2010-01-01

A Deadline-Driven Epidemic Data Collection Protocol Suitable For Tracking Interpersonnel Rendezvous

Avranil Tah
University of Texas at El Paso, avrahit@gmail.com

A DEADLINE-DRIVEN EPIDEMIC DATA COLLECTION PROTOCOL SUITABLE FOR TRACKING INTERPERSONNEL RENDEZVOUS

AVRANIL TAH

Department of Computer Science

APPROVED:

Eric A. Freudenthal, Chair, Ph.D.

Luc Longpré, Ph.D.

Virgilio Gonzalez, Ph.D.

Patricia D. Witherspoon, Ph.D.
Dean of the Graduate School

to my

MOTHER, FATHER

with love

A DEADLINE-DRIVEN EPIDEMIC DATA COLLECTION PROTOCOL SUITABLE FOR TRACKING INTERPERSONNEL RENDEZVOUS

by

AVRANIL TAH

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

December 2010

Acknowledgements

I would like to express my deepest gratitude to my advisor Dr. Eric Freudenthal for encouraging me to write this thesis and his guidance throughout the process. I have learnt a lot of underlying concepts of Computer Science from him. I would like to thank Dr. Luc Longpré and Dr. Virgilio Gonzalez for being my committee members.

I am grateful to my parents, my sister Bidisha, my brother Abhishek, Mr. D. P. Mallik for their love and support during my entire life. I am thankful to Anuradha for her unconditional love and support.

I would like to take this opportunity to thank Bivas Das and Manuel Corona for their help whenever I faced hard time in my project.

Abstract

This thesis describes a peer-to-peer wireless data collection algorithm that uses epidemic communication to propagate time-sensitive sequentially sampled data records from sensors toward infrastructure connected upload stations via mobile data mules. These records are labeled with sequence numbers and delivery deadlines, and are transmitted in sequential order. Delivery deadlines enable transmission prioritization and trigger alarms warning of violations.

The sequential ordering of records simplifies the protocols transmission-control and garbage collection mechanisms: only two monotonically increasing scalar sequence indices associated with a particular sensor must be exchanged between peers prior to selecting which records need to be communicated. One of these indices also serves as an anti-entropy message, indicating which records are known to already have been uploaded by an infrastructure-connected upload station, thereby indicating eligibility for garbage collection.

This data collection algorithm is used to implement a prototype inter-personnel rendezvous reporting system potentially useful for tracking the spread of contagious disease. Radios used to transfer information among peers also serve as proximity sensors. Records are created whenever a peer discovers another. The data collection algorithm transports these records to infrastructure-connected upload stations responsible for their storage and on-line processing.

My contributions include refinement, implementation, and initial testing of the algorithms for data collection and rendezvous reporting.

Table of Contents

Chapter 1

Introduction

This thesis describes a peer-to-peer wireless data collection algorithm that uses epidemic communication to propagate time-sensitive sequentially sampled data records from sensors toward infrastructure connected upload stations via mobile data mules. These records are labeled with sequence numbers and delivery deadlines, and are transmitted in sequential order. Delivery deadlines enable transmission prioritization and trigger alarms warning of violations.

The sequential ordering of records simplifies the protocols transmission-control and garbage collection mechanisms: only two monotonically increasing scalar sequence indices associated with a particular sensor must be exchanged between peers prior to selecting which records need to be communicated. One of these indices also serves as an anti-entropy message, indicating which records are known to already have been uploaded by an infrastructure-connected upload station, thereby indicating eligibility for garbage collection.

1.1 MOTIVATION

The process of manual collection of medical samples is monotonus and error-prone. Automatic collection of these medical samples will reduce the workload for healthcare personnel. The use of wireless technology can enable the automatic collection of these samples.

Inter-personnel tracking system enables hospitals and medical centers to identify the people with whom the patient has come in contact. Even if a patient is diagnosed hours, or days, after admission, the history can be recalled which allows the proper measures to be taken to prevent the infection. Patient tracking in hospitals is vital to help halt the spread

of contagion and to improve patient throughput.

We are implementing an epidemic based data collection algorithm useful for tracking inter-personnel rendezvous. This proposed algorithm automates the process of tracking inter-personnel rendezvous within a hospital. Use of short range radio devices makes deployment and maintenance of the system easier than high-power wireless devices such as Wi-Fi. Chance of interference between delicate medical devices and wireless devices is reduced by the use of low-power devices. So it enables hospitals to keep track of inter-personnel interactions among patients, visitors, health-personnel during the outbreak of any infectious disease.

1.2 SOLUTION

Distributing dynamic information across a large number of computers is a central problem in distributed systems design. Epidemic protocols offer a mechanism for information distribution without relying on central servers. Their simplicity, scalability, and good performance characteristics have made them suitable for information dissemination in ad hoc networks.

The purpose of this thesis is to develop a peer-to-peer wireless data collection algorithm that uses epidemic communication. This algorithm is used to implement an inter-personnel tracking system that will automatically keep track of every users contacts with others within the network. Whenever a peer discovers another, a record is created and stored into memory. The information collected by the users is propagated to other peers by the use of epidemic communication protocol. The information eventually arrives at the server where it gets stored and processed.

Duplicate records are inevitable due to the nature of how epidemic algorithm works. Since an infectious message is transmitted to all in-range peers, all members of the network will eventually have the same data. As other peers try to establish communication with other members, the amount of redundant data increases in an exponentially way. An

anti-entropy protocol is used to delete the already uploaded records efficiently from the gossip-based network system.

Chapter 2

Related Work

Several applications of mobile commerce depend on the knowledge of customers positions. For example, advertisers may want their advertisements to reach customers in a specific location. This is possible if the locations are estimated correctly. The required precision of estimated locations varies with applications. For example, a system that selects which advertisements the customers should receive based upon their positions within a shopping mall, may require more accurate location estimation than to target people who are driving towards that mall.

Location tracking systems are usually designed to provide location information of the tracked person/item. We are examining applications that require interpersonal interaction information such as the meeting time of two or more persons and the duration of the meeting. For example, in order to track the spread of infections within a hospital, it may be beneficial for infection specialists to have a system which keeps track of interpersonnel rendezvous.

To simplify and organize the discussion of related works in the area of wireless tracking system, I used the following two categories: (1) tracking systems based on triangulation method, and (2) tracking systems based on scanning technique.

2.1 TRACKING SYSTEMS BASED ON TRIANGULATION METHOD

The RADAR [2] is a triangulation based location tracking system. This system uses radio-frequency (RF) for locating and tracking users inside buildings. A Radar system normally needs a single active antenna with Range estimation by detecting echoes. This system is capable of locating an object using polar coordinates.

LORAN [17] and GPS [3] systems use multiple active antennas and the receiver must estimate the position by precise knowledge of each antenna and measurement of the propagation delays.

Another system based on triangulation is described in [7]. This paper describes the empirical test results for different indoor scenarios. The system uses received signal strength indication (RSSI) to measure distances of the devices. RSSI can be affected by wall, water and other obstacles. That can reduce the signal strength. This limits the effectiveness of the RSSI-based system.

2.2 TRACKING SYSTEMS BASED ON SCANNING TECHNIQUE

The Active Badge system [22] was an early contribution to the field of scanning based location tracking systems. In this system, sensors placed at known positions within a building pick up the unique identifiers emitted from the IR badges and relay these to the location manager software. Though this system provides accurate location information, but it has some drawbacks as well: (a) it performs poorly due to the limited range of IR, (b) it incurs high installation and maintenance costs, and (c) it performs poorly in the presence of direct sunlight.

