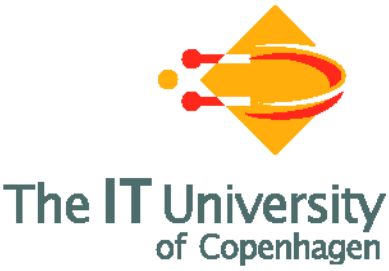

Real-world Bluetooth MANET Java Middleware

Michael Nielsen
Arne John Glenstrup
Frederik Skytte
Arnar Guðnason

IT University Technical Report Series **TR-2009-120**

ISSN 1600–6100 **October 2009**

ISSN 1600–6100

ISBN 9788779492028

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web **www.itu.dk**

Real-world Bluetooth MANET Java Middleware

Michael Nielsen
IT University of Copenhagen
Copenhagen, Denmark
Email: michael1982@itu.dk

Arne John Glenstrup
IT University of Copenhagen
Copenhagen, Denmark
Email: panic@itu.dk

Frederik Skytte
Email: skytte@itu.dk

Arnar Guðnason
Email: arnar@itu.dk

***Abstract*—We present BEDnet, a Java based middleware for creating and maintaining a Bluetooth based mobile ad-hoc network (MANET). MANETs are key to nomadic computing: Mobile units can set up spontaneous local networks when needed, removing the need for fixed network infrastructure, either as wireless access points or wired LAN. This allows for simple sharing of services (Internet access, media storage, databases, etc.) and resources (printers, scanners, GPS units, etc.). Application areas include nomadic offices, field workers' applications, in-home gaming; situations where mobile units are brought together for the duration of the task at hand.**

Based on the low-cost Bluetooth chip which is ubiquitous in handheld units, existing Bluetooth technology provides just piconets: star-shaped networks of up to eight units within a limited range of typically ten meters. BEDnet connects piconets into a scatternet, thus exceeding the eight devices and ten meters limits for spontaneous networking.

While algorithms for Bluetooth scatternet formation and routing have been studied extensively, to the best of our knowledge, BEDnet is the first implementation of Bluetooth scatternet middleware running on real mobile phones. Based on the Java JSR-82 specification, BEDnet is portable to a wide selection of mobile phones, and is publicly available as open source software.

Experiments show that e.g. media streaming over Bluetooth is feasible, and that BEDnet is able to set up a scatternet within a couple of minutes. Surprisingly, experiments showed that structured scatternet topologies are not significantly better than an ad hoc master/slave mesh topology. Experimental results also indicate that for routing a Bluetooth MANET, DSDV is more efficient than AODV, as DSDV requires less processing time per packet.

I. Introduction

Wireless communication is becoming increasingly popular as mobile devices are becoming more widespread. Since many mobile devices are shipped with Bluetooth chips [1], forming ad hoc Bluetooth networks is subject to much research.

A Bluetooth connection is the result of a complex device pairing process, and provides a channel on which many data services can be provided, such as voice, Internet communication, file sharing, printer connection etc. One of the downsides are that these connections are point to point, meaning that a device beyond the relatively short Bluetooth distance of approximately 10 meters will not be able to detect a given service [2].

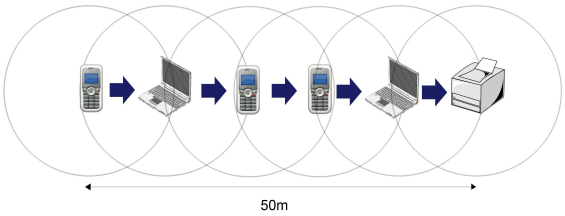

Fig. 1. A real multi-hop routing scenario.

The BEDnet framework provides broadcasting over several hops, in which one scenario (depicted in Figure 1) enables a phone 50 meters away from a printer to use it, by routing through other mobile phones and laptops (all running BEDnet). The framework, developed in J2ME using the JSR-82 API, acts as a middleware between the Bluetooth API and the application. The framework will automatically detect new devices, devices that have left the network and use a MANET routing protocol to efficiently route packets.

II. Related Work

Much existing literature in this field focuses on parts of our subject, egg. framework [3], different scatternet protocols [4]–[10], programming language solutions [11], [12], Bluetooth technology and optimisation [13]–[15] and P2P routing [16], [17], but very few specific implementation studies of MANETs over Bluetooth exist.

These attempts are mainly analysed theoretically [3] or performed on simulators [18], which invariably yields results that are far too optimistic, compared to the behaviour of real-world Bluetooth devices.

One exception to this are the real life studies by [16], [19], [20], in which they implement a MANET rout-

ing protocol over Bluetooth in order to investigate the feasibility of using such networks. They all manage to setup a multi-hop routing scenario, but also conclude that performing communication that requires high bandwidth and low latency is infeasible on the devices they had available. All had difficulties establishing scatternets, and fall back to either using point-2-point [16], [19] or implementing an ineffective packet forwarding method based on: 1) Searching for devices. 2) Connecting and sending packets. 3) Receiver searches for new devices; connects and sends, and so on, until the destination is reached [20].

