MoteNet: A Wireless Sensor Network Testbed

BTech Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Bachelor of Technology

by

K Naresh Kumar
Roll No: 05005029

under the guidance of

Prof. Purushotam Kulkarni

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai
Contents

1 Introduction 1
  1.1 Wireless Sensor Networks .............................................. 1
    1.1.1 Sensor Node .................................................. 1
    1.1.2 Characteristics of WSN ........................................ 2
    1.1.3 Operating Systems ............................................. 2
  1.2 WSN Test beds ...................................................... 3
    1.2.1 Motivation ..................................................... 4
    1.2.2 Functionalities provided by MoteNet ......................... 4

2 Overview 5

3 Related Work 5
  3.1 Available Testbeds ................................................ 6
  3.2 Quality of Testbeds ............................................... 7

4 Technical Details 8
  4.1 Hardware Details ................................................ 8
  4.2 Software Components ............................................. 10
    4.2.1 My SQL Database Backend .................................. 10
    4.2.2 User Interface ................................................ 11
    4.2.3 Execution of the job ........................................ 12
    4.2.4 Status of motes .............................................. 14
    4.2.5 Expanding the Testbed ..................................... 15
  4.3 Summary ........................................................... 15

5 Experiments 16

6 Future Work And Conclusion 17
Abstract

With the growth in the research in wireless sensor networks the need for deployment and debugging sensor applications on real sensor network bed has grown. It is very difficult for one to set up network manually and reprogram the nodes for every application. Data gathering is another challenging task in such applications. Although there are tools available to evaluate sensor network, only real testbed can provide information to understand resource limitations, communication loss etc. MoteNet is permanently deployed sensor network nodes connected through a base station which reprograms nodes to run user application and collect the data for the user. User uploads and schedules their job through an interface. MoteNet helps wireless sensor network researchers accelerate their research. In this report we will go through details of the MoteNet.
1 Introduction

1.1 Wireless Sensor Networks

Advances in the science and technology are interdependent. There is tremendous increase in the semiconductor technologies. According to Moores Law number of transistors on a chip which also implies processing and storage capacity of the chips doubles every year. Because of this it became easy and cheap to developed device which uses radio and sensors to collect information at a given place (using sensors) and communicate with other devices (using radio) on the network. Thus wireless sensor network evolved as an emerging area with wide variety applications. A Wireless Sensor Network (WSN) consists of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations. Development of wireless sensor networks are actually motivated by military though today they are used in many fields.

1.1.1 Sensor Node

Each device in a WSN is called mote or sensor node it is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network. Typical architecture of a mote is given in the figure below [1].

![Architecture of Sensor Node](image)

The main challenge in hardware of WSN is to produce low cost and tiny sensor nodes. Major components of a mote are micro-controller, transceiver, external memory, power source and sensor nodes. Micro controller performs tasks, processes data and controls the functionality of other components in the sensor node. In general as semi-conductor circuits become smaller, they consume less power for a given clock frequency and fit in a smaller area. Micro controllers, operate near one milliwatt while running at about 10 MHz. And most of the circuits can be powered off, so the standby power will be about one micro watt. Using Transceiver one mote can transmit or receive messages in WSN. External memory is used for two purposes, to store application related data and programming memory used to store program of the device. Flash Memory and on chip memory of micro controller are commonly employed as external memory. Many times motes have to operate for long time, since motes are wireless energy sources limit there over all operations. Motes consume power for processing, sensing and communication. processing and sensing consume much less power than communicating. For example the energy...
cost of transmitting 1 Kb a distance of 100 m is approximately the same as that for the executing
3 million instructions by 100 million instructions per second/W processor. Power is supplied to
mote generally using batteries.

1.1.2 Characteristics of WSN

Applications of WSN mainly include controlling, tracking and monitoring. They can roughly be
differentiated as a) Monitoring Space, b) Monitoring things, c) Monitoring interactions of things
with each other. For example WSN are used in area monitoring in which certain number of motes
are setup in the area, which will indicate a base station if there is any change in temperature,
sound etc. Other type of application is environment monitoring to predict nature calamities. To
briefly describe the characteristic of WSN one can say 1) they have limited power 2) they should
be able to withstand harsh environment conditions 3) they should cope with node failures,
and communication failures 4) Since sometimes, it is always difficult to manually configure large
number of nodes, these should be able to organize themselves.5) generally WSN are large scale
deployment 6) Once WSN is set up it is on its own. Typical WSN is shown in the figure below

Figure 2: Typical WSN ([1])

