Bluetooth-based Ad-Hoc Networks for Voice Transmission

Frank Kargl - Stefan Ribhegge - Stefan Schlott - Michael Weber
Department of Multimedia Computing, Albert-Einstein-Allee 11,
89081 Ulm, University of Ulm, Germany, +49-731-50-31310,
frank.kargl|stefan.ribhegge|stefan.schlott|michael.weber@informatik.uni-ulm.de

***Abstract*—In this paper we inspect the possibilities of using Bluetooth for building Ad-Hoc networks suitable for transmitting audio and esp. voice data using synchronous SCO links. We analyze the features or problems that Bluetooth offers for transmitting audio data in a multihop network. As the existing MANET routing protocols that emerged out of the work of the IETF MANET WG (like AODV, DSR etc.) can not be directly used to work with Bluetooth, we present a new routing protocol called Bluetooth Scatternet Routing (BSR) that is influenced by other MANET routing protocols but pays special attention to the restrictions of Bluetooth (like number of connections, connection setup times etc.). The protocol is also inspired by the channel switching concept of ATM. Some initial results of simulations and real-life tests give an impression of the performance and efficiency this protocol can reach in an application scenario.**

I. Paper Outline

After section II gives an introduction to a scenario of voice transmission in a Bluetooth based MANET, sections III to V give a short overview on relevant aspects of Bluetooth, MANETs and earlier work in this field. Section VI analyzes the benefits and problems of mobile ad-hoc networking via Buetooth with special focus on audio/voice transmission. We argue that existing MANET routing protocols cannot be used straightforward with Bluetooth but need adaption. In section VII we describe our contribution: a routing protocol called ”Bluetooth Scatternet Routing” (BSR). As the Bluetooth hard- and software available today doesn’t implement the standard completely - e.g. often scatternet functionality is missing - we have implemented only parts of the protocol directly. Section VIII tries to evaluate the efficiency of BSR with an approach combining measurements and simulation. Section IX concludes our paper with a summary and outlook.

II. Motivation

Many of the research activities in the field of mobile ad-hoc networks (MANET) assume that cheap short to medium range radio transceivers will become very common in the next years. Most of the practical work and implementations focus on IEEE 802.11 Wireless LAN as an underlying physical radio network used for simulations or tests. For use in limited wearable devices like PDAs or cellphones, 802.11 has the disadvantage of consuming more battery power than other technologies like Bluetooth [1][2]. Furthermore it seems that after some startup problems Bluetooth may really become a common feature of cellphones (like the Ericsson T68m and others) or PDAs (like the new Compaq IPAQ 3870).

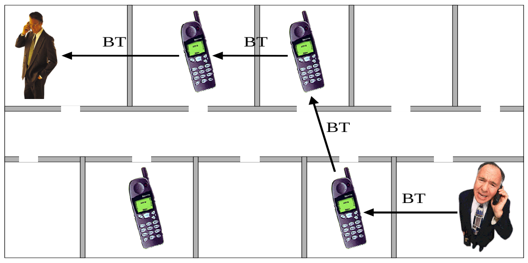


Fig. 1. Example scenario of a Bluetooth ad-hoc network

So the developers of the MANET routing protocols should consider that their protocols might also be used in a Bluetooth environment. The initial goal of our work presented here was to evaluate if the current protocols can be easily used in combination with Bluetooth or what modifications need to be done to enable Bluetooth-based Ad-hoc networking.

First we try to find a scenario where most of the characteristics and problems of MANETs and Bluetooth will appear. In this scenario we connect a large number of Bluetooth-enabled cellphones to a (multi-hop) MANET in order to enable Bluetooth-relayed phone calls between them.

There are a large number of possible application fields for this scenario. Think of a large office building where people move a lot between their office desk, conference rooms or central areas like printer-rooms or the cafeteria. Fixed-line telephones don’t provide a reasonable grade of reachability here. So people tend to call their colleagues at their cellphone in order to find our their current location or to reach them for urgent requests etc. Of course the

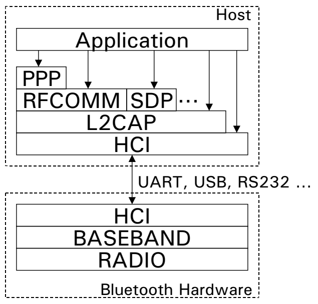
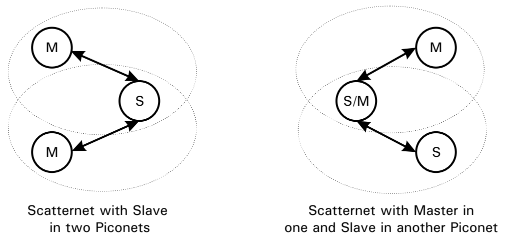


Fig. 2. Bluetooth stack



M
S
M
Scatternet with Slave in two Piconets
M
S/M
S
Scatternet with Master in one and Slave in another Piconet

Fig. 3. Examples of Scatternets

network carrier of the cellular phone network charges for these calls. Given our scenario above people may be able to do the same calls at no cost.

