streams	17
3.3	Buffers required for the example node	19
3.4	Illustration of control packet	21
3.5	Data packet format	21
3.6	Overall modular design of CDP	22
3.7	CDP network stack	25
3.8	Decoding procedure	27
3.9	Mote location map of the WSN in our simulation	30
3.10	Illustration of retransmissions in the sensornet	33
3.11	Comparisions of total data transmitted in the sensornet (in bytes)	34
3.12	Energy consumption in the sensornet	35
4.1	A network view of the gateway	39
4.2	Core view of the gateway	44
4.3	Flowchart of the main control thread of the gateway	48
4.4	An illustration of the test-bed used in gateway system testing	51
5.1	A network view of the overall system implementation	53

ABSTRACT

Erratt, Newlyn S. M.S., Purdue University, December 2012. A Compressed Data Collection System for use in Wireless Sensor Networks. Major Professor: Yao Liang.

One of the most common goals of a wireless sensor network is to collect sensor data. The goal of this thesis is to provide an easy to use and energy-efficient system for deploying data collection sensor networks. There are numerous challenges associated with deploying a wireless sensor network for collection of sensor data; among these challenges are reducing energy consumption and the fact that users interested in collecting data may not be familiar with software design. This thesis presents a complete system, comprised of the Compression Data-stream Protocol and a general gateway for data collection in wireless sensor networks, which attempts to provide an easy to use, energy efficient and complete system for data collection in sensor networks. The Compressed Data-stream Protocol is a transport layer compression protocol with a primary goal, in this work, to reduce energy consumption. Energy consumption of the radio in wireless sensor network nodes is expensive and the Compressed Data-stream Protocol has been shown in simulations to reduce energy used on transmission and reception by around 26%. The general gateway has been designed in such a way as to make customization simple without requiring vast knowledge of sensor networks and software development. This, along with the modular nature of the Compressed Data-stream Protocol, enables the creation of an easy to deploy and easy to configure sensor network for data collection. Findings show that individual components work well and that the system as a whole performs without errors. This system, the components of which will eventually be released as open source, provides a platform for researchers purely interested in the data gathered to deploy a sensor network without being restricted to specific vendors of hardware.

1 INTRODUCTION

This thesis details the design and implementation of a Compressed Data collection system for use in wireless sensor networks (WSNs). WSNs are distributed networks of embedded systems with attached sensors. The driving goals of WSN hardware are, typically, size, energy efficiency and cost. This goal means that nodes are limited in memory and processing power. Additionally, because WSNs are often deployed in extreme environments, they may be powered by batteries. These limitations lead to the requirement that code on motes must be small in size, relatively simple, and energy-efficient. One of the primary goal of many WSN deployments is to collect sensor data for analysis. Despite the fact that this type of deployment is common, there is not a widely-available non-commercial system for data gathering that is easily adaptable. This means that the deployment of a data gathering WSN is often involved and complex due to the varied requirements of a specific deployment. This problem is compounded by the fact that many researchers who are interested in deploying a WSN to collect specific data have little to no experience in programming. The primary goal of this work is to determine if a data gathering wireless sensor network can feasibly be deployed with support for compression at the transport layer in the network stack. Since this concerns itself with both technical feasibility as well as practical feasibility, the use cases for the system as well as some specific challenges arising from those use cases must be addressed.

1.1 Use Cases

My work is useful for both data data collection WSN deployments as well as in deployments where the primary goal is to test new WSN research. In both cases, the users will not have to be familiar with all of the details of the underlying WSN

platform. They can easily deploy their work while only gaining knowledge of the specific areas that need to be customized. This should reduce the amount of time that researchers will be required to spend on work that is not beneficial to their actual research.

1.1.1 WSN Deployment

In this case, the user is a researcher who depends on sensor deployments to gather data that will be used in their research. Additionally, this user may not have very much familiarity with software development. This user is really only interested in reliable collection of their data. Their deployment will typically follow the following pattern:

1. Sample sensors at the nodes.
2. Gather sensor data at the sink.
3. Store and forward data at the gateway.

They will then download their data from the gateway computer and do some complex analysis of the readings. Currently, they usually depend on a commercial system such as those described in Sec. 2.3. The customization of these complex commercial applications can be both time consuming as well as complicated (in the case where they don't have much knowledge of software development). If the user has a more complicated application it may not even be possible with these commercial systems.

1.1.2 WSN Research

In this case, the user is a researcher who wants to test new WSN research in the real world. The scope of possible research done in this case is challenging to define but could include things such as monitoring network performance, testing compression algorithms and testing new per-mote processing algorithms. This user's deployments

will be more varied than the standard data gathering deployments but they will only need to learn how to customize the specific area of the application the are interested in. Their current work-flow generally follows one or both of the following two patterns. Firstly, they may only do their research at the simulation level. A large amount of current WSN research is tested via simulations such as MATLAB or a WSN simulation platform. Secondly, they may deploy their work in one of the many publicly available test-beds. While this work is valuable, simulations may not properly simulate the realistic and unpredictable real-world conditions. Additionally, the publicly available test-beds may not provide the information the user is interested in; this is especially the case if they are interested in gathering network performance measurements for the development or tuning of new research. In this case, as in the other case, I want this work to facilitate the advancement of research that is useful in the real world while reducing the effort required to properly test said research.

1.2 A Compressed Data Collection System

This section briefly describes the goals (Sec. 1.2.1) and organization (Sec. 1.2.2) of my complete data collection system. The system is primarily built upon the compressed data-stream protocol (CDP) and the general WSN gateway for data collection; both of which were developed by Dr. Yao Liang and myself. The overall goal of this system is to attempt to develop a data gathering wireless sensor network system with compression in the transport layer. More information about CDP and the gateway may be found in Ch. 3 and Ch. 4 respectively. Some necessary background information may be found in Ch. 2. More detail about the system and testing of the system may be found in Ch. 5. The direction of future work may be found in Ch. 6.

1.2.1 Challenges

There are three underlying challenges that must be taken into account while exploring the real world feasibility of my system. For more information about the spe-

cific goals and challenges of CDP and the gateway please see their respective chapters. The goals are as follows:

1. Ease of configurability: It should be simple to configure this system for the specific sensors used in a deployment as well as the rate at which those sensors are sampled.

2. Energy efficiency: The system should emphasize energy efficiency. Since motes are often deployed in extreme environments it is vital that this system have good battery life.

3. Ease of use: The system should be easy to use. That is, it should be both easy to deploy as well as easy to gather the logged data.

1.2.2 System Organization

The overall organization of this data collection system is as follows:

1. Motes: The motes consist of an application developed on top of the CDP network stack. The application layer is defined by a sensor module, that will be customized or rewritten by the user, the main timer, the period of which determines sampling rate and may be easily modified, and the stream setup code, that defines what readings are associated with each stream.

2. Gateway: The gateway consists of the general gateway configured for use with the CDP. It should not require any modifications for the general data gathering case.

The primary two new works consist of the CDP and gateway components. These components are described in Sec. 1.2.2.1 and Sec. 1.2.2.2 respectively.

1.2.2.1 CDP

The compressed data-stream protocol (CDP) is a transport layer protocol developed and implemented by myself and Dr. Yao Liang for use in wireless sensor networks. Its primary goal is to attempt to reduce energy consumption by reducing the total data transmitted over the network through compression at the transport layer. As explained in Ch. 3 radio activity is by far more expensive than computation in WSNs. Any reduction in radio activity may significantly improve battery life. It is shown that this goal is, indeed, achievable without header overhead negating compression benefits. Since the overall goal of this project is to improve energy efficiency through the use of the ideas in CDP it is the necessary choice for my system.

1.2.2.2 Gateway

The general WSN gateway for data collection discussed in Ch. 4, developed by myself and Dr. Yao Liang, is a gateway system developed with the primary goals of being easily customizable for data collection WSNs based on nearly any protocol stack as well as being a non-commercial alternative to what is currently available. While there has been a lot of research into WSN gateways as detailed in Sec. 2.2 most of these to not have publicly available implementations. Using this gateway will allow my system to maintain high configurability while also avoiding reliance on a commercial gateway system.

2 BACKGROUND

This chapter provides some background information about several of the topics that are important in understanding my work. Additionally, there is a some background work specific to the compressed data-stream protocol(CDP) in Ch. 3 regarding compression in wireless sensor networks (WSNs) that is not important to the whole of this work but is vital in understanding the CDP. In Sec. 2.1 I discuss WSNs and some of the associated challenges. In Sec. 2.1.1 I detail a few of the available transport layer protocols that provide different functionality than CDP. In Sec. 2.2 I list some of the other works on WSN gateways. In Sec. 2.3 I detail one of the commercially available systems for data collection.

2.1 Wireless Sensor Network Background

Wireless sensor networks (WSNs) are increasingly important for enabling continuous monitoring in many fields including environment sciences, water resources, ecosystems, structural health and health-care applications. In many such applications, a large amount of observation data in a monitoring sensornet needs to be transferred to data sink(s) for analyses (e.g. [1–5]). Consisting of a large number of tiny, battery-powered, autonomous sensor nodes (motes), sensornets are fundamentally constrained by motes energy limitation and communication bandwidth. Energy-efficient technologies, such as energy-efficient communications, cannot only fundamentally address sensornets power limitations but also foster environmental sustainability and the economics of energy efficiency.

2.1.1 WSN Transport Layer Background

While there has been a lot of work done on transport layer protocols for WSNs, as far as I have found, the CDP is the first transport layer protocol to include data compression in the network stack for WSNs. Work on transport layer has attempted to achieve replacement of the Transmission Control Protocol(TCP). Zafar surveys some of the attempts to modify or replace the TCP for WSNs in [6]. Other transport layer protocols attempt to achieve the usual goals such as flow control [7, 8] and reliability [7, 9–11]. While these goals are useful in both traditional networks and WSNs, the additional energy constraints of WSNs provide an additional opportunity for transport layer protocols. CDP attempts to reduce energy consumption by compressing data being sent over the network. It was also found, during testing, that the CDP reduces the Packet Error Rate for a number of reasons discussed in the CDP chapter. This allows the CDP to improve both energy efficiency and reliability.

2.2 WSN Gateway Background

Several studies exist about WSN gateway systems, ranging from ordinary aspects of WSN gateways (e.g., [12–16]) to specific concerns such as security [17]. In [13] static packets are required since all customization is done through XML. Reference [16] presents a system using the Stargate hardware and takes into account that the gateway may be resource limited whereas our assumption is that the gateway machine has access to wall power. While many of the systems reported in the previous work are research projects and not yet available for broader use, Xserve [18] is an industrial gateway available to use, but it is proprietary and therefore does not support the level of customization that users may require. Besides, Xserve is also completely tied to the proprietary WSN management system and the Xmesh network protocol, which significantly limits its potential applications to many real-world tasks. In contrast, our work aims to develop a general user-configurable WSN gateway system to work with any WSN routing protocols and management systems in principle. While the

work in [15, 16] shares some goals with ours, our work was independent from theirs, and addresses some challenging issues not addressed in the previous work.

2.3 Data Collection System Background

One of the most popular data collection platforms is probably the MoteWorks platform originally developed by Crossbow and now developed by Memsic. The MoteWorks platform consists of two components. XServe [18], the software that runs at the server level, acts as both a gateway and a managements system. XServe collects data from the network, sends commands to the network, stores collected data and provides a Web page interface for management of the network. On the motes, the XMesh [18] protocol , an ad-hoc mesh network for WSNs, is used. While MoteWorks is a popular platform it is proprietary. This means that XServe, XMesh and Memsic sensor nodes depend on one another. In fact, XMesh won't even work with all Memsic nodes, it only supports the MICAz and IRIS mote platforms and does not support the TelosB platform. These limitations mean that users are locked-in to the vendor and cannot easily deploy arbitrary mote hardware. In the research world, this limits researchers to making decisions purely based on support and reducing the ability for decisions based on financial constraints or choosing hardware that supports a given deployment the best. Additionally, the proprietary nature of MoteWorks limits the configuration options for a deployment. These limitations motivate my work towards a more open system for data collection in WSNs.

3 COMPRESSED DATA-STREAM PROTOCOL (CDP)

3.1 Introduction

In this chapter, we present the design and implementation of CDP, a compressed data-stream protocol for energy- and bandwidth-efficient data collections for WSNs, to simultaneously address the challenges of both energy limitation and bandwidth constraint in sensornets. The primary goal for the design of the CDP is to determine whether it is feasible to build compression into a transport layer protocol without negating the effects of compression due to header overhead.

