Beddernet: Application-Level Platform-Agnostic MANETs

Rasmus Sidorovs Gohs, Sigurður Rafn Gunnarsson, Arne John Glenstrup

▶ **To cite this version:**

Rasmus Sidorovs Gohs, Sigurður Rafn Gunnarsson, Arne John Glenstrup. Beddernet: Application-Level Platform-Agnostic MANETs. 11th Distributed Applications and Interoperable Systems (DAIS), Jun 2011, Reykjavik, Iceland. pp.165-178, 10.1007/978-3-642-21387-8_13 . hal-01583579

HAL Id: hal-01583579

https://inria.hal.science/hal-01583579v1

Submitted on 7 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Beddernet: Application-Level Platform-Agnostic MANETs

Rasmus Sidorovs Gohs, Sigurður Rafn Gunnarsson, Arne John Glenstrup

rasmus.gohs@gmail.com, sigurdur.rafn@gmail.com,
arne.glenstrup@gmail.com

This paper introduces Beddernet, a platform-agnostic mobile ad-hoc network framework. The Beddernet architecture is designed to work with different networking protocols - the version detailed here supports Bluetooth ad-hoc networks or scatternets.
Although considerable work has gone into researching and designing scatternets, no standard has been agreed upon and no scatternet protocol can be found in Bluetooth specifications. Beddernet fills this gap and can become a useful tool both for research and real-world applications. The standard is open and free to use, and is detailed in a separate Beddernet Specification Document. Beddernet middleware has been tested on Java and Android devices with good results. The reference design of Beddernet is based on the Android Operating System and is available under an open source license.

Keywords: MANET, peer-to-peer, mesh, networking, DSDV, Android, multicast, Bluetooth, mobile.

1 Introduction

Mobile devices like handheld gaming devices and mobile phones are becoming quite accommodating; the latest mobile phones have several connectivity features and powerful application processors. These devices rely mostly on some infrastructure such as a WLAN or a mobile phone network to communicate with each other and the world. This isn't always feasible or desired. One solution to this is to have the devices themselves interconnect and create mobile ad-hoc networks, MANETs. Such networks can enable devices to share data and resource sharing for e.g. collaborative work, file sharing and gaming without any infrastructure or central control.

MANETs do require some processing power and ideally an advanced operating system to run on. As powerful mobile devices with sophisticated operating systems not hampered by these problems are now commonplace, MANETs can have an important place in the world by augmenting infrastructure in places where it is weak,

expensive or non-existent. For this to be possible, devices need a standard to connect and communicate.

This paper proposes a solution to this problem in *Beddernet*, an advanced application level MANET protocol with self-organising and self-healing capabilities. The typical usage scenario would be MANETs consisting of 2 to 20 devices.

The next chapter discusses some work related to this project. Chapter 3 and 4 briefly introduces the technologies and concepts Beddernet relies on. Chapter 5 details the design of Beddernet, its protocols and structure. Several experiments were performed to test Beddernet's performance and functionality. Those experiments are discussed in Chapter 6, Evaluation. Conclusions and perspectives for the project's future are then discussed in the final chapter.

2 Related work

In an ad-hoc network, individual nodes cooperate to create and maintain the network and to route data. A *scatternet* is such a network where the nodes use low power Bluetooth communication for connections.

BEDnet, the predecessor of Beddernet, is a real-world scatternet application based on the Java Platform, Micro Edition (JME). Due to some limits of Bluetooth on JME, BEDnet eschews complicated *Scatternet Formation Algorithms* and has devices connect to each other in a simple mesh creating algorithm.

BEDnet showed good results, formed scatternets reliably, routed data accurately, and proved useful in applications such as turn-based gaming and text messaging, and moderately successful in media sharing applications [1]. Performance was below the theoretical maximum transfer speeds of Bluetooth, but it was believed this could be managed using better hardware and possibly some optimisations in code. The Beddernet project builds on the success of BEDnet and addresses its shortcomings.