Another usage of the system proposed may be on large exhibits or fares where a lot of people with Bluetooth-enabled phones may meet. Again relaying calls between two visitors (like friends visiting different halls that want to meet for lunch) via a Bluetooth-based MANET may save costs compared to the normal cellular phone network.

III. Bluetooth Overview

In this section we will give a very brief overview of the relevant aspects of the Bluetooth standard. For detailed information see [1][2].

When initiated in 1998, the original idea of Bluetooth was to create a cheap wireless replacement for the myriad of data wires that surround today's multimedia devices.

Bluetooth uses a protocol stack of several layers. Figure 2 shows an simplified overview. The Radio Layer describes the physical radio system. Bluetooth devices fall within one of three different radio classes, but nearly all of todays Bluetooth-enabled devices use Class 3 radios, esp. the smaller battery-powered systems like cellphones. These devices have a radio range of about 10m. Bluetooth uses a frequency hopping scheme where the hopping sequence is coordinated by the master of each Piconet (see below).

The Baseband Layer is responsible for transmission and reception of data packets, error detection and encryption (if used).

The Link Controller uses a state machine to control synchronization, connection setup and shutdown. This state machine has different states that are called Standby, Page, PageScan, Inquiry, InquiryScan and Connection. A connected device may either be in the state Active, Hold, Sniff or Park. For details see [1].

When a device wants to connect another device it first has to do an inquiry for its direct neighbors. After receiving the inquiry results it can contact another device using its unique Bluetooth address. When connected the two devices form a so-called Piconet. The initiator of the connection becomes the Master of this Piconet, the other device becomes a Slave. When the Master contacts additional devices, this Piconet will contain multiple slaves up to a maximum of 7.

When a device A that is e.g. a Slave in one Piconet contacts another device B then a new Piconet is formed with A being the Master in this new Piconet and still being a Slave in the original Piconet. Any connected combination of Piconets is called a Scatternet. Figure 3 shows some Scatternet examples. A Scatternet is essentially some form of a MANET where traffic may be relayed between Piconets. Unfortunately the Bluetooth standard does not describe any Routing protocol for this and most of the hardware available today has no capability of forming Scatternets. Some even lack the ability to communicate between Slaves of one Piconet or to be a member of two Piconets at the same time. We will give details on this later.

For building MANETs based on Bluetooth we need to find a suitable routing algorithm and implement this functionality on the application level. Later this may be integrated into the Bluetooth stack itself.

After connecting another device, data may be transmitted using the ACL mode (Asynchronous Connection Less). In Bluetooth the data transmission is controlled completely by the Master. Slaves may only transmit data after being polled by the Master. ACL data transmission implements error detection and retransmission.

The other transmission mode uses SCO links (Synchronous Connection Oriented). SCO links need to be established explicitely and only after an ACL connection has been setup. Each SCO link reserves 64kBit/s of bandwidth. In contrast to the ACL links, SCO links have no retransmission, packets with errors are silently discarded. A master may establish up to three SCO links to its slaves.

The Host-Controller-Interface (HCI) separates the Bluetooth hardware from the part of the protocol stack that is usually implemented in software. Thanks to this standardized interface, the Bluetooth hardware and the Bluetooth stacks usually interoperate very well. The hardware connection is standardized for HCI-UART, HCI-RS232 and HCI-HCI-USB. Others may follow.

The Logical Link Control and Adaption Protocol (L2CAP) layer multiplexes different data streams, manages different logical channels and controls fragmentation. Multiple higher layer modules may access the L2CAP layer in parallel. These higher layer modules may consist e.g. of RFCOMM for emulation of serial connections, OBEX for transmission of serialized data objects or SDP for service discovery.

IV. MANET Overview

This section will give a short overview of MANET routing. For more detailed information see e.g. [3][4][5][6][7]. MANET routing protocols may be divided into two categories: proactive and reactive.

Proactive protocols always try to maintain up to date routing tables for all reachable destinations. All well-known Internet routing protocols like RIP or OSPF fall in this category. The reactive protocols are only activated when a node S wants to send packets to a second node D. In this case S originates a route request that searches the network for a valid path towards the destination. Once the optimal path has been discovered, D sends a route reply to S. Once S has a valid path for D it can start to send its packets. During this process the routing system needs to perform routing maintenance, i.e. it has to check if the route is still valid. If a route breaks many protocols perform a route repair.