A number of collection protocols have been proposed in the area of WSNs, including collection tree protocol (CTP) [19, 20], Flush [11], Fetch [4], Wisden [2] and Fusion [21]. These protocols focused on reliable data transport in WSNs to address wireless link dynamics, and rate and congestion control, but none of them considered a data compression approach. On the other hand, most existing works on data compression algorithms for sensornets are focused on the algorithmic level and only examined by numerical simulations (e.g. [22–24]). Notably, S-LZW [25] is a novel sensor version of the well-known dictionary-based lossless compression Lempel–Ziv–Welch (LZW) algorithm [26], and was implemented as a specific application for some targeted scenarios. The experimental results of [25] clearly demonstrate the advantages of data compression approach to energy savings in a real-world sensornet testbed. Despite those works, there are still some concerns about the merit of the data compression approach in sensornets for energy conservation, in the sense that the packet overheads and additional computations for data compression might eliminate the gain achieved by data compression. Such concerns appear to, in a large degree, root from the fact that there is a lack of development of any general transport protocol based on data

compression for data gatherings in WSNs. This motivates our work. We investigate if and how data compression can be effectively supported in general WSN transport protocols in order to be widely used for energy-efficient data collections in various application situations; we also explore the performance limits of data compression approach built in such a general WSN transport protocol for data collection. To this end, our design of CDP is generic and other lossless and/or lossy compression algorithms can be easily plugged into our protocol system without any changes to the rest of the CDP. We envision that the development of a general transport protocol based on data compression approach, such as CDP, is able to not only provide the first of its kind compression-based transport protocol for easy and wide practical use for data collection in sensornets, but also offers a useful and handy research tool for people to further investigate and validate different compression algorithms and their effectiveness for diverse WSN applications to advance the understanding of benefits and limitations of data compression approach in real-world WSNs.

The rest of the chapter is organised as follows. In Sec. 3.2 we describe the system model of temporal compression for many-to-one data collections in WSNs, and then introduce our unified compression algorithm, referred to as generalised predictive coding (GPC), for both lossless and lossy compression for resource-constrained motes. Sec. 3.3 presents our CDP design and focuses on how to reduce the packet overhead by our novel concept of data stream. Sec. 3.4 describes the implementation of CDP in the nesC language and TinyOS operating system. In Sec. 3.5, we present detailed evaluation of the CDP based on TOSSIM and PowerTOSSIM-z simulation environments using real world sensor data streams. Finally, the conclusions and future work are given in Sec. 3.6.

3.2 System Model and GPC

3.2.1 System Model

WSNs can be modeled by graphs. A graph $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subset V^2$. Nodes in V represent autonomous sensor nodes, and edges in E correspond to wireless links among the nodes. Let $SINK \subset V$ denote a small set of particular nodes referred to as data sinks where observations from individual sensor nodes in V should be gathered. The sensor nodes are battery-operated whereas the sinks are assumed not power limited. Sensor nodes transmitting and receiving are the most energy-consuming operations. For example, studies have shown that about 3000 instructions could be executed for the same energy cost as sending a bit for 100 m by radio [27] and, in general, receiving has comparable energy cost to transmitting. Therefore it is appropriate and desirable for one to reduce the total energy usage at sensor nodes by carefully minimising nodes transmission (and hence the corresponding reception), probably offset by a slight increase of computation operations. This leads to data compression-based approach.

In this paper, we consider temporal sensor data compression in WSN data collection paradigm, in which a few number of data streams (the accurate definition of the data stream to be given later in Sec. 3.3) will be consecutively gathered from each individual sensor node. We first briefly describe our novel general data compression framework referred to as GPC [28] upon which the CDP is developed. The GPC extends the previous work on two modal transmission approach for WSN energy-efficient communication [22, 24], and combines both lossless and lossy compression in the same framework efficiently. In contrast, existing WSN lossless and lossy compression algorithms follow different principles and thus none of those algorithms can be applied to both lossless and lossy compression. For example, recent lossy compression algorithms such as LTC [29] and PLAMLiS [30] are based on piecewise linear approximation, and would result in more compressed bits than the raw data bits when

applied to lossless compression. For a comprehensive survey of recent developments of practical WSN data compression algorithms, see [31].

3.2.2 GPC Framework

The basic idea of the GPC is to, for a given residue distribution model, encode only those residues falling inside a relatively small range $[-R, R]$ ($R > 0$ and is called compression radius hereafter) by entropy coding (referred to as predictive compression mode) and to transmit the original raw samples un-coded otherwise (referred to as normal mode). Clearly, the normal transmission mode in the framework also provides a direct (re)synchronisation mechanism between the predictors at sensor node and the sink. Thus the GPC can overcome two fundamental difficulties associated with traditional predictive coding approaches such as recent LEC algorithm [23]: (i) no mechanism to (re)synchronise the predictors used at both transmitting and receiving sides, and (ii) potentially bad residue distribution shapes (i.e. long tails) in practice having adverse impact on entropy coding performance. Moreover, for lossy compression, our GPC essentially makes use of synchronised iterative multi-step prediction at both sensor nodes and the data sink, in which the predicted output for a given time step will be used as an input for computing the sensing signal series at the next time step, with all other predictors inputs being shifted back one time unit. This is in contrast with the lossless compression where the single-step prediction is used at both sensor nodes and the sink. As prediction errors propagate in this iterative multi-step prediction procedure, eventually a residue would become larger than the allowed error bound. At this point, the compression mode has to be switched to the normal mode in our GPC, and the original raw reading(s) will be transmitted to resynchronise the predictors at both sensor node and sink. The number of raw readings to be transmitted is equal to the input demission of predictor used. Thus, the embedded normal mode for transmitting raw samples in the GPC framework has

also been able to directly support iterative multi-step prediction scheme to facilitate lossy compression.

In our unified GPC algorithmic framework, a compression error bound (denoted as e) is used as the control knob, and lossless compression can be processed as $e = 0$ in our framework. Also, please note that the GPC framework is a general framework in which one has complete flexibility to choose appropriate predictor and entropy encoder based on given tasks. The algorithmic procedure at source nodes of the GPC is presented in Fig. 3.1. The corresponding algorithmic procedure at the sink(s) can be described accordingly and easily.

Notation:

e:	given compression error bound;
x:	sensor reading;
$\hat{x}$:	prediction of *x*;
residue:	$x - \hat{x}$;
R:	compression radius ($e \leq R$);
predictor():	any predictor used to predict the next observation based on the previous observations or predictions;
p:	the input demission of predictor;
flag:	the flag to indicate the predictor (re)synchronisation status;
encode():	encode residues with any encoder being used;
pack(y, n):	packing *n* data items *y* at source node to into a packet;
send():	transmitting at source node.

```
GPC (e):
begin
        flag← TO_SYNC;
        while (not_done)
                if (flag = TO_SYNC)  then
                        pack (x, p); flag ← SYNC_DONE;
                else
                 x̂ ←predictor(); residue ← x − x̂;
                if (e = 0)  then /* lossless compression */
                        if (residue≤R)  then pack(encode(residue), 1);
                        else pack(x, 1);
                if (e > 0)  then /* lossy compression */
                        if(residue ≥ 0 and residue ≤ e) then pack(encode(0), 1);
                        else if (residue > e and residue≤R) then pack(encode(residue), 1);
                        else  flag← TO_SYNC;

        send();
end
```

Figure 3.1. GPC framework

3.2.3 GPC Realisation

In our development of CDP, which employs GPC for data compression, we adopt the simplest linear predictor to predict the next sample based on the last observed sample, that is, $\hat{x}_i = predictor(x_{i-1}) = x_{i-1}$. Then the residue is the difference $r_i = x_i - x_{i-1}$. The choice of this simplest predictor is based on our following considerations. First, we found that sensor observations in many real-world applications, such as environmental monitoring (e.g. [22]), the prediction performance of this simplest predictor is comparable with other more sophisticated predictors including higher order of linear models and non-linear models. Thus, the selection of the simplest predictor can greatly reduce the computation overhead of making the prediction at the motes. Second, the adoption of the simplest predictor WSN-wide improves the scalability of WSN deployment, because the sink only needs to maintain one simplest predictor for thousands of sensors in the sensornet. Otherwise, if individual sensors used their best predictors, the sink would have to potentially maintain thousands of different predictors and thus would suffer from the scalability issue. Third, as our design of CDP is intended to be generic, so that it can be used as a tool in research as well as in applications, our initial selection of a predictor in the GPC only serves as a default predictor, since our design and implementation of CDP allows the default predictor to be easily replaced with any other predictors that could be better for given applications. Furthermore, as described in Sec. 3.3, users can even easily replace the entire GPC, the default data compression framework in CDP, with another compression mechanism.

With the same assumption of the residual distribution model used in LEC [23], we adopted the entropy encoder employed in LEC [23] in the realisation of GPCs predictive compression mode. (We note that any specific residual distribution model and entropy encoder are certainly not tied to the GPC framework, and hence can be easily replaced with other alternatives in the CDP.) The adopted encoder is a modified version of the Exponential-Golomb code of order 0 [32]. Basically, the alphabet of residues is divided into groups to reduce the alphabet size. Thus, any residue ri is

represented in two parts: group code it belongs to and its index in that group. Based on the residual model and entropy encoder adopted, the compression radius R is simply selected as $2^{K-1}-1$, where K is the resolution of A/D converters used in WSN motes. In the normal mode of the GPC, uncompressed raw samples are transmitted. The size of the coding table used in the GPC of our CDP implementation is just $K+1$ entries, whereas S-LZW [25] uses significantly more memory space for its dictionary entries and mini-cache entries (e.g. MAX_DICT_ENTRIES being 512 and MINI-CACHE_ENTRIES being 32 [23, 25]). Table 3.1 gives the coding table when $K = 14$, where S_i represents the residue group code for residue r_i and n_i indicates the number of bits of r_i''s index that follows S_i. For example, if $r_i = 2$ and $r_j = -2$, then their group codes will be 01110 and 01101, respectively. Note that the index for negative r_i is computed by $2^{n_i} - 1 - |r_i|$. When $S_i = 111111111110$, S_i is no longer a group code in the compression mode of the GPC but the code flagging the normal mode of the GPC, which is followed by an original raw sample.

3.3 CDP Protocol Design

In designing CDP we attempt to achieve two specific goals. The first goal is to minimise the packet overhead, so that the protocol overhead does not negate the benefits of the data compression. To this end, a novel concept of streams is developed which is described in Sec. 3.3.1. The second goal is to provide a platform for researchers to develop and test any new compression algorithms effectively. This is achieved by keeping our protocol design modular for easy plugging of different compression algorithms, which is described in Sec. 3.3.2. We note that the reliability of transport is not addressed by CDP, as CDP is intended to be a lightweight transport protocol. The reliability of transport in CDP depends on the reliability of the underlying network layer protocol.

Table 3.1
Coding table ($K = 14$) in GPC of CDP

n_i	S_i	r_i
0	00	0
1	010	$-1, +1$
2	011	$-3, -2, +2, +3$
3	100	$-7, ..., -4, +4, ..., +7$
4	101	$-15, ..., -8, +8, ..., +15$
5	110	$-31, ..., -16, +16, ..., +31$
6	1110	$-63, ..., -32, +32, ..., +63$
7	11110	$-127, ..., -64, +64, ..., +127$
8	111110	$-255, ..., -128, +128, ..., +255$
9	1111110	$-511, ..., -256, +256, ..., +511$
10	11111110	$-1023, ..., -512, +512, ..., +1023$
11	111111110	$-2047, ..., -1024, +1024, ..., +2047$
12	1111111110	$-4095, ..., -2048, +2048, ..., +4095$
13	11111111110	$-8191, ..., -4096, +4096, ..., +8191$
14	111111111110	original raw sample

3.3.1 Data Streams

In order to minimise the packet overhead and memory use in CDP, we introduce a novel concept of streams. In the context of the CDP protocol, a data stream is defined as an aggregate flow of multiple individual sensor data flows from a single mote employing the same compression algorithm with corresponding parameters. Note that if any mote has sensors with K different sampling rates, K individual streams have to be created in CDP for this mote. Fig. 3.2 illustrates an example node with two streams. Stream 1 will contain all data flows from sensors 1, 2 and 3 whereas stream 2 will contain data flow from sensor 4. The motivation of organising the sensor data flows of a WSN into the newly defined data streams in CDP is that, by aggregating multiple sensor flows on a single node, we can minimise the protocol overhead, reduce the number of packet buffers required at motes, and provide flexibility for supporting different compression operations (by either different compression algorithms or different parameters of the same algorithm). As a specific case, a data stream can be an aggregate flow consisting of multiple sensor data flows from a single mote with an identical sampling rate but without any compression at all.