Although Bluetooth is the only widely spread protocol that supports device-to-device connections, little work seems to have been done designing Bluetooth scatternet standards or software for mobile devices. Scattercom [2] is written for the Symbian OS and is based on a proactive routing protocol, but does not offer APIs for third party applications. A project by Ibraheem [3] is implemented in JME and uses a reactive routing protocol. The project seems to target transferring a 4kb file in a maximum of 10 seconds on a two hop scatternet, so it does not seem to be intended to support applications such as interactive real time gaming and media streaming. Finally, Donegan et al. [4] present another JME project, originally designed to facilitate parallel computations over Bluetooth scatternets. Although claimed to be general enough for further deployments, it has not been codified as such.

3 Technologies

Bluetooth communication, scatternet formation/routing, and Android OS, the basic technologies the Beddernet prototype builds upon, are described in the following.

3.1 Bluetooth

Bluetooth is a wireless standard for low powered, short range data exchange. It is implemented in e.g. computers, mobile phones, and video game consoles [5]. Bluetooth devices are uniquely identified by their address and are arranged in star networks called *piconets* [6], each consisting of up to 8 active devices, one of which is designated as *master*. Devices in a piconet communicate using a shared medium. The master assigns specific time intervals, *time slots*, to each connected device to transmit data to or from the master, cf. Fig. 1. More than 7 devices can be registered with the master but are then put into *park mode* [6], where they are considered a part of the piconet but are not assigned time slots. Parking and un-parking of devices has a negative impact on performance [7]. For a device to be able to join a piconet it needs to identify the address and clock of the piconet master [7]. This is done in two phases by the master; inquiry for discovering new devices and paging for establishing a connection.

Each phase consists of two modes; listening (scanning) and transmitting. For two devices to exchange address and clock information they must be in opposite modes.

In the first phase, inquiry, the master discovers the slave address and clock. Next, in the paging phase, the master sends its address and clock to the slave and the devices are connected. To avoid interference, devices hop to a different radio frequency at each time slot. Each piconet uses a specific hopping pattern identified by the master's address and clock. When both identities and clocks have been interchanged, the frequency hopping sequence can be synchronized and data exchange can begin. The device initially in inquiry transmitting mode becomes the master of the connection.

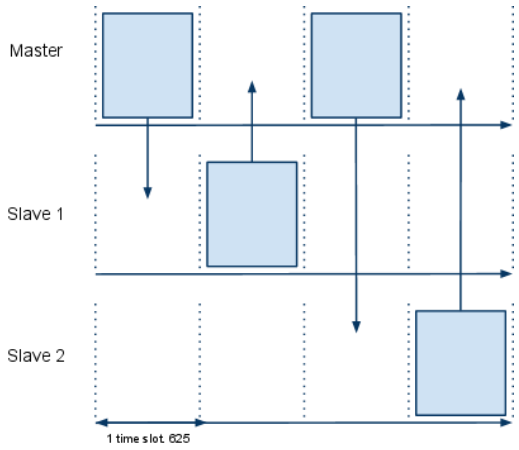

Fig. 1 Bluetooth switching

RFCOMM.
Being the only connection protocol available in both Java and Android, the RFCOMM Bluetooth protocol is used by Beddernet (Fig. 2). This stream-oriented protocol relies on the automatic retransmission and in-order sequencing provided by the lower base-band layer for reliability in transmissions between connected devices.

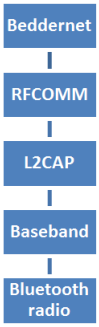

Fig. 2 Bluetooth radio stack

3.2 Scatternet Formation

Thanks to frequency hopping, several piconets can overlap geographically without interference. A node in one piconet can join another, thereby connecting them. It is possible to connect several piconets using this method, the resulting network is known as a Bluetooth *scatternet*, cf. Fig. 3. The fundamental problem of forming a self-organizing scatternet of Bluetooth devices is non-trivial and is an active area of research [7]. Ensuring connectivity requires nodes to agree on a scatternet formation algorithm (SFA), specifying how they interconnect. Several different algorithms have been proposed with different characteristics [7].

A general problem with implementing many of the algorithms is that they make assumptions that impede usage in platform agnostic standards. Some e.g. assume all devices are in range of each other, or that devices have access to such information as the link management in the Bluetooth stack, location of devices or battery levels [1].