When transmitting audio data over a network there are basically four parameters that determine the audio quality. The bandwidth of a link is the maximum amount of data that a link between two nodes can transmit.

The loss rate is defined as the maximum percentage of packets (or sometimes octets, bytes, ...) that a link may drop during operation.

$$lr_{singlehop} = \frac{NumberOfLostPackets}{NumberOfAllPackets} \qquad (1)$$

The delay is defined as the amount of time that a packet needs to be transfered from A to B. In our scenario, we usually measure roundtrip delays. The unidirectional delay d is defined as

$$d = \frac{roundtrip}{2} \qquad (2)$$

Normally we measure the delay of a large number of packets ($D = d_i | i = 1 \ldots n$) and specify only the average delay:

$$avDelay = E(D) = \frac{1}{n}\sum_{i=1}^{n} d_i \qquad (3)$$

The final relevant value is called jitter. It is defined as the standard deviation of the link's delay:

$$jitter = \sigma(D) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(d_i - E(D))^2} \qquad (4)$$

When we introduce buffers at the receiver we can reduce the jitter but increase the delay.

V. Related Work

There are a number of publications describing work to enable Ad-Hoc networking with Bluetooth hardware. E.g. in [8] Lars Wernli and Riccardo Semadeni describe their work on developing a reduced version of the DSR protocol [5] for Bluetooth [1]. This work also demonstrates that you face extremely long delays when you just port the existing protocols to Bluetooth without regarding the special properties of Bluetooth.

Other approaches like Bluetree [**?**] or Bluenet [9] try to establish a Bluetooth Scatternet spanning all reachable nodes.

In contrast our work only established links as needed. This is reasonable because Bluetooth puts tight limits on the number of connections allowed, esp. when using SCO links.

VI. Bluetooth-based MANETs

When using Bluetooth as a physical layer for a MANET, we have to consider a number of restrictions compared to e.g. IEEE 802.11 as used by most protocols.

1) Bluetooth is connection oriented. So in order to send data to another node you have to setup and later tear down a connection.
2) Bluetooth has no "all neighbors" broadcast capability (only point-to-multipoint within the Piconet). So in order to e.g. flood a route request you have first to connect all neighbors and then send a point-to-multipoint packet to these. If there are more neighbors than allowed in a Piconet (7) things get more complicated.
3) When using SCO connections we have a restricted number of connections per master node (3), so any node may only relay one connection and originate another connection. The routing protocol needs to consider this.
4) Bluetooth has very long inquiry and relatively long connection setup times. So the connection setup/disconnect needs to be optimized.

When we want to implement a telephone system using a Bluetooth-based MANET, we have a number of requirements that contrast with above capabilities:

- Users want a short connection setup time similar to a normal phone dialing (a few seconds).
- We need to have relatively few hops. Otherwise the delay will be too long and the user will experience a significant pause and echo.
- We will usually use SCO links for transmission. As stated above this restricts the number of paths that a node can participate in. When using ACL links this restriction is missing but we then have problems guaranteeing the necessary bandwidth. The link type should be configurable.
- In order to optimize the overall throughput and minimize interference we want to minimize the number of Piconets and shutdown unneeded connections.
- To avoid interrupts in audio transmission we need a very efficient and fast route-maintenance and -repair.

No current MANET routing protocol fulfills all of these requirements. So we designed a new routing protocol with the focus on voice transmission in Bluetooth Scatternets. Nevertheless the current protocols provide a number of interesting ideas to use as a basis for this new protocol:

- reactive operation
- route maintenance/repair

Furthermore there are other connection oriented networks that provide interesting concepts. E.g. our data packets don't carry any destination information in the header. Instead the intermediary nodes use the incoming (L2CAP or SCO) channel identifier to decide where to forward the data. This is very similar to the circuit switching approach of ATM. We tried to take these ideas and integrate them in a routing protocol for Bluetooth networks that is described in the next section.

VII. Bluetooth Scatternet Routing (BSR)

We named our routing protocol protocol Bluetooth Scatternet Routing (BSR). It is a reactive routing protocol similar to AODV or DSR but keeps additional information on the state of links and tries to avoid long delays due to inquiry or connection setup. The main components of BSR are presented in the next subsections.

A. Path Discovery

Before participating in a path discovery, any node needs to know its direct neighbors. These are found by regular inquiries and are stored in an internal neighbor list.