Another investigation of using Bluetooth networking in an ad hoc manner is in relation to multi-player games [21]. They have benchmarked the Bluetooth connection, and try to fit a game category to their results. They conclude that realtime games, such as shoot-em-up games, are not possible to run due to too low bandwidth and high latency. As with the previous two examples, they were unable to establish a scatternet but uses point-2-point Bluetooth connections instead. However, all of their experiments were using devices with Bluetooth 1.1. They state that with the update to Bluetooth version 2.0, greater bandwidth and lower connection-time would enable more communication intensive games [21].

III. Enabling technologies

In the following, we describe key aspects of the four basic technologies underlying BEDnet: Bluetooth, Java for mobile devices, Java Bluetooth Application Protocol Interface (API), and Mobile Ad-hoc network (MANET) routing protocols.

A. Bluetooth

Bluetooth networks are basically organised as small groups of devices called piconets. All the devices in one piconet share the same communication channel, which consists of a unique pseudo-random pattern of radio frequencies in which the piconet devices hop synchronously to avoid radio interference with adjacent piconets. Each piconet consists of one master device and up to seven active slave devices. The address space of the active devices in a piconet limits its size to a maximum of 8 devices. In extension to the active devices, it is also allowed for devices to be parked, meaning that it leaves a piconet temporarily while still being synchronised to it. The address space for parked devices allow a maximum of 255 devices in this mode.

The benefit of parked mode is that a single device can leave a piconet temporarily while being active in another. The result is that a device can be a master in one piconet and a slave in others, or just a slave in several

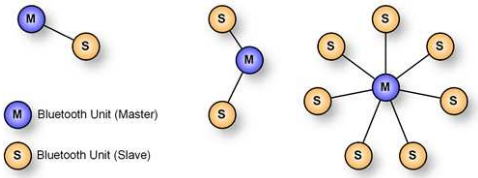

Fig. 2. Bluetooth piconet

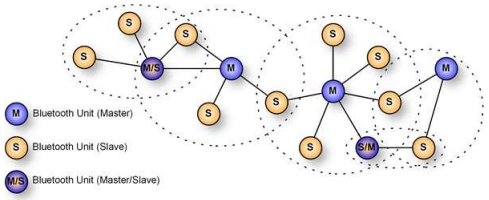

Fig. 3. Bluetooth scatternet

piconets. Devices that in this way are able to perform inter-piconet communication, connect the piconets into a scatternet (cf. Figure 3), allowing for communication between devices out of transmission range.

Connecting two devices via Bluetooth requires two phases [22]:

1)
   - *Inquiry*: This process consists of a sender broadcasting inquiry packets, which do not contain the identity of the sender or any other information.
   - *Inquiry Scan*: In this state, receiver devices listen for inquiry packets, and upon detection of any such packet, the device broadcasts an inquiry response packet. This contains the identity of the device and its native clock.
2)
   - *Page*: When paging, a sender device tries to form a connection with a device whose identity and clock are known. Page packets are sent, which contain the sender's device address and clock, for synchronisation.
   - *Page Scan*: In this state a receiver device listens for page packets. Receipt is acknowledged and synchronisation between the devices is established.

In the inquiry phase, the sender discovers potential receivers. This phase takes at least 10.24 seconds [23], [24], but in our experience it takes longer, typically around 15 seconds, depending on the device.

After the inquiry phase, the device is aware of the Bluetooth addresses of the discovered devices. When performing service discovery in the paging phase, the sender connects to each desired device individually, taking a few seconds. Service searching uses the Service

Discovery Protocol (SDP), which describes how an application acts to contact Bluetooth servers services [25]. SDP uses a simple request/response model and the data a service server stores, is contained in a single Service Record. Some of the information stored is the services ID, name and description. Providing these information to a client is valuable as it can evaluate whether or not the service is relevant for the device and/or application.

When the sender has established a connection it becomes master for the receiver, which in turn becomes a slave.

The nature of this process also implies that the Bluetooth radio cannot be used for communication across existing connections during inquiry or paging phases.

B. Java for mobile devices

The Java Micro Edition (ME) is a subset of the Java Platform providing a collection of Java APIs for the development of software for small resource constrained devices [26]. Over a billion mobile phones supporting Java have been sold giving a 45% market penetration, a number that is constantly growing as new phones produced almost all support Java [27]. This makes Java a good choice when developing mobile applications for portability.

Resource constrained devices that support Java can choose to implement different profiles and configurations [28]–[30]. For mobile devices the most common configuration is Connected Limited Device Configuration (CLDC), and profile Mobile Information Device Profile (MIDP). Together, CLDC and MIDP provide a subset of the features known from Java Standard Edition.