Another popular technology used for personnel and/or asset tracking is radio frequency identification (RFID) [15], which works by means of electromagnetic induction. The system described in [14] is based on scanning technique. Every person in the network wears a RFID tag and an intelligent RFID reader. The methodology of this system is based on person-to-person and person-to-object contacts. Whenever a user comes in contact with another or an object, the intelligent reader stores in memory the ID number and the current time.

A Bluetooth based tracking system is described in [19]. This system uses mobile phone terminals to build a virtual networking by combining GPS and Bluetooth technology with mobile Internet. The Bluetooth scanners perform Bluetooth discovery process and assign location information to found Bluetooth IDs based on GPS coordinates.

During the SARS outbreak in November 2002, Industrial Technology Research Institute (ITRI) of Taiwan implemented a RFID based personnel tracking system [12] to track hospital staffs within the facilities. Each employee was given a RFID tag associated with a distinct ID number. RFID readers were located in different areas of the hospital, connected to the local area network. Every time an employee was within the range of a reader, the reader created a record including the ID number, the area, and the time. Even though this system provided information about the location of an individual at any time, it failed to inform who the person was in contact with.

The Defense Science and Technology Agency of Singapore developed Hospital Movement Tracking System (HMTS) during the SARS outbreak as well [11]. Unlike ITRIs tracking system HMTS not only tracked hospital staff, but patients and visitors as well by providing a RFID tag to every person within the hospital facilities. Even though this approach provides more data about people visits to a certain place than the ITRI system, it failed to inform who the person was in contact with.

2.3 EPIDEMIC COMMUNICATION

Data Transmission protocols play an important role in building these tracking systems. In ad hoc networks, the power supply of individual nodes is limited and wireless bandwidth is limited. Moreover, since nodes can be mobile, routes may constantly change. Thus to enable efficient communication, robust routing protocols must be developed.

Distributing dynamic information across a large number of computers is a central problem in distributed systems design. Epidemic protocols offer a mechanism for information distribution without relying on central servers. Their simplicity, scalability, and good performance characteristics have made them suitable for information dissemination in ad hoc networks [6][20][23]. Epidemic refers to information exchange mechanism where each node can be a source of information, and is capable of information transfer.

The use of epidemic algorithm was introduced for database replication within networks composed of several servers [5]. As the database is modified in a certain site, the update is replicated to other sites until it arrives to a server. Different algorithms based on epidemics, such as anti-entropy and rumor mongering, are used to spread the updates in the database to other sites. This paper also introduces the idea of infective messages, which are the updates propagated to other sites.

[10] describes a partial replica update protocol for distributed databases. The protocol is based on epidemic dissemination of information throughout the network. In order to receive the update, a site communicates with another site and sends it information.

The Bayou system presented the idea of sharing data, i.e. calendars, notes, etc., among mobile users [4]. Every user carries a database and as this is modified, the updates are replicated and propagated to other users using the infra-red port of the portable computer. The update eventually arrives to the servers database and the appropriate modifications are performed. Then the server propagates this update to all the users within to network so that they are aware of this change [18] and [16].

[21] describes an approach based on epidemic protocol to build a semantic overlay for

content-based searching. This approach uses epidemic protocol to cluster peers with similar contents.

[1][13] describe the use of epidemic protocol to disseminate information among sensors or nodes. [9] describes the SPIN protocol, that efficiently disseminates information among sensors in an energy constrained wireless network. SPIN uses the concept of meta-data negotiation to eliminate the transmission of redundant data throughout the network.

2.4 ANTI ENTROPY

Duplicate records are inevitable due to the nature of how epidemic algorithm works. There is a class of anti-entropy protocols that can efficiently delete the already uploaded data from the gossip-based network system. Bayou [4] introduced the concept of anti-entropy in order to resolve inconsistencies, in which multiple values are assigned to the same symbol. It specifies a mechanism for an inconsistent system to eventually converge on a single value.

[8] discusses the suitability of anti-entropy protocol for managing data deletion in an epidemic based system. In our approach already uploaded samples need to be deleted in order to free the devices memory. Anti-entropy protocol which exploits the sequential nature of data collection and upload events is a suitable technique for this purpose.

Chapter 3

Method

3.1 OVERVIEW

The proposed peer-to-peer wireless data collection algorithm is used to provide information about a users previous contacts with other users inside a network. Records collected by the users get propagated in an epidemic way to other peers and finally arrive to the infrastructure connected upload station where it is stored and processed.

Whenever a radio discovers another, a record is created including the identification number of that peer and a delivery deadline of that record. Next they exchange sensor stream status which contains values such as sensor number(s-no), Upload-Index and Maximum-Index for that radio, and a delivery deadline. There are several samples corresponding with a specific sensor (S-no). The delivery deadline is assigned to each individual record to enable transmission prioritization. Priority of a sensor is equal to the priority of the highest prioritized record of it. Upload-Index represents servers copy of the index for S-no. Maximum-Index represents the index of the last collected record by the radio for S-no. Peers send and receive records based on the comparison of sensor stream status. Thus both users exchange records of previous contacts with other peers. When a peer discovers an upload station as communication partner, it deletes all the samples from its memory after uploading the newer samples to the server successfully.

Duplicate records are inevitable due to the nature of how epidemic algorithm works. The protocol overcomes this obstacle by the use of an anti-entropy message, indicating which records are known to already have been uploaded by an infrastructure-connected upload station, thereby indicating eligibility for garbage collection.

3.2 DATA STRUCTURE

3.2.1 SENSOR STREAM DATA(SSD)

SSD is a python dictionary where each record is stored. Dictionary is the set of key-value pairs where the keys are unique. Whenever a radio discovers another, it creates a record including the sensor number of the nearby radio. The keys are the unique sensor numbers and the values are the records collected by these specific radios. For example, SSD looks like the following;

{'001FE2E092C0': [(234542, '00CFD2E00FC0'), (376545, '001FE2F02180')], '001FE2E092BD': [(467654, '00215C6F2F55'), (487650, '00217CFF2C03')]}

The keys are 001FE2E092C0 and 001FE2E092BD. The samples collected by the radio whose unique sensor ID is 001FE2E092C0 are (234542, '00CFD2E00FC0') and (376545, '001FE2F02180') where the first field of the tuple is the delivery deadline of that record and second field is the tracked radios unique ID number. Whenever the peer discovers an upload station, it sends newer records to the upload station and updates the Upload-index and Maximum-Index for each sensor and deletes all records from its memory.

3.2.2 SENSOR STREAM STATUS(SSS)

SSS is also a python dictionary which contains the sensor number as key and Upload-Index, Maximum-Index and priority of this sensor as values. For example, SSS looks like the following;

{'001FE2E092C0': [0, 1, 234542], '001FE2E092BD': [0, 1, 467654]}

The Upload-Index, Maximum-Index and priority of the sensor 001FE2E092C0 is 0, 1and 234542 respectively.

3.2.3 RECORDS TO SEND

Format for the samples to be sent is ['001FE2E092C0', ('001FE2E092BD',20000), ('011FECF0FD09',12000)]. The first field of the list indicates sensor ID and it is followed by samples associated with this sensor.

3.3 DESCRIPTION OF THE ALGORITHM

Each Bluetooth radio in client mode performs a scan for in-range Bluetooth radios. Whenever a peer discovers another, a record is created including the identification number of both peers and a delivery deadline of that record. This is followed by the establishment of connection between peers. Next they exchange sensor stream status which contains values such as sensor number (S-no), Upload-Index and Maximum-Index for this sensor, and a delivery deadline. There are several records corresponding with a specific sensor (S-no). The delivery deadline is assigned to each individual record to enable transmission prioritization. Priority of a sensor is equal to the priority of the highest prioritized record of it. Upload-Index represents servers' copy of the index for S-no. Maximum-Index represents the index of the last collected record by the sensor for S-no. Each radio maintains a heap based on the priority of the sensors. The peers exchange current records based on the comparison of sensor stream status. From sensor stream status, radio checks if it has newer Upload-Index for any S-no. For any sensor, an older Upload-Index gets updated with a newer index received from another radio. Since value of Upload-Index denotes the sensors' status in the servers' database, therefore, once a radio gets a new Upload-Index it eventually deletes all records for S-no from its memory up to the new Upload-Index. If the new Upload-Index is even higher than its Maximum-Index then the radio updates its Maximum-Index with the value of the new Upload-Index and deletes all the records from its memory for that specific

sensor. The radio reorganizes the heap after receiving the remote sensor stream status.