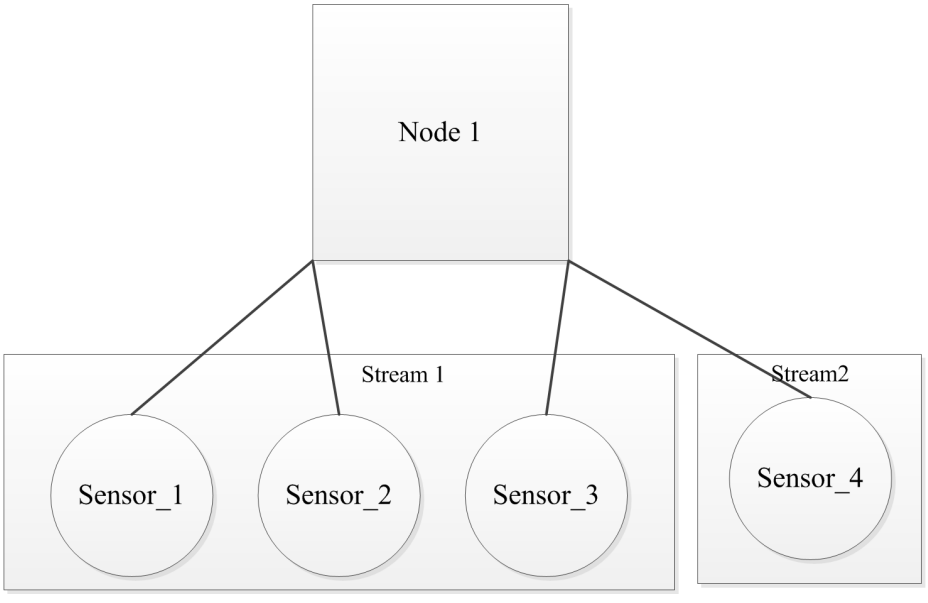


Figure 3.2. Example of a node with two streams

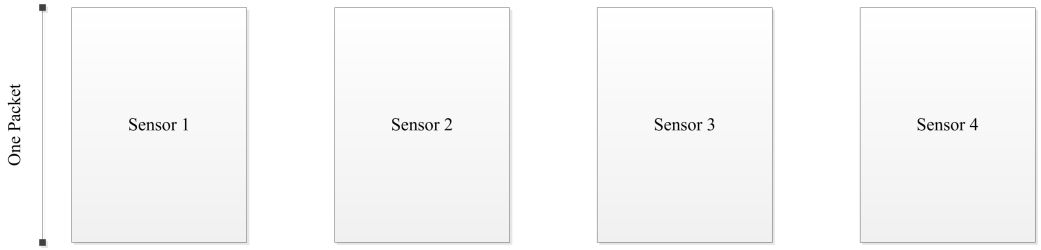
3.3.1.1 Overhead Reduction

Overhead reduction: Protocol overhead is a major issue in designing a compression protocol. Owing to the small size of packets, it is vitally important that we do not negate the benefits of compression by introducing a high overhead to our packets. By only requiring information such as compression algorithm and sensor information be sent once, at stream setup, we reduce much of the information that would be required in each data packet. After stream setup each data packet only requires the stream id as additional header information. Additionally, the requirement of a consistent sampling rate within each stream eliminates the need to identify each compressed sensor reading in the data segment of the packet. This reduces each packets overhead by $m * \log_2 n$ bits, where m is the number of readings in that packet and n is the number of sensors associated with that node.

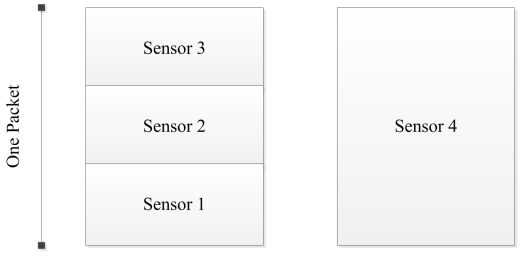
Owing to the limited memory of motes, buffer space may be prohibitive. Commonly used WSN operating systems, such as TinyOS, do not support dynamic memory allocation, which makes the efficient use of memory quite challenging. Streams can reduce buffer overhead. To illustrate, let us consider an alternative solution in which each packet only contains data from a single sensor. In this case, the packet overhead would only require the algorithm id and the sensor id. Although this solution could eliminate much of the packet overhead, it would introduce a significant memory overhead. This is because each node will have to maintain a packet-sized buffer, because of lack of dynamic memory, for each of its connected sensors. Taking the example node given in Fig. 3.2, Fig. 3.3 shows how packet buffers would work (i) without streams (Fig. 3.3(a)) and (ii) with streams (Fig. 3.3(b)). Clearly the solution with streams can significantly reduce this memory overhead whereas the solution without streams could quickly undermine the practicality of a compression-based protocol by requiring excessive memory as the number of sensors connected to each mote increases. Assuming s sensors per stream, then the amount of buffer space required by CDP will merely be $1/s$ of the memory which would be required otherwise for the alternative solution of a single sensor flow per packet.

Owing to the lightweight design consideration of CDP, the sampling rate configuration that is usually either supported by WSN configuration management or implemented by applications is not specified in CDP. The only requirement is that the sampling rate for all sensors data flows within a stream should be identical, either static or dynamic. This is a necessary design decision to provide the benefits of streams.

Moreover, our concept of streams allows a simplification of data collection in a complicated WSN where multiple compression algorithms (e.g. lossless compression and lossy compression) are used at the same time in addition to diverse sampling rates, because of the different physical variables and mote locations in a WSN large-scale deployment. When several sensors data flows of a mote are grouped into a data stream, data packets only need to carry the stream id instead of individual sensor flow identifiers.



(a) One packet buffer for each sensor



(b) One packet buffer for each stream

Figure 3.3. Buffers required for the example node

3.3.1.2 Stream Setup and Control Packet

To set up a data stream, one should specify a sampling rate shared by all data flows in the stream, a compression algorithm used with given parameters, and how to distinguish individual sensor data flows collected in the stream. Since all sensor data flows in a stream use the same sampling rate, the relative order of data from individual sensors can be fixed for an easy identification of individual sensor data flows within a stream. In our design of CDP, we consider individual motes in a sensornet are autonomous. The stream setup specification for individual motes is achieved through control packet(s) exchanged between the motes and the sink by CDP during stream creation process. Once a stream is set up via control packet, data flows from the stream can be collected forever via data packets.

Each stream setup packet begins with the 16 bit node ID but this ID is retrieved from the lower layer packet (i.e. cross-layer information) to avoid additional overhead in CDP. Additionally, each stream is assigned a stream id to identify which stream a data packet belongs to, as each mote can support up to eight independent data streams in CDP. Stream id, compression (selected among all the implemented algorithms) and sensor list are specified in their corresponding fields in CDP control packet. The sensor list for the created stream will specify the relative order of all sensors data flows belonging to that stream by their identifiers within the mote. The control packet structure is illustrated in Fig. 3.4. The general packet structure is given in Fig. 3.4(a), whereas the control packets for the two streams illustrated in Fig. 3.2 are shown in Fig. 3.4(b). The dotted field of node ID is 'virtual' as it does not exist in the CDP control packet to minimise the overhead. the node ID is actually obtained from lower level protocol.

3.3.1.3 Data Packets

Data packets in CDP are very simple. They simply contain the node id, stream id and the compressed data. Node id is, again, retrieved from lower layer header information.

The payload section will consist of a cycle of one reading from each sensor repeated until the packet is filled. As CTP does not guarantee delivery, it is important to ensure that a packet loss does not introduce any errors to subsequent packets received at the sink. We can simply use GPC normal mode for resynchronisation at the beginning of each packet to achieve this. Fig. 3.5 shows the general structure of a CDP data packet. Similar to the CDP control packet, the dotted field is 'virtual'

3.3.2 Modular Design

In order for CDP to be useful as a tool for researchers to investigate and test compression algorithms it is vital that we design CDP in such a way that the GPC may be easily replaced by another compression algorithm. This consideration led to our modularised design for CDP. The entire design of our CDP is broken down into three major components: network access, utility and compression. Fig. 3.6 shows the overall modular design of CDP.

Node ID: 16 bits	*Stream ID*: 3 bits	*Algorithm*: 2 bits	*Sensor List*: 4 bits each

(a) General control packet structure

Node ID = 0000000000000001	*Stream ID* = 001	*Algorithm* = 00	*Sensor List*: 0001, 0010, 0011

Node ID = 0000000000000001	*Stream ID* = 010	*Algorithm* = 00	*Sensor List* = 0100

(b) Illustration of control packets

Figure 3.4. Illustration of control packet

Node ID: 16 bits	*Stream ID*: 3 bits	Payload

Figure 3.5. Data packet format

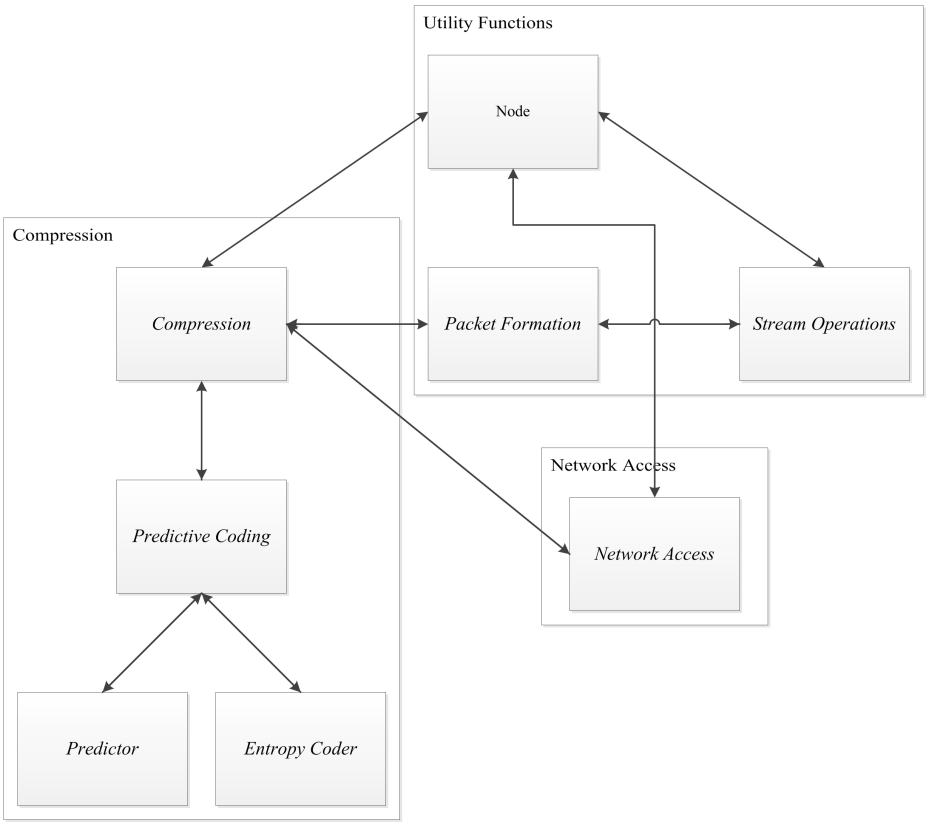


Figure 3.6. Overall modular design of CDP

3.3.2.1 Network Access

This module provides the interface for initialising the lower level network, sending packets and receiving packets. This module will maintain separation of network services from the rest of CDP so that the underlying network protocols may be changed based on the requirements of the application. Although CDP is a collection-based protocol, it is possible with some logic in the network access module to implement CDP on top of any protocol that provides a path from each mote to the sink. If CDP is being used on top of a primitive network stack, additional logic may be added in this module to improve the performance. The network access module will, additionally, provide a platform for lower level protocol developers to easily test underneath CDP. This would allow lower level network protocols to be designed to cater to compressed data without actually implementing the compression itself.

3.3.2.2 Utility Modules

Three separate modules are designed to support the underlying structure of CDP. The packet formation module is responsible for building and reading packet headers. This module allows for modifying header structures, packet types and the method for actually building the header. The module specifically defines methods for building configuration and data headers as well as reading received headers. The stream operations module is responsible for everything related to streams. It provides methods for building and sending streams as well as passing stream data to other modules that may require these data. These packet formation and stream operations modules define the basic operation of CDP. Modifying these modules will, obviously, change the fundamental way that CDP operates. The node module simply provides an easy to use interface, by abstracting away design details, to applications using CDP.

3.3.2.3 Compression Modules

This major component consists of the group of modules including compression, predictive coding, predictor and entropy coder (see Fig. 3.6). This group of modules allows for new compression algorithms to be implemented and plugged into CDP with minimal modification on corresponding module(s) in this group. Furthermore, the design for GPC allows for changing implementations of predictors and/or entropy coding tables with little or no code change of the module(s).

The compression module passes the sensor data off to the selected algorithm related to a stream. With a standardised interface for compression algorithms, this module will allow a new compression algorithm to be merged with CDP by simply implementing the algorithm in a new module that conforms to the interface and modifying the compression module to point a specific algorithm id value to that module.