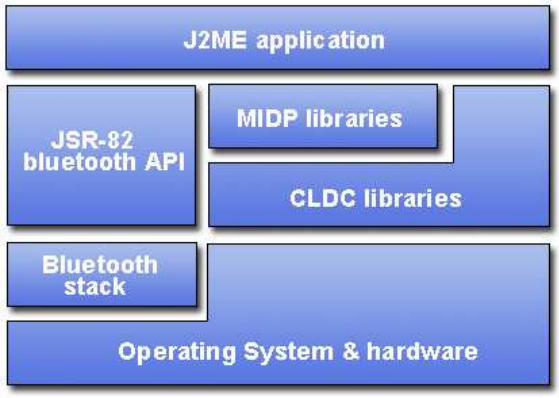

Fig. 4. Configuration for Small Devices in Java ME

In addition to the profile other Java Specification Request (JSR) packages can be implemented as needed, like JSR-82 package described in the next section. Figure 4 shows how the configuration and profiles relate to the application, Operating System (OS) and hardware.

For a mobile device, the application created is called a MIDlet. A number of MIDlets are bundled into a *MIDlet suite,* which allows method invocation among the contained MIDlets. MIDlet suites are packed into Java archive files that can be deployed on mobile phones via Bluetooth or HTTP.

C. Java Bluetooth API

To use Bluetooth in a Java ME application one must use Java's Bluetooth API, called JSR-82 [31]. JSR-82 lies as a middleware between the application and the Bluetooth stack, allowing the same access to the stack no matter how it is implemented. The stack can be written in native code, C or Java, but they must all offer the same interface, defined by JSR-82, to be called Java compatible. When this interface is supported, a developer can use JSR-82 methods in theory without considering the underlying stack implementation; in practise, the underlying implementation must in some cases be considered [32].

The rather limited interface provided by JSR-82 can restrict an application attempting to use Bluetooth, and from our experience the manufacturers often use the most conservative settings when implementing their stack. That means that in most cases, paging and inquiring are not allowed during connections, master switching (switching the rôles of two devices after connecting) is turned off and only one service discovery can be performed at a time; when using the JSR-82 API, these quirks must be considered. As a developer there is nothing to be done about this except try to work around it as good as possible.

While JSR-82 provides wide portability, its limited interface restricts the control of the Bluetooth operations. JSR-82 does not provide any way of accessing methods or attributes in the Bluetooth stack [33], limiting the program's abilities to determine what JSR-82 actually does when performing its actions. One cannot detect what connections are currently open or active for a device, what role the device plays (master or slave) in accordance with the connections, or how and when polling of communication with other devices is done.

When attempting to create a scatternet, this makes it hard to know how it is actually done in the Bluetooth stack, as assumptions about the device roles have to be made, based on the method invocations. Also, when a device is connected to several devices, the method in which the application listens for incoming packets runs

asynchronously to the Bluetooth radio polling scheme, which could be a possible optimisation issue.

D. Mobile ad hoc networks (MANETs)

The main characteristics of MANETs are:

- Distributed operation: A device cannot rely on an underlying network. It must provide security and routing so they work under distributed conditions.
- Dynamic network topology: Devices are, generally speaking, in motion. The network must be able to handle new devices as well as devices moving in and out of range.
- Low power devices: in many cases the devices are battery driven, affecting CPU processing, memory, signal processing and more.

During the last decades, many routing protocols have been developed, with various properties, such as scalability, robustness and network traffic optimisations [34]. Generally, the development has taken two directions: Proactive and on-demand routing. Proactive routing primarily depends on a routing table (hence it is also referred to as table driven routing protocols), while this is not necessarily a property of the on-demand protocols. On-demand routing will try to establish a route when communication is needed, reducing idle state overhead and improving scalability [34]. This can be especially relevant when operating in a highly dynamic topology.

In this paper, we have chosen a routing protocol of each flavour in order to investigate which works better on the BEDnet framework:

1) Proactive: Destination Sequenced Distance Vector (DSDV)
2) On-demand: Ad-hoc On Demand (AODV)

Both routing protocols are quite simple, and thus well-suited to run on resource-limited mobile devices.

1) DSDV routing: Each device maintains a routing table and its own sequence number, which is kept as an even number, and only updated when broadcasting its routing table. An entry in the routing table consists of a sequence number as well as distance in some metric (e.g., hop count), next hop and destination address. When a device wants to send a packet to another device, it looks up the destination address to find the next hop address.

Routing table updates are performed in two ways; full dump and incremental update. The full dump is a periodical broadcast of the entire routing table to all neighbours. The incremental update is more frequent and is a broadcast of changes in the routing table that have occurred since the last full dump.

Upon receiving an update, the device compares each route in its routing table to the received routing table entries. If a route is not present in its own routing table it is added by incrementing the hop count and setting the sending device as the next hop. If the route exists in the receiver's routing table, the hop count and sequence number are compared to the existing values. If the received route has a lower hop count or a higher sequence number, it is accepted as the new route to that destination.