When a node S wants to send data to another node D it first picks an arbitrary neighbor X and connects to X using an ACL connection. It then sends a Path-Request (PREQ, figures 4a and 5). Each PREQ contains a PREQ-ID which is incremented each time a node initiates a request. For

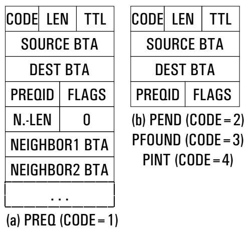


Fig. 4. BSR packet formats

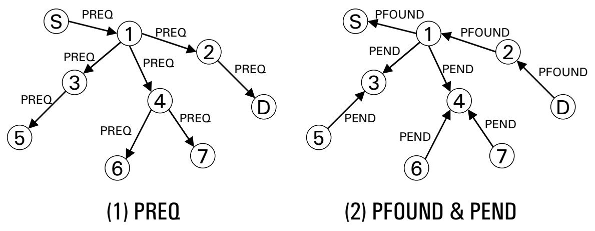


Fig. 5. Path Discovery

simplicity, we assume that there is no overflow in PREQ-IDs. The PREQ furthermore contains a Time To Live field (TTL) that is decremented by each intermediary node. Finally each packet includes a list containing all of the last node's neighbors.

On receiving a PREQ, X checks whether it already received a PREQ packet from S with destination D and a given sequential-ID. If not it forwards the packet to all neighbors (excluding the originator of the data) in the same manner as described above, decrementing the TTL-field by one. All neighbors that are included in the packet's neighbor list (i.e. the neighbors of X's predecessor) are excluded from this, because it is expected that X's predecessor has already forwarded the PREQ to them; establishing another expensive ACL-connection is not necessary.

When a node X receives a second PREQ packet for the same path request, it decides (based on the TTL) which of the two paths is shorter. It then sends a Path-End (PEND, figure 4b) packet to the other host pruning the path. The same happens when a node has no neighbors to relay the PREQ packet to or it has received PENDs from all of its neighbors. This way unnecessary Bluetooth connections are closed after a short time.

Finally the destination node D will receive the PREQ. It then sends a Path Found (PFOUND, figure 4b) packet back the discovered path to S. Note that using this mechanism the path discovery will find a short but not necessarily the shortest path. To really find the shortest path, the destination would need to wait for an undefined time to collect all PREQ packets and find the request with the highest TTL. As this may take a very long time (given the long connection setup times that we will see later) this is not acceptable. So the goal is to get a connection quickly even if the path used may not be optimal. One of the optimizations below describes a way to use late-coming PREQs with better TTLs.

When a node along a path receives a PFOUND, it forwards the packet towards the source of the PREQ S. At the same time it sends PEND packets to all other nodes to which it forwarded the PREQ packets earlier. As the connection setup necessary for the PREQ delivery takes much longer than just forwarding a packet over an established connection, the PEND packets have a very good chance of overtaking the PREQ. In this case the PREQ is canceled so that the Bluetooth network is not stressed too much.

When we want to use synchronous transfer and a PFOUND packet reaches S, we need to establish SCO links along the path found. Otherwise the already established ACL connections can be used right away. Finally the audio data can be transmitted. Each node stores a mapping of incoming and outgoing channel for each connection. So whenever a node receives data on one channel it only needs a simple table lookup to find the corresponding outgoing channel.

In case of SCO channels, all intermediary nodes can (at maximum) relay only one connection and initiate or terminate one other connection at a time. So if a node X is already relaying one connection and receives a PREQ for a node other than itself it needs to reply with a PEND. If however the PREQ is searching a path to X, then X can accept the connection. Possibly X must also perform a number of master-slave switches to become master of all three SCO links.

B. *Path Maintenance*

Path maintenance consists of two tasks: *Path Monitoring* and *Path Repair*. For Path Monitoring we need to constantly monitor the nodes's established connections in order to detect link failures and react accordingly. Bluetooth takes over the task of link monitoring, so we don't need to send alive packets or similar.

When a link failure is reported, Path Repair takes place. In the simplest case, all nodes detecting a broken path send Path-Interrupted (PINT, figure 4b) packets towards source and destination. The source of a connection can then re-initiate a path discovery. See below for possible optimizations.

C. *Path Removal*

When an end node of a connection wants to terminate the connection, it simply sends a PEND packet along the path and shuts down the Bluetooth connection towards this neighbor.

D. *Optimizations*

There are a number of ways to optimize the above protocol. First we noticed that the order in which neighbors receive PREQ is completely random. If we had a connection to the same destination D earlier and that connection died for any reason, we might want to remember the former next hop neighbor for that connection and send the PREQ packet there first. This is called *Preferred Neighbor* optimization.