1.1.3 Operating Systems

After looking at the WSN applications one can say that the WSN network must allocate
limited hardware to multiple concurrent activities, such as sampling sensors, processing, and
streaming data. A node should be aware of its connectivity and also information needed for
routing the packet. One has to be very choosy about the operating system employed Operating
Systems which we use like Linux require more processing and RAM, they are complex and
they are more suitable for computer hardware. Operating systems for wireless sensor network
nodes are typically less complex than general-purpose operating systems both because of the
special requirements of sensor network applications and because of the resource constraints in
sensor network hardware platforms. For example, sensor network applications are usually not
interactive in the same way as applications for PCs. Because of this, the operating system
does not need to include support for user interfaces. There are many operating systems serving
different purpose like Tiny OS[16], Contiki[6], MANTIS[17], SOS[8] and Nano-RK. TinyOS is the
first such operating designed to be suitable to WSN. It is event driven operating system, that is when ever an event happens like message receiving etc. it call corresponding event handler to handle the event. It is developed on nesc. TinyOS is based on an event-driven programming model. When an event occurs like receiving message etc. corresponding event handler is called to finish the task. User can write his or her own task in those event handlers. Event handlers post the tasks that are scheduled by the TinyOS kernel some time later. Tasks on TinyOS are non-preemptive and are scheduled in FIFO order. TinyOS is developed over Nesc. NesC is designed to detect race conditions between tasks and event handlers. TinyOs is a open source software many users contributed to it. Because of this TinyOs is widely used and it supports many platforms like micaz, telos etc. We initially will support TinyOS application on our platform.

Contiki is designed to support loading modules over the network and supports run-time loading of standard ELF files. Contiki is also event driven but creates threads for every application. The Contiki kernel is event-driven, like TinyOS, but the system supports multi threading on a per-application basis. Task are scheduled in FIFO order. Unlike the event-driven Contiki kernel, the MANTIS kernels are based on preemptive multi threading. SOS is an operating system for mote-class wireless sensor networks developed by the Networked and Embedded Systems Lab (NESL) at UCLA. Like TinyOS and Contiki, SOS is an event-driven operating system. SOS uses a common kernel that implements messaging, dynamic memory, module loading and unloading, and other services The prime feature of SOS is its support for loadable modules.

Mat[15] is general architecture that allows users to build a wide range of virtual machines. It is developed to address challenges like reprogramming large number of mote in a sensor network. Once deployed, a network must be reprogramming is a challenge because mote are physically unreachable and wireless reprogramming can be a significant energy cost, primarily due to code size. Virtual machines are a compelling solution to these challenges. In sensor networks, communication is the greatest energy cost; reducing the traffic necessary to reprogram a mote can allow more frequent reprogramming, a longer network lifetime, or both. By abstracting high-level operations into VM byte codes, a VM program can be very short, on the order of bytes instead of kilobytes. Mate architecture allows to build Virtual machines. A Mat VM provides the advantages of concise code and safe execution and simplifies network programming. To reprogram a network a mote is added which runs new propagating capsules. When other motes hears this capsule it installs and begins forwarding it. After a short amount of time, every mote in the network runs the new code, with no effort or management by the user. Mate is still developed to add many features. Current versions are available in tinyos website.

1.2 WSN Test beds

WSN testbed is a testbed which consists of motes deployed over a certain area and sensor network programs can be tested on them. Testbed can be broadly classified in to two types domain specific and platform specific. Domain specific testbeds are concerned with the exploration of a specific WSN application domain such as habitat monitoring, health care, factory surveillance, etc. Platform specific testbeds are concerned with the exploration of technical solutions for networking, fault recovery, energy consumption, etc. While the two testbed categories may differ in requirements for data collection and analysis, there may be requirement overlaps in terms
of remote control, exclusive access control, etc. Our aim initially is to set up platform specific testbed.

1.2.1 Motivation

The life time of a sensor node depends on energy available to it. Algorithms employed in WSNs are distributed algorithms and more concerned about energy usage. WSNs are deployed in large numbers in various environments, with ad-hoc communications as key. For this reason, algorithms and protocols need to address the following issues[2]:

1. Lifetime maximization
2. Robustness and fault tolerance
3. Self-configuration

Debugging of a WSNs is also notoriously hard problem. Because WSNs are vast distributed system and because distributed algorithms are run on them there it may not be sufficient to know the status of one node. This makes the problem even worse. WSN applications also need to know about distance between each node and link capacity between them inorder to come up with optimize measurements for their applications. But real details of the experiment like energy constraints, reusability and scalability can only be understood by real simulation of the experiment on the real hardware. If one want to do this manually for every application he has to set up a different network and reprogramme each node and collection of data could be challenging. In doing manually one has to spend more cost(for node) and its not reusable by any others. One have to write his own programmes to collect data from debugging. Number of nodes also are dependent on application so one has to re-set the network to include more nodes. This approach is useful if number of nodes are few. Moreover all of this could be quite frustrating and slows down research process. To address these issues we developed MoteNet. It consists of set of computers to which few motes are connected. Each application is automatically ran on the scheduled time and data collected is logged and submitted to user.