If a device leaves the network, the neighbour device will detect this. If a device detects a broken link, it increments the sequence numbers for all routes using this link to an odd number, indicating an invalid route. This update is then followed by an immediate incremental update, ensuring fast propagation.

2) AODV routing: This protocol was developed as an improvement over DSDV, since tests have shown that DSDV has a large overhead as the network size increased. The somewhat simple solution to this overhead problem was to have the nodes only store routes recently used, and if a new route is needed, it would request it from other devices in the network in a broadcasting manner.

When a node requests a route it does not have in its routing table, it will broadcast a Route Request (RREQ) packet to its neighbours. Upon receiving a RREQ, a node will evaluate the packet, and if the packet's destination address is the node's address, a Route Reply (RREP) packet is unicast back along the return path. As the source node receives the RREP packet, the route will be set up, and it can transmit the required packets to the destination node. If a node has already seen a RREQ it is disregarded, to keep paths short.

Each RREQ uses sequence numbers to ensure that the routes are loop free and to make sure that if the intermediate nodes reply to route messages, they reply with the latest information only [34].

Should a node fail, or move, when data is being transmitted, the device that recognises the link break will send a Route Error (RERR) packet along the route to the source node, which will broadcast a new RREQ packet.

IV. Framework design

A. framework compartmentalisation

In order to achieve a truly robust and flexible framework, that can handle a wide variety of applications, the following requirements need to be met [11]; a) platform independence and b) the possibility of interchanging components or modules of the system.

Platform independence can be achieved by using the Java programming language. The second requirement is met by designing the framework in layers, with each

layer capable of being replaced as long as it adapts to a specified interface. The BEDnet framework consists of three layers:

- Application: The part that the user will interact with, such as a chat program or game.
- Routing: In order to send packets, the system will use the routing layer. This layer will hold all details on how to route packets to other devices in the network.
- Datalink: Will handle all Bluetooth specifics, such as maintaining links to neighbours, sending and receiving packets through them.

B. Scatternet formation

An ad-hoc mesh topology is able to connect the devices, allowing the devices to communicate. However, when comparing this topology with the more structured topologies, egg. trees or rings [22], it could be more inefficient due to more overhead or longer routes. In the past years, many interesting Scatternet Formation Algorithm (SFA)'s have been introduced. Most of these protocols have only been tested in simulators, with a certain abstraction level, and thus it is necessary to evaluate whether they can be expected to perform well in a realistic environment, using J2ME with the JSR-82 API.

Any SFA we use for BEDnet must satisfy the following properties:

The SFA cannot assume that every device is within communication range. Some SFA's assume this [4], [5], [22], and these can therefore not be expected to work properly in a realistic environment.

The SFA must not perform Master-Slave (MS) switching frequently. As role switching is blocking the Bluetooth connection on the involved devices, the longer time it takes to perform a switch, the longer a connection is inaccessible [31]. In practise this is done by closing and reestablishing the connection with the new master slave configuration. We have observed that this process takes a few seconds to perform, which could sever a network for the duration.

Several Bluetooth procedures must not run concurrently. Some SFA's are using separate concurrently running Bluetooth processes perform maintenance procedures and discover new devices. This enables a device to maintain topology while being able to locate new devices and perform the actual communication [6], [22]. Due to the properties of Bluetooth this is not possible unless a device has several Bluetooth chips. As this is not the case in most of todays Bluetooth enabled devices, all Bluetooth related tasks must be run sequentially.

The SFA have no access to any operating system, or Bluetooth stack specific parameters. A number of protocols assume access to system, or even hardware-specific data, such as the power level or full access to all components in the Bluetooth stack. The L2CAP layer is the lowest accessible layer when working in J2ME with the JSR-82 API [31].

Since common scatternet procedures, such as master/slave switching and maintaining a specific topology is time consuming and a complex affair, we want to limit that as much as possible. Currently, no published SFAs that have been optimised for the Bluetooth stack, and for that reason we implement a simple SFA, that creates an ad-hoc Master-Slave mesh, which requires a minimum of maintenance and does not force Bluetooth roles on any devices.

C. Bluetooth device discoveries and connections

As most of the Bluetooth operations are blocking, it is impossible to e.g. perform a device discovery while sending packets to already connected devices. However, one element that is not blocking is listening for devices that are performing device discoveries. Thus, it is possible to communicate while listening. For this reason, devices that already have open connections are also listening for incoming devices.

If a device wants to connect to an existing network, the quickest way will be to start performing a device discovery, as the other devices will be listening for joining devices. However, if no network exists, that approach will not work, as no devices will be listening. To solve this, a random wait algorithm for connecting devices is used. This will randomly alternate between listening and searching. The time a device spends listening is determined by measuring the time it takes to perform a full device discovery, which ensures that other devices are able to establish a connection to it.