GPC consists of three modules: predictive coding, predictor and entropy coder. The predictive coding module implements the GPC framework and should not be modified if the default GPC is used. It supports defining a compression radius and

error bound as well as choosing and executing sync operations and lossless/lossy compression. The predictor module supplies a module with a well-defined interface for predictors. The coding module does the same as the predictor module for entropy coding. This allows for new predictors and/or entropy coding techniques to be implemented to match our interfaces and simply switch in and out for easy testing as well as customised GPC algorithms to optimally fit the sensor data characteristics at given tasks.

3.4 Protocol Implementation

For our reference implementation, we adopted TinyOS 2.1 [33] as the underlying platform, due to the fact that TinyOS is an open source operating system for WSNs developed in the nesC programming language [34] and is widely used both in the research community and real-world WSN applications. CDP is intended, once fully tested, to be useful for real-world WSN applications and the research community, the combination of TinyOS wide use and open source nature makes it an ideal underlying platform for our CDP development. Also, the nesC programming language paradigm provides for a nearly one-to-one mapping of our design modules into TinyOS. Fig. 3.7 illustrates the TinyOS based network stack environment in which our CDP is implemented. As we can see from Fig. 3.7, the NetworkAccess module has three commands (Init, SendCompressed and PushSend) and two events (sendDone and receive) that are wired to the CTP implementation in TinyOS. Command Init initialises the underlying CTP protocol and gets everything ready to send. Command SendCompressed queues compressed data to be sent, where the data will not actually be sent until we have a full packet. Command PushSend forces an incomplete packet to be sent, for more time sensitive data. Event sendDone signals after a packet was successfully sent and Event receive signals at the root when a compressed message has been received.

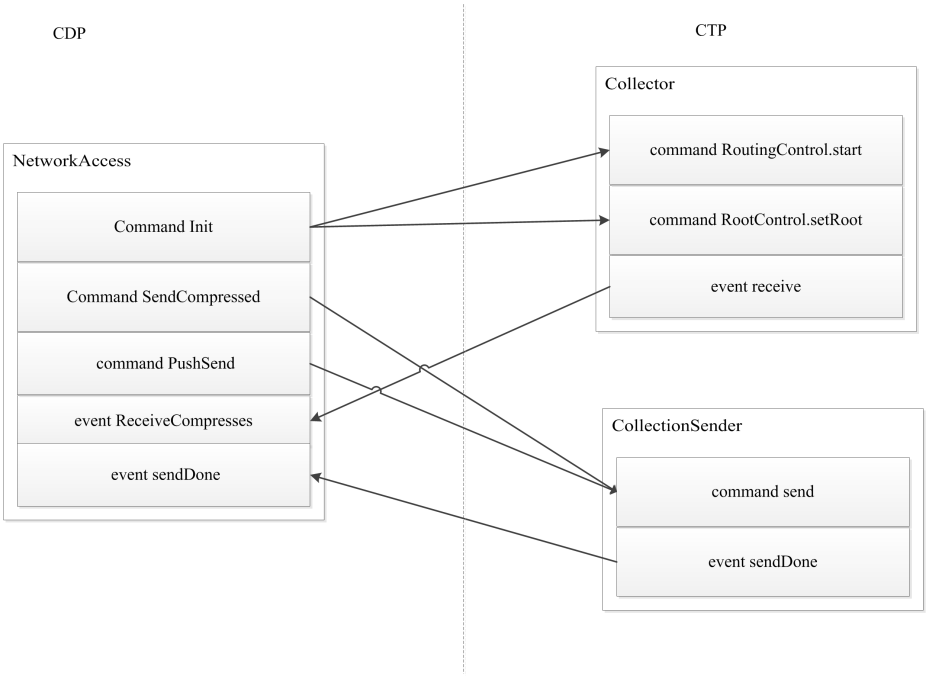


Figure 3.7. CDP network stack

3.4.1 Collection Tree Protocol

In our CDP implementation, we adopted the CTP, a tree-based collection protocol, as the underlying routing protocol [19,20]. As an integral part of TinyOS environment, CTP provides a number of benefits over other routing protocols. It is a collection protocol, which provides CDP with the proper routing for data collection. It has been shown that CTP outperforms similar collection-based protocols and is quite reliable. The two primary benefits of CTP over other collection protocols are because of data-path validation and adaptive beaconing [19]. These two methods allow for 73% fewer packets and greater than 90% packet delivery rate [19]. Additionally, CTP reduces topology repair latency by 99.8% [19]. More details about CTP can be found in [19, 20].

3.4.2 Packet Buffers

CDP tries to maximise the amount of data in each packet to help minimise the impact of packet header overhead. However, nesCs avoidance of dynamic memory allocation means that we must keep full-sized packet buffers. This can be an issue, because of the limited memory of sensor motes. CDPs design allows for minimal buffering in the following two ways.

By using streams, we must keep a one packet-sized buffer per stream. With default 29 bytes of packet on motes with 802.15.4 radios such as the CC2420, the maximum buffer size for up to eight streams per node is 232 bytes. In actual implementation, the maximum buffer size will be slightly smaller, as we only keep the data segment in memory. However, many applications usually have a smaller number of streams for motes, which can be improved by providing a MAX_STREAM_NUM constant at compile time so unnecessary buffer space will not be allocated. On the data sink side; we maintain a configurable buffer of packets which may be kept low in the typical case where the data are forwarded to the gateway immediately after reception.

To achieve maximum data segment size for compressed data we create and maintain two additional buffers on each mote. The first buffer, pending, maintains s independent sections of $n_s * B$ bits of data, where s is the number of streams, n_s is the maximum number of sensors per stream and the B is the maximum size of one samples coding, rounded to the nearest byte. From Table 3.1, B would be 12 bits (i.e. the size of normal mode code of the GPC, see Table 3.1) plus the size of an original raw sample. This buffer is maintained to determine if a whole set of readings will fit in the remaining space in the packet. If not, the current packet will be sent and a new packet will be started. The second buffer, raw, maintains the raw data corresponding to data in pending so that the original raw data may be sent in normal mode if a new packet needs to be generated.

3.4.3 Decoding Implementation

In our implementation of the entropy coding used in GPC, we tried to keep the code as simple and easy to maintain as possible. While encoding is straightforward, decoding seemed to require a complex block of code. We decided to use an array-based finite state machine for decoding. To read the S_i from Table 3.1 and return the n, we begin in an initial state and read bits to change state until we attain one of the final states. This tells us how many bits the following reading will occupy. The operation of the decoding algorithm is described in Fig. 3.8. An example state table is given in Table 3.2. Note that decoding is performed at the gateway which is not energy-limited. Additionally, performing decoding at the gateway provides for the ability to use asynchronous compression algorithms. That is, since the gateway is not computationally limited in the way sensor nodes are, it is appropriate for compression algorithms whereby compression is computationally simple but decompression is computationally complex. This may allow for the development of new and novel compression algorithms for use in WSNs.

```
Notation:
curr_state:                 Current state of reading S_i
read_bit():                 Returns the next bit in the packet
NUM_STATES:                 Total number of states in array
NUM_FINAL:                  Total final states in array
state_table[2][NUM_FINAL]:  Table containing all states

get_n():
begin
curr_state = 0

      /*transition states until reaching a final state*/
while(curr_state< (NUM_STATES – NUM_FINAL))
bit = read_bit()
curr_state = table[bit][curr_state]

      /*return n*/
Return curr_state – (NUM_STATES – NUM_FINAL)
end
```

Figure 3.8. Decoding procedure

Table 3.2
State table for the codes 00 and 01

		Current state			
		0	1	2	3
Bit	0	1	2	2	3
	1	−1	3	2	3

3.5 Simulations and Analyses

We have performed simulations to evaluate CDP. Sec. 3.5.1 briefly describes the two simulators used in our experiments, TOSSIM [35] and PowerTOSSIM-z [36]. Sec. 3.5.2 describes the simulation setup in detail, including a randomly generated sonsornet node location map and real-world sensor data sets used. It should be noted that our reference implementation, compiled for the MICAz platform consumes an extra 429 bytes of RAM and 2,978 bytes of ROM over CTP alone. This was analyzed by compiling a sample application that simply sends some value over the network repeatedly. In MICAZ motes there is 128 kilobytes of ROM and 4 kilobytes of RAM so this is reasonable but may be reducible through some optimizations. Compiled sizes may vary based on platform. We provide the simulation results and analysis of CDP performance and energy consumption in Sec. 3.5.3 and Sec. 3.5.4, respectively.

3.5.1 TOSSIM and PowerTOSSIM-z

TOSSIM is a simulator for TinyOS-based WSNs [35]. TOSSIM works by replacing a few key TinyOS modules during compilation, primarily the hardware-reliant modules, with simulation code allowing the TinyOS code to be compiled to the simulator instruction set. The code is broken down into events, discrete portions of code and queued as discussed in [35]. This allows for efficient simulation of large networks. Another advanced feature of TOSSIM is its environmental noise modeling. TOSSIM

allows its simulations to take in an environmental noise trace and then attempts to accurately simulate real-world noise using closest-fit pattern matching as discussed in [37]. A commonly used real environmental noise trace with TOSSIM today is called as Meyer Heavy noise trace which was taken at the Meyer library at Stanford during heavy 802.11 activity [37]. However, TOSSIM is unable to accurately simulate the per-instruction time and anything else that relies on it, including energy usage. PowerTOSSIM-z [36], a port of PowerTOSSIM [38], attempts to add accurate energy modeling into TOSSIM. PowerTOSSIM-z achieves this goal by logging power state transition information during simulation. CPU energy usage modeling is more complex and is discussed in depth in [38]. Although PowerTOSSIM-z has some limitations [36], PowerTOSSIM-z performs without needing to directly emulate the mote hardware, allowing for faster energy modeling than is available in more traditional emulation environments.

3.5.2 Simulation Setup

In our simulations we generated a sensornet topology using the topology generator included in the TinyOS package. We generated a WSN with total 33 nodes (i.e. one data sink and 32 motes) randomly distributed over a 100m by 100m area, as shown in Fig. 3.9. We used the first 5000 lines of the Meyer Heavy noise trace for our simulations. By reducing the length of the noise trace we vastly reduced the memory and time requirements of running a simulation. Owing to the nature of noise modeling in TOSSIM and routing in CTP, the network topology may vary greatly over time. Links have various qualities, for example, the link from node 6 has a link to node 0 which is at -74.99 db gain but its link to node 2 has a -16.69 db gain. This layout should give a realistic location map motes in the simulated multi-hop sensornet with different number of hops to the data sink for the evaluation of CDP for data collection. Examination of the topology during our simulations shows that the number of hops

from motes to the sink ranges from one to nine, and discounting routing loops that may occur at times.

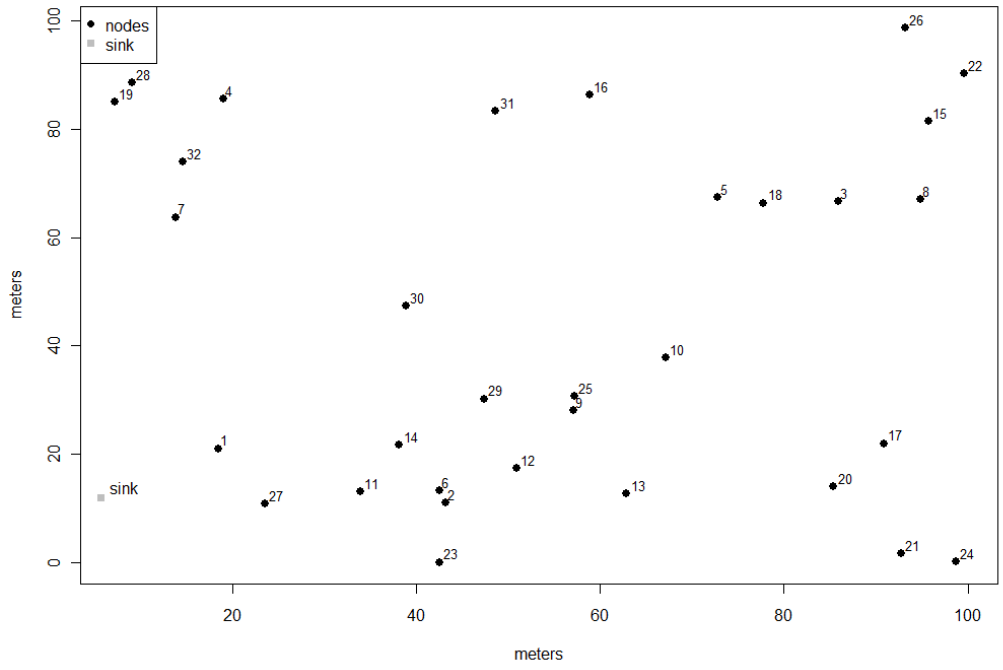


Figure 3.9. Mote location map of the WSN in our simulation

We adopted the publically available real-world WSN Patrouille des Glaciers (PDG) 2008 datasets provided by SensorScope [3] for the simulation of lossless data collection. Sensor nodes use temperature and humidity data from a node in the original data collection. In our simulations, each set of data is replicated on four nodes at various distances from the sink node, as shown in Table 3.3. The first two columns of Table 3.3 show the assignment of nodes in our simulations to station ids in the original SensorScope data. Each node in the CDP simulations uses a single stream, composed of both humidity and temperature data, with the sampling rate set to be one reading every 100ms. To thoroughly and fairly evaluate the performance of the CDP, CTP alone is simulated as a performance baseline. The selection of CTP alone as the performance baseline is due to the fact that CTP is popular, effective, well

evaluated and integrated with tinyOS environment. In addition, the CDP runs on top of CTP. In the CTP simulations each packet contains four humidity and four temperature readings rather than a single set. This allows CTP to behave more favourably against the aggregation inherent in our CDP.

