Tufts University
College of Engineering
Department of Electrical Engineering & Computer Science

Bluetooth Email Application

Martin Shek
Tulyatep Uawithya

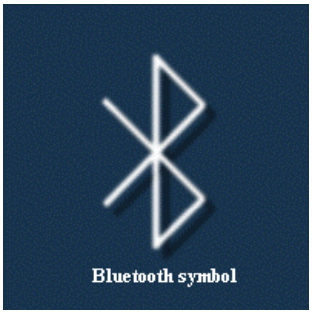
Advisor: Professor C. Hwa Chang
December 28, 2001

I. Abstract

We design and develop an email application using the Bluetooth technology as an Internet-bridge. Using the Ericsson Bluetooth Application Development Toolkit and its model applications, our email application has two main components, BT_EmailServer and BT_EmailClient. The BT_EmailServer component resides on a computer that does have network connection (i.e. Internet connection). The BT_EmailClient component resides on a computer or PDA that does not have network connection. Through the Ericsson Bluetooth modules, the two components are able to communicate and transmit data with each other.

II. Introduction

Bluetooth is essentially a cable-replacement technology. This technology allows electronic devices such as computers, PDAs, mobile phones to communication with each other wirelessly, without the burden of cables and wires. Bluetooth uses short-range radio chips for communications. Some of the advantages of the Bluetooth chips are robust, low power, low cost and they can operate in noisy frequency environments. The technology uses the ISM band at 2.4 GHz. This frequency is reserved for industrial and scientific communications in many countries.



III. Why Bluetooth?

When any two devices want to communicate with each other, they need to operate with the same protocol. That specific protocol ensures the devices to agree on a standard procedure before any data transmission can occur. The first issue is the medium of communication: wire or wireless. The protocol will have the standard for the data exchange.

- Serial communications: the data can be sent one bit at a time.
- Parallel communications: the data can be sent in groups of bits.

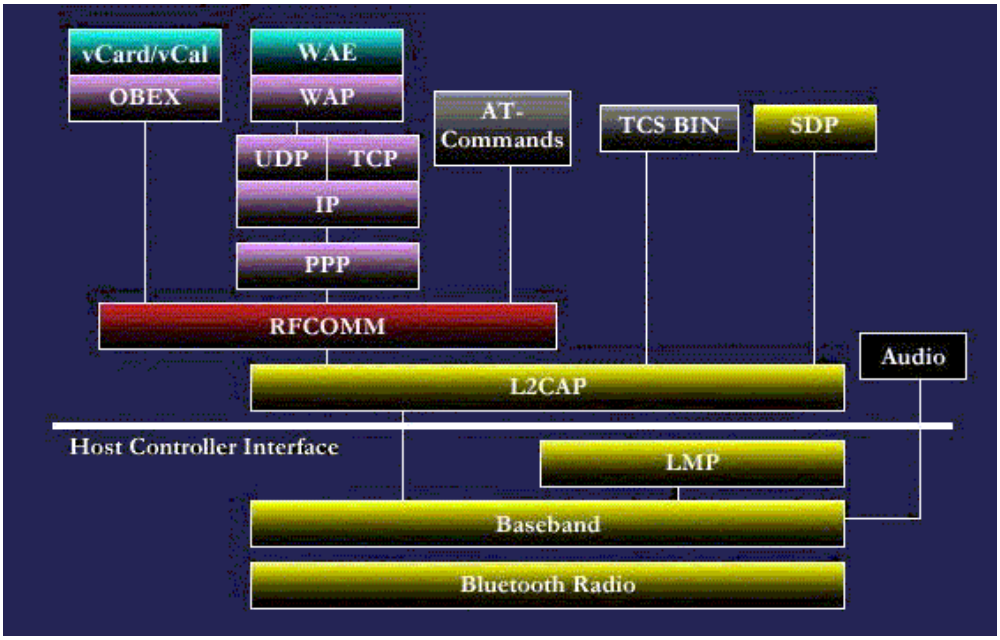
Different devices uses different scheme to exchange information. The predetermined protocol would govern the mean of communication. In addition, the devices need to know how to send, receive and interpret the data being exchanged. Bluetooth is a standard protocol that will allow all electronic devices to talk and interact with each other without the users to have to worry about which type of cable to connect or which protocol standard to use.

Bluetooth is not only a luxury to have, but this type of technology is essential nowadays. Companies that manufacture computers, entertainment systems and other electronic devices have different technical specifications from each other, with Bluetooth; these devices would be more users friendly and more compatible.

IV. Bluetooth Specifications

The Bluetooth technology is developed by a group of electronics manufacturers (ex: Intel, Toshiba, Motorola) worldwide creating the Bluetooth Special Interest Group (SIG). Bluetooth describes the protocol of a short-range frequency-hopping radio link between devices. Bluetooth radio modules operate in the unlicensed ISM band at 2.4GHz, and avoid

interference from other signals by hopping to a new frequency after transmitting or receiving a packet. Compared with other systems in the same frequency band, the Bluetooth radio hops faster and uses shorter packets.



A. Bluetooth Radio

Bluetooth radio is a short-distance, low-per radio that operates in the unlicensed spectrum of 2.4 GHz. It uses a nominal antenna power of 0dbm and so the range of communication is 10 meters. The Bluetooth radio spreads the spectrum by using frequency hopping (FH). It uses 79 hops spaced 1 MHz apart, starting at 2.402GHz and finishing at 2.480GHz. The hop rate is 1600 hopes per second, which means a single hop slot of 625microseconds. The maximum gross data rate of 1 Mbps can be achieved, but overhead reduces the practical data rate to approximately 721 kbps.

For less interference from other devices operating at this frequency, fast frequency and small data packets are used. For voice, CVSD is used which can withstand high bit error rates. For packet headers a highly redundant error correction is used.

B. Baseband

The baseband layer controls the radio. Baseband is the hardware that turns received radio signals into a digital form, which can be processed by the host application. It also converts digital or voice data into a form that can be transmitted using a radio signal. Baseband provides a way with which devices can synchronize their clocks and establish connections.

In the stack, it's the layer right above the Bluetooth radio layer. It manages both ACL (Asynchronous Connection-Less links) and SCO (Synchronous connection-Oriented) links, handles packets and does paging and inquiry to access and inquire other Bluetooth devices in the range. It uses a time-division duplex (TDD) to transmit and receive data.

C. Link Manager Protocol (LMP)

The link manager software runs on a microprocessor and manages the communication between Bluetooth devices. The LMP handles piconet management, link configuration and security functions. A piconet is a group of devices connected to a common channel. This common channel is identified with its unique hop sequence. The one that initiates the connection is called master and rest of them are slaves. The maximum number of active members in a piconet is eight, but many more parked devices can be connected to a piconet. The channel is managed by the master with the help of link managers on each device.

To communicate with each other, the devices must establish a piconet. Devices can be part of many piconets. The LMP provides the functionality to attach/detach slaves, switch roles between a master and a slave and to establish ACL/SCO links. LMP also handles the low power modes-hold, sniff and park, designed to save power when the device doesn't have data to send. Link configuration asks include setting link parameters, quality of service and power control if the device supports it. Authentication of devices to be linked and managing link keys is also taken care by LMP.

D. Host Controller Interface (HCI)

It provides command interface to the baseband controller and link manager, and access to hardware status and control registers. It resides on the Bluetooth hardware and accepts communications over the physical bus. The HCI is functionally broken up into 3 parts: HCI firmware, HCI driver and Host Controller Transport Layer.

HCI firmware is located on the actual Bluetooth hardware and makes it possible for the hardware to access baseband commands, link manager commands, hardware status registers, control registers and event registers. HCI Driver is located on the software entity, Host and is responsible for notifying the events. Host Controller Transport Layer makes it possible for HCI driver and firmware to communicate. This layer provides the ability to transfer data transparently.

E. Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP is used for protocol multiplexing above the basic Bluetooth layers, for packet segmentation and reassembly, and to convey Quality of Service

information. Via protocol multiplexing this layer allows multiple applications to use a link between two devices simultaneously. It also reduces the size of packets provided by the application to the size of packets accepted by Baseband. The maximum packet size that L2CAP can accept is 64 Kbytes and it reduces it to 2745bits, which is the acceptable size for Baseband.

L2CAP is also responsible for the reverse procedure meaning that it combines the small packets from the Baseband layer to present it to the application. L2CAP allows applications to demand Quality of Service on certain parameters link peak bandwidth, latency and delay variation. L2CAP checks if the link is capable of providing it and provides it if possible. Basically, L2CAP provides the network layer function to application and higher protocols.

F. RFCOMM

This protocol is a simple transport protocol, which provides emulation of serial ports over the L2CAP protocol. 60 simultaneous connections between two Bluetooth devices can be supported by this protocol. RFCOMM has to accommodate two different types of devices. The first type of devices is the communication endpoints and the Second type of devices is the part of the communication segment such as modems.