In the protocol described above, the Path Repair will take a very long time, because after a link failure a completely new path discovery needs to be initiated. This can take a number of seconds, so clearly nobody will tolerate regular breaks of multiple seconds during a call.

The first optimization for this case, called *Local Repair*, makes use of the fact that a single link failure may be healed locally. We assume that all nodes know all other nodes in a path (this information has to be added to the PREQ and PFOUND packet). So when a link fails, the intermediary nodes that form this link don't generate PINT packets immediately but instead try to do a local repair by sending a new PREQ with a very small TTL and a destination of the next but one neighbor along the path. If this request succeeds, the transmission can continue with only a very short interruption. If the request fails again, a PINT packet needs to be sent to the ends of the path.

An alternative to the *Local Repair* strategy is to keep a backup path readily set up, so that a node can switch to an alternate path in case of a link failure. This *Backup Path* strategy implies that destinations receiving multiple PREQ packets may answer two (or even more) of these requests with a PFOUND packet. In this packet you have to note the priority of the path. The source can then use the path with the highest priority and in case of a failure switch immediately to the second path. In our protocol we use a backup path only when it is completely separated from the primary path. When using a backup path in combination with SCO links this backup path of course consumes SCO connections which are very limited (see above).

A last optimization is called *Dynamic Path*. As stated earlier, our current protocol doesn't necessarily create shortest paths but instead uses the first PREQ request that reaches the destination. PREQs with a better TTL that reach the destination later are answered by a PEND packet and discarded. With *Dynamic Path* we can answer these requests with a PFOUND packet marking this path as ”alternative”. When the source has already set up a path to the destination and receives another PFOUND it can then decide to use this second path and discard the first path by sending a PEND. *Dynamic Path* has a number of problems when different paths are not separated and PEND packets from the first path tear down the second before it's established. We are still investigating solutions to these problems.

VIII. Protocol Evaluation

The first goal when developing a protocol is of course to implement it on real world hardware. Unfortunately the available Bluetooth adapters have a number of restrictions that make it impossible to implement our protocol on the current hardware. We started our work with PCMCIA cards 'Brainboxes BL-620' which use a Bluetooth chipset from Cambridge Silicon Radio (CSR) called BlueCore01b. This cards currently have only a very limited multipoint capability and the firmware release notes [11] state that a device can't be a slave in one Piconet and a master in another or a slave in both Piconets. It is therefore nearly impossible to build larger Bluetooth networks. Other hardware (like the Ericsson test modules [12] have similar restrictions. Thus we decided to take a different testing approach. We measure characteristics of a single Bluetooth Piconet and then transfer these values to simulation results of our protocol. The Bluetooth stack we use is OpenBT 0.0.8 [13] compiled for a Linux 2.4.16 kernel. As OpenBT has no support for SCO links we use ACL for our tests.

A. Single Piconet

We want to measure the following values: Connection setup time, single link delay and jitter. In order to get realistic results, we implement a scenario where one computer records an audio signal from the soundcard's line in and transmits it to a second computer using a Bluetooth ACL link (RFCOMM Profile). On the receiving side the audio signal is played via the normal soundcard line out. An oscilloscope is used to produce the audio signal, record the resulting signal and measure the delay. The complete test scenario is shown in figure 6.

We measure a total delay of about 200 ms where the delay also varies with the RFCOMM baudrate. At 115200 baud the delay is 270ms, at 460800 baud 164ms. In a separate installation we test the delay of the PC soundsystem alone (without Bluetooth transmission) and find it to be about 136ms. So for the Bluetooth transmission alone remains a delay of about 28ms.

Next we want to find out some information about roundtrip delay and jitter in Bluetooth. As it is not practicable to measure jitter with an oscilloscope, we use a

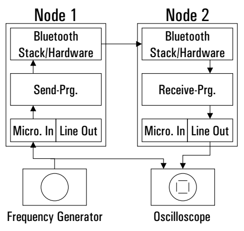


Fig. 6. Test Scenario

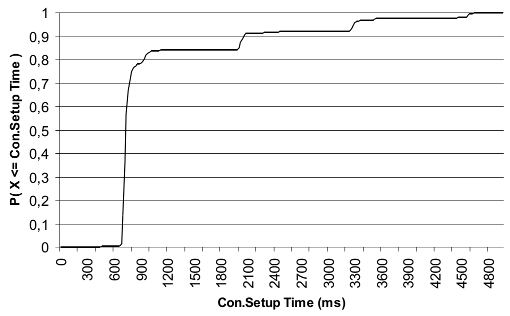


Fig. 7. Distribution Function of the connection setup time

different setup, where on the one side a program gene rats data packets with timestamps at a fixed rate (100ms per packet). On the other side of the connection there is a simple repeater sending back the received packets as fast as possible.