3.5.3 Performance Evaluation

We first conducted TOSSIM simulations to study the general performance of the CDP, including retransmissions, and total protocol overhead (including lower layers). Owing to the randomness of environmental noises with TOSSIMs noise modeling, each simulation trial would have somewhat different result. To effectively eliminate the random fluctuations of the simulation results, all our simulation results are based on the average of five individual simulation trials.

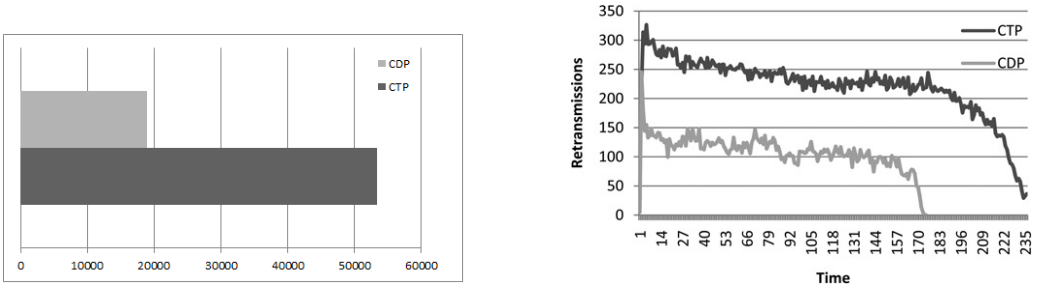
We first compute the empirical static compression ratio of CDP based on our simulations. Since in the benchmark of using CTP alone the sensed raw data are sent directly over CTP, we can use the total bytes of the original raw data payload sent by CTP and the total bytes of CDP packets (including CDP protocol overhead) sent by CTP for the same amount of sensed raw data to compute CDP compression ratio. The CDP compression ratio, as defined by $1 - D'/D$ where D′ is the size of CDP packets and D is the size of the corresponding raw data, is computed and listed in Table 3.3. Note that these actual CDP compression ratios include the CDP protocol overhead, which is in contrast to the algorithmic level compression ratios [22–24] that were obtained without including any protocol overhead. The fourth and fifth columns of Table 3.3 show the total number of static packets of using CTP alone and CDP for each sensor source, respectively. That is, sending the same amount of data by CDP requires significantly less static packets than those by CTP alone.

In addition to simply reducing total data payloads by the significant static compression ratios presented above, CDP is found to be able to reduce retransmissions significantly. Fig. 3.10(a) shows the average total number of retransmissions over five

Table 3.3
Node assignment to sensorscope data and CDP compression ratios

Nodes	PDG 2008 ID	# Hum\Temp pairs	# CTP packets	#CDP packets	Compression ratio
$1, 9, 17, 25$	1	3665	916	671	35.23
$2, 10, 18, 26$	3	3041	760	538	37.56
$3, 11, 19, 27$	4	3041	760	541	37.12
$4, 12, 20, 28$	5	1403	351	267	32.29
$5, 13, 21, 29$	7	4535	1134	772	40.31
$6, 14, 22, 30$	10	3657	914	618	40.55
$7, 15, 23, 31$	15	4662	1166	818	38.23
$8, 16, 24, 32$	16	3072	768	555	36.09

trials in CDP versus CTP alone. Fig. 3.10(b) illustrates the average retransmission dynamics over five trials of CDP against CTP alone in the sensornet. This difference is due to two factors. First, the frame error rate can be significantly reduced due to compression [22]. Additionally, since the compression rates among nodes vary, CDP adds an ad hoc delay between different nodes transmissions that reduces the likelihood of collisions.



(a) Comparison of retransmissions (b) Retransmission dynamics

Figure 3.10. Illustration of retransmissions in the sensornet

Moreover, CDP reduces lower layer overhead by minimising the total number of packets sent. CTP and the protocols below it, such as IEEE 802.15.4 for example, use a combined 21 bytes (i.e., 8 bytes of header for CTP, 10 bytes of header for IEEE 802.15.4 and one byte of active message type for TinyOS) of overhead per CTP packet. On average, this reduced the total lower layer overhead, accounted for both transmissions and retransmissions, from $1,528,340$ bytes to $739,960$ bytes over the five trials. That is, the average of total lower layer overhead reduction is more than half of the original.

Overall, the average total size of data packet transmissions and retransmissions sent by CDP is significantly less than that sent by CTP alone as shown in Fig. 3.11, in which the average total bytes contributed by transmissions and retransmissions have included all corresponding packet overheads of IEEE 802.15.4, CTP, and CDP accordingly. The dynamic CDP compression ratio, due to reducing retransmissions

and corresponding lower level overheads, has an overall average of 55.23% over five trials, realising additional savings over the static compressions shown in Table 3.3.

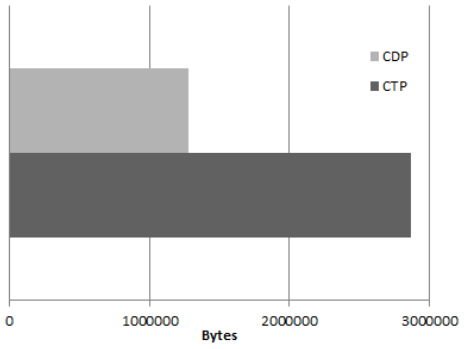


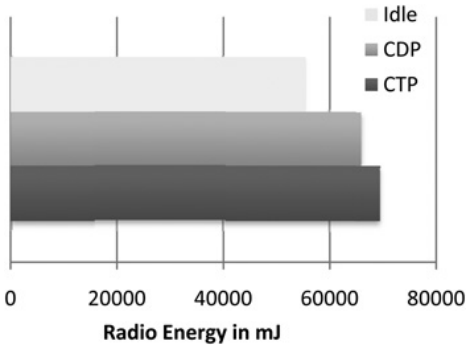
Figure 3.11. Comparisions of total data transmitted in the sensornet (in bytes)

3.5.4 Energy Evaluation

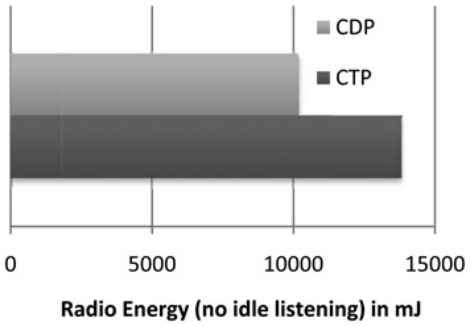
In addition to our TOSSIM simulations on CDP perfor4mance, we have conducted further simulations with PowerTOSSIM-z to attain energy usage statistics for CDP against CTP alone. To gather truthful energy usage statistics, it is necessary to make some small changes to our previous simulations reported in Sec. 3.5.2. This is because idle listening at each node was included in the previous simulations. It was found, however, that such idle listening actually dominated the overall power consumptions in the simulated sensornet. To obtain truthful energy usage statistics for CDP against CTP alone, we actually conducted three sets of simulations with powerTOSSIM-z: (i) CTP alone simulations, (ii) CDP simulations and (iii) idle listening simulations. In the idle listening simulations, the radio transceiver on each mote was turned on but disabled any protocol activities above the physical layer. The simulations of idle listening were run for the same length of time as the CTP and CDP simulations to determine the energy cost of idle listening. Again, to eliminate the random fluctuations of the simulation results because of environmental noises, we averaged the energy usage results of five simulation trials of CTP, CDP and idle listening, respectively.

Then, the energy cost of the idle listening is excluded to determine the actual energy savings of data collections using CDP over CTP alone.

The simulation results of average single-node radio energy consumption obtained using PowerTOSSIM-z are shown in Fig. 3.12(a), where the idle listening is a large portion of radio energy consumption in the simulated sensornet. In Fig. 3.12(b), by removing the idle listening energy consumption from both CTP alone and CDP simulations, we observe that data compression through CDP reduces radio energy consumption by about 26.2% in real-world sensor data situations, in comparison to CTP alone for data collections.



(a) Comparisons of radio energy usage in the sensornet



(b) Radio energy comsumptions of CDP against CTP (no idle listening)

Figure 3.12. Energy consumption in the sensornet

Next, we study motes microcontroller energy usage for CDP processing. While typically sensor nodes power consumption is dominated by its radio, we want to verify that CDPs compression operations do not cause any significant increase in CPUs power usage and thus would not negate the benefits of radio energy savings. Our simulations show that 1the sensornet operating CDP only consumed an average of 113.9 mJ for CPU energy usage per mote. Even assuming zero CPU energy usage for CTP alone, the increase of CPU energy usage per node for CDP data compression is negligible compared to the average radio energy savings of 3618 mJ per node by CDP. Thus, the total energy savings (considering both radio and CPU energy usages)

of CDP over CTP would be about 25.4%, close to the energy reduction ratio of 26.2% where the CPU energy usage of CDP is not included.

3.6 Conclusions

In this paper, we have presented the design and implementation of CDP, an energy-efficient data compression-based transport protocol for data collections in WSNs. The key features of our CDP design and implementation are: (i) exploiting our novel unified algorithmic framework GPC, to effectively provide both lossless and lossy compressions; (ii) minimising the protocol overhead and at the same time providing considerable flexibility for complex network data gathering operations where diverse sampling rates and both lossless and lossy compression algorithms with different parameters are simultaneously supported; (iii) demonstrating the merits of data compression-based general transport protocol for network energy efficiency via simulations using real-world sensor data; (iv) providing a research platform for developing and testing new data compression algorithms for networked data collection in that new algorithms can easily replace the current default one in the CDP without affecting the rest of CDP implementation. To our knowledge, CDP is the first of its kind transport protocol for energy-efficient WSN data collections. Our simulation evaluation on CDP shows that, in addition to remarkable compression ratios, CDP can significantly reduce retransmissions in noisy WSN, which does not only reduce the data retransmissions but also reduces the total lower layer packet overhead, altogether resulting in substantial savings on total transmitted bytes. Moreover, our simulation shows that the proposed CDP design and implementation enables the energy usage of CDP protocol processing including data compression is negligible for real-world sensor data collections. Our future work includes to further thoroughly evaluate CDP in a real watershed monitoring WSN testbed, which is currently in deployment. We also plan to use the combination of the origin, collect_id and seqno

from CTP to allow us to uniquely identify packets [20], in order to attain a higher level of reliability with little or no additional packet overhead.

4 GENERAL GATEWAY

4.1 Introduction

WSNs are being widely applied to the areas of environmental research and monitoring, commercial/manufacturing processes, health care, military, and others. In most monitoring WSNs development and deployment, one of the most vital goals is to collect and aggregate the sensor data (e.g., environmental data, patient data, battlefield data) for analysis. While this work is popular not much work has gone into making deployment based research easier and more cost-effective for researchers without a lot of technical knowledge. To this end, WSN gateway development is critical for data gathering in diverse WSNs in real-world tasks.

A WSN gateway connects a WSN to the Internet, as illustrated in Fig. 4.1, where remote users can easily retrieve WSN data through the Internet in a real-time manner, and analyze their collected data either online or offline. As we can see, there are four primary components in Fig. 4.1. The first component is WSN motes and their application code for data sampling and communication. The second component is the WSN sink and its code to collect all mote data and pass them on to the connected gateway machine. The third component is the gateway system which is in charge of receiving data from the sink, doing any preliminary processing necessary, storing the data to a local database, and forwarding the data to the data management system or application via the Internet. It should be noted here that complex data analysis is not expected to be done in the gateway; only simplistic and necessary work should be done such as transforming the data into an easier to use format. The fourth component is the data management system on the back-end server which receives the data from the gateway, provides functionalities for monitoring/viewing live data

and other more complex data analysis, and backs the data up to a database. In this framework, the data management systems database is the primary backup while the gateways database is a secondary backup in case the management system is down or unreachable, which provides the WSN with two levels of backup for reliable data connection.