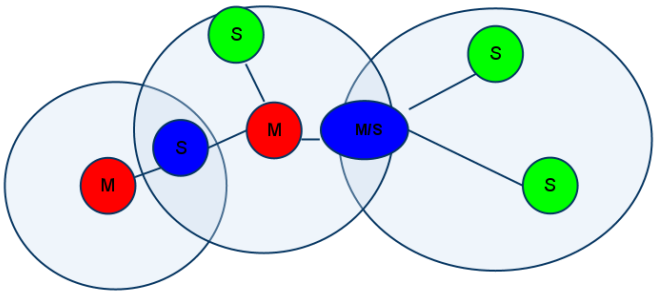

Fig. 3 Scatternet - three piconets form a scatternet via bridge nodes (blue)

3.3 Routing

MANETs such as Bluetooth scatternets are more volatile than normal computer networks; devices can appear spontaneously, move around, and then disappear again. In the face of such network churn, special routing protocols have been designed, broadly speaking in two classes: *proactive* and *reactive*. Proactive protocols attempt to maintain a recent list of all nodes and/or routes on a network by regularly exchanging routing information updates. Reactive protocols like Ad hoc On-Demand Distance Vector (AODV) find routes on demand, usually by flooding request packets. Simulations suggest that AODV is better suited than e.g. the proactive Destination-Sequenced Distance Vector (DSDV) algorithm for highly volatile ad-hoc networks [8], but actual experiments have shown that in some cases the route lookup takes an inexpediently long time, outlasting even the actual Bluetooth transmission time [9]. AODV also requires more processing per packet than DSDV [1]. These properties were factored in when designing Beddernet.

4 Mobile programming frameworks

Beddernet is designed to be simple and platform agnostic. A reference implementation has been created on the Android mobile platform [12]. Applications on Android can run as background services and can communicate with other applications on a device making it very suitable for a Beddernet reference implementation.

5 Framework design

Beddernet adheres to a 3 layer architecture having a data-link, a routing, and an application layer. All communication between Beddernet devices, both for maintaining the scatternet and for transmitting data, is via discrete Beddernet messages. The first byte (or bytes in special cases) of each message is a control byte. It denotes what type of message follows. Different message types are used for maintaining routing information, carry data etc.

The following sections describe the function of each layer.

5.1 Datalink Layer

The Datalink layer contains the functionality that concerns the actual connection medium, Bluetooth in the case of Beddernet. This layer holds all connections to neighbour devices and sends and receives Beddernet messages from the routing layer.

Scatternet Formation.
A reliable scatternet framework must make sure connected scatternets are created, but also maintain the scatternet as nodes appear, move around and disappear. Beddernet

attempts to accomplish this with a two-phased algorithm that first creates a mesh based scatternet and then enters an active maintenance phase.

The Beddernet framework is designed to be a general framework and does not assume information like battery status or location is available. Therefore the simple but functional mesh algorithm described below is used.

Phase 1 - Mesh creation

As a node starts Beddernet, it tries to establish a connection with other devices in range. It randomly alternates between listening and transmitting modes until a connection can be established. This random factor prevents devices from being constantly in the same mode and ensures that a device eventually connects with other devices if they are in range. When connection has been established, knowledge of other devices is exchanged, thereby quickly establishing a fully connected scatternet.

Phase 2 – Maintenance

A Beddernet device that connects to another device stops scanning as frequently and enters a maintenance phase. In this phase it spends most of the time being discoverable, allowing for incoming connections, but only performing device discoveries intermittently. As device discovery is generally a power intensive procedure that interrupts communication [13], it should be done as rarely as possible. To achieve this, Beddernet uses a dynamic maintenance algorithm that slows scanning frequency linearly with the number of connected Bluetooth neighbours.

The time T between devices discoveries is thus regulated by the following formula:

$$T = \begin{cases} (N+1) \times T_0 \times X, N < 7 \\ \infty, N \geq 7 \end{cases}. \quad (1)$$

where N is the number of connected neighbours, T_0 is some constant time interval and X is a random number between 0 and 1. The maintenance protocol runs continuously, regularly scanning for new devices. This enables two or more established scatternets to merge automatically. (cf. Fig. 3).