G. Service Discovery Protocol (SDP)

This protocol provides a means for applications to discover which services are available and to determine the characteristics of those available services. This protocol is needed since Bluetooth devices are supposed to detect other Bluetooth enabled devices and the devices available in the range can change dynamically. For discovery purposes, SDP uses request/response model where each transaction consists of one request protocol data unit (PDU) and one response PDU. All the SDP PDU's consist of a header and PDU-specific parameters. The header contains the following three fields: PDU ID which identifies the type of PDU, TransactionID which identifies request PDU's and is used to match response PDU's to request PDU's, ParameterLength which specifies the Length of all parameters contained in the PDU. For the cases, when response cannot fit in a single response PDU, the SDP generates a partial response along with a continuation state parameter.

SDP contains service records of all the services that are maintained by it. The service record consists of service attributes, which in turn consist of two components called attribute ID and an attribute value. The attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. The attribute value is a variable length field whose meaning is determined by the attribute ID associated with it.

Whenever a client wants to browse an SDP server's services, it creates a service search pattern containing the UUID (universally Unique identifier) that represents the root browse group. All services that may be browsed at the top

level are made members of the root browse group by having the root browse group's UUID as a value with in the BrowseGroupList attribute.

V. Avoid Interference

One of the ways Bluetooth devices will avoid interfering with other systems is by sending out very weak signals of 1 milliwatt. By comparison, the most powerful cell phones can transmit a signal of 3 watts. The low power limits the range of a Bluetooth device to about 10 meters, cutting the chances of interference between your computer system and your portable telephone or television. Even with the low power, the walls in your house won't stop a Bluetooth signal, making the new standard useful for controlling several devices in different rooms.

With many different Bluetooth devices in a room, one might think the devices would easily interfere with one another, but it's unlikely that several devices will be on the same frequency at the same time because Bluetooth uses a technique called spread-spectrum frequency hopping. In this technique, a device will use 79 individual randomly chosen frequencies within a designated range, changing from one to another on a regular basis. In the case of Bluetooth, the transmitters change frequencies 1,600 times every second, meaning that more devices can make full use of a limited slice of the radio spectrum. Since every Bluetooth transmitter uses spread-spectrum transmitting automatically, it is unlikely that two transmitters will be on the same frequency at the same time. This same technique minimizes the risk that portable phones or radios will disrupt Bluetooth devices since any interference on a particular frequency will last only a tiny fraction of a second.

When new Bluetooth-enabled devices come within range of one another, an electronic detection will take place to determine whether they have data to share or whether one needs to control the other. The user doesn't have to press a button or give a command -- the electronic detection happens automatically. Once the detection has occurred, the devices -- whether they are part of a computer system or a stereo -- form a network. Bluetooth systems create a Personal-Area Network (PAN) or "piconet" that may fill a room. Once a piconet is established, the members randomly hop frequencies in unison so they stay in touch with one another and avoid other piconets that may be operating in the same room.

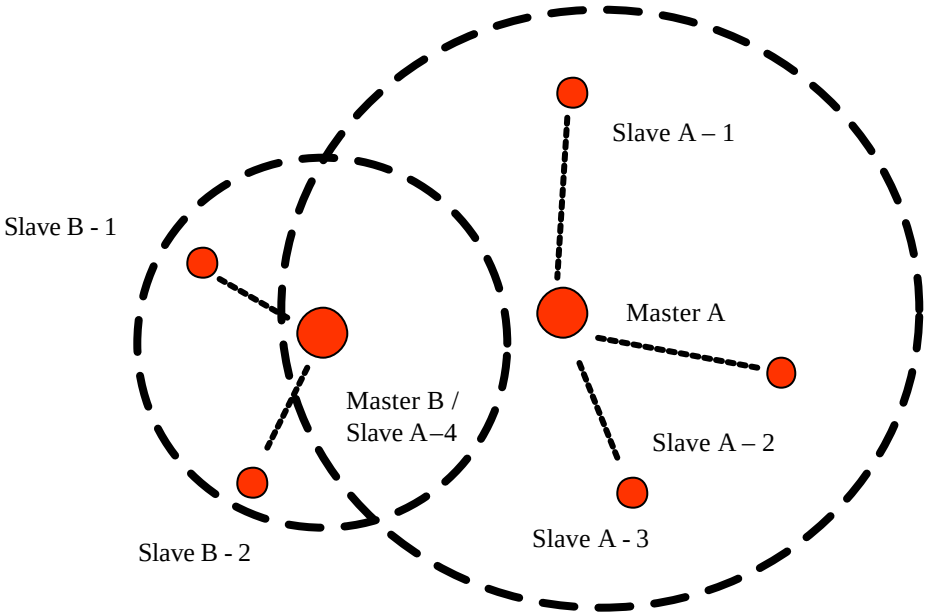
A piconet has a master and up to seven slaves. The master transmits in even time slots, slaves in odd time slots. The packets can be up to five time slots wide, and the data in a packet can be up to 2,745 bits in length. There are two types of data transfer in Bluetooth technology

- SCO: Synchronous Connection Oriented Link

  A synchronous connection for reserved bandwidth communications (voice). Up to three simultaneous synchronous voice channels can be supported. Each voice channel can support 64 kbps links. Voice channels use the Continuous Variable Slope Delta Modulation (CVSD) voice-coding scheme, which is a very robust scheme in handling dropped and damaged voice samples.

- ACL: Asynchronous Connection-Less Link

  An asynchronous connection and is used primarily to transmit ACL packet data. At a time, devices can support only one ACL link and data packets can be retransmitted. The data channel can support 721 kbps in either direction while permitting 57.6 kbps in the return direction, or a 432.6 kbps symmetric link. ACL is a point-to-point (master to one slave) or broadcast to all the slaves. ACL slaves can only transmit when requested by master. Slots not reserved for SCO links can be used for ACL links.



VI. Email

A. How email was created

Ray Tomlinson who worked for Bolt Beranek and Newman (BBN), the company hired by the United States Defense Department in 1968 to build ARPANET. ARPANET is a growing multimillion-dollar network launched in 1060 by the Defense Department's Advanced Research Projects Agency (ARPA)

with the aim of electronically linking dozens of major computer science labs throughout the country. At the time, the ARPANET consisted of 15 nodes, located at places like UCLA in California, the University of Utah in Salt Lake City, and at BBN in Cambridge, Massachusetts.

In 1971, Tomlinson was tinkering around with an electronic message program called SNDMSG, which he had written to allow programmers and researchers who were working on Digital PDP-10s, one of the early ARPANET computers, to leave messages for each other.

But that program was not an email. There was a number of then existing electronic message programs, the oldest dating from the early 1960's. SNDMSG only worked locally. The way the program worked was it allowed the exchange of messages between users who shared the same machine. The users could create a text file and deliver it to a designated "mail box". A mailbox is also a file, the user can add anything to the mailbox but they cannot erase what was already there.

Tomlinson was also working on an experimental file transfer protocol called CYPNET, which transfers files among linked computers at remote sites within ARPANET. He then thought that the CYPNET could add material to a mailbox file as the SNDMSG.

What he did was he adapted the CYPNET to use SNDMSG to deliver messages to mailboxes on remote machines through the ARPANET. Tomlinson also chose the @ symbol to distinguish between messages addressed to mailboxes in the local machine and messages that were headed out onto the network. Then he sent an email to himself by using two PDP-10 computers, which were wired together through the ARPANET. Basically that was the first email that ever sent through a network, from computer to computer.

B. What is email?

Electronic mail or email is the a term given to an electronic message, usually a form of simple text message, that a user types at a computer system and is transmitted over some form of computer network to another user, who can read it.

Earlier when email was invented, the email systems can only be sent and receive in an office where every person was equipped with the same software. Then, with the expansion of the Internet, the manufacturer of email systems introduced the capability to connect to the Internet for transferring message outside of the local network. This can be done by using some software interface that converts the local messages into a recognized standard form suitable for transfer over the Internet.

Email servers exchange messages using the SMTP protocol. Client applications log into servers to send and receive email with a variety of protocols, commonly including POP3, IMAP and MAPI.

C. Email addresses

Early email servers passed email to other institutions over dial-up telephone modem links. Therefore each server had to know the path of computer each email had taken in its way there, so it could send a reply back using the same path.

These paths were stored in what was called a “bang” notation, with the “!” character separating each computer in a path, and were recorded at the top of each email, so you could tell which computers the message had traveled through on its journey, as in the following real bang path:

utzoo!decvax!harpo!eagle!mhtsa!ihnss!ihuxp!grg

Then the domain names were introduced. They will be used instead of the “bang”. Addresses came to be associated with a single known domain, independent of the path taken to get there, An email address is unique for each domain name, and expressed in the form “name@domain”, as follows:

jonathan@tufts.edu
nokia@hotmail.com

Some email addresses specify more than one level of domain name:

Bobby@mail.server.gis.net

Each Internet domain has a single email server that maintains the accounts for users with addresses at that domain. Since there is a one-to-one mapping between domain names and email servers, any email server can reach any other email server using the standard routing protocols built-in to the Internet network.