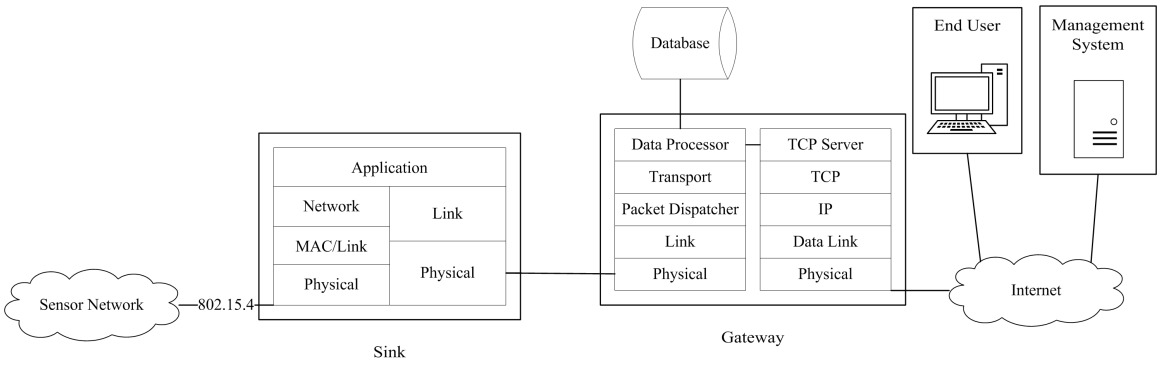


Figure 4.1. A network view of the gateway

However, while many advances in WSNs have been made, there is still a lack of general software platforms for WSN gateways that can be easily deployed in various WSN applications, with a variety of WSN protocols and management systems. Besides, a desirable WSN gateway platform should be able to support WSN transport layer. Due to the resource limitations on WSN motes, WSN transport layer protocols can be quite asymmetric in terms of processing, where the protocol processing at individual motes can be easily performed (e.g., some data compression mechanisms) but the final processing may overwhelm if performed at sink mote. This motivates our work. In this paper, we present our design and implementation of a general WSN gateway software system in a novel way for various WSNs. The unique characteristic of our work is that our developed WSN gateway is aimed to be easily configurable for nearly any WSN data collection applications, with diverse data managements and WSN protocols. In particular, it supports WSN transport layer at the gateway which is assumed not to be power limited. In this way, the developed gateway should

directly facilitate heterogeneous WSNs management [39] and support various WSN routing protocols and transport-layer data collection protocols such as compressed data-stream protocol CDP [40].

The rest of this chapter is organized as follows. Sec. 4.2 presents our design criteria for the gateway. Sec. 4.3 describes important design features. Sec. 4.4 discusses some specific implementation decisions. Sec. 4.5 gives our initial testing of the gateway system. Sec 4.6 presents our conclusions and outlines future work. Additionally Sec. 2.2 briefly discusses previous work on gateway systems for WSN.

4.2 Criteria of the Gateway

We begin with defining some important design criteria that we follow in this work.

- The focus is a general user-configurable WSN gateway architecture and system for effective and efficient data collection.

- The gateway must easily support both static and dynamic packet formats. Much of the current research on gateway design for WSNs relies on the fact that packet formats are static by using methods such as XML [13] to define packets. However, in many situations, variable packet size and dynamic packet formats are necessary. For example, the packet in CDP [40] is dynamic.

- The gateway should be separate from WSN management system, but at the same time, must provide a simple interface for the management system. The separation of gateways from WSN management system is to facilitate network management for heterogeneous WSN systems [39], where different gateways could be adopted for different WSNs, with a unified management system.

- The gateway must reliably log all received packets to a database. To eliminate possible packet loss, it is desirable to conduct logging at the gateway instead of leaving that solely to the management system.

4.3 Gateway Design

In addition to our criteria, we describe some specific design goals first in Sec. 4.3.1 that we attempt to achieve. Sec. 4.3.2 then details the actual design of the system in a top-down manner.

4.3.1 Gateway Design Goals

In designing our gateway we have four primary goals.

- The gateway should not make any assumptions regarding the protocol stack running on the motes. The reconfiguration of the sensor network to use a new network stack should not require a major change in the gateway. This enables the gateway to easily integrate with different WSNs that implement different network stacks. Any changes that must be made should be handled in one specific and well-defined configuration mechanism.

- The gateway should be easily configurable for different applications. It should provide all of the base functionalities including receiving packet data from the network for decoding and processing. Additionally, there should be a system for writing data to the database as well as forwarding it on to the network. These functionalities should not require any changes by the user in the general case. The user should provide code to support the packet structure, data processing, and converting the processed data into formats suitable for writing to the database and the network. To achieve this, we need to separate the code that must be customized, based on the application, from the core of the gateway.

- The gateway functionality should require little to no specific control code on the motes and base station. When any control code is required on the motes it is desirable to keep it very modular in the form of small libraries with well-defined interfaces that can be easily used by any applications. Consequently,

any application related functionality will be clearly separated from the control code of motes in our modules.

- New WSN applications should require little to no modification of the core gateway. The gateway will provide well-defined generic interfaces to support any new application to be added. These well-defined gateway interfaces should be well documented for the deployment of new applications.

4.3.2 Top-down Design of Gateway

We take a top-down design approach. This top-down methodology facilitates in illustrating how our criteria and goals are considered and how they have shaped our design decisions. The discussion of our design is primarily at the architecture level of the gateway. We then present the gateway system design at module level, illustrating interactions between user modules and core modules. Finally, we describe each of the individual modules and interfaces in depth.

4.3.2.1 Gateway Architecture

We focus on the software architecture of our WSN gateway system. Basically, our WSN gateway has dual network stacks. As shown in Fig. 4.1, one network stack (referred to as WSN stack) connects to WSN sink, while the other network stack (i.e., a standard Internet network stack) connects to the Internet. As WSN network layer is terminated at the sink, the sink usually connects to the gateway directly through data link layer. The gateway aims to support a wide range of physical layer and data link layer of the WSN stack via gateway configurations. One unique feature of our gateway architecture is that we have considered a generic transport layer residing in the gateways WSN stack, which can effectively and efficiently support WSN transport layer protocols that may be deployed in WSN motes, such as CDP [40]. Incoming packet streams flow through the gateway dual network stacks as follows. A packet is

first received at the physical layer and the data link layer of the Gateway machine. Then, Packet Dispatcher switches the packet into the relevant transport protocol. The transport layer performs any necessary processing and passes the results to the Data Processor system. The Data Processor will execute any processing that must be done, convert the packet into appropriate formats, and then write it to buffers for the database as well as the TCP server which is a generic component in our gateway system.

Upon receiving data in its buffer, if a connection is available, the TCP Server will send the data out to the back-end server over TCP, through its Internet network stack. Similarly, the database buffer will be written to a database if the database is currently available.

4.3.2.2 User and Core Gateway Spaces

We will now discuss the Gateway from the perspective of separation of user space from the core space. Fig. 4.2 shows the Gateway Core and the Customization Core and their relationship. The explicit separation of these two spaces makes it easier to determine how to design the interface between the core gateway systems and the systems that users may want to customize. Choosing an appropriate boundary for the separation of these spaces will dramatically improve the ease with which a user can customize the gateway for their application.

In the Gateway Core, four primary functions take place. First, the transport layer does processing based on the transport layer used in the WSN stack. Second, packet dispatching occurs. Upon receiving a packet, the dispatcher must determine where to send the packet. To properly achieve this, the gateway provides an abstract type that defines the required functionality of a Data Processor. This type is detailed in the next paragraph. Third, the gateway core contains a Database Handler. Listening to a thread-safe buffer, whenever incoming data become available, the Database Handler connects to the database, writes the information received from the Data Processor,

and logs any errors that might occur. At the same time, the TCP Server will maintain an open server socket and listen to a thread-safe buffer. As data become available on the buffer, the server will check to see if it has a current connection. If there is a connection it reads the entry from the buffer and sends it to connected destination port. It will then log any errors that might occur. It is only required that the gateway supports a single TCP connection, since it is expected that a Management System will be connected for multiple and concurrent users, with which sophisticated data retrievals and management services will be provided for end users. The use of thread-safe buffers is important to the gateway design since processing of data in the gateway requires parallelization of the system in case that data are received faster than they can be processed.

In the Customization Core three processing functions are handled. First, the Customization Core performs any application layer processing that is needed (e.g. converting raw sensor data). Second, it converts the data into a query and buffers it for the Database Handler. Third, it reformats the data into a format suitable for sending over TCP and buffers it on the TCP buffer. This Customization Core system is built on top of three abstract data types provided by the Gateway Core:

- Data Processor: Functionality defined includes initialization of the object by the Packet Dispatcher using the packet and starting the processing task.

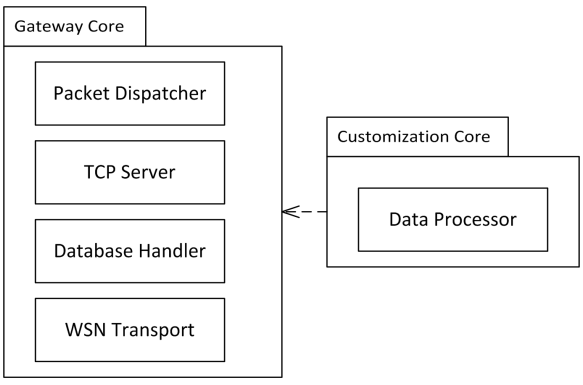


Figure 4.2. Core view of the gateway

- Query: The functionalities defined include creating the query, retrieving the database schema information (used during gateway startup to create tables if needed), and retrieving the query.
- TCP Packet: Functionalities defined include the creation of the packet object and retrieval of the packet in a format suitable for transmission.

4.3.2.3 User and Core Modules

This section presents a more in-depth view of the modules discussed above to detail both the elements of the Customization Core and the Gateway Core. Our discussion of the Customization Core describes how the module can be used to support all of the gateway criteria, while the discussion of the Gateway Core details how the primary functionality of the gateway works and how each module should operate.

In the Customization Core the only defined module is the Data Processor. A users implemented version of this module is responsible for receiving the result of transport layer processing, performing processing of the data in that packet, and formatting it for both the database and the TCP Server. That is, this module is defined as an abstract type for users implementation. This approach supports our design criteria as elaborated as follows.

To meet the gateway criteria of being able to support both static and dynamic packet formats this approach allows a deployment of the gateway to have multiple implementations of the Data Processor with each implementation representing one specific type of packet. The user may use his/her implementation to process packets in whatever way is most useful and necessary.

To meet the criteria of providing a simple interface for connections with a management system and database, this approach enables the gateway to connect to any management system that the user wishes to use, with a well-defined interface.

The Data Processor type helps achieve the goals of both providing an easily configurable system and making no assumptions regarding packet types. To prepare for a

deployment, the user implements an object that conforms to the Data Processors defined interface. This object receives, as input, the appropriate packet object from the transport layer and is able to process that packet in any manner deemed appropriate by the user. The user need not worry about filtering packets because the Gateway Core only sends packets associated with that Data Processor. By providing both a Query abstraction, which provides a method for setting and retrieving a SQL query, and a TCP Packet abstraction, which provides a method for setting and retrieving a TCP packet, the Data Processor enables the user to write data to the database and TCP sockets using any format they require. The Gateway Core removes the necessity of writing the low-level TCP and database code and allows the user to simply design in terms of SQL statements and packets.

The Gateway Core consists of four different modules. The WSN Transport Layer module receives packets and offsets transport layer processing which was not performed at the sink. The Packet Dispatcher module receives packets from the Transport Layer module and forwards them to the appropriate Data Processor. The Database Handler module writes data into the database. Similarly, the TCP Server module is responsible for transferring data over a TCP socket through a standard Internet stack.

The Packet Dispatcher is registered as a listener for all packets in which the Gateway is interested, and is the main control thread of the Gateway. As a packet is received its type is determined. If the packet has a corresponding Data Processor, the dispatcher will create an instance of the Data Processor and pass that packet, the database buffer, and the TCP buffer to it, and then post this Data Processor task for execution.

The Database Handler is a thread that will monitor the database buffer and write the data into a database. It waits for new data on its buffer. As data is received, it connects to the database and writes the query to the database. On the other hand, once a gateway query has been executed, any possible errors would be logged. If there are more gateway queries available it will continue writing them to the database. If

there are no more queries to be written it may disconnect and sleep until a new packet is added to its buffer.

The TCP server is a thread very similar in operation to the Database Handler. Started by the main thread, it establishes a TCP server socket and waits for data to arrive in its buffer. When a packet is available for sending, one of two cases will occur: (1) if no connection has been established to the server socket the packet will be discarded; (2) if a connection either from the user or management system has been established to the server socket, the packet will be written out to the connected client.