V. Implementation

The implementation was written to support CLDC version 1.0 and MIDP version 2.0. This maximises compatibility, as not all mobile devices with Bluetooth chips support the newer CLDC 1.1 and MIDP 2.1.

The BEDnet framework has been written as a single MIDlet and can be bundled with any MIDlet Suite. Many mobile devices are unable to execute several Java applications at the same time, thus the framework, and the applications that use it, are designed to run as a single application.

A. Data structure

Figure 5 shows an overview of the BEDnet framework. Each layer was developed as its own package containing the interface for which the above layers could use. Another sub-package, containing the framework interface was also developed.

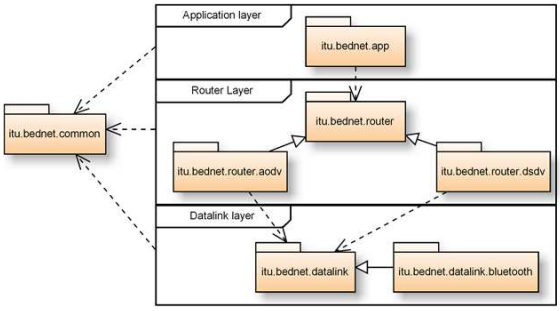

Fig. 5. Package diagram for BEDnet

The Datalink layer package consists of 7 classes and the router layer consist of either 27 (AODV) or 15 (DSDV) classes. The application layer package does not have any classes by default, as this is up to each developer to write.

VI. Experiments

In order to establish the highest available performance, a lightweight benchmark application was developed. This application is able to measure the latency (as round trip time) and throughput over a single hop. The BEDnet framework will be benchmarked against the obtained values from the benchmark application.

A. Single hop benchmark

In order to investigate the practical limits of the bandwidth and latency of Bluetooth, this lightweight benchmark application was developed as a very simple MIDlet that was only able to communicate point-2-point (Server/client communication).

Bandwidth was measured in two ways; 1) Average transfer rates were determined by transferring 10Mb data and 2) peak bandwidth is determined as the fastest transferral of 250kb in 50kb chunks. Latency was measured as Round trip times (RTT), where a total amount of 1000 packets were sent.

The results are depicted in table I. It clearly shows that the BEDnet framework does not utilise the full bandwidth of Bluetooth. The reasons for this are most likely related to the increased complexity of the application and the increased packet processing time. Packet handling are probably the most time consuming element in BEDnet; each packet has to be opened, read and compared to the internal Bluetooth address in order to establish whether a packet should be delivered to the application layer. If the packet destination address does not match with the devices address, it has to look up the next hop in the routing table, which also consumes time. As the time it takes to perform the processing is CPU dependent, the older mobile phones this framework was tested on, also shows the least performance.

Bluetooth v1.1	benchmark	BEDnet
Bandwidth avg	323±11 kbps	44 kbps
Bandwidth peak	633 kbps	45 kbps
RTT	50±4 ms	65±10 ms
Bluetooth v2.0	Benchmark	BEDnet
Bandwidth avg	866±244 kbps	141±24 kbps
Bandwidth peak	1792 kbps	163 kbps
RTT	46±4 ms	55±5 ms

TABLE I
SINGLE-HOP BENCHMARK OF BANDWIDTH AND LATENCY.

B. AODV vs. DSDV

The processing times were measured by sending 2000 packets, and subtracting the time spent in the Bluetooth stack. Noticeably, even though the Sony Ericsson has a CPU clocked at much lower speed, it is able to process the packets faster. When using the DSDV protocol on Sony Ericsson k750i, the processing time was too fast to be able to measure with Java's system call to retrieve the time.

Table II shows the time spent for DSDV and AODV in milliseconds for Sony Ericsson and Nokia mobile phones. The reason that we distinguish between brands here are the substantial differences observed. Even with a slower CPU, Sony Ericsson appears to be handling the packets faster than Nokia. The reasons for this could be differences in the operating systems priority of the virtual machine and threads and the implementation of the virtual machine.

We have also observed a noticeable difference in handling times between the two routing protocols. We believe that this observation lies in the difference of the two applications' complexity. Mobile phones have slow CPUs compared to laptops, and very limited memory, which can have a severe impact on performance when using many threads with a given Time-To-Live (TTL), which stresses the garbage-collector and thus lowers the performance. AODV heavily relies on route timeouts in order to be on-demand, whereas DSDV stores the routes much longer, which could be less stressful on the virtual machine.

Device	CPU	DSDV	AODV
Nokia 6120	ARM-11@369MHz	1.3ms	27ms
Sony Ericsson k750i	ARM-9@110MHz	0ms	4.5ms

TABLE II
AODV AND DSDV PROCESSING TIME COMPARISON.

C. Network join times

Figure 6 shows the connection time of a single device to a BEDnet network, with all network devices within range. All communication between the devices in the network was turned off. The figure shows that there is an increase in connection times when more devices are in the vicinity, which is most likely a result from the extra service records the joining device has to process. The increase in time is almost 100%, from approximately 13 seconds with no devices in range to approximately 25 seconds with 6 devices in range.