The Maximum-Index values among the connected radios are also compared. Records of the highest priority sensor are sent first. The radio having higher Maximum-Index for a sensor starts sending records to its partner. The sender starts sending records with index starting from the value one more than receivers' Maximum-Index and stops until index of sent samples reaches the higher Maximum-Index. The records are always disseminated in an ascending monotonic sequence. The receiver updates its Maximum-Index by index of last received record irrespective of normal or undesired termination of connection.

When one of the communicating radios is an upload station, the transmission of samples is only towards the upload station. This is because the upload station does not need to send any record to the radios. If a radio finds higher index value from the upload station for any S-no then it updates its Upload-Index and Maximum-Index with that index value and deletes all samples from its memory. Otherwise, the radio transmits new samples to the upload station and upload station increases index for S-no to the latest index of samples arrived from the radio. An upload station uploads records to Database sink for collection and processing. Both the above mentioned deletions of samples are examples of anti-entropy mechanism applied along with a gossip protocol.

The acquired information is stored in the local memory of Bluetooth radios and will eventually be uploaded to the central storage through mobile radios. Epidemic communication helps to upload samples from peer which is not within range of an upload station for a long period of time by allowing the propagation of samples to the server through in-range peers. The following section describes the algorithm by state machine diagram.

3.3.1 Protocol Description

This section describes the implementation of WRifes communication protocol. In order to simplify the implementation, it is divided into three interconnected subsystems, each responsible for distinct aspects of the protocol. The connection subsystem (fig 3.1) is responsible for the pairing of radios and establishing a peer-to-peer channel. During the period

when radios are connected, two interlocked subsystems are responsible for transmission and reception of messages. After completion of the protocol or unexpected disconnection, the connection subsystem resets the radios back to initial state and again begins the process of pairing.

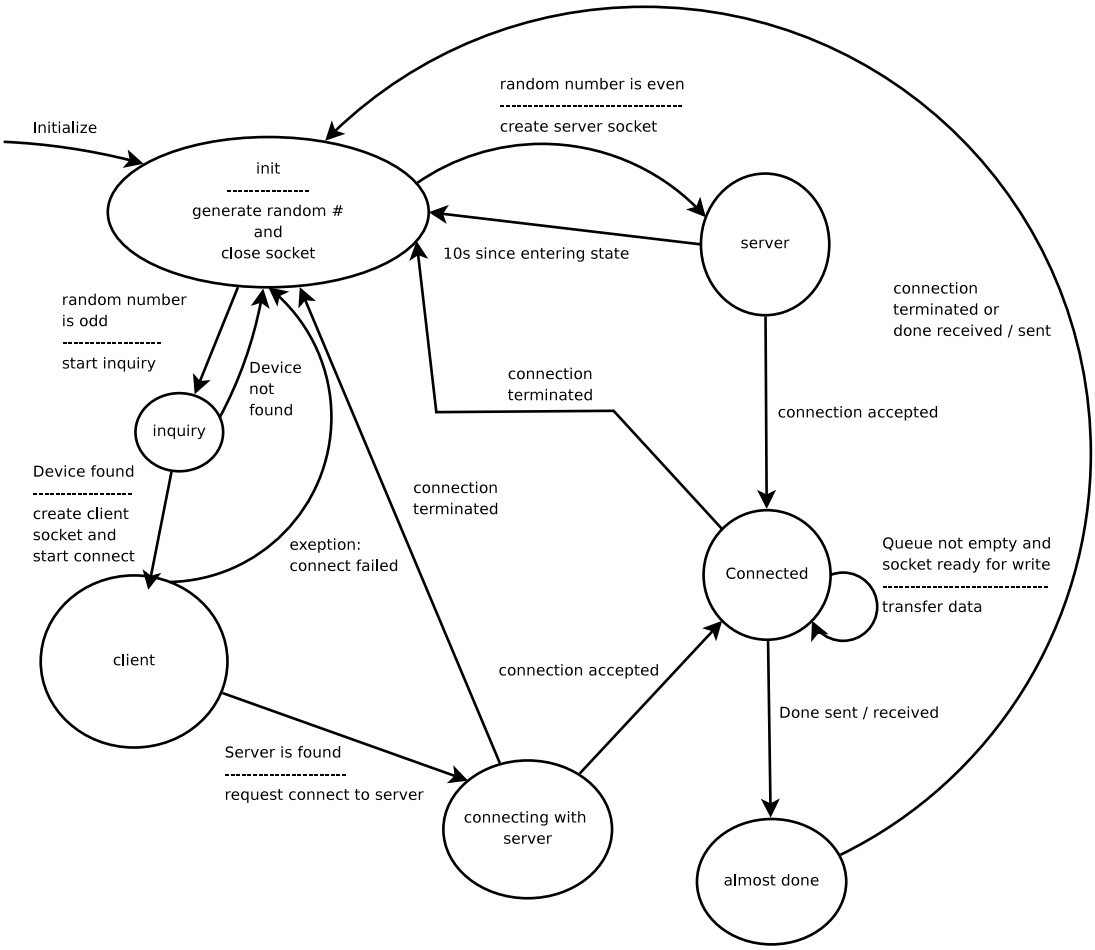

Figure 3.1: Connection state diagram

CONNECTION SUBSYSTEM

WRifes communication protocol is fundamentally peer-to-peer. We used low-cost Bluetooth radios to implement and test the protocol. Unfortunately Bluetooth does not directly