4.4 Implementation

For our implementation of the gateway, we decided to use TinyOS [33] as our WSN platform for the initial implementation. TinyOS is widely used and familiar due to our previous work on CDP [40]. TinyOS libraries are primarily offered in C++ and Java. We chose to use Java for our implementation because it is used extensively in our groups WSN network and data management project [39]. Additionally, we are using PostgreSQL for the gateway database. To meet design goal 1 we use the Message Listener interface to implement our Packet Dispatcher. This allows the user to leverage the Message Interface Generator (MIG) provided by TinyOS to obtain a Java representation of the packets being received. To avoid requiring a very specific system on the motes and sink we have used a version of the Base Station application provided in TinyOS 2.1.1 that was modified to use the Collection Tree Protocol (CTP) [19] for upstream communication.

During our implementation of the gateway, some specific aspects have to be addressed in an attempt to effectively achieve all of the design goals that we set. For example, we consider how to handle the asynchronous nature of our gateway design. We also have to determine the best way to handle a users interaction with the system.

The Java Runnable interface and is used to achieve the asynchronous nature of our gateway. The TCP Server, Database Handler, and Data Processors are all

implemented using the Runnable interface. Data Processors execute in a Thread Pool to prevent the creation of too many threads if data are incoming too quickly to process. The Thread Pool limits the number of running threads which eliminates the overhead incurred when too many threads exist. Also, each packet is processed independently at the gateway. To forward information from the Data Processors to the Database Handler and TCP Server we use BlockingQueue.

During our gateway implementation, we made the decision to make a small change to how data flows through the gateway. This is an implementation specific decision that is made to utilize the MIG generated packets more effectively. To make swapping of network stacks easier, we simply merge the WSN transport layer processing into the Data Processor. The Data Processor may be implemented by separate objects for transport layer processing and higher level processing so that a transport layer processor may be provided by the protocol developer and utilized by end users. The flow of the gateway can be seen in the flowchart of the main loop in Fig. 4.3. When the gateway starts the TCP Server and Database Handler threads are started first. Then the gateway starts its main loop. This loop consists of checking for a received packet and passing that packet on to the Data Processor. The Data Processor performs any transport layer processing, sensor data processing, query formatting, and TCP packet formatting and then exits.

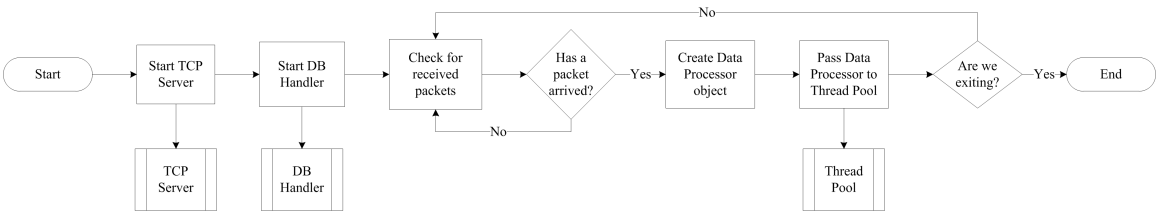


Figure 4.3. Flowchart of the main control thread of the gateway

Property files are used to provide a configuration interface to users of the gateway. We have chosen this configuration file based interface because the gateway is designed

as server software and does not handle any data visualization. This system can also allow Customization Core preferences to be stored in the same central properties file.

4.5 Testing

We have conducted two separate tests of our gateway implementation. Firstly, we have performed a load test. Secondly, we have tested the gateway in several small real-world test-beds.

4.5.1 Load tests

For our load test we aim to test the maximum throughput that our gateway can achieve and maintain. To this end, we designed a special version of the sink application, in which the sink sends packets to the gateway computer repeatedly at the maximum sustainable rate. The testing hardware consists of the sink, a single MICAz [41] mote connected to a MIB520-USB [42], and the gateway computer with a single-core Intel Atom processor (1.6Ghz) and 2 gigabytes of RAM, running a desktop distribution of linux. The MIB520-USBs USB interface is limited to 56.7 kBd due to USB being implemented by using a FTDI FT2232D [43] chip to handle conversion from the MICAz microcontrollers serial output to USB. The test application was setup to send 100,000 packets and was conducted 5 times. During each run of the test every packet was received and processed. This test revealed that, even on modestly powered hardware, our gateway is able to maintain a packet reception rate greater than a MICAz/MIB520-USB based sink could possibly forward data to the gateway machine.

4.5.2 Real-World Test-Bed Tests

In addition to our load test, we performed the field testing of our developed gateway by running an example application in three independent small real-world

test-beds, including the long-running backyard test-bed operated by the University of Pittsburgh [44]. The primary goal of these tests is twofold. Firstly, we want to find and eliminate any potential bugs that could appear only during real-world usage. Secondly, we want to verify that no significant issues arise when our gateway is being used in realistic and uncontrollable conditions. In all test-bed tests the sink node hardware is identical to our load testing and each individual mote in a WSN test-bed is a MICAz mote connected to an MDA300 [45] data acquisition board. The test software consists of the sink, which receives a CTP message on its radio and passes that message on to its USB interface, and the mote application, which samples temperature, humidity, and several analog-to-digital conversion (ADC) channels. Sampling is varied between sampling every 1 minute to sampling every 10 minutes.

Initial tests, primarily attempting to achieve our first goal, were run on a test-bed in our lab, consisting of five motes and a sink. During testing, each time a bug is found and eliminated a new test is started. Prior to testing in the Pittsburgh test-bed two tests were successfully performed for about two weeks.

We have performed testing at the University of Pittsburghs backyard test-bed [44]. Fig. 4.4 shows the layout of the test-bed consisting of one sink and nine motes. The red node is the indoor sink and gateway machine. This test ran until the batteries were depleted, sampling once every 10 minutes. No gateway-related errors were experienced during this test.

In addition, we set up a seven mote indoors test-bed in Indianapolis for some further testing of our gateway system. This test resulted in the discovery and elimination of a minor bug in the gateway.

4.6 Conclusion and Future Work

In this paper, we have explored the design and implementation of a general and configurable WSN gateway software system for data collection. Our presented and developed WSN gateway system can be easily configurable for nearly any WSN

data collection applications. This system is configurable for both WSN research and practice, including the support of any newly developed WSN transport layer by researchers/users, static or dynamic packet formats, and various conversions of sensor data needed for diverse data gathering WSNs. Our developed gateway system has been thoroughly tested and validated via load test and real-world field tests including two indoors WSN test-bed and one outdoors WSN test-beds.

While our WSN gateway has performed well for data collections, we have plans to make our gateway more robust and full-featured in our future work. Firstly, we are working on developing a downstream command system of the gateway. By defining a command format and using callback functions we can develop this system in such a way that the users can provide their own command functionality. Secondly, we plan to extend our current gateway design to enable the user to integrate his/her new Data Processor by requiring only the addition of some information to the configuration file. Finally, we plan to introduce a user-friendly and robust error logging system using a pre-existing logging framework.

Figure 4.4. An illustration of the test-bed used in gateway system testing

5 SYSTEM

In this chapter I describe the composition of the combined data gathering system and some of the customizations that may be done. This is the culmination of the CDP and the general gateway for data collection projects. The two are brought together to build a complete data collection system. It consists of motes sending compressed sensor readings over the network which are then collected, processed and stored at the gateway. Firstly, this chapter details how the system was built. Secondly, it discusses challenges in bringing the system together. Thirdly, it details testing of the system.

5.1 System Design

My system deploys the CDP on the motes and the general gateway for data collection on the gateway. The network view of the system, as seen in Fig. 5.1, shows the software layout, which works as follows:

1. Sensors are sampled at some rate at the application layer.
2. As a set of samples is received, it is sent to the CDP.
3. CDP compresses the data.
4. As a packet is filled CDP sends a packet over the Collection Tree Protocol(CTP).
5. CTP delivers the packet to the sink node.
6. The sink node forwards the packet to the gateway machine.
7. The general gateway decompresses the packet, converts the readings into useful information, saves it to a database and forwards it over a TCP connection.

8. A management system, such as our group's system [39], receives the TCP data, stores it, and presents it in a human readable format for users to analyze.

The entire system can be broken down into one component for the motes and a second component for the gateway machine.

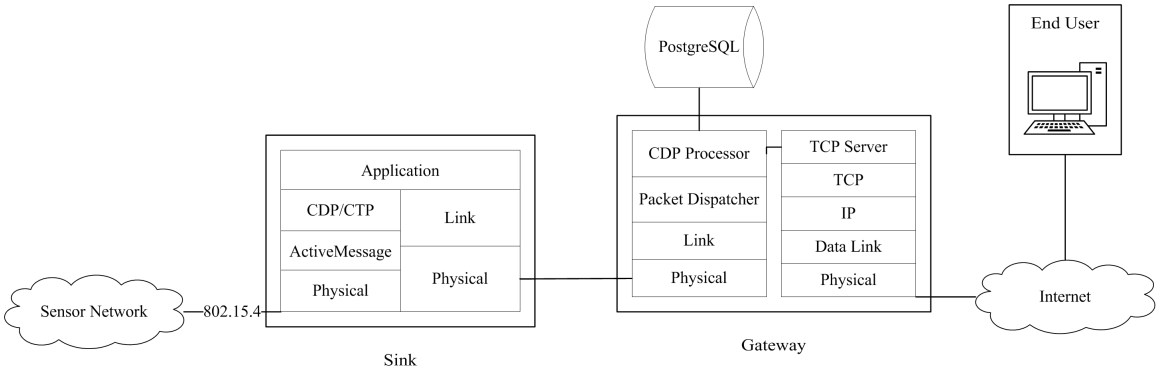


Figure 5.1. A network view of the overall system implementation

5.1.1 Motes

The motes consist of a network stack and an application layer. The network stack is responsible for compressing and transmitting data to the sink. The application layer is responsible for setting up options and passing them on to the network layer.

The process of sampling is done at the application layer and can easily be customized based on a number of variables. Firstly, the sampling rate may be modified as desired. Secondly, the sensors should be sampled at the application layer. It is expected that the user will load the sensor drivers and sample them in the sampling timer event. Thirdly, the CDP streams that correspond with given sets of sensors should be defined. Lastly, any per-node processing of data is done at the application layer. This provides significant configurability for WSN deployments.

In the network stack, CDP's modular design allows for a couple of simple customizations. The compression algorithm may easily be swapped, only requiring that

the user write a new compression module. Additionally, the underlying network stack may easily be swapped by implementing a new network module. Network modules essentially just send data to the sink so any protocol stack with facilities for sending to a sink node may be used.

5.1.2 Gateway

The process of translating data for storage and forwarding to the management system is done at the gateway. The modular design of the general gateway for data collection allows for additional network-wide processing of data to be performed. This can be achieved by modifying the provided Data Processor object described in Sec. 4.3. This allows for a number of customizations.

5.2 Challenges

While this combination of technologies was rather simple, there was one area that required some work. Since the gateway relies on packets being generated by the Message Interface Generator(MIG) of TinyOS there was a slight catch in deployment. Since I was interested in collecting CTP headers as well as data, the reliance on the Packet Dispatcher described in Sec. 4.4 relies on the AM_Message type and CTP data packets typically have the same AM type it was a challenge to differentiate between the CDP setup packets and the CDP data packets. This is currently solved by requiring that a Data Processor implementation verify the received packet type.

5.3 Testing

While significant testing of each individual component has been performed, some additional testing is required to verify the complete system performs as expected. This testing is done in a simple manner with primary focus being verification that everything functions properly. Due to time constraints, it is not practical to verify

energy efficiency or any other functionalities that require long-term testing. This is not a large drawback due to the fact that it was previously shown that energy efficiency will be improved over the same application running without compression.

The test setup is similar to the indoor testing of the gateway. 4 motes are set in my living room with an additional mote being used as a sink node. The sink node is connected to a low-powered machine running a single-core Intel Atom processor with 2 gigabytes of RAM. To keep testing simple, each node has one stream with two data sources. The data sources are sampled every 1 minute and are comprised of previously generated random data. The data is comprised of 100 pairs of values which is sampled in sequence repeatedly. It should be noted that the complete Data Processor gateway class for the CDP test was not written. Rather, the compressed data is written to the database and is decompressed by manually. This was done both due to time constraints as well as the desire to look at the complete packet data to determine if packet error occurred. During testing, all data was received correctly and the initial data-sets were able to be reconstructed without error. While these tests show that the system works reasonably well, it does not show with any certainty that the system is free of defects. Unfortunately, due to time constraints and issues beyond my control, in-depth testing was unable to be performed and must be performed at a later date. More information regarding the issues and further testing strategies may be seen in Ch. 6 and Ch. 3.6.

6 FUTURE WORK

While significant testing and analysis has been done on the individual components of my system, there is still significant work to be done to address some limitations of the work that has been done. The primary limitations that need to be addressed are as follows:

1. Lack of real-world deployment of the system.
2. Further analysis is needed, in addition to real-world deployment, to validate the performance and architecture of the system.
3. While it is shown that CDP is beneficial over no compression, the benefits over application layer and more simple approach to compression at the protocol level should be quantified.