1.2.2 Functionalities provided by MoteNet

A testbed is a platform for experimentation for large development projects. Like any other, our testbed has fixed number of nodes setup across an area and each of these nodes can communicate with some of the nodes in the testbed. We would like to provide following feature on the test bed.

1. A remote user should be able to use the test bed
2. A user can choose to run his program on all motes or select some nodes on which he wants to run
3. A user can mention time and amount of time he wants to run his program. That is he can schedule his program
4. System should be able get the connectivity of the nodes set up and provide it to user. It should also be able to report any bad nodes in the network.
5. System should be able to reprogram the motes remotely and autonomously at the scheduled time

6. When a program (precise definition of job is provided in details section) is running it should be able to collect all information related to job that is message packets received at a node or message packets which are being communicated over the radio etc.

7. When scheduled time is over all the collected information should be stored and make it available to user in a readable format.

8. Errors occurred during running the job or installing the job should be reported

User can write his application in such a way to collect information he wants for analysis of application and send it on serial port. For example to calculate energy consumed at every active node in the application, application can send all message packets which include number of sent and read packets to serial port and based on number of packet by multiplying with average amount of energy used for sending/receiving he can calculate energy used at every node. Our aim is to set up a test bed initially to provide above minimum support.

2 Overview

This report has details of the MoteNet testbed that is set up at IIT Bombay. We will discuss the key implementation details followed by the testing of the test bed. Section two explains related work done regarding the setting up of testbed. Similar testbed that are set up around the globe. In section three, we will go through all technical details of the project like hardware, software modules and deploying test bed. Section four explains various experiments run on the project and discusses various pros cons in the architecture. In section five we will discuss the future of the project, what advance features can be incorporated in to the testbed.

3 Related Work

The testing and validation of embedded code on real hardware is the key to successful development of sensor network applications. Because of increase in the need for deploying, debugging sensor network application many simulators like TOSSIM, SENSE, etc. are developed. TOSSIM is a discrete event simulator for TinyOS sensor networks. One can compile applications for TOSSIM and run on PC instead of mote. The primary design goal of SENSE (A Sensor Network Simulation) is to address factors such as extensibility, reusability, and scalability, and to take into account the needs of different users. Since all simulators run on a PC users can examine their TinyOS code using debuggers and other development tools. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. As one can notice these simulators, although help in debugging etc. does not provide information to understand resource limitations, communication loss and energy constraints at scale. A few WSN testbeds already exist. This section provides an overview of already setup testbeds.
3.1 Available Testbeds

**MoteLab:** MoteLab\[10, 3, 4\] is a WSN testbed developed at Harvard University for use with TinyOS. The testbed consists of set of Mica2 motes wired to the LAN at the deployment site. The motes are controlled by Crossbow MIB600 gateways. A single mote is attached to a multimeter in order to provide power consumption measurements. MoteLab features a web interface for setting up scheduled jobs. A job is defined by a set of flash image files that are assigned to a set of motes. Jobs are started by a job scheduler, and any mote output is logged to a local database. Special Java classes may be uploaded to perform custom data logging into specified tables. Real-time data I/O access is provided for each mote over TCP/IP by the TinyOS Serial Forwarder. This requires the TinyOS radio stack to be running on the motes in order to provide message framing. The TinyOS Serial Forwarder has not been integrated with the user account system, so in fact anyone may connect to the Serial Forwarder interface of any mote in the testbed. Only one job can be run on the entire test bed at any given time. That is, the MoteLab testbed does not allow different jobs to execute simultaneously. The basic functionalities provided by the MoteNet are similar to the motelab.

**Mirage:** The IRB Mirage testbed system\[7\] uses a micro-economic auction scheme for testbed resource allocation. In particular, it allows users to bid on the desired testbed resources and uses a resource discovery algorithm to assign the specific resources. The Mirage system at IRB provides users with a project ssh account from which a set of TinyOS related tools are available. Real-time mote I/O is performed using the TinyOS Serial Forwarder.