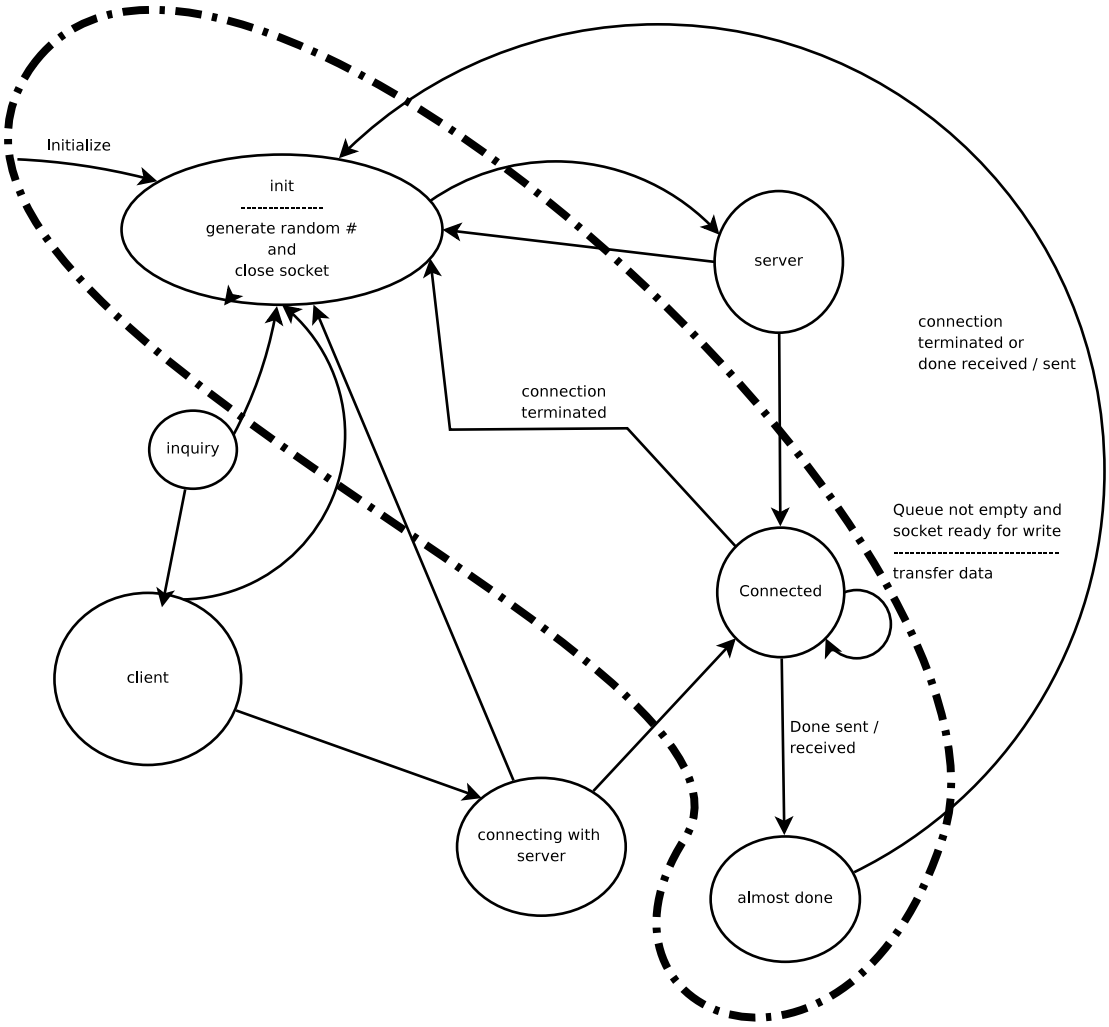

Figure 3.2: Common states of server and client

provide peer-to-peer communication, but instead provides server and client abstractions. In order to create a connection between two Bluetooth radios, one must be set to server mode and the other to client mode, and only a radio in client mode can detect and connect to a server. A priori selection of client or server mode is inappropriate for this protocol because the radios set in the same mode cannot connect. To permit any pair of mules to connect, they repeatedly randomly choose to be a server or client. Once the connection is established between two radios, subsequent communication is peer-to-peer.

SERVER

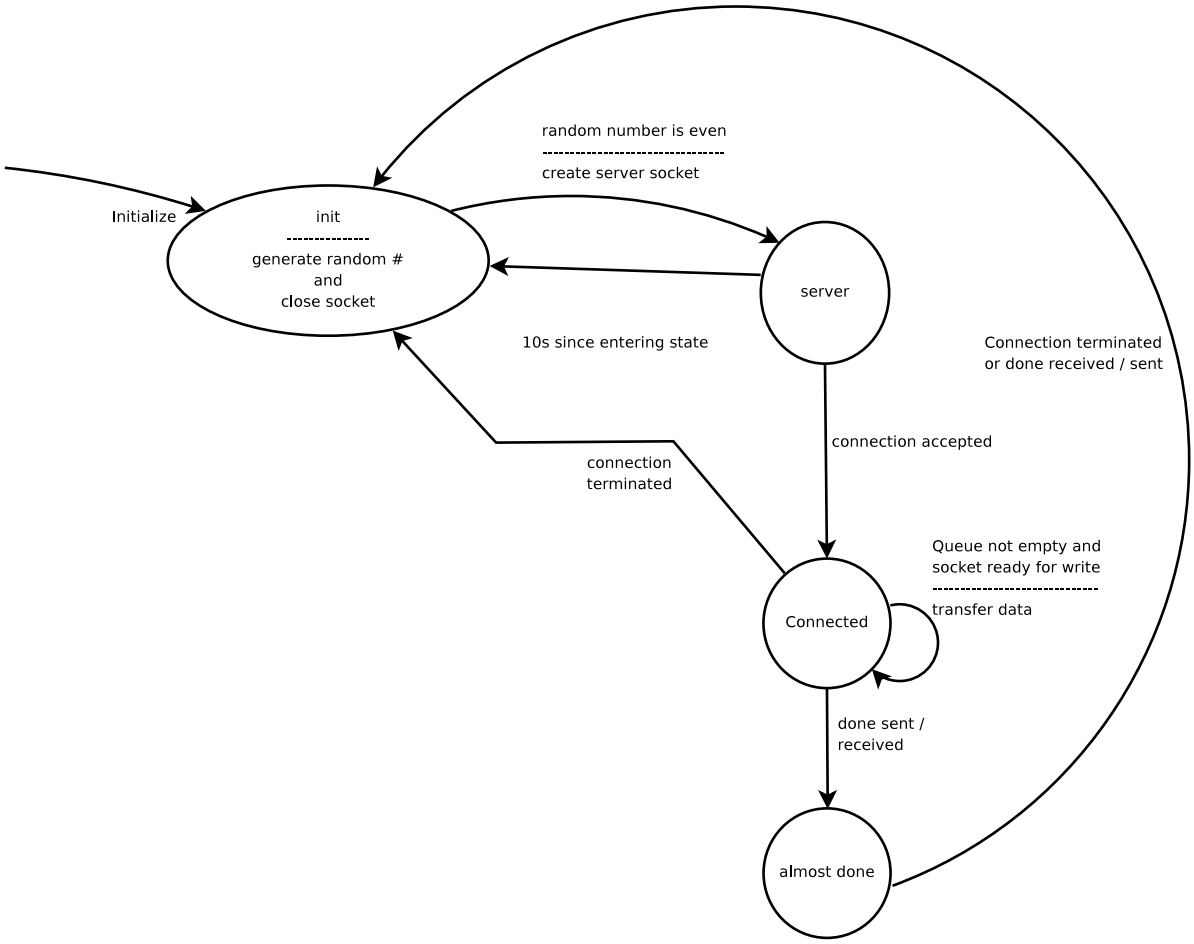

Figure 3.3: Server state diagram

As illustrated by Figure 3.3, the server portion of the connection subsystem is selected by the generation of an even random number. To permit a client to connect, the radio is set into server mode for up to ten seconds. If a client request for connection is received within this timeout, a server attempts to accept the connection and switches to the connected state. Once the connection is established the send and the receive subsystems manage the communication between connected peer. After completion of the protocol, the server is reset back to the initial state and starts the process of pairing again.