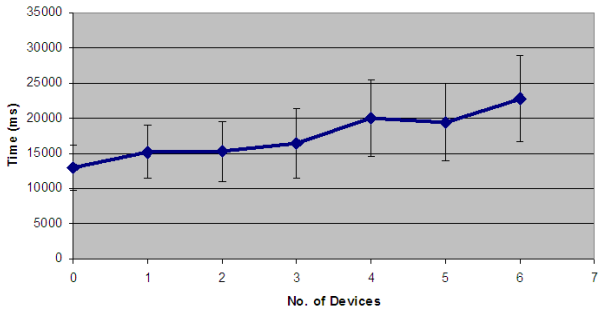

Fig. 6. Network join times

Experiments with bootstrapping (forming a BEDnet network from scratch) shows that it takes between 45 and 82 seconds to connect 2 to 5 devices.

D. Scatternet formation

We tested whether or not it pays off to have a structured scatternet topology or if an ad-hoc mesh is sufficient. A linear topology consisting of 3 to 5 devices is set up in different scatternet topologies, and the RTT is measured by sending 100 packets 10 times over the network diameter (the longest distance). Figure 7(a–g) show the setups used.

Figure 7(d–e) shows the two formations we tested with four devices, a mesh scatternet and an optimised scatternet, containing 3 and 2 piconets respectively. From Table III we see that in this case the mesh scatternet performed better, with 22% higher RTTs for the optimised scatternet.

With five devices we are able to form three different linear scatternet topologies, with 2, 3, and 4 piconets. In table III it is shown that all three topologies have nearly

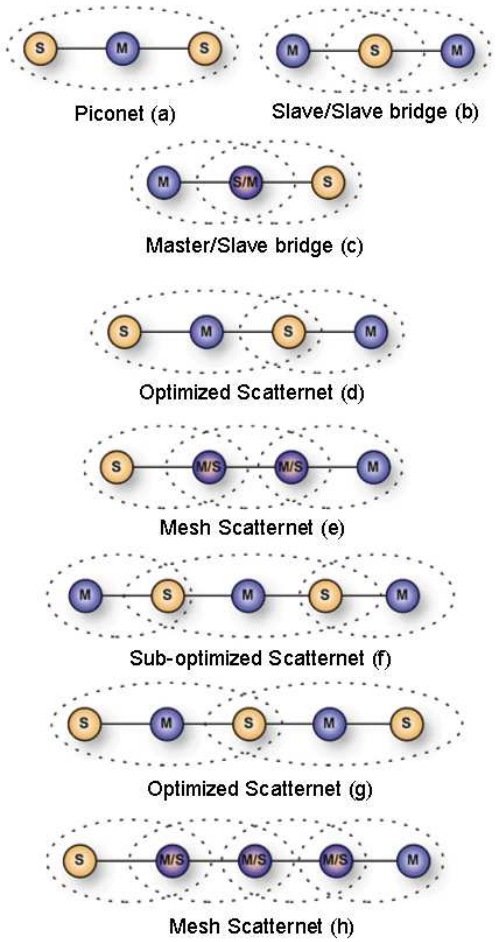

Fig. 7. Linear scatternet formations for 3 to 5 devices

Topology	RTT
Two hops	
Fig. 7(a)	113±8 ms
Fig. 7(b)	109±8 ms
Fig. 7(c)	100±11 ms
Three hops	
Fig. 7(d)	254±93 ms
Fig. 7(e)	207±87 ms
Four hops	
Fig. 7(f)	459±102 ms
Fig. 7(g)	516±90 ms
Fig. 7(h)	469±20 ms

TABLE III
RTTS IN DIFFERENT SCATTERNET FORMATIONS FOR 2,3 AND 4 HOPS

the same RTTs, which indicate that the topology is not the bottleneck in the system.

VII. Conclusion & Future Work

In this paper we have presented the BEDnet framework, which we have shown is able to route packets in a multihop fashion on real life devices over Bluetooth connections. We have showed that it is possible to form a scatternet, and also shown that when using J2ME with the JSR-82 Bluetooth API there is no performance penalty when a device is both a master in one piconet and slave in another. As we have also shown that the connection phase is still very time consuming in practise, we recommend that the usage of structured scatternet topologies, like rings etc., are not used.

When considering the choice of MANET routing protocol we have shown that DSDV, a quite simple protocol that works well in smaller networks [35], requires less processing time for each packet than AODV, which makes it a good candidate for a routing protocol in Bluetooth networks. If BEDnet is to be used in large networks (several hundreds of devices), AODV should be considered [36].

Currently, BEDnet does not utilise the full bandwidth of Bluetooth, but only around 15%, which could be explained by the complexity of the framework. Experiments have shown that the upper limit of the network performance is approximately 1800kbps when using Bluetooth v2.0 with EDR, which we expect the BEDnet framework to be able to approach in the near future.