D. Email servers

Each Internet domain has a corresponding email server. When you send an email, you first send it to your email server, which then contacts the addressee's email server. The two email servers then hold a brief conversation according to the rules defined by the Simple Mail Transfer Protocol (SMTP). Your server will ask the other server if the user name is valid, and, if it is, then the email will be transferred, and the other server stores the email until the addressee logs on and downloads it.

By far the most common SMTP server in use is the venerable sendmail system, first distributed for free with the Unix operating system. The lists of commands that can be exchanged during an SMTP session between two email servers are listed below. The first command of an SMTP conversation must be the HELO command. A mail transaction is begun with the MAIL, SEND, SOML, or SAML commands. The last command in a session must be the QUIT command.

Command	Expanded Command
DATA	DATA
EXPN	EXPAND
HELO	HELLO
HELP	HELP
MAIL	MAIL
NOOP	NOOP
QUIT	QUIT
RCPT	RECIPIENT
RSET	RESET
SEND	SEND
SAML	SEND AND MAIL
SOML	SEND OR MAIL
TURN	TURN
VRFY	VERIFY

F. Email clients

An email client communicates with a server to login, get mail status and send and receive email. There are three most common client connection protocols, POP, IMAP and MAPI.

- **POP**

Also known as Post Office Protocol. POP3 has become the most common email client connection protocol. The POP3 protocol enables any email program anywhere on the Internet to connect to any email server to perform the usual

email functions, such as reading and sending, as long as they have a valid account and password. We can Telnet into our email provider on port 110 and exchange POP3 commands with our server just as though we were an email application. POP3 is an open Internet standard. The common POP3 commands and responses are listed in the following table:

Command	**Responses**
getwelcome ()	Gets the greeting from the server.
user (username)	Login with a username. If valid username, server will respond with request for password.
pass_ (password)	Send password. If valid, server response will be two numbers, message count and mailbox size.
stat ()	Get the mailbox status. Response is two numbers, message count and mailbox size.
list ([message])	Get list of messages. An option "message" gets information on a specific message.
retr (message)	Get message number "message".
dele (message)	Delete message number "message".
rset ()	Remove all deleted message markings.
noop ()	No operation. Do nothing. Really. Needed in unusual programming situations.
quit ()	Quit. Commits all changes, unlocks the mailbox, and ends the server connection.
top (message, lines)	Gets just the first "lines" number of lines of message number "message". Useful on low bandwidth lines to get just the first part of long messages.
uidl ([message])	Gets a unique id list -- a message digest including unique ids. The option gets the unique id for the specific message "message".

- **IMAP**

The Internet Message Access Protocol (IMAP) is a more modern protocol than POP3, and was first developed at Stanford University in 1986. The current version is IMAP4, and provides similar services to the POP3 protocol, but

includes additional features such as the ability to keep your email on the server after downloading a copy to read locally.

The IMAP features can be useful in several situations, for example when you are traveling and don't want to download your email onto a laptop, because then you won't have them on your home computer. It can also be useful to use on low-bandwidth devices like PDA, enabling you to select a few emails from a list of subject headers before downloading just the ones you want.

- **MAPI**

The Messaging Application Programming Interface is a standardized set of C functions originally developed by Microsoft and supported by many third party vendors for Windows platforms.

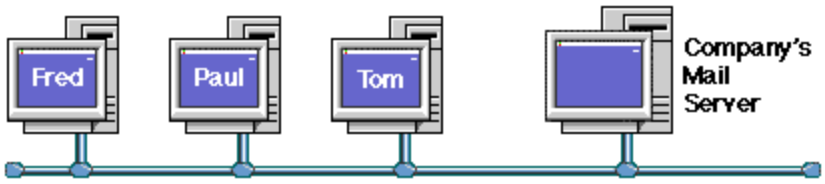
G. Overview of Email Services

The followings are over of email services taken from http://www.vicomsoft.com.

The following examples will start with a fictional computer network, and will lead through the basics of how email functions, and it's relevance to the Internet. An example email system to illustrate the basics could be as follows:

Simple office email system

Email is required within a company, but not out to the rest of the world. A very simple email system could be installed and maintained, giving interoffice communications:



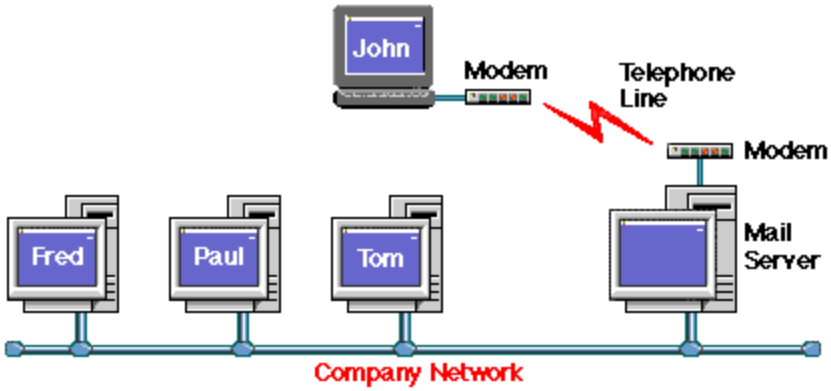
In the above example, the three workstations are connected to a computer network within a company office. If one user wishes to send email to another user, then the message is simply typed and sent to the mail server, addressed to the recipient using their email name, which would simply be the first name of a user, such as "Tom".

For example: if Fred wants to send a message to Tom, he types his message on his email client, addressing it to Tom. His email client then sends the message to the mail server, where it is stored for Tom. When Tom next checks to see if there is any mail for him, his email client will collect his messages and allow him to read them. Because this email system works only within the office, each recipient can be referred to using only their email name.

This system could easily be expanded to allow for remote users if some form of dial in support is added to the network using a modem (A modem is a device

that sends computer signals down a telephone line, effectively making a telephone system a part of a computer network). This would increase the flexibility of the system enormously.

Remote user with access to office email

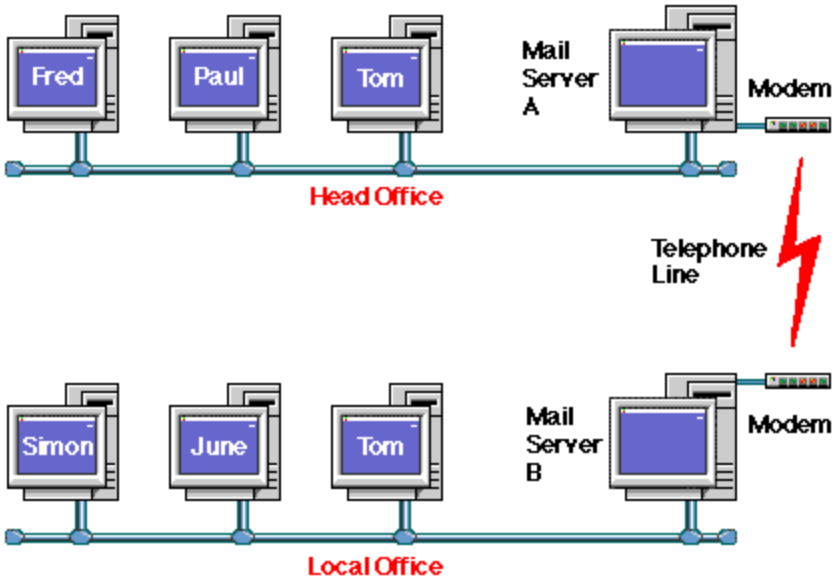


When John wants to send email to Paul, he types his message within his email client, then, when he wants it to, his email client dials into his company computer network using a modem and telephone line, sends his message to Paul, then collects any waiting messages for him. Once the messages have been collected, the modem disconnects from the phone line, and John can read any messages that were collected.

Because John's computer connected using the telephone lines, he can collect his mail from anywhere he can plug his modem into a telephone socket. If the company also had another remote user who also connected through the telephone network, then messages could also be transferred to them as easily as to one of the workstations in the office. The advantages and flexibility of an email system starts to become clearer when compared against traditional memo and telephone systems.

The next step is to allow email messages to be able to be sent to another office or company.

Simple email between two offices



This diagram shows a simple email system to give internal email between two offices, which are connected via a telephone line.

Mail is sent internally within an office using the same methods as discussed earlier, but as there are two separate sites, this adds an additional complication in addressing the recipient. As can be seen in the above diagram, there are two Toms available to send email to. How can you specify the correct Tom to send your message to? There are two ways:

1. Change Tom's email name to be something else. This usually is implemented by using the users second name or initial, such as "S" of the second name "Smith", so the second Tom's email name would be "Tom.S" (there is no actual standard way of implementing email names, other than trying to keep them short and easy to remember).