5.2 Routing Layer

As discussed earlier, reactive protocols tend to scale better than proactive ones. As a Beddernet usage scenario was presumed to be typically 2-20 devices, this was not seen to justify the added complexity of such reactive protocols. Therefore, Beddernet uses DSDV. This also makes the implementation of advanced features such as multicasting [10] and service discovery simpler than if using AODV [11].

Multicast.

Multicasting can save bandwidth and increase throughput in some scenarios (Fig. 4) and is included in the Beddernet protocol [15], using a stateless explicit multicast algorithm because of its simplicity and efficiency [10]. The special Beddernet multicast message header can contain multiple Bluetooth addresses. The number of

addresses is indicated by a control byte that precedes the address list, supporting up to 255 destination addresses within a single multicast message. The protocol could be extended to support reverse multicast, by having each intermediary device aggregate replies before returning them towards the multicast source but this is not a part of the specification.

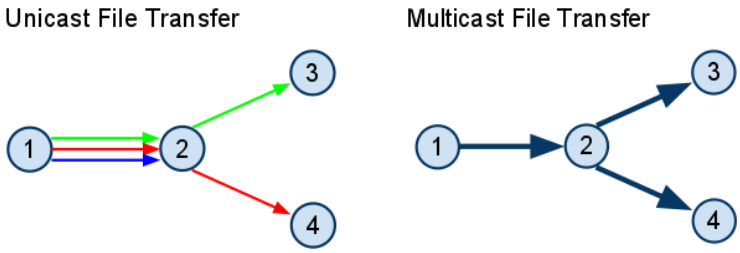

Fig. 4 Unicast vs. multicast file transfers

5.3 Application Layer

Beddernet is designed to work with several concurrent applications running on different platforms and devices without interference. The following discusses briefly how this is done in Beddernet.

Unique Application Identifier.
Applications in Beddernet are given a 64-bit Unique Application Identifier (UAI). It is obtained by hashing the application's human readable name into a 64-bit sequence. This identifier is then used to route messages to the correct application on the destination device making it possible to run several applications concurrently. Although this method does not guarantee collision-free application routing, it makes the risk of collisions very improbable [16]. It two applications do get the same UAI, application designers can modify the name they provide to Beddernet. Information about active applications on a device is propagated proactively in Beddernet, embedded in the DSDV routing messages, cf. Table 1 and Table 2. This proactive approach entails an overhead of 8 bytes per control message.

Table 1. Route Broadcast Message

Type	Senders address	Recipients address	Is route down?	Number of RTE	Routing Table Entries
1 byte	6 bytes	6 bytes	1 boolean	1 int	1-* RTE

Table 2. Routing Table Entry (RTE)

Type	Destination Address	Number of Hops	Sequence Number	Number of UAIH	UIAH
1 byte	6 bytes	1 int	1 int	1 byte	1-255 longs

6 Evaluation

To test the practical performance of Beddernet a series of tests were run on the Android reference implementation and on a JavaSE implementation, created for this purpose. Tests on the JavaSE version were carried out on several homogeneous and stationary Windows XP SP3 workstations with identical unbranded and generic class 2, version 2.0 + EDR Bluetooth hardware.

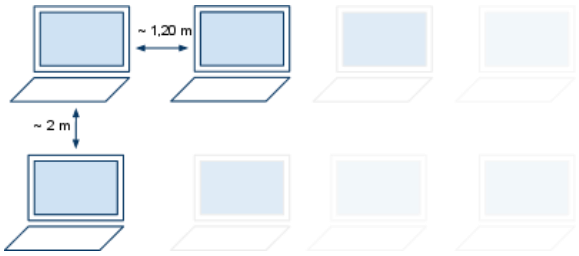

Fig.5 Default test setup

6.1 Performance

To measure performance and explore the cost of routing a message through intermediary nodes, bandwidth and latency was measured in a linear scatternet where up to six devices were connected in a chain; making up a scatternet of five piconets. (cf. Fig. 6) RTT and average throughput was measured between the first and last devices. The last device in the chain was then disconnected, performance measured again, etc. until only two devices were left.