**Emulab and mobile emulab:** Emulab\[5\] is a common framework for providing access to different network testbed environments. An interesting application of this framework is the Mobile Emulab. A WSN testbed consisting of a set of static motes and a set of mobile motes is provided. The motes are mobilized by a set of robots that are remotely controllable, allowing the testbed users to modify the WSN topology on-the-fly and experiment with dynamic topologies.

**Emstar:** EmStar\[11\] is a flexible WSN development environment. The concept is to enable developers to gradually shift the boundary between simulation and reality when developing new WSN applications. A range of tools allow the developer to perform pure simulations, hardware supported simulations, and finally test the application in a close-to-reality deployment. EmTos is an enhancement of EmStar that allows simulation of TinyOS code.

**Kansei:** The Kansei testbed framework\[9\] was developed for the ExScal project at Ohio State University. The framework is designed to support static, portable, and mobile sensor arrays. The ExScal project is a hybrid approach to simulation of large scale WSN deployments. It features a large static array of nodes that are interconnected using both wired and wireless back channels. Artificial sensor readings may be injected into the WSN either based on physical models or by scaling up sensor traces from smaller deployments. Users are able to pre-schedule multi phase experiments. The experiments are executed by a Director module and results are stored server side for later retrieval. The Kansei system is designed for mote platform portability. Interfaces are exposed using web services and web pages, and integration of third party modules is possible. This platform consists of five robotic mobile nodes that operate on the transparent Plexiglas mobility plane Kansei because of its design focus on sensing and scaling has unique
characteristics (i) Heterogeneous hardware infrastructure (ii) Time accurate hybrid simulation engine for simulating substantially larger arrays using testbed hardware resources. (iii) High fidelity sensor data generation and real-time data and event injection. (iv) Software components and associated job control language to support complex multi-tier experiments.

**Twist:** The TWIST testbed[18] was developed at TU Berlin. It uses a back channel to control the state of each mote in the WSN. A special feature of the TWIST testbed system is the use of socket naming to enable automatic detection of mote locations when motes are moved from one socket to another. The TWIST testbed also allows remote control of the power state of each mote.

### 3.2 Quality of Testbeds

With so many simulators and Testbeds one wants to look into some of the characteristics of these which evaluate them. Moreover because different testbed have different architecture one would be worried about the validity of the results. In paper Apples, Oranges and Testbeds[14] they characterize MoteLab at Harvard, Mist Lab and TOSSIM by presenting some of the empirical data they obtained by running some experiments on these testbed.

In order to compare testbeds, experiment is done based on application(surge) using two protocols (Min Hop and Mint Route) on two testbeds (MoteLab and Mist lab). Surge is an application in which message generated at different nodes are sent to one central node (sink). Two protocols are employed to build the application. In Min hop route to the sink is formed depending on number of hops but in Mint Route it is formed depending on link quality. Because there are no standard parameters to characterize a test bed author has chosen the following parameters at testbed level 1) number of node (number of active nodes) 2) connectivity (average number of neighbors over all active nodes) 3) link capacity (max raw radio output over any link [bytes/s]) 4) link quality (raw packet success rate). To allow comparisons across testbed based on an application, one should also consider parameters at application level. Selected parameters are 1) good put (fraction of injected messages delivered at the sink) 2) hop count (average hop count of the delivered messages) 3) tree depth (average hop count over all paths to the sink) 4) route quality (one hop message success rate averaged over all used links).

This experiments are done on mist lab, motelab (micaz bed) and motelab-sky (telos). It has been observed that both multi hop and mint Route perform equally good on motelab-z and mist lab. But On motelab-sky for low message rates Mint Route performs very bad (goodput < 50) and loses to MultiHop. For high rates, however, Mint Route wins by a factor of two. It is observed from log files that mint Route is very slow in setting up a spanning tree, once it is set it out performs multihop. The reasons for the divergence in the results are not yet known. One surprising result is divergence in motelab and motelab sky although they uses same radio chip. It is not clear if characterization is wrong or its application dependent. There is some research needed to clarify factors which characterize reliability of the testbed.
4 Technical Details

MoteNet consists of software components for managing Ethernet connected sensor network testbed. The figure below shows how the testbed is setup. There is a central server known as Major Node and other computers to which motes are connected are Minor Nodes. Major Nodes handles user request and copies all programmes to Minor Nodes. Minor Node runs the programmes at scheduled time and sends collected data to the Major Node. User access testbed and collected data by connecting to Major Node.