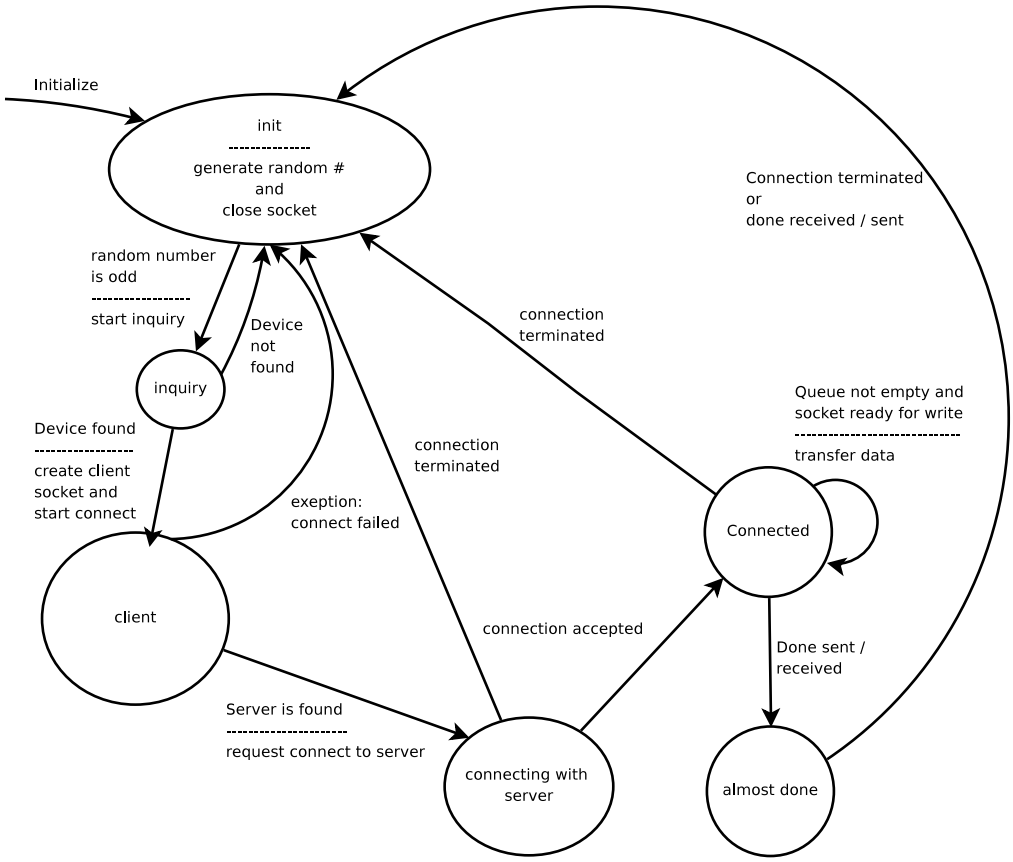

Figure 3.4: Client state diagram

CLIENT

Lines 168 to 191 of the track-soft (see Appendix) program implement the client component of the connection system. If a radio is set in the client mode, it starts searching for nearby radios. If an in range peer is found it creates the client socket and request to connect with that peer. Radios switch back to the initial state if there is no server at that time to connect. Client switches to the connected state once the connection is established.

COMMUNICATION PROTOCOL

fig 3.5 and fig 3.6 provide the regular expression for received samples and sensor stream status. Recived sample contains sensor ID, and one or more tuple consists of delivery

deadline and recorded sample. Sent or Received sample stream contains sensor ID and upload-index, maximum-index and priority of that specific sensor.

Lines 113 to 137 of the track-soft program (see Appendix) implement the receive subsystem. The receive state machine is responsible to identify the regular expressions of sensor stream status (fig. 3.6) and received samples (fig. 3.5).

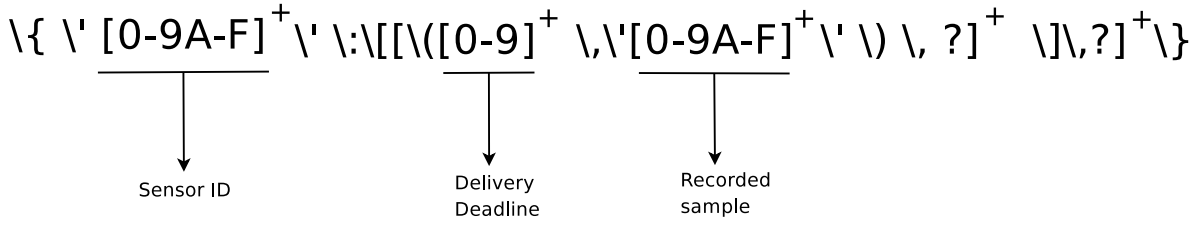

Figure 3.5: Regular expression of received samples

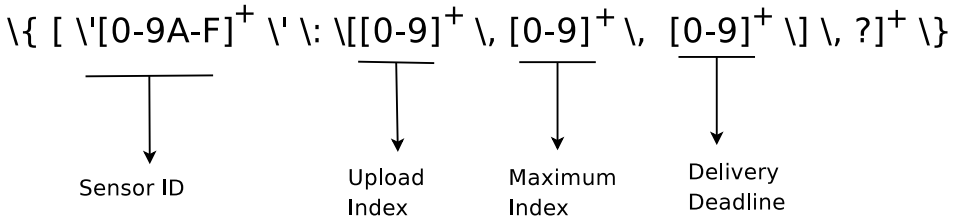

Figure 3.6: Regular expression of sensor stream status

RECEIVE

Parsing of the received sensor stream status and received data is described in fig. 3.7. Device enters in this state when the socket is readable. First the device receives sensor stream status from the connected pair. After receiving sensor stream status successfully, it compares the local and remote sensor stream status and deletes the already uploaded samples from its memory. It stores the necessary entry in heap based on the priority of the samples which need to be transmitted to the connected peer. This is followed by receiving

new records from the other device. Device remains in this state until it receives done from other end.

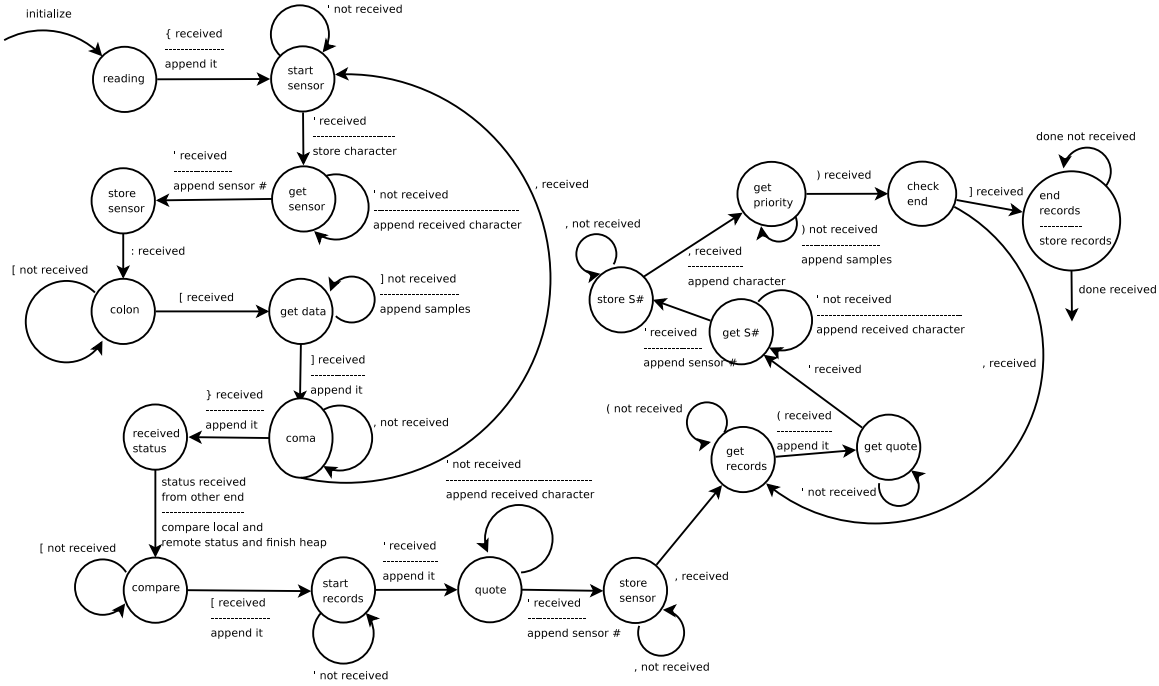

Figure 3.7: Receive state diagram

SEND SUBSYSTEM

Lines 91 to 110 of track-soft program (see Appendix) implement the receive subsystem. Figure 3.8 describes the steps of sending newer samples to the connected peer. A device enters in this state when the socket is writable. First it sends sensor stream status to the connected pair. Samples of the highest priority sensor are sent first. The device having higher Maximum-Index for a sensor starts sending samples in an ascending monotonic sequence to its partner. The sender starts sending samples with index starting from the value one more than receivers Maximum-Index and stops until index of sent samples reaches the higher Maximum-Index or the sensors priority becomes equal to the priority of another sensor.