This chapter discusses some of the direction of future work for each component as well as the system as a whole. This provides a clear direction for continuing work on this project to get closer to the final goal of an open source release.

6.1 CDP Future Work

While CDP has been thoroughly simulated and appears to perform well there are a number of tests that will allow a more systematic evaluation of its performance. These will hopefully lead to additional evidence that the system is appropriate and well-designed. The following is the planned future work for the CDP.

- Implement the more naive approach whereby each sensor is compressed in its own packet with the minimal header to provide the same functionality as CDP. While Dr. Yao Liang and myself tested against CTP alone, an analysis of the

performance of CDP versus the naive approach may help to show in what ways the CDP is beneficial as well as in what ways it is limited.

- Deploy the CDP in the real world for a long-term analysis of all benefits of the protocol that we saw in simulation. These include improved energy efficiency, reduced retransmissions and reduced total data transmitted over the network. Additionally, this test will attempt to determine the effect on Packet Error Rate (PER) by setting up a stream that sends predictable data over the network and comparing it to an uncompressed version of the same data sent over the CTP.

- Analyze the reliability and performance of the CDP using a model such as that proposed in [46]. In the case of this model I can verify performance and reliability of CDP by using Discrete Time Markov Chains (DTMCs) to model and analyze the protocol implementation. In the case of the CDP this analysis may require some significant modification of any analysis methods to make them appropriate for analysis of a nesC program. This analysis will allow me to give some more concrete analyses of the benefits of CDP, especially over a more naive approach and may allow for an improvement of the CDP design.

6.2 Gateway Future Work

The gateway designed by Dr. Yao Liang and myself has had significant small scale real-world test deployments as well as a single load test that shows that even a low-powered computer can receive data at the maximum rate a MIB520 sink node can forward data to it. While this is a significant start there are a number of analyses that could either verify the performance of the gateway or provide specific tips on where the design may be improved. These are in addition to the future works discussed in 4.6.

- Perform more in-depth load testing of the gateway to estimate the maximum complexity of a Data Processor implementation. This will primarily be done by

determining the maximum number of operations on our low-end computer that may be performed in a Data Processor. Additionally, an attempt will be made to find a specific mathematical formula to estimate the maximum complexity of a Data Processor implementation given the gateway computer hardware as well as the variables in the gateway application (e.g., maximum concurrent Data Processors and number of SQL insert statement that will be batch written to the database at a time).

- Perform load testing using either a simulated sink or an actual sink that can send data at a faster rate than the MICAz over the MIB520. One possible solution is the MIB600 which interfaces with a gateway machine over ethernet and should be able to forward data at the maximum 802.15.4 rate. This will either provide me with a maximum possible rate that the gateway may receive data at or show that the gateway is suitable for any 802.15.4 network.

- Perform similar analyses to those in 6.2 to analyze our gateway based on its architecture. This approach will also likely require modification to an existing model for analyses due to the fact that it is difficult to analyze concurrent systems.

6.3 System Future Work

On top of the simple testing that has been performed and testing of the components proposed previously in this chapter there are a number of improvements that need to be made to the system. First and foremost is that a complete Data Processor for CDP be implemented. Beyond this, a more thorough test of the system must be performed to verify that all data is received and processed correctly. Once this has been reasonably shown, the following step will be to find users that are interested in deploying sensor networks that are willing to detail the use of the system and determine whether the primary goal of ease of use was achieved in their opinion. This will

likely be a mostly qualitative analysis but if users can be found that will deploy a system using both my work as well as XServe some quantitative analysis may be done regarding the amount of time required to learn the system and deploy their specific network. Additionally, as modularity is a key design element of this system some additional measurements on the modularity is warranted using common metrics such as separation of concerns, cohesion and coupling. This analysis will attempt to make my modular design goals explicit and measurable.

7 CONCLUDING REMARKS

This thesis has presented a complete system for compressed data collection in WSNs. This system is needed because current complete solutions for data gathering in WSNs is mostly commercial and not easily adaptable for varying real-world requirements. The overall goal of this system is to gather and store sensor data from WSN nodes. The motes are built on a network stack consisting of the CDP running on top of the CTP. At the gateway, the general gateway system for wireless sensor networks, developed by Dr. Yao Liang and myself, is running to decompress, store and forward received data.

The CDP is a protocol that was developed to allow for compression and transmission of sensor data that has been split up into logical streams. CDP also allows for simple swapping of compression algorithms so that a user may choose the most appropriate one for their scenario. In its current state I have chosen to use the Generalized Predictive Coding(GPC) framework developed by Dr. Yao Liang [28]. This compression is quite effective in compressing temporal data and is a good choice for the type of sensors typically used in WSNs. It is shown that CDP is both effective at compressing data as well as reducing energy usage.

The general gateway for wireless sensor networks is a highly customizable gateway for data gathering WSNs. It has been shown that the design of this gateway is valuable for nearly any data gathering WSN. Since this gateway focuses on offsetting WSN transport layer processing, it is a good choice for use with a protocol such as CDP. Doing decompression on the gateway eliminates the possibility that the sink will be overcome by processing. The gateway has been shown to be stable in both load tests as well as in several real-world test-beds.

I show how the CDP and general gateway can be built into a system for data gathering. While this combination was relatively simple due to the highly customiz-

able and user-focused design of both systems there was one small complication due to the Message Interface Generator(MIG) of TinyOS. Additionally, I detail a testing setup and methodology for analysis of testing the system as a whole. Data supports that this system is both reliable as well as more energy-efficient than a more basic solution.

Unfortunately, due to time constraints and a problem with a hardware driver in TinyOS 2.x, real-world testing of the system was not able to be performed. Plans have been made to address this along with several other limitations of this work mentioned in the Future Work section.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Kim Sukun, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 254–263, 2007.

[2] Jeongyeup Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri. A wireless sensor network for structural health monitoring: Performance and experience. In *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pages 1–10, May 2005.

[3] SensorScope. *PDG 2008 Deployment*, 2008 (accessed July 2011). `http://sensorscope.epfl.ch`.

[4] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the ACM Symposium on Operating System Design and Implementation*, pages 381–396. USENIX Association, 2006.

[5] Wen-Zhan Song, Renjie Huang, Mingsen Xu, B. A. Shirazi, and R. LaHusen. Design and deployment of sensor network for real-time high-fidelity volcano monitoring. *Parallel and Distributed Systems, IEEE Transactions on*, 21(11):1658–1674, 2010.

[6] Saima Zafar. Article: A survey of transport layer protocols for wireless sensor networks. *International Journal of Computer Applications*, 33(1):44–50, November 2011. Published by Foundation of Computer Science, New York, USA.

[7] F.K. Shaikh, A. Khelil, A. Ali, and N. Suri. Trccit: Tunable reliability with congestion control for information transport in wireless sensor networks. In *Wireless Internet Conference (WICON), 2010 The 5th Annual ICST*, pages 1–9, March 2010.

[8] Jeongyeup Paek and Ramesh Govindan. Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks. *ACM Trans. Sen. Netw.*, 7(3):20:1–20:45, October 2010.

[9] Gitanjali Shinde, Swati Joshi, and Shami Jhodge. Diversity coded directed diffusion for wsn. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pages 355–359, New York, NY, USA, 2012. ACM.

[10] Özgür B. Akan and Ian F. Akyildiz. Event-to-sink reliable transport in wireless sensor networks. *IEEE/ACM Trans. Netw.*, 13(5):1003–1016, October 2005.

[11] Sukun Kim, Rodrigo Fonseca, Prabal Dutta, Arsalan Tavakoli, David Culler, Philip Levis, Scott Shenker, and Ion Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *Proceedings of the Fifth International Conference on Embedded Networked Sensor Systems*, pages 351–365. ACM, 2007.

[12] L. Steenkamp, S. Kaplan, and R.H. Wilkinson. Wireless sensor network gateway. In *AFRICON, 2009. AFRICON '09.*, pages 1–6, September 2009.

[13] Lili Wu, J. Riihijarvi, and P. Mahonen. A modular wireless sensor network gateway design. In *Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on*, pages 882–886, August 2007.

[14] M.-E. Raluca, M.-E. Razvan, and A. Terzis. Gateway design for data gathering sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, pages 296–304, June 2008.

[15] Hsueh-Chun Lin, Yiao-Chiang Kan, and Yao-Ming Hong. The comprehensive gateway model for diverse environmental monitoring upon wireless sensor network. *Sensors Journal, IEEE*, 11(5):1293–1303, May 2011.

[16] Haoran Chen, Dahai Han, and Hongping Feng. Research on standard architecture of sensor networks gateway. In *Advanced Intelligence and Awarenss Internet (AIAI 2010), 2010 International Conference on*, pages 287–290, October 2010.

[17] Juhan Kim and Dooho Choi. esgate: Secure embedded gateway system for a wireless sensor network. In *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*, pages 1–4, April 2008.

[18] John Suh and Shana Jacob. Distributed sensory systems and developer platforms from crossbow technology. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 319–319, New York, NY, USA, 2005. ACM.

[19] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the Seventh ACM Conference on Embedded Network Sensor Systems*, pages 1–14. ACM, 2009.

[20] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, P. Levis, and A. Woo. The collection tree protocol, 2007. available at `http://www.tinyos.net`.

[21] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating congestion in wireless sensor networks. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems*, pages 134–147. ACM, 2004.

[22] Fei Huang and Yao Liang. Towards energy optimization in environmental wireless sensor networks for lossless and reliable data gathering. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–6, 2007.

[23] Francesco Marcelloni and Massimo Vecchio. An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks. *Comput. J.*, 52(8):969–987, 2009.

[24] Yao Liang and Wei Peng. Minimizing energy consumptions in wireless sensor networks via two-modal transmission. *SIGCOMM Comput. Commun. Rev.*, 40(1):12–18, 2010.

[25] Christopher M. Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the Fourth ACM International Conference on Embedded Networked Sensor Systems*, pages 265–278. ACM, 2006.

[26] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[27] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[28] Yao Liang. Efficient temporal compression in wireless sensor networks. In *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, pages 466–474, 2011.

[29] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow, and D. Estrin. Lightweight temporal compression of microclimate datasets [wireless sensor networks]. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 516–524, 2004.

[30] Chong Liu, Kui Wu, and Jian Pei. An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):1010–1023, 2007.

[31] Tossaporn Srisooksai, Kamol Keamarungsi, Poonlap Lamsrichan, and Kiyomichi Araki. Practical data compression in wireless sensor networks: A survey. *J. Netw. Comput. Appl.*, 35(1):37–59, 2012.

[32] Jukka Teuhola. A compression method for clustered bit-vectors. *Inf. Process. Lett.*, 7(6):308–311, 1978.

[33] The TinyOS 2.x Working Group. Tinyos 2.0. In *Proceedings of the Third International Conference on Embedded Networked Sensor Systems*, pages 320–320. ACM, 2005.

[34] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, pages 1–11. ACM, 2003.

[35] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, pages 126–137. ACM, 2003.

[36] Enrico Perla, Art Ó Catháin, Ricardo Simon Carbajo, Meriel Huggard, and Ciarán Mc Goldrick. PowerTOSSIM z: realistic energy modelling for wireless sensor network environments. In *Proceedings of the Third ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, pages 35–42. ACM, 2008.

[37] HyungJune Lee, A. Cerpa, and P. Levis. Improving wireless simulation through noise modeling. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 21–30, 2007.

[38] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the Second International Converence on Embedded Networked Sensor Systems*, pages 188–200. ACM, 2004.

[39] Miguel Navarro, Diviyansh Bhatnagar, and Yao Liang. An integrated network and data management system for heterogeneous wsns. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 819–824, 2011.

[40] Newlyn Erratt and Yao Liang. Compressed data-stream protocol: an energy-efficient compressed data-stream protocol for wireless sensor networks. *Communications, IET*, 5:2673–2683, 2011.

[41] Crossbow Technology. MICAz wireless measurement system data sheet.

[42] Crossbow Technology. MIB520 usb interface board data sheet.

[43] Future Technology Devices International ltd. FT2232D dual ISN to serial UART/FIFO IC data sheet. 2006.

[44] Xu Liang, Tyler W. Davis, Miguel Navarro, Diviyansh Bhatnagar, and Yao Liang. An experimental study of wsn power efficiency: Micaz networks with xmesh. *International Journal of Distributed Sensor Networks*, 2012:14, 2012.

[45] Crossbow Technology. MDA300 data acquisition board data sheet.

[46] Vibhu Saujanya Sharma and Kishor S. Trivedi. Architecture based analysis of performance, reliability and security of software systems. In *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, pages 217–227, New York, NY, USA, 2005. ACM.