![Figure 3: testbed-setup](image)

4.1 Hardware Details

One would like to select the hardware for mote based on following factors cost, reliability, ease to use, should be remote programmable. Some of the hardware available are telos family mote[13], mica family mote[12], intelmote2 etc. Mica mote which was developed before telos was is a first from mica family and is designed more carefully to serve as a general purpose platform for WSN research. It has more memory, extensive sensor interface and very flexible radio interface. Mica was useful for development, but unsuitable for deployments. The boost converter provided a stable voltage but used excess quiescent current. Radio ranges is short and are unreliable. Later motes from mica family have redesigned some component still energy used in transmitting bit and waking up time is more compared to telos.

Telos is developed at UC Berkeley to enable wireless sensor network research. Main characteristics of Telos mote are: minimal energy consumption, easy to use, and increased software and hardware robustness. Telos design is based on the principle that a sensor node is most time in inactive mode, so in order to decrease amount of power consumption by a mote one has to minimize standby time and wakeup time. Table below shows the comparison of the components of the mica and telos mote. MSP430 micro controller used in telos has lowest power consumption in sleep and active modes compared to other. It operates at 1.8V (two AA batteries). Its wake up time is less than 6ms. Telos uses the chipcon CC420 radio in the 2.4GHz band a wideband radio with higher data rate which allows shorter active periods further reducing energy consumption. The radio crystal use on telos was carefully chosen to be a low ESR 16 MHz crystal.; low resistance is needed for minimizing start up time of the crystal. Telos can communicate with any number of devices sharing the same physical layer, including devices from other vendors.

To get a brief idea of telos power consumption, for example in a network with each mote reporting data for every three minutes mica motes ran for 453 days while telos mote ran for 945 days. Cost of the telos mote is less than mica2 mote. By using USB on every mote, any Telos may operate as an experimental device on a lab bench, a gateway to a PC or as a node.
in a large testbed. In the lab USB provides an easy and robust way to interface, program, and experiment with motes. As a gateway, no programming or interface board is required and any node may act as the gateway. Remote programming of telos mote needs Ethernet USB interface while MIB-600 is needed for remote programming of telosb mote. The infrastructure cost of a 60-node Telos testbed is approximately 1,000(USD). In contrast, a 60-node Mica2 testbed costs almost 21,000(USD). Telosb mote can be connected to any sensor which can give information in analog form. Telos platform provide 16 pin on which sensor can be setup. Thus Telos mote is better than mica and also cost for test bed and remote programming is less. We are using Telosb motes as node in our test bed.

Test bed should be in such a way that it can support as many applications as possible. Distributing motes uniformly will allow as many applications as possible. Test bed should be set up in such a way so that there always exists bunch of nodes which can be used to run the program. Factor on which setup of the test bed depends are density of the node distribution on the test bed and number of nodes available. Nodes in the test bed should be distributed such that there are places where nodes are close enough so that collision probability is more and places where nodes are far enough where collision probability is less. User can chose suitable configuration of nodes on which he wants to run his application. To support applications of large scale, number of nodes has in the testbed has to be more.
To set up a test bed to achieve our tasks we need motes and some hardware to connect them to a central server so that we can re-program the motes. In other projects described above like motelab they use MIB-600 for remote programming of motes. Using MIB-600 one can connect to mote through Ethernet. MIB-600 is a costly hardware, so we decide to connect the motes to different computers in Circular hall at Kresit. Note that these computer are connected to each other. Now one can reach mote from central server over Ethernet through the computer to which it is attached. The central server which stores information about all jobs in database, is called a **Major Node**. There is only one Major Node. Now all the computer to which motes are connected are called Minor Node. Minor Nodes are connected to Major Node over Ethernet. This type of set-up although avoids cost but has its own disadvantages which will be discussed later.

Number of motes connected at each minor node can be varied. Mini usb hub is used to increase usb ports at the minor node. Since computers in circular hall are spread all over the room a very good network can be formed with different link qualities etc.

### 4.2 Software Components

Software Component of MoteNet include UserHandler, JobHandler, Jobscheduler ,SQL Database, MoteTracker. User handler is only program running on the major node all other software modules job scheduler, job manager, Motetracker will be running on the Minor Node. Database is at the Major Node In this section we will go through function and details of each of this components. Before we go into details, let us define Job. A job submitted by user should have following things:

1. Main Executables files which are to be run the motes. Each job may have any number of main executables.
2. Class files of the packet structure which user expects us to capture at node.
3. A mapping between main executable files and motes on which it has to be run.