2. Refer to users at a separate office or site with an additional piece of information, which defines their location, such as "local office." So to send mail to Tom at the Local Office, you would address his messages to "Tom@local.office". Notice the "@" symbol which is read as "at" and that there are no spaces allowed within an email name or address.

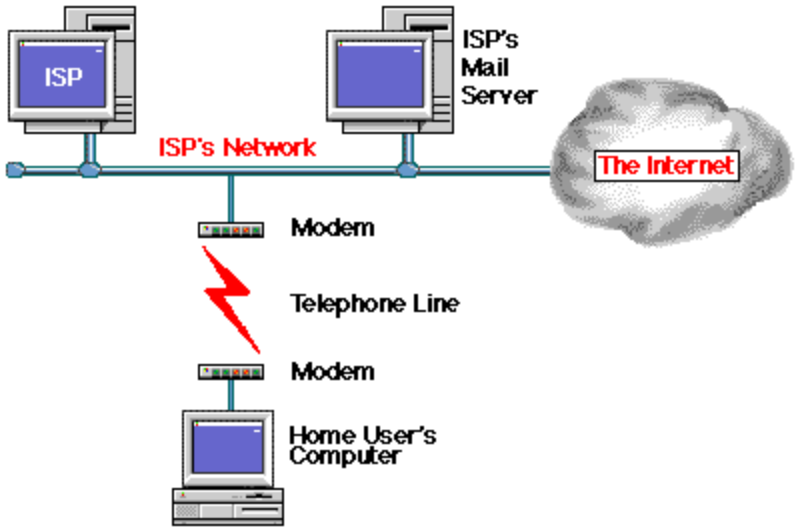
The second method is the preferred option as it allows for future expansion of the system, especially if there is the potential for a number of local offices. These

could be referred to as "Local.Office.A" "Local.Office.B" or possibly by location, such as "New.York.Office" and so forth. These addresses are known as "Domains" and simply give the location of the user who the message is destined for within the company. (Note that these are not "Internet Domains", but internal company ones).

Note that the telephone line would only be used intermittently, when mail was destined for the other office, and could also be used for remote users as well. Using a combination of the discussed options so far, it can be seen that a comprehensive company email system can be assembled without too many problems.

The options discussed so far only allow for internal email with a company or organization. The next example is to allow for email access to the global Internet.

A single user dialing into an Internet Service Provider (ISP)



When a single user dials into the Internet via an Internet Service Provider (ISP) they are effectively dialing into the ISP's network in the same way as in the earlier example "Remote user with access to office email."

The only major difference is that the ISP's computer network is itself connected to the Internet, and may have a large number of modems to support their users.

The home users email is stored at the ISP's mail server in exactly the same way as within the simple company email system introduced above. The home user can connect to the ISP's network, send their messages and collect their waiting email, then disconnect.

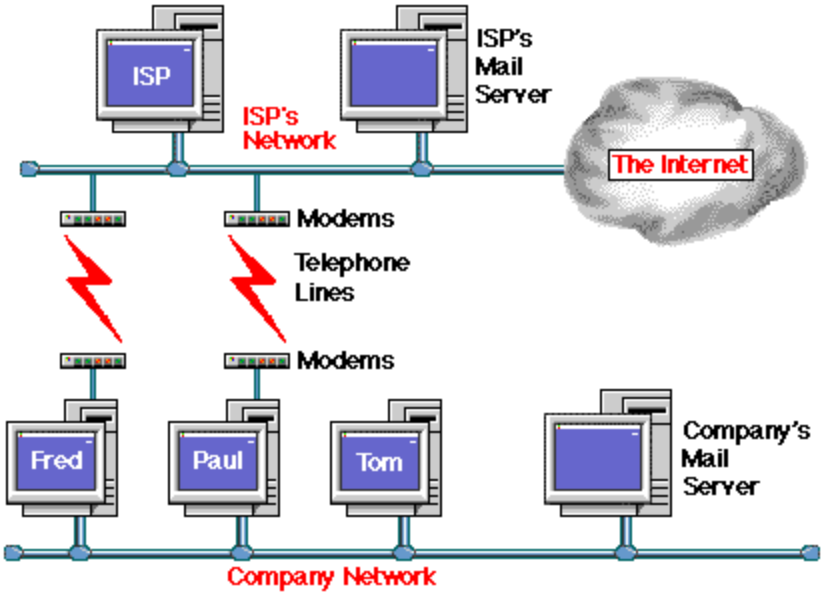
Each network connected to the Internet has a Domain name associated with it, to ensure email --and other traffic-- gets directed to the right recipient. In the

above diagram the ISP would have their own domain name, which points any email destined for a user in their domain to their mail server.

So, for example, if the ISP is called "Provider" and the domain that they own on the Internet could be called "provider.com" then all email to the home user is directed to "home.user@provider.com" which will result in the mail being stored on the ISP's mail server, ready to collect by the home user email client.

A single office user could also use the same system to collect and send mail using an ISP, but this would not have any direct relationship, or link, to the internal email systems that have already been discussed.

A number of users on a network dialing in to an ISP



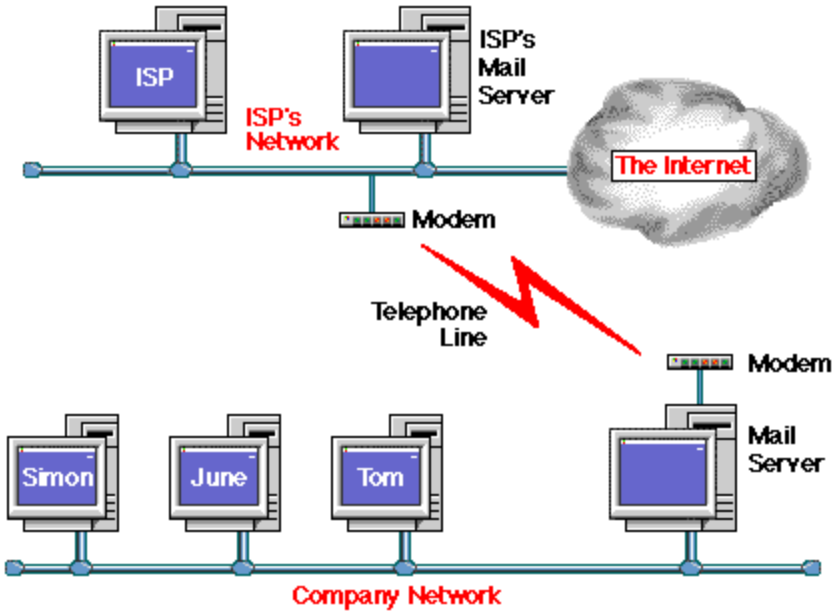
In this example Fred and Paul have two email addresses: One for internal mail within a company, and one for Internet email. This can sometimes occur if most of the email that a user reads or sends is internal within a company network, yet the user wants access to global Internet email. Each user would have an email account on both the company mail server and the ISP's mail server.

An Internet email user can contact Fred and Paul using email directly. However if an Internet user wanted to send a message to Tom, then they could not without having to send it to either Fred or Paul and asking them to forward the message. This arrangement allows for company email within the confines of the office network, but gives Internet email facilities to users who need them, in this case Fred and Paul. If Tom wanted to send email to an Internet address, rather than within the confines of the company, then he would have to ask either Fred or Paul to send it on his behalf.

Note that there may not actually be individual modems for all users, but some form of modem sharing may occur. If there were more than two Internet email

users, then connecting the office network mail server --rather than individual machines--to the Internet would probably be more efficient and flexible. Tom would then have been able to send his email message to another company himself, rather than asking another user to do it for him.

A company network connected to an ISP



In this example the company network is connected to the ISP's network by modem.

This adds the additional complication that the email addresses within the company network must be of a form that other users on the Internet can use. As the company network is connected to the Internet through an ISP, then the company could use the "Internet Domain" of the ISP for addressing their own email -- which means that each user could be addressed in the form "user@company.provider.com" (note that this is one possible method of addressing: each service provider may have their own way of addressing individual companies) or they could register their own Internet Domain. This would mean that a user is addressed as "user@company.com" where "company.com" is the Internet domain registered.