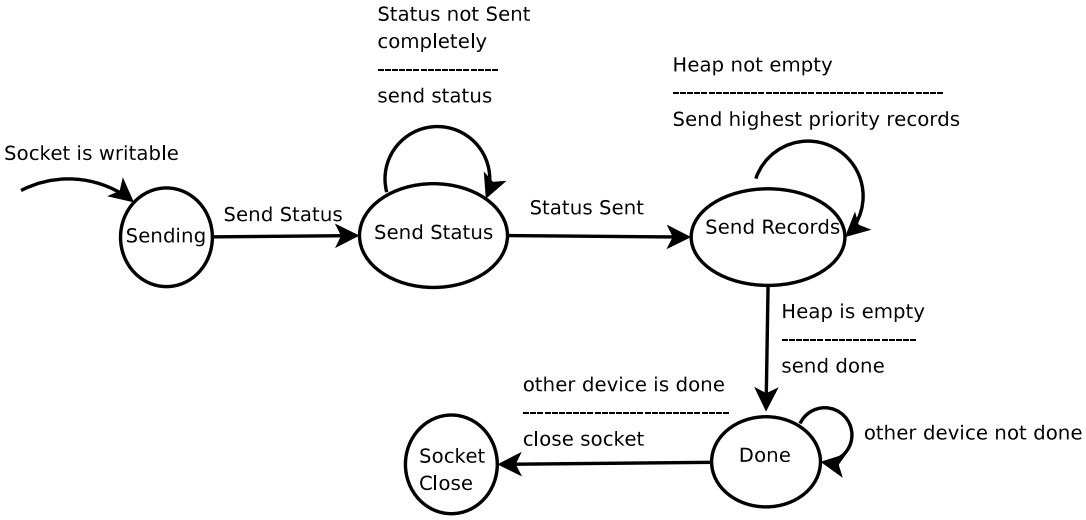

Figure 3.8: Send state diagram

If the sensors priority becomes equal to the priority of another sensor, sender starts sending samples of that other sensor. Device keeps sending records to other peer until all the records are sent and heap is empty. It sends done when the heap becomes empty and waits for the other device to be done. Device closes the socket when it receives done from the other end.

STOP SUBSYSTEM

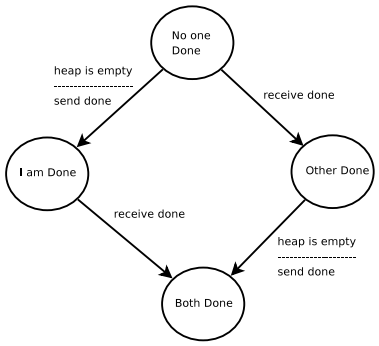

Figure 3.9: State diagram of stop state machine

Fig 3.9 describes the steps of terminating the connection between two connected radios. Lines 105 to 110 and 101, 127,128 implement the stop state machine. If the device has no sample to send and it receives a done from other end, itll close the socket and terminate the connection.

Chapter 4

Experimental Results

4.1 REQUIREMENTS

1. Bluetooth enabled laptops with the ability to run Python 2.6
2. Bluetooth enabled desktops with the ability to run Python 2.6
3. Python 2.6
4. PyBluez 0.18
5. Bluetooth 2.0

4.2 TESTING

The proposed algorithm is implemented through the program, Track-Soft (see Appendix). It carries out all the proposed tasks in right order. The algorithm was tested in Bluetooth enabled laptops and desktops. The devices exchanged records between each other and updated their records as per the algorithm. Bluetooth-enabled devices involved in the testing process belong to class 2, which have a communication range of approximately 10 meters.

In table 4.1 it can be observed that before communication device '001FE2E092C0' had higher Upload-Index and Maximum-Index than device '001FE2E092BD' for both sensor IDs. During communication device '001FE2E092BD' received new records from device '001FE2E092C0' and sensor stream status was updated to its newer value.

Table 4.1: Communication between two peers

Device ID	Device Type	SSS before communication	SSS after communication
001FE2E092C0	Laptop	{'001FE2E092C0':[1,4,2], '001FE2E092BD':[7,8,9]}	{'001FE2E092C0':[1,4,2], '001FE2E092BD':[7,8,9]}
001FE2E092BD	Laptop	{'001FE2E092C0':[0,2,2], '001FE2E092BD':[1,3,12]}	{'001FE2E092C0':[1,4,2], '001FE2E092BD':[7,8,9]}

Table 4.2: Communication between two peers

Device ID	Device Type	SSS before communication	SSS after communication
001FE2E092C0	Laptop	{'001FE2E092C0':[5,9,2000], '001FE2E092BD':[1,3,8000]}	{'001FE2E092C0':[5,9,2000], '001FE2E092BD':[3,7,3000]}
001FE2E092BD	Laptop	{'001FE2E092C0':[0,2,1900], '001FE2E092BD':[3,7,3000]}	{'001FE2E092C0':[5,9,2000], '001FE2E092BD':[3,7,3000]}

Table 4.3: Communication between a peer and an upload station

Device ID	Device Type	SSS before communication	SSS after communication
001FE2E092C0	Laptop	{'001FE2E092C0':[5,9,2000], '001FE2E092BD':[1,3,8000]}	{'001FE2E092C0':[9,9,0], '001FE2E092BD':[8,8,0]}
011FE2BCEF91	Desktop	{'001FE2E092C0':[5,5,0], '001FE2E092BD':[8,8,0]}	{'001FE2E092C0':[9,9,0], '001FE2E092BD':[8,8,0]}

Table 4.2 demonstrates that device having newer records sends data to other device and after communication both devices update their sensor stream status with newer values based on their sensor stream data.

It can be noticed from both tables that after successful completion of the communication both device had same sensor stream status.

Table 4.3 shows the communication between a peer and an upload station. It can be observed that the transmission of records is only towards the upload station and after communication device updated its sensor stream status with the index value received from the upload station.

In conclusion, it was found that every record that was originated in a certain device arrived to other communicating device and finally reached to the servers database, which is the main purpose of this algorithm.

Chapter 5

Conclusion

The contribution of this work is a peer-to-peer wireless data collection algorithm that uses epidemic communication to propagate time-sensitive sequentially sampled data records from sensors toward infrastructure connected upload stations via mobile data mules. In spite of the limited communication range of Bluetooth, epidemic protocol allows the replication and propagation of the collected information to in-range peers, which leads to data transmission to the server where it is stored and processed.

It has been demonstrated that transmission prioritization of samples has been achieved by the use of min-heap.

5.1 FUTURE WORK

Future improvement for the system can be achieved by prioritizing the transmission of sensor stream status.

Bluetooth takes at least eight seconds to establish a connection with another Bluetooth device. It is important for the system to reduce the pairing time by use of other technologies, such as NFC, RFID etc.

It is expected that this work will serve as reference for future applications in the area of wireless data collection.

References

[1] Buddycast: an operational peer-to-peer epidemic protocol stack. In *14th Annual Conf. of the Advanced School for Computing and Imaging*.

[2] Paramvir Bahl and Venkata N. Padmanabhan. Radar: an in-building rf-based user location and tracking system. In *Complexity of Computer Computations*, pages 775–784, 2000.

[3] Geoffrey Blewitt. 6. gps data processing methodology: From theory to applications.

[4] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The bayou architecture: Support for data sharing among mobile users. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 2 –7, 1994.

[5] Alan Demers, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22:8–32, January 1988.