User submits a job through interface provided at the WebServer. Creation of a job and scheduling of job are considered separate because this allows us to store configuration of job and user can schedule it at different amount time at different times of the day.

#### 4.2.1 My SQL Database Backend

To store the information about jobs, user, scheduling, test bed configuration, data collection one can use either files or database. We need refer to data we store for some information for example, in moteNet before scheduling a job we want to make sure that no other job is scheduled at the same time to find out which we have go through data stored. Because database allows us to query on data and retrieval is fast we are using database. PSQL database is set up on the central server. To store information in database, one need to have some structure of tables which can be obtained from ER diagram. One draws ER diagram based on the specifications of the project. As explained before our goal is to allow a remote user to use a test bed . Test bed has several motes. Each mote can interact with few other motes. Working condition of all motes has to be monitored frequently. Every user has some files which include main files(executable)
User create a job by choosing which main and class files to use and on what motes. Every job has name and description. Each day is divided into slots of particular size (say five minutes). On the mote lab only one job runs at a time. If a job is scheduled on a particular slot no other job can be run during that time. User when scheduling a job selects some slots. Based on the time, job has to be ran on the moteNet. Based on the above specification regarding moteNet, ER Diagram is shown in the figure below. Mote, User, Mainfile, Classfile, Jobs, slots are the entities. How each entity interacts with other is shown in the ER-Diagram. Mainfiles is made a separate entity because a user may want to use same main file in many jobs. The attributes of each entity are shown.

![ER- Diagram](image)

This ER diagram can be converted into database schema by writing each table for each entity and each relationship. In a relationship primary key are primary key of two connected entities. Schema is normalized The specifications also talk about storing the data collection. Data collected at each node is collection of message packets of particular form. For storing these we create a database for every user which he can access from outside. In database a table will be created for every message format packet and corresponding values will be inserted into each field. For example for message structure containing fields count, version, data. We create a table with those field adding some fields like time etc. as Message(count,version,data,timestamp) and dump all message packet of that form in to that table. We write them back to files and provide data to use in the downloadable format.

### 4.2.2 User Interface

When user logs into motenet he will be provided with set of options like create job, schedule job etc. Each of the options is explained below.

1. **Registration of User**: User can register to the motenet by providing details and setting
a user name and passwd for him. Every user has to create an account for him to use the motenet. When ever a user is registered, in major node and minor node a folder for this user is created in motenet home directory. Directory structure at major and minor nodes is shown in the figure below.

![Directory Hierarchy](image)

Figure 6: Directory hierarchy

2. **Status of Motes**: Motelist is the command used to view status of motes in the testbed. This command when executed by user will show the status of each mote on the motenet. It show which mote id is connected to which ip address. It indicates mote condition. Mote can have four conditions it ok or its removed or its not working or its busy. This command is help ful because user can verify the before creating the job. Condition of the motes will be obtained from database which is being updated by motetracker program running on the minor node. The screenshot below shows the status of different motes.

3. **Create Job**: This option is chosen by user when he wants to create a job. He specifies main file names and class details in a file. In another file he specifies details of mapping of main files to mote. When a new job is created a new folder (with name job'jobID') is created under this user at the major node. All the main files and class files are copied on to major node. details of the files are entered into database tables.

4. **Scheduling job**: When user queries to schedule a job, continuous slots are shown with each slot length 5 min. slots are shown from 10 mins from now. He specifies on which slots the job is to be scheduled. He specifies this by choosing slot numbers. Once he choses the slots corresponding data is entered in to databases. This job folder is copied on the minor nodes on which job is scheduled. The screenshot below shows slots shown while scheduling job.

5. **View data**: This command gives the path to the files where data collected for previous jobs of the user is stored. Data collected at each mote is written to a separate file. There is an error log file which has errors occurred during the execution of the job. User can download data from these folders.

### 4.2.3 Execution of the job

To execute job at correct time, it has to keep track of next job to scheduled. Each thread running on different minor nodes will contact database and look for the next job to be scheduled. It checks every one minute for the job to be scheduled ten minutes from now. This is required as one might edit job schedule or add new job during ten minutes. If no job is found the default
job will be still running on the motes. Each minor node checks independently whether there is a job to be run on motes connected to it.