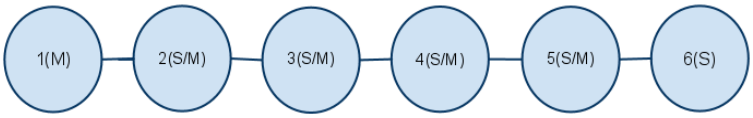

Fig.6 Multi-hop bandwidth and latency test

6.2 Latency

As expected, latency increases linearly with the number of hops in a route (cf. Fig. 7) although some tests showed that congestion can be a factor in overloaded scatternets.

Fig. 7 Multihop RTT

This effect shows clearly that latency dependant applications are strongly affected by the number of hops between devices.

6.3 Bandwidth

Bandwidth between two connected devices is around 600ms under the default lab conditions (cf. Fig. 8) while a two hop file transfer is half as fast. This is expected, as the total bandwidth available has to be split in two; the intermediary node reads from one device and then writes to the next.

Fig. 8 Multihop bandwidth

As another hop is added into the route, sending data through two intermediaries, bandwidth suffers another drop in speed, 44% from the last bandwidth measurement. This drop seems high as the bandwidth available between device 3 and 4 is logically

similar as between device 1 and 3. Additional penalties are then incurred as more hops are added, although much smaller.

Table 3. Multi-hop test results

Hops	1 (base)	2	3	4	5
Bandwidth	697 kbit/s	279 kbit/s	156 kbit/s	127 kbit/s	113 kbit/s
RTT	35 ms	101 ms	153 ms	225 ms	297 ms
Percentage of base bandwidth	100%	40%	22%	18%	16%
Percentage of RTT base	100%	288%	437%	643%	848%

One possible reason for this performance drop may be that the increase in the number of piconets leads to some inefficiencies in exchanging data between the piconets.

To test this, an experiment was carried out. Two different 3-hop scatternets were created (cf. Fig. 9), one containing 3 piconets, the other with only 2. Bandwidth was 13% higher in the 2 piconet setup, suggesting that some bandwidth is lost when nodes hop between piconets.

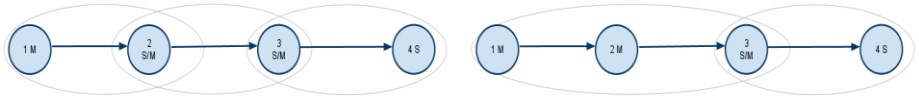

Fig. 9 Different three hop scatternet configurations

6.4 Message size

The design of Beddernet allows for arbitrarily sized messages, some experiments were carried out to assess if some set maximum/minimum size in the specifications would be advisable.

Larger message sizes were shown to increase transmission speed in a simple bandwidth test with different message sizes. Profiling shows [14] that Beddernet has negligible overhead in CPU usage and overhead is a small percentage of total data sent, so most of the gains of using larger message sizes were presumed to be due to the costs of initiating RFCOMM transfers [14].

Although large message sizes improve bandwidth, very large messages sent across a scatternet were speculated to cause problems for latency dependent applications

because of possible congestive effects. A test designed to explore this showed that large messages can completely occupy a connection for several seconds leading to a negative impact on latency for competing transmissions [14]. A message size of 5000 bytes gave a good balance between responsiveness and bandwidth in tests and has been designated as the maximum and default message size in Beddernet.

6.5 Topology

Previous performance tests focused on the number of hops in a linear scatternet. To explore what effect topology may have, another test was conducted. Bidirectional bandwidth was measured between two devices. Then, another device was added to the piconet and the test repeated between the original two devices (Fig. 10). This resulted in a 32% drop in throughput. Adding more devices lead to additional performance drops.