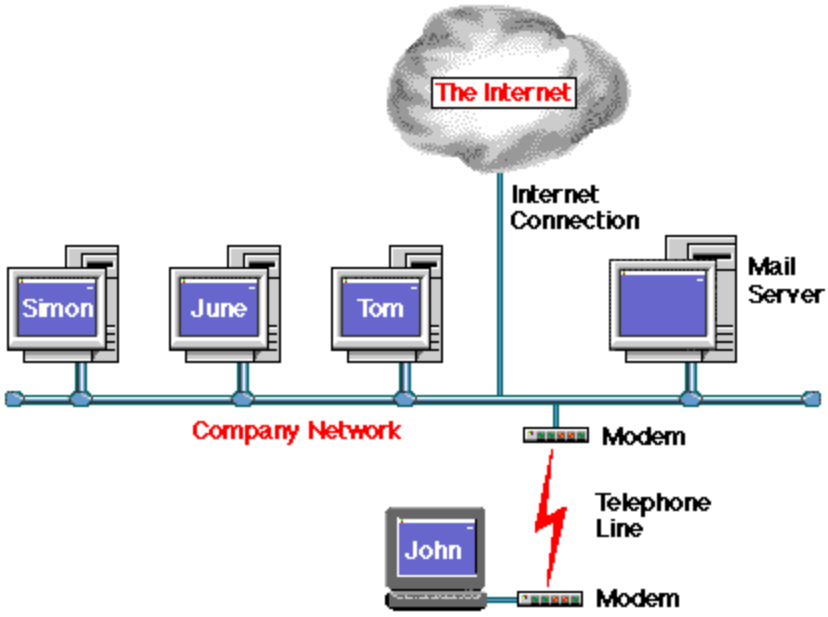
When Fred, Paul or Tom want to send email to a recipient on the Internet, they send the message in the same way as sending it internally within the office, but also must specify the "Domain" of the person they are trying to contact within the email address.

For example, if Tom wants to send a message to "Fred" who is an email user within another company in the US, then he would address the mail message to "Fred@thecompany.com" where "thecompany.com" is the domain for the

company where Fred is based. (Domains will be discussed more fully later in this FAQ).

With this arrangement the company mail server sends and collects email on behalf of the office network users. The users themselves never actually connect to the Internet. This allows the local Company Network and telephone connection to be used efficiently with the most flexibility. Used in conjunction with dial-in remote users to the company network, as discussed earlier, this system would allow for remote users to have access to global Internet email when dialing in to the Company Network.

A Company Network connected to the Internet



This gives all the flexibility of internal email within the company, but also allows Internet access for remote users to the company mail server for collecting and sending messages. The Internet connection would have to be full time in order to implement this arrangement.

Note that the actual physical "Internet Connection" could be one of a number of different connection methods, depending upon the potential traffic requirements to and from the Internet. Also some form of Firewall protection would be a sensible option. (A Firewall allows specified traffic through it, preventing unauthorized access both into the company network, and out onto the Internet).

Remote users could access the company network either via a direct dialed connection, or via the Internet. Also local dial-in users could access the Internet through the Internet Connection, effectively turning the Company into a private ISP!

How does email get from one email client to another email client?

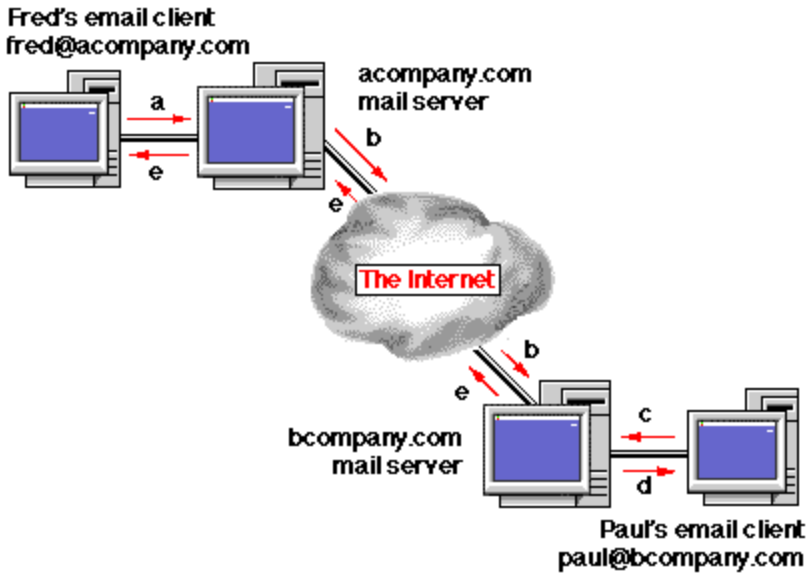
a. Fred wants to send an email message to joe@acompany.com. The email client on Fred's machine sends the message to the email server.
The mail server checks to see if it has an account with the user name "Joe." If this account exists then the message is stored, ready for Joe to collect. If there is not an account for Joe, the message is returned, with an explanation that Joe does not have an account, so the message could not be delivered.

b. Joe checks his email at a later time. Joe's email client asks the email server if there is any mail for Joe.

c. As there is mail waiting for Joe--from Fred-- the email client downloads the waiting message from the mail server. Joe can then read the email message and reply to Fred, if he wants, using his email client.

d. If Fred had sent mail to "tom@acompany.com", instead of "Joe@acompany.com" and Tom did not have an email account created on the mail server, Fred would receive a message back telling him that Tom did not have an email account, so his message could not be delivered.

How does email get from one email client to another when they are at different locations?



Fred wants to send an email message across the world to "paul@bcompany.com"

a. He creates his email message with his email client, which sends the message to the acompany.com mail server.

b. The mail server compares the domain name of the destination email address (i.e. bcompany.com) with the domain name it has been told to look after (i.e. acompany.com). These domain names are different, therefore the acompany.com mail server will send the message to the mail server that looks after email for the bcompany.com domain. (How it finds the bcompany.com mail server will be dealt with in Part two of this FAQ)

c. Paul checks his email at a later time. His email client asks his email server if there is any mail for Paul.

d. As there is mail waiting for Paul --from Fred-- the email client downloads the waiting message from the mail server. Paul can then read the email message and reply to Fred, if he wants, using his email client.

e. If Fred had sent mail to "tom@bcompany.com", instead of "Paul@bcompany.com" and Tom did not have an email account created on bcompany.com's mail server, Fred would receive a message back telling him that Tom did not have an email account on the bcompany.com mail server, so his message could not be delivered.

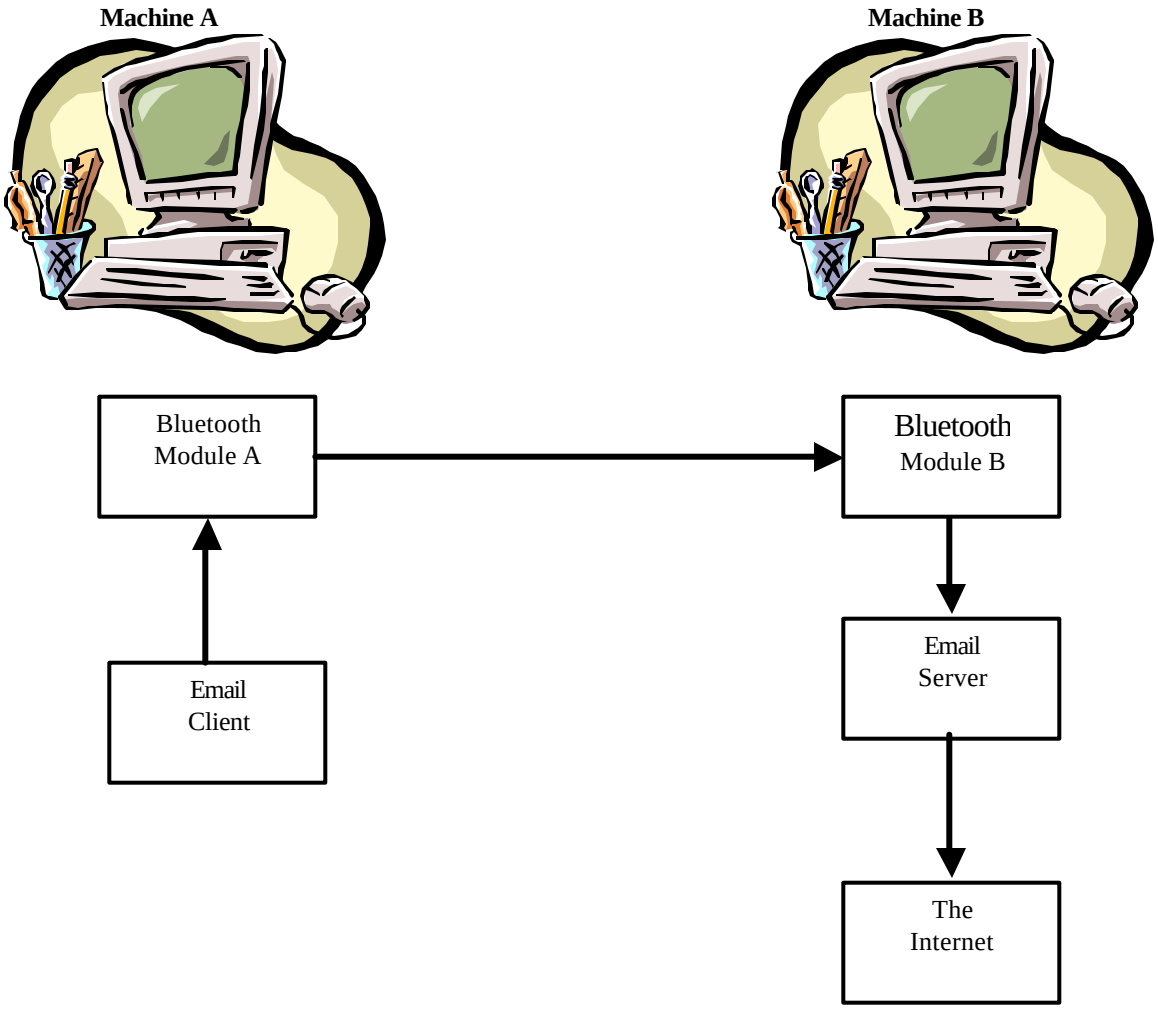
VII. Combining Bluetooth and Email

A. Project Overview

The purpose for this project is for us to take an email client then combining it with the use of Bluetooth module. A projected result is when we have two

computers, one with Internet connection and the other one without; we will be able to send an email from the one without the Internet connection via Bluetooth module.

