A Distributed Sensor Network Management System

by

Anders Hustvedt

A thesis submitted to the
Faculty of University of Colorado in partial
fulfillment of the requirements for the degree of
Bachelor of Science degree in Computer Science
Department of Computer Science
2006

This thesis entitled:
A Distributed Sensor Network Management System
written by Anders Hustvedt
has been approved for the Department of Computer Science

Richard Han

Prof. John Bennett

Prof. Mishra Shivkavant

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Anders Hustvedt, (BS. Computer Science)

A Distributed Sensor Network Management System

Thesis directed by Prof. Richard Han

SWARMS is a system for controlling a large number of sensor nodes in a wired sensor network.

Contents

Chapter

1 Introduction 1

1.1 Contributions of this thesis 2

2 Related Work 3

2.1 Programming nodes 3

2.2 Motelab 3

3 Goals and Requirements 4

3.1 Program many nodes at a time. 4

3.2 Have a consistent naming scheme. 4

3.3 Log data into a database 5

3.4 Have a time for each log message. 5

3.5 Allow easy access to the log messages from prior jobs 5

3.6 Put old log messages into a "Snapshot" 5

3.7 Performance and scaling 6

4 Architecture of the system from a performance prospective 7

4.1 Architecture Overview 7

4.1.1 Node 8

4.1.2 Node-mate 9

4.1.3 Cluster Controller 9
4.1.4 Head Server 10
4.1.5 Database 10
4.1.6 Web Interface 11
4.1.7 Clients 11
4.1.8 Logical groupings of the nodes 12

5 Issues with supporting Hardware/software 13
5.1 Telos Nodes 13

6 Sample Applications 14
6.1 Sound Recorder 14
6.1.1 Overview 14
6.1.2 On the nodes 14
6.1.3 How this changed the testbed 14
6.2 Performance Testing Suite 15
6.2.1 Overview 15
6.2.2 On the nodes 15
6.2.3 The client 16
6.2.4 How this changed the testbed 16

7 Performance Testing Results 17
7.1 Assumptions 17
7.2 Packets per second 18
7.3 Percent Lost Packets per second 19
7.4 Latency from packets 20
7.5 Latency from bandwidth 21
7.6 Conclusion 21

8 Future work 23

8.1 Authentication for clients 23

8.2 Security 23

8.2.1 User access limitations 23

8.2.2 Protocol security 23

8.3 Documentation 24

8.4 More Robustness 24

Bibliography 25

Figures

Figure

1.1 This is a normal sensor node, specifically a MoteIV Tmote Sky[1] with sensor package. 1

4.1 Architecture of the testbed management system. 8

7.1 Input packets per second versus output packets per second . . 18

7.2 Input packets per second versus output packets per second . . 19

7.3 Latency as a function of the input packets per second 20

7.4 Latency as a function of the input bandwidth 21

Chapter 1

Introduction

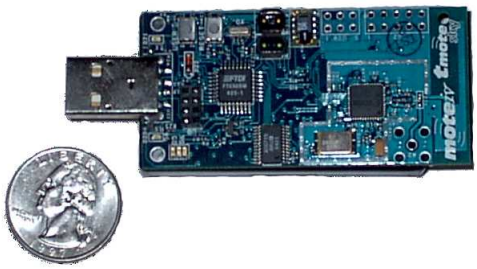

Figure 1.1: This is a normal sensor node, specifically a MoteIV Tmote Sky[1] with sensor package.

Sensor nodes are small microcomputers with sensors and a radio. Volatile and non-volatile storage is usually less than a mebibyte, clock speed is usually less than 10 megahertz, all powered by 2 AA batteries. The normal interfaces are a serial line or over the radio. The only on board display is 3 LEDs, and some models have a button.

Individually, a sensor node doesn't have much utility, since there is only so much data that can fit into the on board storage. Usually, sensor nodes are arranged in a network where every node is connected through the network to a node that is connected to a computer, using wireless links between nodes. The node connected to the computer is called the "Base Station," and is the connection from the computer to the sensor network.

1.1 Contributions of this thesis

This thesis is a performance analysis of a sensor network management system that another student at CU and I developed over the course of the year.