When no program is scheduled default job will be running on each mote. When there is a job to be executed, each minor node installs program to be run on each mote. only one mote is reprogrammed at any time. If there occurs any installation errors, they will be logged. After reprogramming each mote serial forwarders are started to listen to them. A thread is started which connects to serial forwarder running on the port, to collect packets forwarded by serial forwarder. As message are read they are stored in buffer. When buffer size goes more than 100 packets, it is written to a file. Data collected at each mote is written to a different file. User may run same job in different slots, so each time job is run, data collected is stored in different folders. One for each slot. If slots are continuous data job is continued running and data for all continuous slots are stored in first slot folder. When a jobs end time is reached, First serial forwarder is stopped and motes are reprogrammed with default job. If any error occurs during installation of the job or when job is running this will be caught as exception and logged in to error log file. When the job is running if any of the mote got disturbed or removed, it will be recorded in the error log file.

<table>
<thead>
<tr>
<th>MoteId</th>
<th>Moteport</th>
<th>Ipaddress</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>/dev/ttyUSB2</td>
<td>10.129.112.142</td>
<td>REMOVED</td>
</tr>
<tr>
<td>4</td>
<td>/dev/ttyUSB3</td>
<td>10.129.112.142</td>
<td>REMOVED</td>
</tr>
<tr>
<td>5</td>
<td>/dev/ttyUSB4</td>
<td>10.129.112.142</td>
<td>REMOVED</td>
</tr>
<tr>
<td>1</td>
<td>/dev/ttyUSB0</td>
<td>10.129.112.142</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>/dev/ttyUSB1</td>
<td>10.129.112.142</td>
<td>OK</td>
</tr>
</tbody>
</table>

Figure 7: Status of motes
4.2.4 Status of motes

First important task is to giving id’s to the motes. Since motes are removed and put back frequently, we need to develop a system which revisits the mote id’s when a mote is connected. There is unique mapping between usb ports on computer to mote id’s. That is every port on the minor node is assignd a mote id, when ever a mote is conneced in that port, the id of that mote is port id. When a mote is connected to minor node in a new port, that port is assigned a new id, if it is connected in old port, the id of mote is id of that port. It is important to know the status of each mote in the testbed. User while creating job wants to know which motes are working and which are not. He will select those motes, which are available. A thread running at each minor node keeps updating status of each mote. When no job is scheduled a default job is running on each and every job. This default job will send a data packet after every time interval. To know the status of motes, thread will look at output of the motelist command. If already detected mote is not in the list immediately its status is being updated in the data bases as removed. If a new mote is detected it is given an id, and details are added to the database. If the removed mote is connected back, its status is changed to ok. Thread also listens to the data buffer at each of the motes if any motes data buffer is empty implies default program although installed is not working on this mote. So condition of mote is changed to bad. At any given time, this mote tracker will keep track of those motes on which job is not scheduled before or
after two minute from current time. This is because a job cannot be installed on mote when
some program is listening to its serial port. When user issues command each mote information
is picked up from the database and displayed.

4.2.5 Expanding the Testbed

Admin has the privileges to expand test bed by adding a new minor node. To expand the
test bed first admin connects motes to the computer. He has to enter details like ipaddress,
username, password, homedir of the new minor node into database. On the new minor node,
folders will be created for each user in the home directory. Before starting the program, admin
should create two files one each for details of this node and major node. This files will be used
by programs. Once the program is started, program detects newly connected motes and assign
id and enter their details in the database. When user check available motes, these added motes
will be displayed.

4.3 Summary

In this section we will put together all the components seen so far and look at the functioning
of MoteNet. Important modules in the motenet are UserHandler, MoteTracker, Jobscheduler,
JobHandler, dataLogger. The diagram below shows all the components put together

![Diagram of MoteNet](image)

Figure 9: MoteNet
Functioning of the motenet can be summarized in the following steps:

1. User if has no login id registers himself with the MoteNet
2. User created job is stored in to database by User Handler module
3. Mote Tracker keeps updating status of the motes in the database
4. When user schedules the job, this information is also stored into database
5. Job scheduler keeps looking for current running jobs
6. It informs Job handler when a job is found.
7. JobHandler then installs job on the relevant motes and starts data logger to collect the data.
8. After the end time is reached datalogger is stopped and job Handler copies all data files to Major Node
9. User can collect data at the Major Node

Only copying the job on to the minor nodes, is work of major Nodes. Once it has copied the job, minor node deals with execution of the job. It will run program on the motes connected to it and send data back to major Node. Note that time synchronization problem between minor nodes is solved by NTP protocol. Time in all the minor nodes and major node are referred to a global clock.