[6] Reetu Dhar. Performance evaluation of gossiping algorithms. In *MS Dissertation*. Southern Connecticut State University, 2009.

[7] Silke Feldmann, Kyandoghere Kyamakya, Ana Zapater, and Zighuo Lue. An indoor bluetooth-based positioning system: Concept, implementation and experimental evaluation. In *International Conference on Wireless Networks*, pages 109–113, 2003.

[8] E. A. Freudenthal, V. Gonzalez, and B. A. Carter. An anti-entropy protocol suitable for managing data deletion in an epidemic data transmission system. In *Proceedings of the Biomedical Engineering Recent Developments.*, pages 1–2, 2008.

[9] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th*

annual ACM/IEEE international conference on Mobile computing and networking, MobiCom '99, pages 174–185, New York, NY, USA, 1999. ACM.

[10] JoAnne Holliday, Divyakant Agrawal, and Amr El Abbadi. Partial database replication using epidemic communication. In *ICDCS*, pages 485–493, 2002.

[11] RFID Journal. Singapore fight sars with rfid, 2003.available at http://rfidjournal.com/article/articleview/520/1/1/. accessed june,2010.

[12] RFID Journal. Taiwan uses rfid to combat sars. 2003.

[13] V.W.-H. Luk, A.K.-S. Wong, R.W. Ouyang, and Chin-Tau Lea. Gossip-based delay-sensitive n-to-n information dissemination protocol. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1 –5, 2008.

[14] F Martinez, P Perea, P Puga, and A Sosa. Radio frequency identification tracking system. 2005.

[15] The Association of the Automatic Identification and Data Capture Industry. Radio frequency identification (rfid), a basic primer. 2009.

[16] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. *SIGOPS Oper. Syst. Rev.*, 31:288–301, October 1997.

[17] Cycle Identification Problem, A. J. Fisher, and Department Of Computer Science. The loran-c.

[18] Douglas B. Terry, Karin Petersen, Mike J. Spreitzer, , and Marvin M. Theimer. The case for non-transparent replication: Examples from bayou. pages 12–20. IEEE Press, 1998.

[19] T.A. Vaattovaara. Global bluetooth tracking system based on mobile phones. 2009.

[20] Robbert Van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. Technical report, Ithaca, NY, USA, 1998.

[21] Spyros Voulgaris and Maarten van Steen. Epidemic-Style Management of Semantic Overlays for Content-Based Searching. pages 1143–1152. 2005.

[22] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, January 1992.

[23] R. Zhou and K. Hwang. Gossip-based Reputation Aggregation for Unstructured Peer-to-Peer Networks. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10, March 2007.