When sending 100.000 packets we find the roundtrip delay to be on average 56ms. When we set the delay to be the half roundtrip delay we find it again to be 28ms which is consistent with our former value. The roundtrip jitter is measured as 11ms so the jitter for a unidirectional transmission is around 5.5ms. In our tests we find 0.4% of all the packets to have a delay larger than 800ms whereas the rest of the packets have delays of less than 100ms. We assume that these very long delays occur when transmission errors happen. After a timer expiration a retransmission is initiated by the baseband layer. As all other packets reach their destination much faster we don't see any packet delays $>$ 100 ms and $<$ 800ms.

In an SCO scenario these retransmissions wouldn't occur so that 0.4% of the packets would actually be lost. So we set the loss rate to 0.4% and drop these packets from our calculation of delay and jitter.

Finally we need to find out the times for a connection setup. Again we use some simple test programs to do a large number of connects and disconnects in a loop. Running this loop 300 times we measure an average connection setup time of 1100ms. Figure 7 shows the distribution function of connection setup times measured.

B. Multihop Scenario

Next we will use the results of the last section to calculate values for a multihop scenario. The remaining questions to solve are: How is the connection setup time, the loss-rate, the delay and the jitter affected in a multihop scenario? How does the number of clients in a given area affect the overall connectivity?

In order to calculate the multihop connection setup time, we simple have to multiply the single-hop connection setup time by the number of connection setups necessary:

$$t_{BSR} = n * t_{BT} \tag{5}$$

A connection with 6 connection setups e.g. needs a connection setup time of 6 * 1100ms = 6,6s. We need to keep in mind that a BSR Path Discovery setup needs much more connection setups than hops in the resulting path because for sending a PREQ we need to contact in worst case all neighbors before finding the next hop in a path.

The loss rate in an established multihop path can be calculated as follows: Let P_{0h} be the probability of a packet to be delivered from source 0 to destination h. The intermediary nodes are called 1, 2 etc.. The overall loss rate is then

$$\begin{aligned} lr_{0h} &= 1 - P_{0h} \qquad (6)\\ lr_{0h} &= 1 - \prod_{i=1}^{n} P_{(i-1)i} \\ lr_{0h} &= 1 - \prod_{i=1}^{n} (1 - lr_{BT}) \\ lr_{0h} &= 1 - (1 - lr_{BT})^h \end{aligned}$$

So a Bluetooth connection with 3 hops will have a loss rate of $lr_{SD} = 1 - (1 - 0.004)^3 \approx 1.2\%$. The delay of a given multihop connection can be calculated as:

$$delay_{SD} = h * delay_{BT} \tag{7}$$

So again a 3 hop connection will have a delay of $delay_{SD} = 3 * 28ms = 84ms$. Finally the jitter in a multihop path can be calculated as follows: Let $D = \{d_i | i = 1 \ldots n\}$ be the delay of a large number of packets and D_{ij} be the D for a link from i to j where the hops are numbered from 0 to h.

$$\begin{aligned} jitter_{SD} &= \sigma(D) \qquad (8)\\ jitter_{SD} &= \sqrt{\sigma(D)^2} \\ jitter_{SD} &= \sqrt{VAR(D)} \\ jitter_{SD} &= \sqrt{VAR\left(\sum_{i=1}^{h} D_{(i-1)i}\right)} \\ jitter_{SD} &= \sqrt{VAR\left(\sum_{i=1}^{h} D_{BT}\right)} \\ jitter_{SD} &= \sqrt{h * VAR(D_{BT})} \\ jitter_{SD} &= \sqrt{h} * \sigma(D_{BT}) \\ jitter_{SD} &= \sqrt{h} * jitter_{BT} \end{aligned}$$

Again with our results from the last section we can calculate the jitter for a 3 hop connection $jitter_{BT} = sqrt3 * 6ms \approx 10ms$.

C. BSR Simulation

With the results of the last sections we are ready to simulate the behavior of a large Bluetooth Ad-Hoc network. For this purpose we use a simple BSR simulator written in Java which simulates a configurable number of nodes on a rectangular area. The nodes don't move during a simulation run.

Each run has three phases. In a first phase the simulator places the nodes at random positions in the area. Next it determines for all nodes their neighbors. A neighbor is any node that resides within a circular area with a diameter of 10m around the node. Normally this would be done by inquiries but as a simplification we dropped this from the simulation. Then the simulator randomly picks a source and destination node and performs a BSR Path Discovery.