Some of the areas that I researched related to the SWARMS system was a sound recorder application where the nodes would record sound from the microphone on the sensor board, and send the sound samples over the serial line. I then made an application that would get the sound from all of the nodes connected to SWARMS, and create a spectral density graph while playing back the audio. The high data rate of the sound application and its impact on the SWARMS system made it apparent that a performance testing suite would be an important part of the SWARMS system.

The performance suite is a client that is designed to find out what the limits of the SWARS system are, including bandwidth and latency.

Chapter 2

Related Work

2.1 Programming nodes

Previously the primary interface to the nodes, for programming and receiving the data back was a terminal. This requires an account on the computer, with the terminals running locally or remotely via ssh. For getting data from more than one node, one terminal is used per node. Also, if a bunch of data comes in from several nodes at the same time, it can be hard to determine the relative order of the data, and the absolute time of the data.

2.2 Motelab

Motelab[3] is a testbed application that is under development at Harvard. It uses a batch programming model, where jobs start at a specific time, then run for a specific time. Instead of the output being real-time, it is available at the end of the job, so no runtime diagnostics or changes can be made.

Chapter 3

Goals and Requirements

3.1 Program many nodes at a time.

Currently, it is hard to program more than a few nodes at a time. If the nodes use a USB interface to the computer, then that adds another complexity, since it is impossible for a single computer to have more than 100 USB devices. If you want to program more than 100 nodes at a time using the previous methods, the code image would have to be on both computers, then run the programming utilities once per node on each computer.

3.2 Have a consistent naming scheme.

One thing that scientific experiments need is repeatability. If the user of a testbed is trying to run a test where the base station is in the corner of a grid, then the testbed management software needs to make sure that the nodes have consistent names from trial to trial so that the data can be compared. By default, in Linux USB sensor nodes are assigned a name based on the order that they are initialized in, which is almost random if the USB system gets reinitialized. However, the Telos nodes have a serial number that can be used in the assignment of the device name, allowing for a consistent naming scheme.

3.3 Log data into a database

If all data is kept in a central place, and not just put in to the buffer of a terminal, then it is possible to do historical analysis of data. This

3.4 Have a time for each log message.

Having an accurate time stamp for all data is invaluable, since this allows comparisons between other nodes, or with other experiments that are run simultaneously in remote places.

3.5 Allow easy access to the log messages from prior jobs

Once a trial or job has been run, unless the job was a simple functionality check, the data will need to be post-processed. The data should be in a format that is convenient to use.

3.6 Put old log messages into a "Snapshot"

Insertions into a database table get significantly slower after 300,000 rows in a table, so we clear out the log table to make insertions faster. Instead of just deleting the old logs, snapshots provide a convenient way of archiving the messages for later retrieval, while allowing the current job to have a clean log table to use.

Since the snapshots are currently an XML file, this provides an easy way for data from several testbeds to be combined. Currently the only filter from XML is to a comma separated spreadsheet file.

3.7 Performance and scaling

The system should scale in terms of number of nodes, and the rate that the nodes send data. Due to addressing limits of USB, the system needs to be able to use more than one computer to host sensor nodes.

Chapter 4

Architecture of the system from a performance prospective

Up to now we have been working on a generic test bed, with the work shared between me and another student at CU.

4.1 Architecture Overview

Our architecture is designed to be scalable, as a result of the limitations of the USB addressing ability of a computer. A "Node Cluster" is a computer with nodes attached to it. The parts above the cluster controller could be one computer, or each part could be on a separate computer, but it is the data management part of the testbed. Rails is the web development environment that we use. All of the daemons and the web interface are written in the same language, Ruby[2], which makes connections between them easy, since ruby has built in network transparency.