Diagram:



From the diagram, machine A is without the Internet connection. The user at this machine wants to send an email, therefore he will use the Bluetooth mail, which sends an email via Bluetooth module. The Bluetooth module B receives the signal from Bluetooth module A. Then it calls the email program and sends the email out to SMTP or the Internet.

B. How it works?

First of all, we have to write an email program using Visual C++. The program will send an email out whenever it is called. A program will require a mail server for example smtp.tufts.edu, recipient email address, sender email

address, subject and message. We can divide the project into two parts, the server and the client.

C. The Server - BT_Security

BT_Security is a program that also comes with the Bluetooth PC Reference Stack. You are required to run the security before running any other program. For this project we have to modify the BT_Security so it will recognize when we call the BT_Mailserver. When we call the BT_Security, the program asks for what service we want to perform. Then it will wait for the client to establish connection.

D. The Server - BT_MailServer

For this part we will modify the BT_ChatServer program. The interface will remain the same but the function will be different. We still displaying all the messages that we receive from all the fields from BT_MailClient. But after we receive the entire message the program calls the email program that we have written. Then it will send the email to whatever address is input in the BT_MailClent program.

E. The Client - BT_Security

For the client part, when we call the BT_Security, the program looks for a Bluetooth module that might be connected to the computer. If it found the module then we select it and tell the program to look for any services available. If the server has already specified what kind of service, then such services will be available to the client. The client chooses the service and click connect, which will connect the two modules together.

F. The Client - BT_Mailclient

We will modify a sample program that comes with the Bluetooth PC Reference Stack. The program is called BT_Chatclient. It lets user talks to another computer wirelessly using only Bluetooth module and BT_Chatclient program. We will add the forms that will allow user to put mail server, recipient email address, sender email address, subject and message. This will be our BT_Mailclient.

VIII. Bluetooth Email Application Documentation

A. Ericsson Bluetooth Application Development Kit

This project is developed and implemented using the Ericsson Bluetooth Application Development and Training Toolkit. The Toolkit was designed for performing small research and development. The Toolkit includes a Bluetooth

chip on a circuit board with a USB interface and an Application Program Interface (API). On the CD, it includes the documentation of the Ericsson API and two sample programs. One of them is the BT_Chat.exe program, which allows two modules to "chat" with each other, very much like an instant messenger.

B. Project Overview

For our project, we decided to modify the BT_Chat sample program that was part of the Ericsson Bluetooth module to implement our Email Application. The main reason to develop our project by modifying the BT_Chat program is that the security and connection issues in Bluetooth are very complicated and hard to implement from beginning with the time we had. Therefore, by modifying the BT_Chat program, it allows us to not have to worry about the security and connection issues between the two Bluetooth modules and also it will allow the new program to have two features: Chat (original) and Email (our project).

The API and the source code for the BT_Chat application are both written in Visual C++. We spent a great deal of time learning and mastering Visual C++ in order to understand the source code and develop our application.

C. Project Schedule

September 2001 -	Define Project
October 2001-	Learn Visual C++
November 2001-	Project Implementation
December 2001-	Debug & Final Report

D. File Structure

Ericsson's BT_Chat program has three parts:

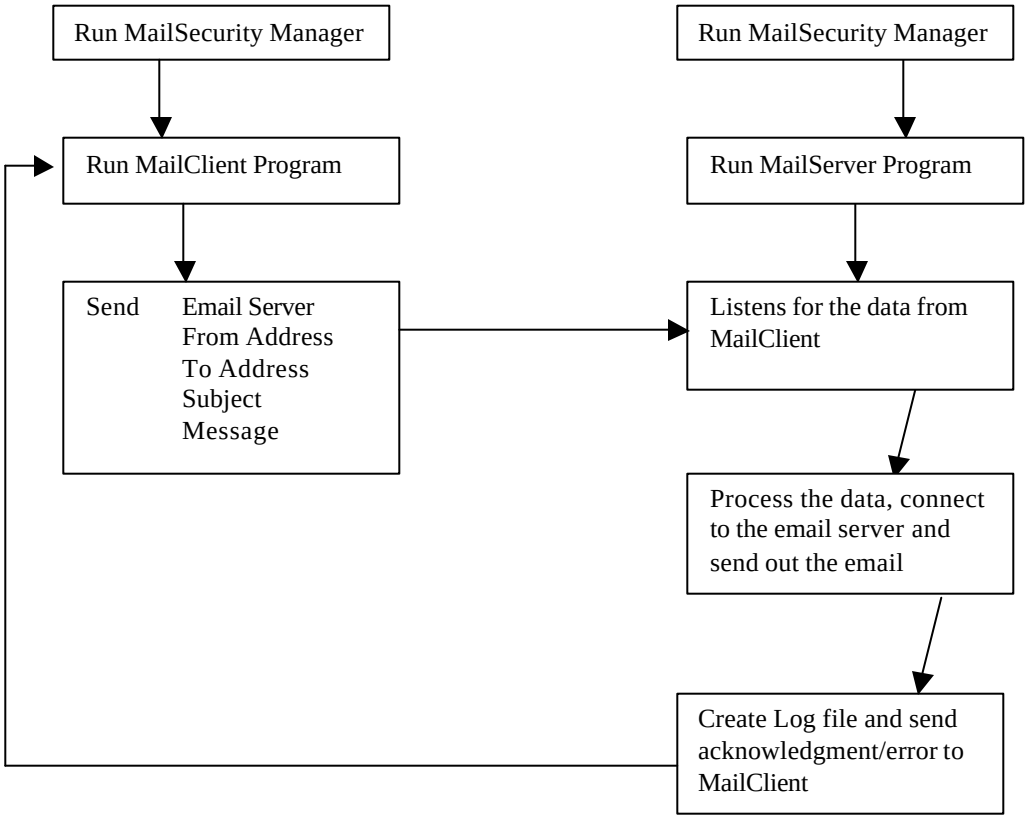
- BT_ChatSecurity
- BT_ChatServer
- BT_ChatClient

Our project's BT_Mail Program has the following three parts:

- BT_MailSecurity
- BT_MailServer
- BT_MailClient

Please note that the BT_MailServer application is for the Bluetooth Server, the device with Internet connection, while the BT_MailClient application is for the Bluetooth Client, the device without Internet connection. BT_MailSecurity must be initiated before running BT_MailServer or BT_MailClient.

E. The Flow of the BT_Mail Application



Both the "server" and "client' computers have to run the BT_MailSecurity Manager. On the "server" computer, it will run the BT_MailServer program. On the "client" computer, it will run the BT_MailClient program. The BT_MailClient program will automatically detect the BT_MailServer program. The BT_MailClient is then ready to send the data to the "server" computer.

BT_MailServer will make sure all the necessary data are received before attempting to connect to the mail server (smtp.tufts.edu). Then the BT_MailServer uses the SMTP protocol to connect to the mail server specified by the BT_MailClient. As the email server (smtp.tufts.edu) sends data back to the BT_MailServer, BT_MailServer checks the data received for error. If an error occurs with the mail server, BT_MailServer sends an error message to BT_MailClient; otherwise BT_MailServer sends an acknowledgment to BT_MailClient.

The activities are put into a log file with a time stamp on the "server" computer. The log file is especially helpful, because by looking at the log file, one can determine if an email has been sent or the kind of error that occurred

while attempting to send the email. Another feature we have put in the BT_MailServer is a messaging option, where the "server" computer can message the "client" computer. This may be useful when the user is on the "server" computer knows that the internet connection is down, and he can alert the user on the "client" computer with the messaging feature.

F. Source Code

For the BT_MailClient program, it is important to note that we are sending strings to the "server" computer. Each individual input field is converted into a string then sent in serial to the "server". Since there is setup time for each transmission, we found out that it is necessary to put a delay in between each string transmission.

Sleep(100);

This statement will allow the Bluetooth "client" module to have the necessary time to setup for another data transmission. Also, it prevents data from being lost, because the "server" module also needs time to receive the data being sent by the BT_MailClient.

For the BT_MailServer program, it will send back eight different acknowledgement/error messages back to the BT_MailClient program once the incoming data have been processed. The descriptions of the eight messages are shown in the table below.