Appendix

```
from bluetooth import *
from heapq import *
import random
from select import *
import thread
import time


#local_mac = '00:1F:E2:E0:92:C0'                    #put the local mac here
sensor_no = ''.join(local_mac.split(':'))
macs = []
times = []
heap = []
DSS = {}
DS2 = {}
S4S = {}
DSS[sensor_no]=[]
uuid = "4cf5d9e7-ac6b-4467-a743-ec4a424bcfee"
dt=[]
rd=""

def add_to_heap(priority, S_n, last):
  heappush(heap, (priority, S_n, last))

def get_first():
  return heappop(heap)

min_I = 0
max_I=0

def inquiry():
        global macs
        global local_mac
        global times
        global sensor_no
        global master
        master = 0
        devices = discover_devices(duration=5, lookup_names=True)
        print devices
        for values in devices:
                macs.append(values[0])
        print "MACS is", macs
        get_ready()

def get_ready():
```

```
47              global macs
48              global DSS
49              for values in macs:
50                      exp=random.randint(1,30000)                     #delivery deadline for the tracked sample
51                      ip_data = (exp,values)
52                      DSS[sensor_no].append(ip_data)
53              create_S4S(DSS)
54
55
56      def create_S4S(data):
57              global min_I
58              global S4S
59              print data
60              for values in data.keys():
61                      a = []
62                      if (len(data[values])>=1):
63                              for valuess in data[values]:
64                                      if(len(data[values])):
65                                              a.append(valuess[0])
66                                      priority = min(a)
67                                      S4S[values]=[S4S[values][0],S4S[values][0]+len(data[values]),priority]
68                      else:
69                              S4S[values]=[0,0,0]
70
71                      print S4S
72                      add_to_heap(S4S[values][2],values,0)
73
74
75
76      def main():
77          global DS2
78          DS2={}
79          while(True):
80              exp=random.randint(1,30)
81              if ((exp % 2)==0):                  #call server or client based on the random number
82                  server()
83              else:
84                  client()
85
86      def Datagram(socket):
87           step_r = 0
88           step_w = 0
89           global S4S
90           global DSS
91           global dt
92           print "Entering Asynchronus Mode"
```

```
 93      while True:
 94          try:
 95            readable, writable, excepts = select( [socket], [socket], [], 1)
 96
 97              if socket in writable:
 98
 99                  if (step_w == 0):
100                      socket.send(str(S4S))#send
101  #                   socket.send(str(1))
102                      print "Init Vector successfully sent"
103                      step_w = 1
104                  if(step_w == 2 and step_r > 0):
105
106                      while(len(heap)>0):
107                        socket.send(str(data_to_send()))#send
108                      socket.send(str(0))
109                      step_w = 3
110                      print "Information about Intersection data successfully sent"
111
112                  if(step_w == 3 and step_r == 2):                #send and receive done
113                      print type(remoteInitVector)
114                      print S4S
115                  print "clossing connection"
116                  socket.close()
117                  break
118
119              if socket in readable:
120                  if (step_r == 0):
121                    print "trying to recieve pair Init Vector"
122                    remoteInitVector = socket.recv(512)#recieve
123                      print "pair Init Vector successfully recieve"
124                      print "pair Init Vector" ,remoteInitVector
125                      recvstatus = recv_status(remoteInitVector)
126  #                   if(eval(remoteInitVector)==1):
127                      step_r = 1
128                      print "pair init vector succesfully receive = ", remoteInitVector
129                  if (step_r == 1 and step_w > 0):
130                    print "recieving remote data"
131                      RemoteStream = socket.recv()
132                      print "adding data in DSS"
133                      recvdata = recv_data(RemoteStream)
134                      if (recvdata == 0):             #done received from other end
135                          step_r=2
136                      print "data successfully recieved"
137                      #data.crypt =  AES.new(KEY,AES.MODE_CBC)
138                      #RemoteStream = data.decrypt(eval(RemoteStream))
```

```
139                         print RemoteStream
140                         #RemoteStream = data.concatenate(RemoteStream)
141                         print "revieved data = %s" % RemoteStream
142
143                 if (step_r == 1 and step_w == 1):
144                         step_w = 2
145             except Exception, e:
146
147                 if (e == "(11, 'Resource temporarily unavailable')" and step_r > 0 and step_w > 0):
148                         main()
149
150
151     def server():
152         global uuid
153         print "Generating the service: %s" % uuid
154         sock=BluetoothSocket( RFCOMM )
155         sock.bind(("",PORT_ANY))
156         sock.listen(1)
157         advertise_service( sock, "wrife", service_classes = [ SERIAL_PORT_CLASS ], profiles =
158     [ SERIAL_PORT_PROFILE ] )
159         sock.settimeout(10.0)                           #timeout is set 10 secs
160         print "Service created Sucsessfully"
161         print "waiting for connection"
162         try:
163           client_sock, client_info = sock.accept()
164           print "connection accepted"
165           sock.setblocking(False)
166           print "Entering Protocol"
167           Datagram(client_sock)
168         except Exception,e:
169           print "server side exception %s" % e
170           sock.close()
171           print "Pair disconnected, Connection Terminated"
172         main()
173
174     def client():
175             try:
176                 global uuid
177             global local_name
178                 sock = BluetoothSocket( RFCOMM )
179                 global macs
180                 global DSS
181                 inquiry()
182                 print "MAC is", macs
183                 print "macs[0] is :", str(macs[0])
184                 try: sock.connect((str(macs[0]), 1))                    #try to connect with the first found device
```

```
185                 except: pass
186             sock.setblocking(0)
187             print "Association successfull"
188                 print "Entering Protocol"
189                 macs=[]
190                 Datagram(sock)
191                 sock.close()
192                 create_S4S(DSS)
193                 print "Connection Terminated"
194                 print "final S4S is", S4S
195             except Exception,e:
196                             print "client caught exception %s" % e
197
198             main()
199
200
201     def data_to_send():             #Send sample until the heap is empty
202       global DS2
203       info = get_first()
204       data=[]
205       if (len(heap) < 1):           #last element in the heap
206             data.append(info[1])
207             data.append(DS2[info[1]][info[2]:])
208
209       temp = get_first()
210       add_to_heap(temp[0],temp[1],temp[2])
211       data = []
212       data.append(info[1])
213       for sample in DS2[info[1]][info[2]:]:
214         if (sample[0] <= temp[0]):
215           data.append(sample)
216         else:
217           if (DS2[info[1]].index(sample)+1 == len (DS2[info[1]])):
218             break
219           add_to_heap(sample[0],info[1],DS2[info[1]].index(sample))
220           break
221
222       return data
223
224     def addSample(data):
225        global min_I
226        global max_I
227        global DSS
228        global e
229        global S4S
230        if data[0] not in DSS.keys():
```

```
231                     DSS[data[0]]=[]
232                     for i in range(1,len(data)):
233                             DSS[data[0]].append(data[i])
234                     S4S[data[0]] = [min_I, max_I, data[1][0]]
235     else:
236             for i in range(1,len(data)):
237                     DSS[data[0]].append(data[i])
238             max_I = max_I+len(data[1:])
239             S4S[data[0]] = [min_I, max_I, data[1][0]]
240     e = ""
241
242 def addSample_forsend(data):
243     global min_I
244     global max_I
245     global DS2
246     global e
247     global S4S
248     if data[0] not in DS2.keys():
249                     DS2[data[0]]=[]
250                     for i in range(1,len(data)):
251                             DS2[data[0]].append(data[i])
252     else:
253             for i in range(1,len(data)):
254                     DS2[data[0]].append(data[i])
255
256 def comp(status):
257             global S4S
258             ls=S4S
259             for items in S4S.keys():
260                     if items in status.keys():
261
262                             if (status[items][0] > ls[items][0]):
263                                     print 'local status has lower min_I'
264                                     print "previous S4S is", S4S
265                                     S4S[items][0]=status[items][0]
266                                     print "now S4S is", S4S
267
268
269                             if (ls[items][1]>status[items][1]):
270                                     diff = ls[items][1]-status[items][1]
271                                     print diff
272                                     send=DSS[items][-diff:]
273                                     print send
274                                     print "len of send is:", len(send)
275                                     data=[]
276                                     data.append(items)
```

```
277                                     print "send[0] is :", send[0]
278                                     for i in range (0,len(send)):
279                                             print "send[i] is", send[i]
280                                             data.append(send[i])
281                                             print "in main data is:", data
282                                     addSample_forsend(data)
283
284                     if items not in status.keys():
285                             print "remote doesnt have that key", items
286                             data = []
287                             data.append(items)
288                             for i in range (0,len(DSS[items])):
289                                     data.append(DSS[items][i])
290                             print "key is not present"
291                             print data
292                             addSample_forsend(data)
293
294             for items in status.keys():
295                     if items not in S4S.keys():
296                             S4S[items][0] = status[items][0]
297
298
299     def recv_data(v):
300             global rd
301             zero = 0
302             w=0
303             for i in range (0,(len(v))):
304
305                     if (w==0 and v[i]=='['):
306                             rd = rd + v[i]
307                             w=1
308                     if (w==1 and v[i]=='\'):
309                             rd = rd + v[i]
310                             w=2
311                     if (w==2 or w==3):
312                             if (v[i]!='\'):
313                                     rd = rd + v[i]
314                                     w=3
315                     if (w==3 and v[i]=='\'):
316                             rd = rd + v[i]
317                             w=4
318                     if (w==4 and v[i]==','):
319                             rd = rd + v[i]
320                             w=5
321                     if (w==5 and v[i]=='('):
322                             rd = rd + v[i]
```

```
323                     w=6
324             if (w==6 and v[i]!='('):
325                     rd = rd + v[i]
326                     w=7
327
328             if (w==7 and v[i]!=','):
329                     rd = rd + v[i]
330                     w=7
331             if (w==7 and v[i]==','):
332                     rd = rd + v[i]
333                     w=8
334             if (w==8 and v[i]=='\''):
335                     rd = rd + v[i]
336                     w=9
337             if (w==9 or w == 10):
338                     if( v[i]!='\''):
339                             rd = rd + v[i]
340                             w=10
341
342             if (w==10 and v[i]=='\''):
343                     rd = rd + v[i]
344                     w=11
345
346             if (w==11 and v[i]==')'):
347                     rd = rd + v[i]
348                     w=13
349
350             if (w==13 and v[i]==','):
351                     rd = rd + v[i]
352                     w=5
353             if (w==13 and v[i]==']'):
354                     rd = rd + v[i]
355                     w=14
356                     addSample(eval(e))
357                     rd=""
358
359             if (w==14 and v[i]=='0'):       #done received
360                     return zero
361
362
363
364 def recv_status(s):
365         e=""
366         z = "1"
367         w=0
368         for i in range(0,len(s)):
```

```
369
370                 if (w==0 and s[i]=='{'):
371                         e = e+s[i]
372                         w=1
373
374                 if (w==1 and s[i]=='\"'):
375                         e=e+s[i]
376                         w=2
377
378                 if (w==2 or w==3):
379                         if (s[i]!='\"'):
380                                 e=e+s[i]
381                                 w=3
382                 if (w==3 and s[i]=='\"'):
383                         e=e+s[i]
384                         w=4
385                 if (w==4 and s[i]==':'):
386                         e=e+s[i]
387                         w=5
388                 if (w==5 and s[i]=='['):
389                         e=e+s[i]
390                         w=6
391                 if (w==6 or w==7):
392                         if (s[i]!='[' and s[i]!=','):
393                                 e=e+s[i]
394                                 w=7
395
396                 if (w==7 and s[i]==','):
397                         e=e+s[i]
398                         w=8
399
400                 if (w==8 or w==9):
401                         if (s[i]!=','):
402                                 e=e+s[i]
403                                 w=9
404
405                 if (w==9 and s[i]==','):
406                         e=e+s[i]
407                         w=10
408
409                 if (w==10 or w==11):
410                         if (s[i]!=',' and s[i]!=']'):
411                                 e=e+s[i]
412                                 w=11
413
414                 if (w==11 and s[i]==']'):
```

```
415                     e=e+s[i]
416                     w=12
417
418             if (w==12 and s[i]==','):
419                     e=e+s[i]
420                     w=1
421
422             if (w==12 and s[i]=='}'):
423                     e=e+s[i]
424                     comp(eval(e))
```

Curriculum Vitae

Avranil Tah was born on May 24, 1986 in Chandannagar, West Bengal, India. He received his Bachelor's Degree in Information Technology from Haldia Institute of Technology, India. In the fall of 2008 he enrolled in The University of Texas at El Paso to pursue a Master's Degree in Computer Science. He plans on a fall 2010 Graduation and plans to join PhD in The University of Texas at El Paso.