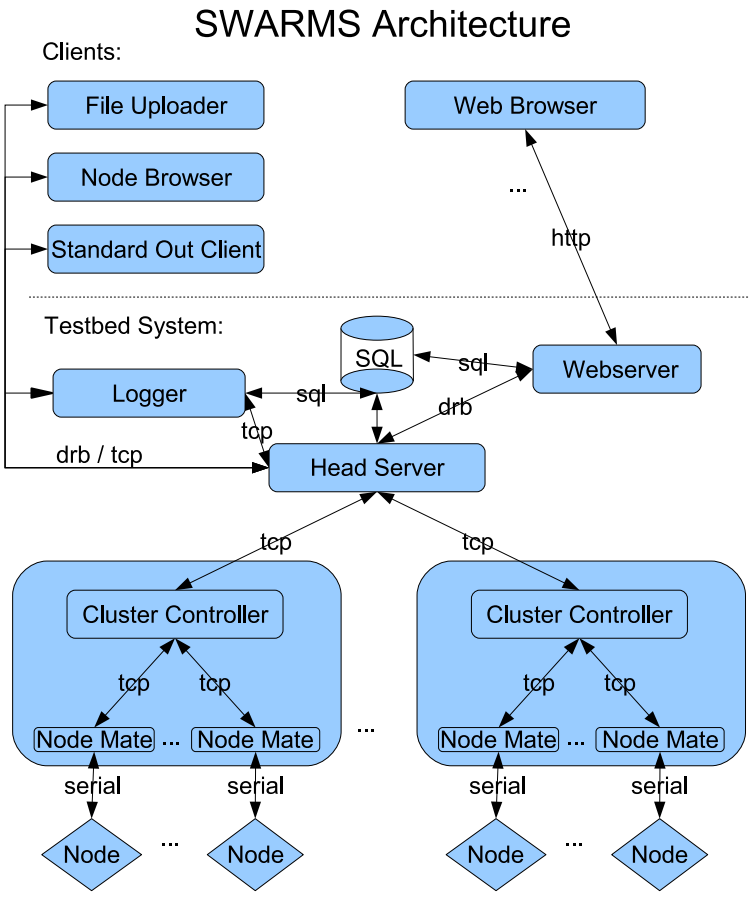

Figure 4.1: Architecture of the testbed management system.

4.1.1 Node

Nodes are directly connected to a computer, either through a USB or serial connection. SWARMS currently accesses the nodes through the /dev file system, so in theory, any device that create a file in that file system that is readable and writable should be supported, with just a minor change in the node-mate to handle the device/embedded operating system specifics.

4.1.2 Node-mate

4.1.2.1 Overview

The node-mate receives commands from the cluster controller, along with the location on the local file system where the code image to be loaded is. Messages are received from the node, and then forwarded to the cluster controller over an established TCP connection.

This is also where hardware specific information about the parsing of data coming from the node is handled, along with how to program a node. Programming a node might involve a program that is distributed by the vendor, and that is invoked here.

As the name implies every node-mate that is run is responsible for the communication to only one node.

4.1.2.2 Performance implications

Since the node-mate does all of the computation related to the specific sensor node including access to the hardware interface. This will scale well since there is a limited number that can be run on a single computer due to USB address limitations. Since the data is sent directly to the cluster controller running on the same computer, the communication delays are small.

4.1.3 Cluster Controller

The cluster controller is responsible for controlling all of the node-mates on a computer, and for receiving commands and code images from the head server. The cluster controller is also used to see what nodes are attached to a computer, by scanning the /dev file system with a wildcard search.

Only one cluster controller is run on a computer, and only if that computer has nodes attached to it.

4.1.3.1 Performance implications

The cluster controller has a single TCP connection to the head server, which usually will go over a network connection, which limits the number of connections that the head server has to process.

4.1.4 Head Server

The head server responds to commands from the web interface and clients, such as "Program Nodes" or "Listen to Nodes," and getting the necessary information out to do that. Commands can also come in though a client besides the web site. The head server also spawns a process that receives the log messages from the node-mate, and inserts them into the database.

The head server is the only daemon in the daemon tree that has a connection to the database, and is responsible for getting data from the database to the other daemons, as they need it.

4.1.4.1 Performance implications

This the primary bottleneck, where all of the messages from all of the nodes are distributed to all of the clients connected to the head server. To improve performance of the system, this is the part of the system that should be optimized first.