Per run the simulator outputs the number of Bluetooth connection setups that were altogether necessary to find the destination and the length of the resulting path. Of course there is also the possibility that a connection can't be established because there is no valid path from source to destination.

Our first test shows how the number of nodes in a fixed area (30m x 30m) influences the path length. Whereas 5 nodes provide a connectivity of only 33% this probability raises linearly until 25 nodes where we have a nearly 100% coverage (see figure 8a). 15 nodes (P=75%) seems to be a reasonable number where the use of BSR makes sense. Figure 8b shows that the connection setup time also raises with the number of nodes. With 100 devices a connection setup takes 17.5s. Even with the 15 nodes that provide a reasonable connectivity, we still have average setup times of 3.5s. During a call we can't tolerate such long interrupts so we definitely need optimizations like Local Repair or Backup Paths.

Next we want to evaluate how the size and shape of the area influenced the network. First we change the size of a quadratic area that is populated with 70 nodes. As you can see in figure 8c, the density of nodes is most of the time high enough to allow nearly 100% connectivity.

The average path length raises until 4.6 hops (figure 8d). This leads to a delay of up to 128ms which is tolerable for phone calls. The connection setup times rise up to 13s (figure 8e). Further tests clearly indicated that the path length as well as delay and jitter is mostly influenced by the diagonal length of the area. Figure 8f shows the dependency between diagonal length and the latency for a fixed number of nodes (70). This indicates that areas with a diagonal length of some 100 meters are not suited for BSR audio transmission.

To sum up the major simulation results:

- The connection setup time depends largely on the number of clients. Without optimizations in Path Maintenance the use of BSR is not reasonable, as otherwise the repair of defect paths would interrupt a phone call for some seconds. There should be no more than about 55 nodes involved in the Path Discovery as otherwise the initial connection setup would take more than 10s.
- The path length that largely influences delay and jitter depends on the diagonal length of the area. To limit the delay and jitter this length should be limited to 100 or 200m.

IX. Summary and Outlook

With our work we try to answer the question whether it is reasonable to build Bluetooth-based MANETs and use them for audio transmission. The results indicate that this depends largely on the intended use and environment. The area for such a network is limited to a field of about 100m x 100m because in larger areas the delay and connection setup times get unacceptable large. Furthermore the number of nodes in this area is restricted, too. If there are more than about 50 nodes in the area, we again have very long connection setup times.

The last aspect is a problem for scenarios like large fair halls, because there you might have a larger number of people within a hall. But e.g. within company buildings the use of our BSR protocol might enable people to make phone calls from cellphone to cellphone without the use of a traditional carrier.

For a more efficient use of BSR there need to be a number of changes done to the Bluetooth specifications. First of all the connection setup times need to be reduced drastically.

In our ongoing work we will try to add the optimizations outlined earlier to our simulations as well as build a real-world prototype as soon as suitable Bluetooth devices become available.

References

[1] Bluetooth SIG: Bluetooth Specification Version 1.1. 2001, http://www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf.

[2] J. Bray, C.F. Sturman: Bluetooth - Connect without Cables. Prentice Hall, New York, 2001.

[3] S. Corson, J. Macker: Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. IETF 1999, RFC 2501.

[4] V. Park, S. Corson: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. Proceedings of IEEE INFOCOM ’97, April 1997; pp 1405-1413.

[5] D.B. Johnson, D.A. Maltz: Dynamic Source Routing in Ad Hoc Wireless Networks. Mobile Computing, T. Imielinksi and H. Korth, eds. The Kluwer International Series in Engineering and Computer Science (vol. 35), Kluwer Academic Publishers, Norwood, Mass. 1996; pp. 153-181.

[6] C. Perkins, E. Royer: Ad-Hoc On-Demand Distance Vector Routing. Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications, Feb. 1999; pp. 90-100.

[7] E.M. Royer, C.-K. Toh: A Review of Current Routing Protocols for Ad-Hoc Mobile Networks. IEEE Personal Communications 6(2), April 1999; p. 46-55.

[8] L. Wernli, R. Semadeni: Bluetooth Unleashed - Wireless Netzwerke ohne Grenzen. 2001, http://www.tik.ee.ethz.ch/ beutel/projects/sada/sa_Semadeni_Wernli.pdf.

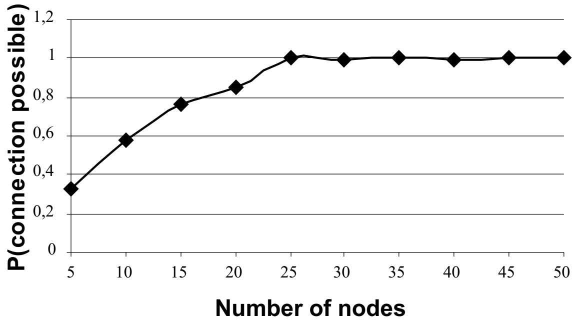
[9] Z. Wang, R. Thomas, Z. Haas: Bluenet - a New Scatternet Formation Scheme. In Proceedings of the 35th Annual Hawaii International Conference on System Sciences, January 2002.