5 Experiments

It is important for the user to know the connectivity between motes on the testbed. For this we wrote a program called Connectivity job, which is treated as any other job. This job is ran, once in week. This has a tinyos program written to get information about connectivity of different motes in the testbed. The mote on which this is installed will broadcast a 20 message packets in some interval of time. Structure of the message packet is very simple [nodeid,counter]. Similarly all the motes in the testbed will broadcast their own messages. When a mote receives packet from other mote, it pushes the packet on to serial forwarder. Message class file is submitted along with the job. The packets at collected at each mote are stored on to a file by data logger. This file is processed later for each mote. If a mote has received a packet from a mote implies that this mote is reachable from that mote. Quality of link is calculated as ratio of number of packets received and number of packets sent by the other node. A graph is drawn to indicate connectivity between motes. User can view this graph from command line. This job is ran as admin job on the moteNet. For example, in the experiment below, we used nine motes three at each minornode. After collecting data and analysis, graph obtained is shown below.

Various tests have been done to check correctness of the code. First we checked if the job is installed correctly whether data is logged correctly. Since there is some time between after installation of program and starting serial frowader, first few packet sent by the program may not be caught by serial forwader. So User has to write his program in such a way to avoid this. For every packet received before writing it to file a time stamp is written to the file. Other tests
Figure 10: mote connectivity

Graph showing connectivity between motes

include checking if errors occurred during execution of the job are correctly written to error log or not. For this, a mote has been manually disturbed just before execution of the job and also during execution of the job. Comparing with the motelab, our system is much cheaper as we are not using MIB-600 crossbow. One of the disadvantages is that job files are redundantly stored at each computer. One of the reasons is that, one cannot remotely reprogram mote connected to another computer by executing command in the current environment. Another disadvantage is that test bed is setup in a place where it could be disturbed a lot, since minor nodes are also used by students. There are series of test cases which are being tested during the development of the system. One set of tests include testing of allotment of motes to each mote. Some of the interesting issues to look into are like, does the job start at scheduled time? Because, as we have seen before, installation on all motes do not start at the same time. Motes at minor nodes are installed one after another.

6 Future Work And Conclusion

Currently what have been developed is a basic version and there is lot of scope in to the project. Motenet can be extended in many ways. MoteNet puts to many uses. Set up like motenet is very useful in developing robust protocols and services for integrating wireless devices with many other fields of application like medical care etc. A web interface can be provided and it can be
made more user friendly. Although web Interface is easy to use for user, scripting interface might be more powerful to more serious researchers. The following are the advanced concepts on which future work can be carried out

**Energy consumption statistics:** Energy consumption is a major area of interest in the WSN domain. Many experiments have the sole purpose of measuring the energy profile of a specific WSN application. Therefore, a uniform way of collecting energy consumption statistics would be a useful feature for a WSN testbed. However, measuring the energy consumption requires specialized hardware and also energy profiles are very specific to the mote platforms in use.

**Artificial sensor data injection:** Some WSN applications, especially those involving environmental monitoring, can only be tested properly if they are exposed to the target environment. Some of the target environments are naturally not very easy to access. The Kansei testbed[9] has simulation tools to aid experiments that require environmental sensor readings. The general approach is to supply each of the motes with a real-time feed of artificial sensor data. The sensor data is produced centrally either by scaling up existing data traces or by simulating well-understood physical phenomena. This enables the embedded software developers to study the behaviour of a real WSN without the need to actually deploy the WSN in the field of application. Also, the cost to acquire the sensors that are capable of producing the data is saved.

**Mote access and control:** Basically, a testbed user should be able to get exclusive access to some motes on the testbed, upload software to the motes, and view output generated by the motes. There are applications which require a direct connection to the mote while job is running to inject data and monitoring. Providing through web interface direct connection to the node is quiet a challenging task. Regardless of the mote reservation policy, the system should allow multiple simultaneous users to control several motes at a time. To support jobs to run side by motelab can be partitioned into lab

Apart from above issues, if more motes are available and a place is dedicated especially for motenet then, the architecture of the system might have to be changed, now it will become more simple. Our aim in designing motenet is to provide a real test bed on which a user can upload his files and get the data collected in downloadable format. Motenet meets its design goal by running applications on real test bed. Motenet allows lab to be shared by large group of people. Motenet once setup would be help full in accelerating research and teaching purposes.
Acknowledgements

I would like to thank my guide, Prof. Purushottam Kulkarni for the consistent directions he has fed into my work and being available all the time when I needed. Meeting minutes were especially helpful in developing ideas.
References


[16] developed at UCB open source. Tinyos: Operating system for sensor network.