4.1.5 Database

The database holds all of the code images, job assignments, and the snapshots. Since Ruby on Rails provided a database abstraction, an effort

has been made to code just to that abstraction, so that the testbed can be database independent. This has been fairly successful, with one deployment of the testbed using PostgreSQL, and the rest using MySQL.

4.1.5.1 Performance implications

Since the database can be run on a dedicated computer, with a large number of processors and ram, the impact of the database on the rest of the system can be made very small. For a running job on the testbed system, the database is just where the logger stores the log messages. The database is a terminating point in the flow of data, and so doesn't have any dependendent parts. The website does use the data in the database, but it is a pull mechanism.

4.1.6 Web Interface

The web interface is the primary method of using the testbed software. Jobs are started here, and data is view and exported from here. Currently, there is the ability to view data as it is coming in, either for the entire testbed, or for a specific set of nodes.

4.1.7 Clients

Clients can connect to the head server and get the log messages in real time as they are sent from the nodes. Clients can issue commands to the nodes. Clients can control the head server to start/stop jobs and database logging.

Currently, the database logger is implemented as a standard client, with the only special features are that it is spawned by the head server, and that it has access to the database directly.

4.1.8 Logical groupings of the nodes

For our system, nodes have a type, which determines the specific node-mate that will be used for that node. Then nodes are grouped together into "Groups," which is just a set of nodes. Groups are used for viewing the data, and for programming. A job is a set of code images that get loaded onto a group, and a job can be started, so that the code images get loaded onto the corresponding groups. For viewing the data as it comes in, there is the "live view," where the last message per node for all nodes is refreshed at a specific interval.

Chapter 5

Issues with supporting Hardware/software

5.1 Telos Nodes

The usb to serial converter on the Telos nodes needs to be read before the buffer in the chip is full, otherwise the node will cease to send data until it is reset. This means that if the node-mate takes to long to read from the serial port, the node will be lost.

Chapter 6

Sample Applications

6.1 Sound Recorder

6.1.1 Overview

Sound recorded on the sensor node is sent through the testbed to a client program that plays back the sound, along with displaying a spectral density graph.

6.1.2 On the nodes

For the nodes, they just sampled the ADC connected to the microphone, and sent that over the serial line as fast as possible. With the ADC running in programmed IO mode, this worked at 3,000Hz, and was intelligible. However, problems with getting the ADC to work in auto-sampling mode prevented sound recording at higher frequencies.

6.1.3 How this changed the testbed

This motivated turning off the database, so that data isn't logged into the database. Instead, a client could save the data as a set of audio files, or just play them back and not record the data at all. Also, this motivated the performance client, since the sound application was sending data as fast as the node could get it over the serial line.

6.2 Performance Testing Suite

6.2.1 Overview

The performance suite a client that measures throughput from the system, and latency from the client, to the node, and back to the client. Since this client specifies the data load that the nodes should be sending out, the difference between what it recieves and what it should recieve can easily be calculated to get the packet loss. Since the data load from the nodes goes from light to heavy several times, the recovery of the system is also tested.

The performance suite is an example of a script that sends command to the testbed and the nodes, and then receives the output from the nodes.

Variables used:

Database: On, off

Packet Size: 1, 2, 4, 8, 16, 32, 64 bytes

Delay: 1, 4, 5, 7, 10, 25, 50, 100, 150, 200 miliseconds

Samples per data point: 10 samples

Time per data point: 1 second

For each of the input variables, (databse state, packet size, and delay between packets), the state is set, and then samples are taken, each "Time per data point" long, and repeated "Samples per data point" times.

6.2.2 On the nodes

The program on the nodes is really simple: it just sends out a specific size packet over the serial line at a specific rate. The rate and size are configured by input over the serial line, so that they don't have to be reprogrammed to

try a different setting. Also, a special packet can be sent to the node that causes an instant "ACK" packet to be sent, for latency measurements.

6.2.3 The client

The client is a simple ruby script that goes through each combination of values and runs through the tests, gathering statistics about the rate that packets come back, and the latency to each node.