A packet round trip was found to take approximately 50ms for one hop, and increasing rather fast to around 500ms for four hops, which indicate that in this current version of BEDnet, low latency applications, such as real time gaming, should be avoided when networks can increase only a few hops. The benchmark application was not significantly faster, with RTTs almost the same as BEDnet. However, we don't believe that the main difference lies in the single hop scenario, but in the multihop, since factors like how BEDnet polls its connections for packets, how long it takes to look up a route process the result of the lookup into a send command to the next hop device.

In the near future we will perform bandwidth and latency experiments using more devices and focus on interesting topics such as investigating in the frameworks robustness, implementing multihop service search in a variant of [37], enabling devices out of range retrieving service records from each other, and develop a .NET platform that could enable laptops to use the BEDnet framework and e.g. act as a proxy for the Internet.

We believe this area to be promising, and we will continue to develop the BEDnet framework. We have showed that nomadic computing on Bluetooth enabled devices is now possible, as we can now operate beyond the 10 meter range, point-to-point and eight devices restrictions that currently are on todays Bluetooth enabled devices.

References

[1] Forrester and 80211.Com-report. (2003) Wi-Fi versus Bluetooth - Forrester Opines Against the Conventional Wisdom. [Online]. Available: http://www.mobileinfo.com/news_2003/Issue20/Forrester_WiFiBluetooth.htm

[2] BluetoothSIG, "Specification of the Bluetooth system v2.0," *Specification Volume 0*, 2004.

[3] C. Bisdikian, "A framework for building Bluetooth scatternets: A system design approach," *Pervasive and Mobile Computing 1*, pp. 190–212, 2005.

[4] M. Sun, C. Chang, and T. Lai, "A Self-routing Topology for Bluetooth Scatternets," *I-SPAN*, 2002.

[5] L. L.-Y. Shek and Y.-K. Kwok, "An integrated approach to scatternet traffic management in Bluetooth ad hoc networks," *Computer Networks 45*, pp. 99–118, 2004.

[6] M. Tekkalmaz, H. Sözer, and I. Körpeoglu, "Distributed Construction and Maintenance of Bandwidth-Efficient Bluetooth Scatternets," *2005. ICC 2005. 2005 IEEE International Conference on Communications*, pp. 3223–3229, 2005.

[7] L. L.-Y. Shek and Y.-K. Kwok, "Efficient Multi-Hop Communications in Bluetooth Scattemets," *The 14m IEEE lntemational Symposium on Persona1,lndoor and Mobile Radio Communication Proceedings*, pp. 755–759, 2003.

[8] M. Kalia, S. Garg, and R. Shorey, "Scatternet Structure and Inter-Piconet Communication in the Bluetooth system," *IEEE National Conference on Communications New Delhi*, 2000.

[9] T. Lin, Y. Tseng, K. Chang, and C. Tu, "Formation, routing and maintenance protocols for the bluering scatternet of Bluetooths," *Proceeding of the Hawaii International Conference on System Science (HICSS-36), Big Island*, 2003.

[10] C.Blundo and E. Cristofaro, "Configuring Bluestars: Multihop Scatternet Formation for Bluetooth Networks," *IEEE Transactions on Computers, Vol. 52, No. 6*, pp. 779–789, 2003.

[11] N. Pabuwal, N. Jain, and B. N. Jain, "An Architectural Framework to deploy Scatternet-based Applications over Bluetooth," *2003. ICC '03. IEEE International Conference on Communications*, pp. 1019–1023, 2003.

[12] J. . E. Group and jsr 118-comments@jcp.org, *Mobile Information Device Profile for Java 2 Micro Edition, Version 2.1*. Sun Microsystems, Inc. and Motorola, Inc., 2006.

[13] P.-C. Wei, C.-H. Chen, C.-W. Chen, and J.-K. Lee, "Support and optimization of Java RMI over a Bluetooth environment," *Concurrency Computat.: Pract. Exper.*, pp. 1–21, 2002.

[14] S. B. Handurukande, S. Ganguly, and S. Bhatnagar, "Fast Bluetooth Service Discovery for Mobile Peer-to-Peer Applications," 2006.

[15] F. Ferraguto, G. Mambrini, A. Panconesi, and C. Petrioli, "A new approach to device discovery and scatternet formation in Bluetooth networks," *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004.

[16] S. S. Kristiansen, "Bluetooth enabled Peer2Peer services in ActorFrame," *Master of Science in Communication Technology*, pp. 1–117, 2006.

[17] C.Blundo and E. Cristofaro, "A Bluetooth-based JXME infrastructure," *Proceedings of the 9th International Symposium on Distributed Objects, Middleware, and Applications*, pp. 667–682, 2007.