Eight Acknowledgment/Error Messages	Description
"Email sent!"	No error was found
"Cannot create socket!"	BT_MailServer has problem with socket
"Cannot connect to server!"	Mail server could be down
"No responses from server!"	Mail server could be down
"FROM Address Syntax Error!"	Bad address
"TO Address Syntax Error!"	Bad address
"Server Error!"	Server timed out
"Message not accepted!"	Server could not send out the email

G. Future Improvements

This email application can be improved in many ways. One of them is to add more features to the existing version. We can modify the current version such that it can handle multiple addresses, send file attachments and add an address book. Also, we can change the flow of the application such that a user can write the email anytime with or without another Bluetooth module in range. Once a Bluetooth module is in range, it then sends the data automatically. This implementation would be more practical and useful to the users.

With the use of Bluetooth, many more applications can be implemented. For example, instead of an email application, we can do a telnet application, FTP or web browser. All of the above mention applications use the same basic idea, one computer has Internet connection and the other one does not. This includes both desktops and laptops.

We can also adapt the Bluetooth so it can be used with PDA or other hardware. But we have to develop how to connect the Bluetooth module to PDA devices, and then the applications can be implemented.

IX. Acknowledgment

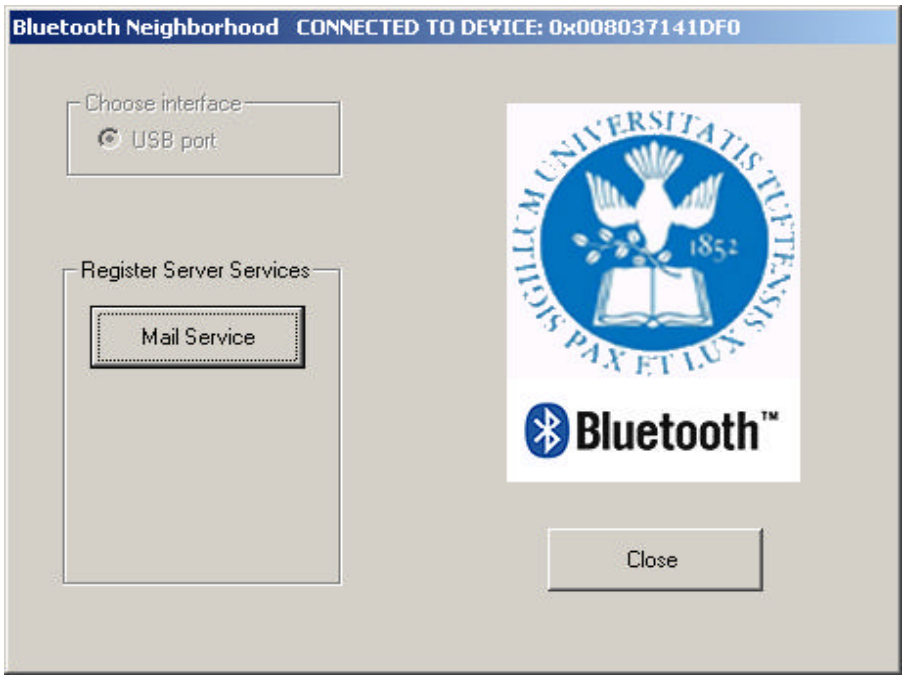
The design and development of the Bluetooth Email Application are done my Nick Hong, Martin Shek and Tulyatep Uawithya. We would like to especially thank Professor C. Hwa Chang for advising and helping during the project.

X. References

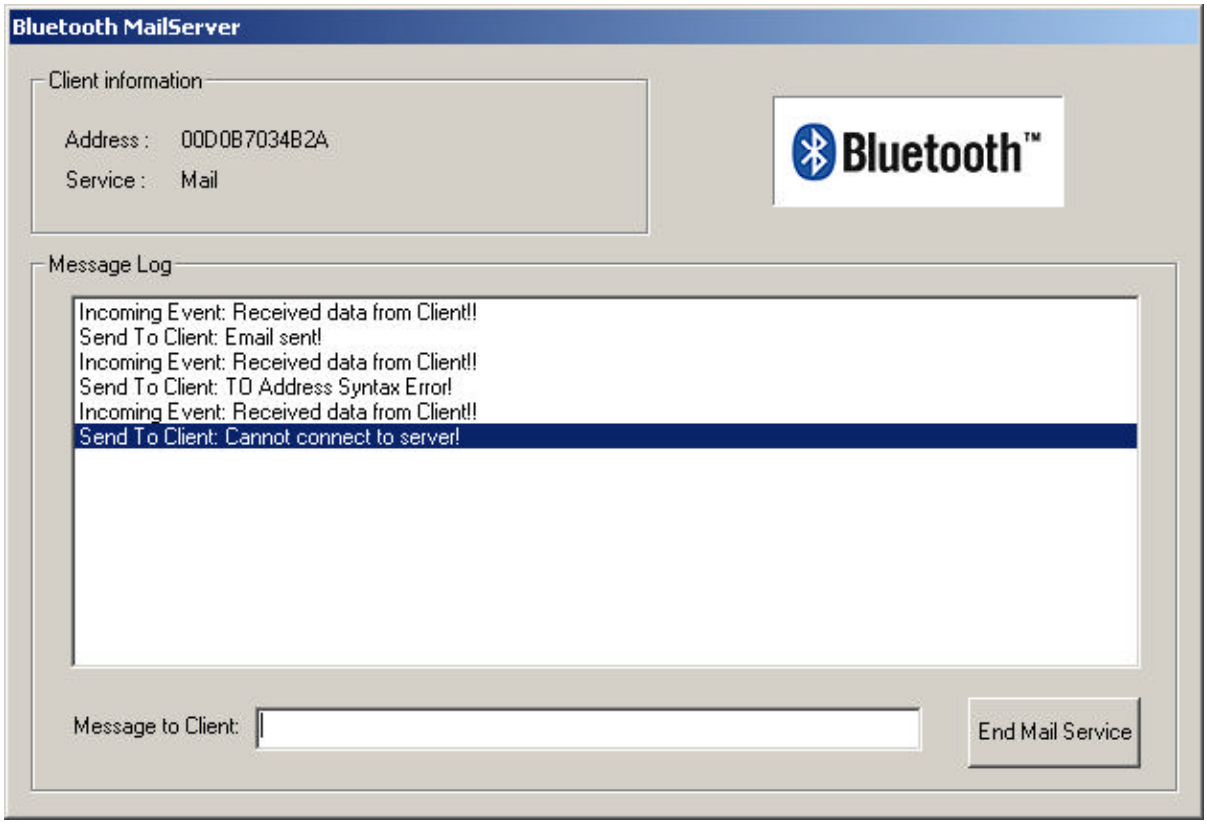
Ericsson Bluetooth Application Development Toolkit Documentation
B.E.S.T. Design Project 2000- by Ken Fan, Manraj Saini & Martin Shek

www.bluetooth.com
www.bluetooth.org
www.palowireless.com
www.howstuffworks.com
www.ericsson.com/bluetooth
www.3com.com
dir.yahoo.com/Computers_and_Internet/Internet/History
www.clearswift.com/subpages/email_news_hist.htm
www.pretext.com/mar98/features/story2.htm
www.vicomsoft.com/knowledge/reference/email.history.html