6.2.4 How this changed the testbed

Just getting the performance client to work was dificult. The commands to the nodes had to go through the system correctly, which exposed a few concurrency problems with reading and writing to the serial device. Beyond that, quite a few changes were made to reduce the packet loss, and to aid in the recovery of the system after the nodes stop sending large number of packets.

The performance suite also is a complete system validation for SWARMS, where communication from the client down to the sensor node and back is tested, and the resulting data can be used to determine the packet losses, and the round trip time latency. Just getting the performance testing suite to run was a daunting task, as quite a few parts of the SWARMS system worked reasonably well under light loads, but crashed under heavy loads.

Chapter 7

Performance Testing Results

This data is from 10 nodes connected to a cluster controller on one computer, with the head server on a separate computer. The MySQL database is on the same computer as the head server. The web server is a separate computer as well, but that isn't used during the execution of the performance testing.

7.1 Assumptions

This data assumes that the rate of packets and the size of the packets doesn't change during each test.

7.2 Packets per second

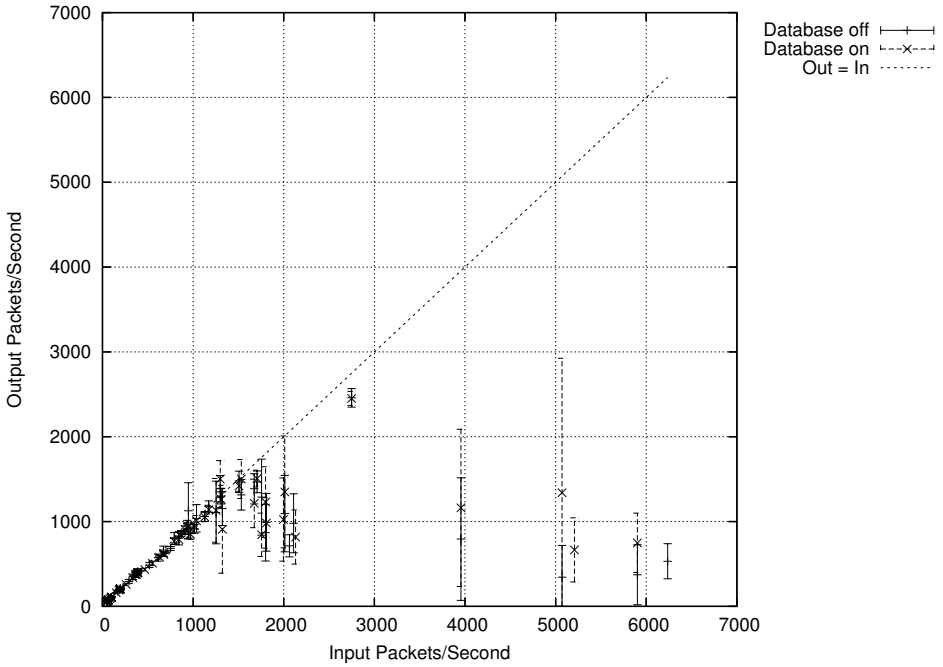

Figure 7.1: Input packets per second versus output packets per second

This graph shows the input packets sent in to SWARMS over a serial link. The increasing variance of the packes after 1,000 packets/second, and the deviation from the y=x line mean that SWARMS has problems handling more than 1,000 packets per second, but that it will do a best-effort transport of the messages. The data points beyond 3,000 packets per second show that the system is capable of not failing even when it doesn't have the processing power to handle the packets coming in from the nodes.

These data points were not taken in order, so some of the points on the left show the system after the points on the right, showing that the system also recovers after a load higher than the system can handle.