[18] Y. Noishiki, H. Yokota, and A. Idoue, "Design and implementation of ad-hoc communication and application on mobile phone terminals," *3rd International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*, pp. 208–214, 2006.
[19] S. Ali, "The Feasibility of Mobile Adhoc Routing Over Bluetooth - and a discussion about the realism of simulations," *Masters thesis in Computer Science at University College London*, pp. 1–62, 2007.
[20] N. Jain, "Scatternet Formations for Multimedia Applications over Bluetooth Personal Area Networks," *Masters Thesis: Dept. of Computer Science and Engineering, at The Indian Institute of Technology, Delhi*, 2002.
[21] C.Blundo and E. Cristofaro, "Issues related to development of wireless peer-to-peer games in J2ME," *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, 2006.
[22] R. M. Whitaker, L. Hodge, and I. Chlamtac, "Bluetooth scatternet formation: A survey." *Elsevier - Ad Hoc Networks 3*, pp. 403–450, 2005.
[23] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A Formal Analysis of Bluetooth Device Discovery," *International Journal on Software tools technology transfer manuscript*, 2006.
[24] R. Woodings, D. Joos, T. Clifton, and C. Knutson, "Rapid Heterogenous Connection Establishment: Accelerating Bluetooth Inquiry Using IrDa," 2001.
[25] E. A. Gryazin, "Service Discovery in Bluetooth," *Group for Robotics and Virtual Reality, Department of Computer Science, Helsinki University of Technology*, 2000.
[26] SunMicrosystems. (2008) Java ME technology. [Online]. Available: http://java.sun.com/javame/technology/index.jsp
[27] W. Hardy, "Journal 6: JavaOne and Trends in the Java Framework," *Web Developer & Designer*, 2006.
[28] K. Topley, *J2ME in a Nutshell*. O'Reilly, 2002.
[29] SunMicrosystems. (2008) Connected Limited Device Configuration (CLDC). [Online]. Available: http://java.sun.com/products/cldc/
[30] T. Appnel, "Introducing MIDP 2.0," *OnJava.com*, 2002.
[31] M. Milikich, "Java APIs for Bluetooth Wireless Technology (JSR 82)," 2005.
[32] SunMicrosystems. (2008) JSR 82 - Bluetooth API 1.1. [Online]. Available: http://java.sun.com/javame/technology/msa/jsr82.jsp
[33] A. Gudnason, M. Nielsen, and F. Skytte, "Designing and implementing Bluetooth Scatternets in Java," 2007.
[34] P. Misra, "Routing protocols for ad hoc mobile wireless networks," *Ohio State University - http://www.cis.ohio-state.edu*, pp. 1–20, 2000.
[35] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications*, pp. 234–244, 1994.
[36] R. Khalaf, A. El-Haj-Mahmoud, and A. Kayssi, "Performance comparison of the aodv and dsdv routing protocols in mobile ad hoc networks," 2005.
[37] A. Greede and D. O'Mahony, "A service driven routing protocol for Bluetooth scatternets," *Network And Telecommunications Research Group (NTRG) Computer Science Department*, 2006.
[38] F. Skytte, A. Guðnason, and M. Nielsen, "BEDnet—Bluetooth Enabled Device ad-hoc Network," Master's thesis, IT Univ. of Copenhagen, 2008.

ACRONYMS

AODV Ad hoc On-demand Distance Vector
API Application Protocol Interface
BEDnet Bluetooth enabled device ad-hoc network
BT Bluetooth
CLDC Connected Limited Device Configuration
CRC Cyclic Redundancy Check
DAC Device Access Code
DSDV Destination-Sequenced Distance Vector
DSR Dynamic Source Routing Protocol
ERR Exhaustive Round Robin
FEC Forward Error Correcting
FHS Frequency Hopping Synchronization
FHSS Frequency Hopping Spread Spectrum
GUI Graphical User Interface
HCI Host Controller Interface
HEC Header Error Check
IDE Integrated Development Environment
IP Internet Protocol
JSR Java Specification Request
JVM Java Virtual Machine
L2CAP Logical Link Control and Adaptation Protocol
LCDUI Liquid Crystal Display User Interface
LM Link Manager
LMP Link Management Protocol
MANET Mobile Ad-hoc network
ME Micro Edition
MIDP Mobile Information Device Profile
MS Master-Slave
MTU Maximum Transmission Unit
OS Operating System
OSI Open Source Initiative
OLSR Optimized Link State Routing Protocol
PAN Personal Area Network
PDU Protocol Data Unit
PLsWRR Pseudo Random cyclic Limited Slot Weighted Round Robin
PRR Pure Round Robin
PSB Parked Slave Broadcast
QoS Quality of Service
RF Radio Frequency
RFCOMM Radio Frequency Communication
RTT Round Trip Time
RVM Routing Vector Method
SDP Service Discovery Protocol
SIG Special Interest Group
SFA Scatternet Formation Algorithm
UML Unified Modeling Language
UUID Universally Unique Identifier
WAP Wireless Application Protocol
ZRP Zone Routing Protocol