[10] G Zaruba, S. Basagni, I. Chlamtac: Bluetrees - Scatternet Formation to enable Bluetooth-based Ad Hoc Networks. Proceedings of IEEE International Conference on Communications, 2001, pp 273–277.

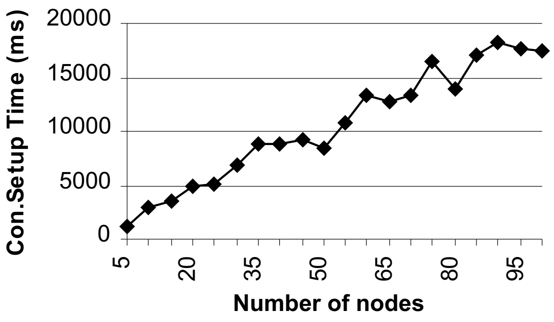
[11] Cambridge Silicon Radio: BlueCore01 12.7 Firmware Release Notes. 2001, http://www.csr.com/, (Only available at request).

[12] Ericsson Microelectronics: ROK 101 007 Bluetooth Module Datasheet. 1999, http://www.comtec.sigma.se/.

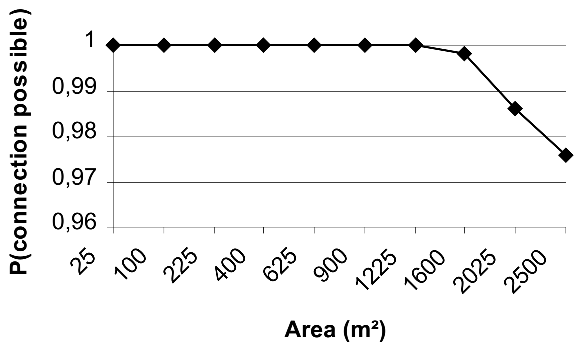
[13] AXIS: AXIS OpenBT Stack. 2002, http://sourceforge.net/projects/openbt/.



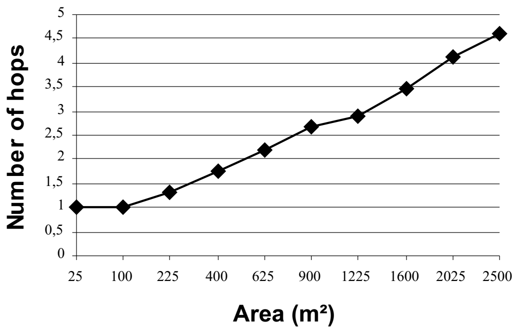
(a) Fixed area: connection propability depending on number of nodes



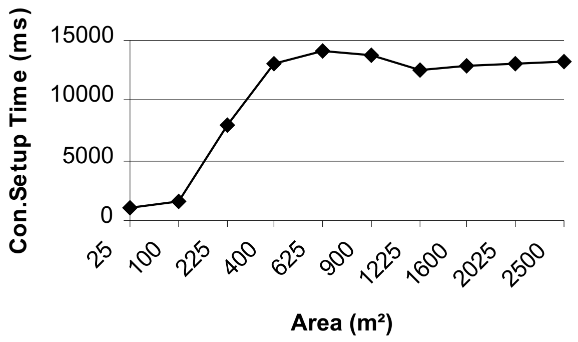
(b) Fixed area: average connection setup time depending on number of nodes



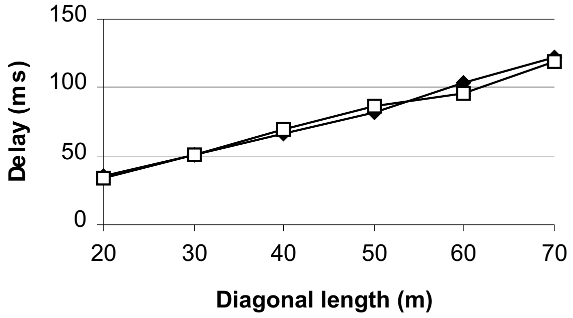
(c) Fixed number of nodes: connection propability depending on area



(d) Fixed number of nodes: average path length depending on area



(e) Fixed number of nodes: average connection setup time depending on area



(f) Fixed area and number of nodes: average delay depending on diagonal length

Fig. 8. Simulation Results