7.3 Percent Lost Packets per second

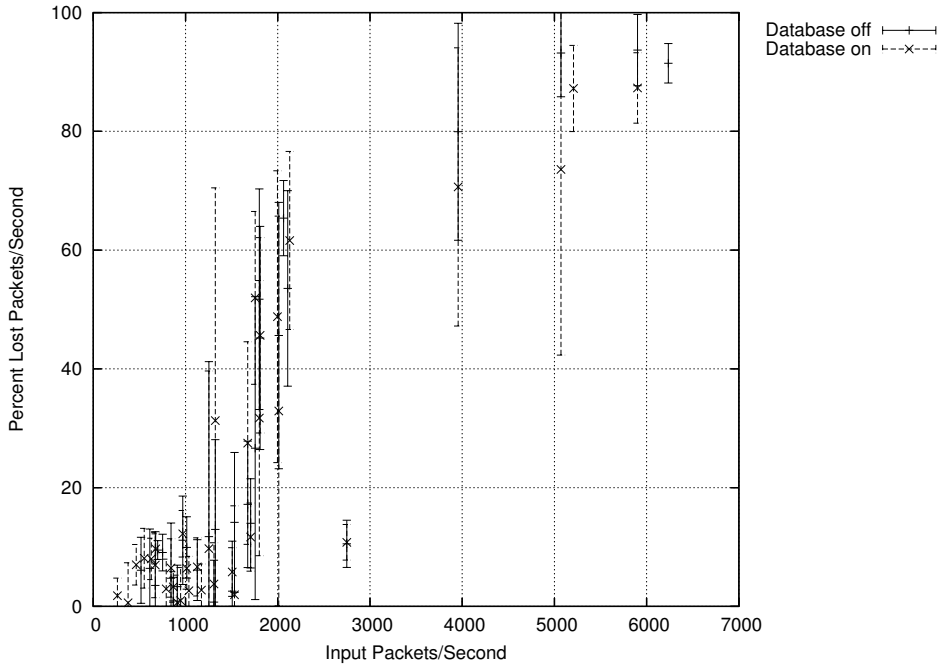

Figure 7.2: Input packets per second versus output packets per second

This shows the number of packets that are lost, as a function of the packets/second that all of the nodes togeather are sending. If the system were perfect, this would be a flat line at 0. This is just the difference between the output packets per second and the line showing input=output packets per second from the previous graph.

7.4 Latency from packets

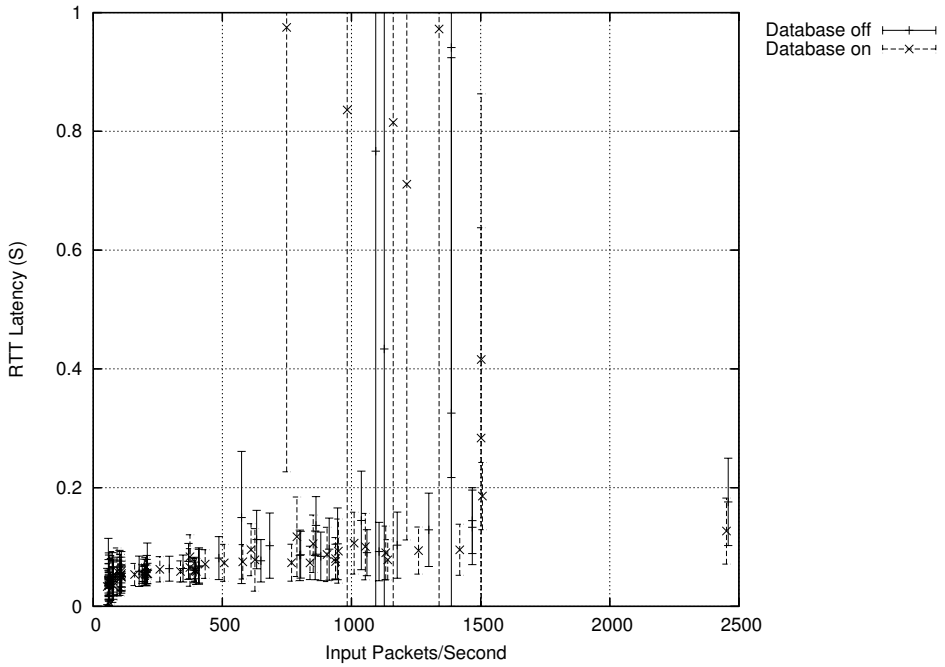

Figure 7.3: Latency as a function of the input packets per second

This shows that at low levels of system usage, up to about 100 packets/second, the latency is below 0.04 seconds. However, above about 200 packets/second, the latency goes to about 0.06 seconds. The range for the x-axis is small because the excessive packet loss interefered with the latency measurments.