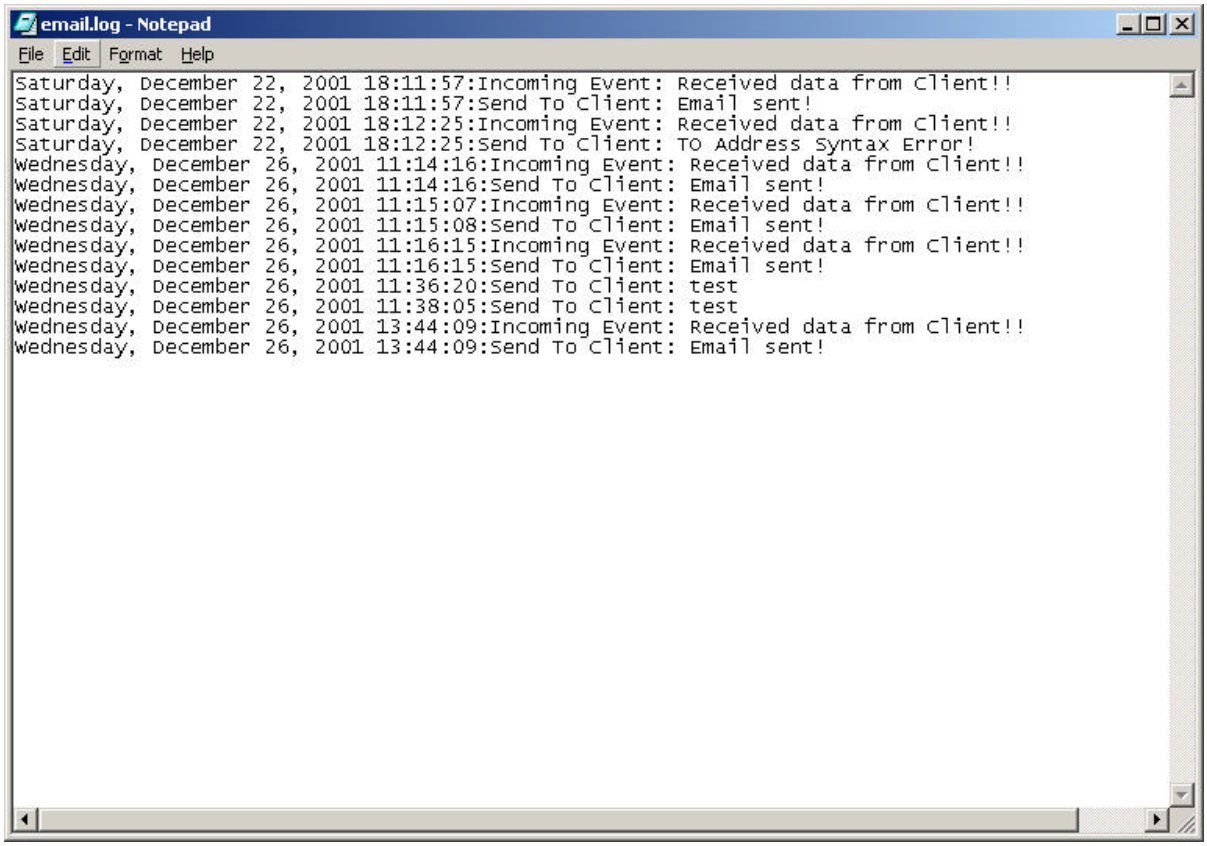
XI. Appendix A: BT_MailServer



Security for the “server” computer Screenshot

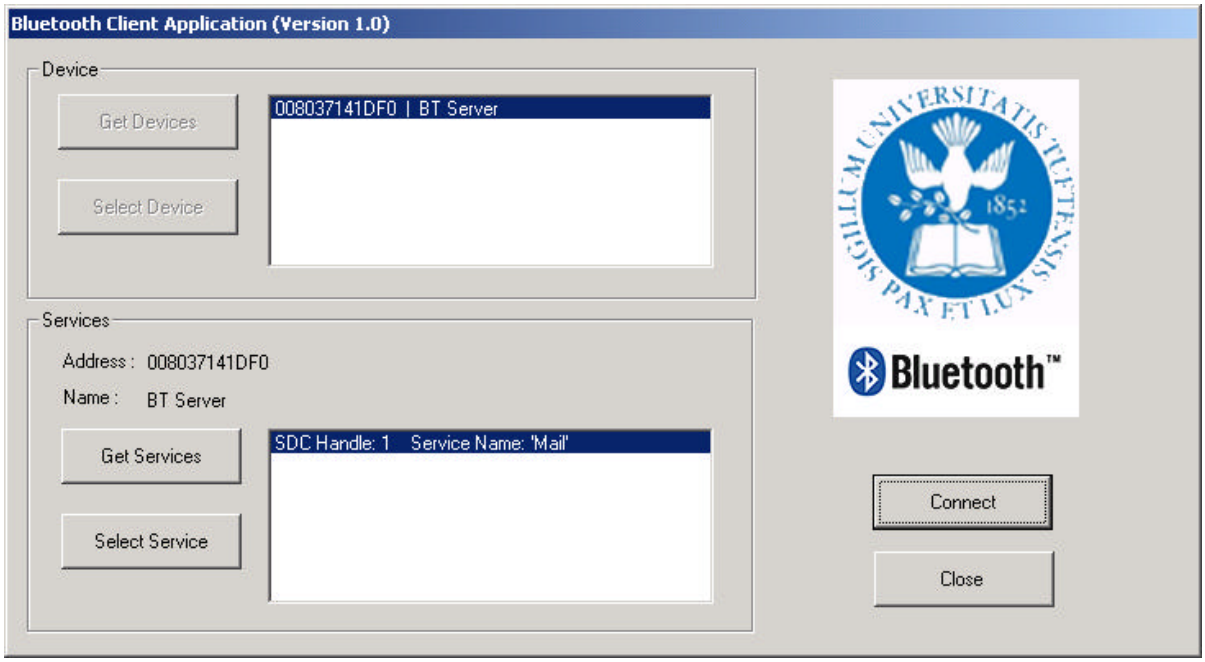


BT_MailServer Screenshot

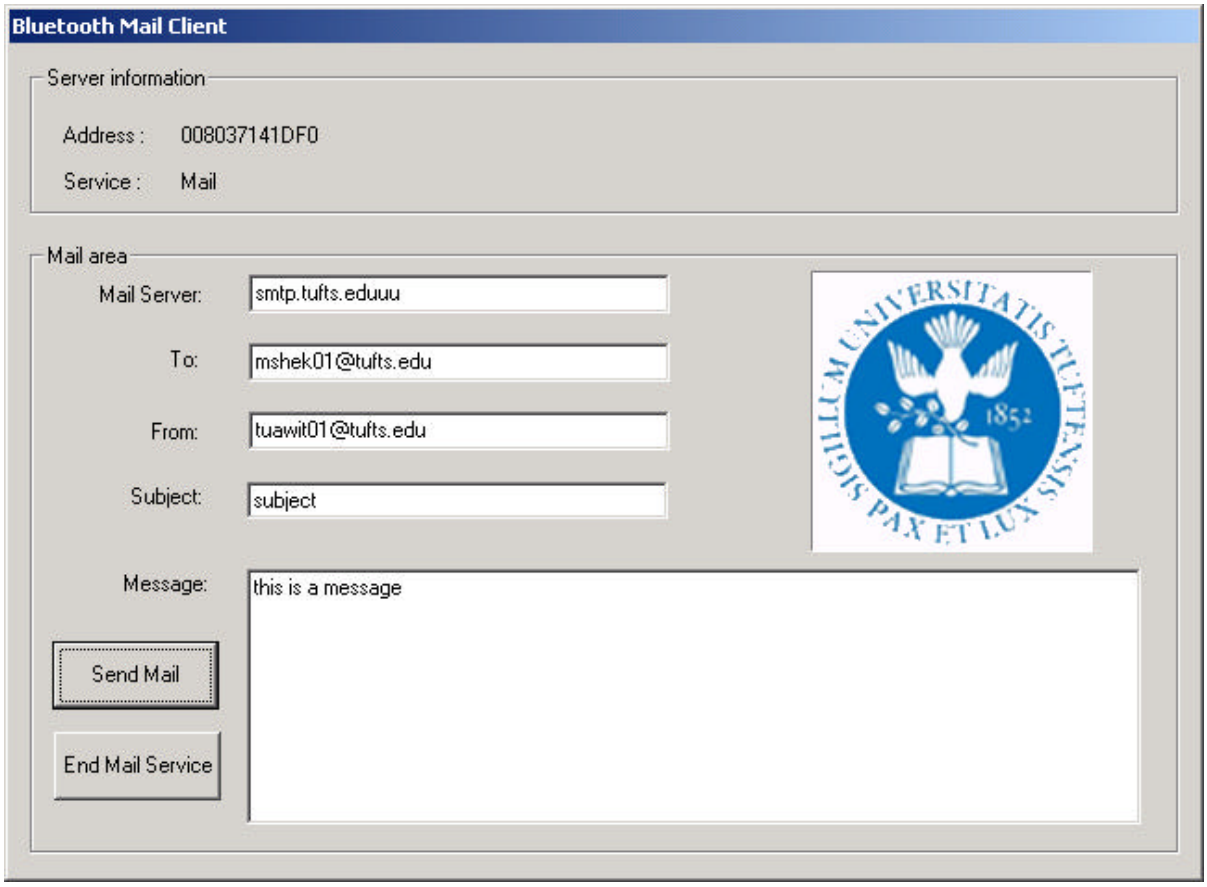


Screenshot of the Log file

XII. Appendix B: BT_MailClient



Mail Client Application



Screenshot for the BT_MailClient

XIII. Appendix C: Terms & Acronyms

ACL	Asynchronous Connection-Less Links
ACO	Authenticated Ciphering Offset
API	Application Program Interface
ARPA	Advanced Research Projects Agency
BD_ADDR	Bluetooth Device Address
FH	Frequency Hopping
GAP	Generic Access Profile
HCI	Host Controller Interface
IMAP	Internet Message Access Protocol
L2CAP	Logical Link Control and Adaptation Protocol
LMP	Link Manager Protocol
POP	Post Office Protocol
PDU	Protocol Data Unit
SCO	Synchronous Connection-Oriented
SDP	Service Discovery Protocol
SMTP	Simple Mail Transfer Protocol
TDD	Time-Division Duplex
USB	Universal Serial Bus