7.5 Latency from bandwidth

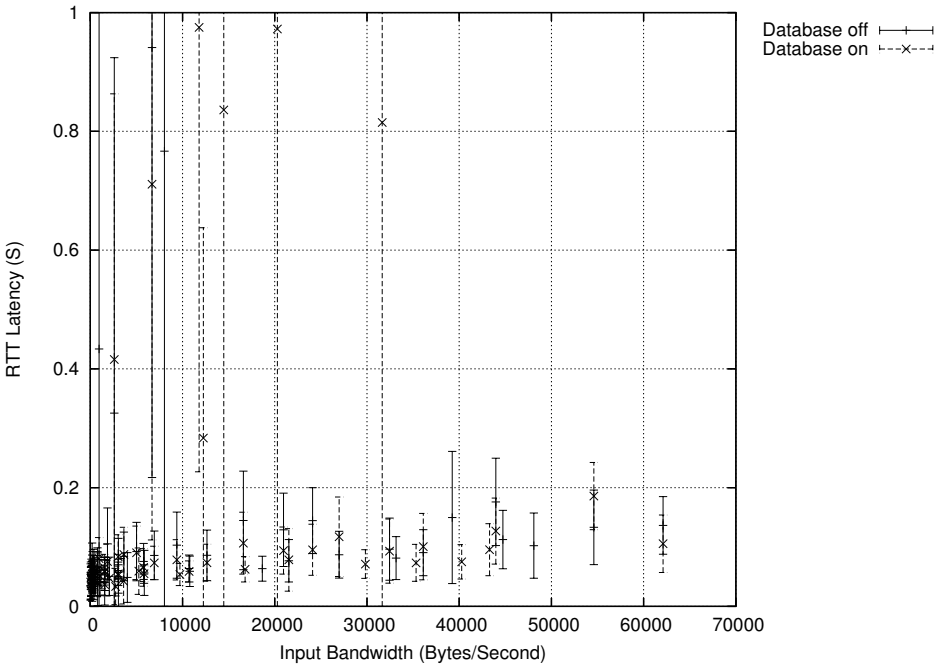

Figure 7.4: Latency as a function of the input bandwidth

Since a lot of the computation in the testbed is on the basis of packets, bandwidth would have more of an effect if the data is going over a network link. Even with the nodes being on another computer, aross a single 100BaseT link, bandwidth doesn't play a significant role in the latency of the system.

This means that if a certain amount of data needs to be sent, it is best to use packets that are as big as possible. That also has the added benefit of reducing the percentage of overhead on the serial line out of the sensor node.

7.6 Conclusion

The system performed well up to a threshold, but after that threshold an excessive number of packets get lost. This could be improved by making

the head server more efficent, or even making the message redirection part be in a more efficent language.

Chapter 8

Future work

Currently we are planning on getting SWARMS ready for being released as a software package that users outside of CU can benefit from.

8.1 Authentication for clients

Currently, clients are not authenticated for a correct reservation, or to authenticate as a valid user of the testbed. this needs to be changed so that the clients have the same level of authentication as the users of the website.

8.2 Security

8.2.1 User access limitations

One thing that should be considered is user level speration, where a user doesn't have access to the jobs or snapshots of another user. This would be important if users outside the devlopment group are given access to the system.

8.2.2 Protocol security

Currently the protocol and communication between the parts of the system are not encrypted or authenticated. Cluster controllers need to authen-

ticate to the head server, and node mates need to authenticate to the cluster controllers.

8.3 Documentation

Currently, there isn't any documentation that covers the system and how to install and use it.

8.4 More Robustness

The system still has bugs left, but the Performance Testing Suite is making some of them more obvious.

Bibliography

[1] Moteiv Corporation. Tmote sky wireless sensor node. http://www.moteiv.com.

[2] Yukihiro Matsumoto. http://ruby-lang.org/en, 2006.

[3] Matt Welsh. Motelab. http://motelab.eecs.harvard.edu/, 2006.