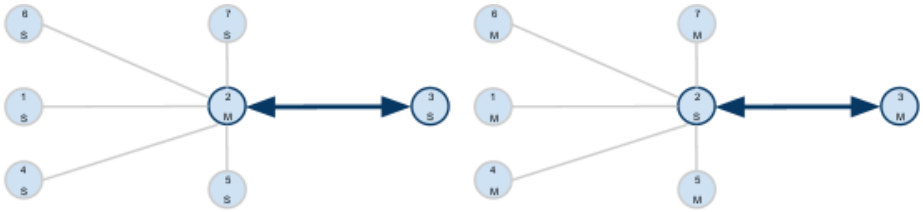

Fig. 10 Piconet bandwidth test, multiple slaves vs. multiple masters

The results from this experiment seem to indicate that the master device divides the available total bandwidth equally between all connected devices rather than assigning active devices more slots.

Conversely, changing the setup so that a single slave was connected to multiple masters lead to only a slight decrease in performance compared to the previous, single piconet test. This almost constant throughput is speculated to be because of a node's ability to go into sniff mode. In sniff mode a device can be absent from one piconet for a longer period of time while being engaged in another without losing connectivity [6].

It would seem that devices only use the sniff mode to negotiate between different piconets and not to increase bandwidth within a piconet. If this effect is common in Bluetooth hardware implementations, it may have a considerable effect on the performance of SFAs in real-world settings. Designing a SFA that leverages this factor and takes other experimental results into account could show some real improvements over older designs. The algorithm would minimise hops while preventing masters from having many slaves. The topology produced could e.g. resemble an inverted Bluestar [17].

Scatternet proposals with a very high number of masters could raise questions of interference issues though, due to the larger number of piconets. Calculations from

[18] indicate that this is not a critical concern as e.g. $R = 4$ concurrent piconets would only experience an interference related drop of $I = 4\%$ in a simplified worst case scenario, ignoring error correction etc., using the formula:

$$I = 1 - \left(1 - \frac{1}{79}\right)^{R}. \tag{2}$$

At this point, tests have shown that both routing through intermediary nodes and having extra nodes in a piconet causes considerable performance drops, cf. Table 3. To give a better picture on scatternet performance in real-world usage, a new test was designed, combining these two factors.

A new 3 hop scatternet was set up and bandwidth measured. Then, 2 inactive nodes were added to the scatternet, connecting to the two intermediary nodes as shown on Fig. 11. Bandwidth was measured again, revealing a 39% drop in throughput.

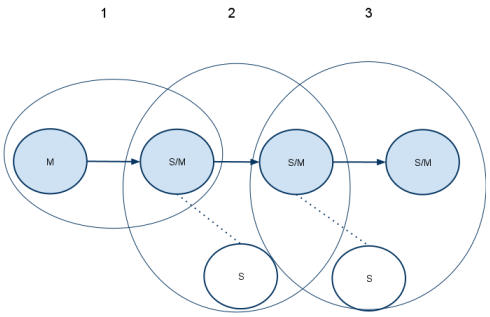

Fig. 11 Multiple hops with extra nodes, the two inactive nodes added are white.

These results are somewhat surprising. The devices are already performing far under their available bandwidth capacity but still incur bandwidth penalties as devices are added, even if these new devices are inactive.

Table 4. Multiple hops with extra nodes

Hops	2	3	4	5
Simple chain	303 kbit/s	198 kbit/s	165 kbit/s	151 kbit/s
With two inactive nodes	198 kbit/s	142 kbit/s	119 kbit/s	107 kbit/s

6.6 Multicast Performance

The multicast feature of Beddernet was tested by setting up a scatternet as on Fig. 4. Transfer using multicast were 53% faster than using unicast. This isn't surprising as each message only needs to be sent through three individual connections and not five as with unicast. This experiment shows the promise of using multicast in scatternets for applications such as streaming media to multiple nodes.

7 Conclusion and Future Work

Despite the possible utility of mobile ad-hoc networking, such networks are not yet a standard feature of mobile devices. The Beddernet project was started to provide a free and open standard to enable multi-platform scatternets, both for research and real-world projects.

Implementations of the simple Beddernet protocol have been shown to work on different platforms with good results. Performance has been tested and although highly dependent on scatternet topology, shown to be sufficient to enable different useful applications. Performance is only expected to improve as mobile processors and Bluetooth adapters become faster. The Beddernet project is considered to have reached its technological goal. The real success of Beddernet, however, depends on its usefulness to research and in real world deployments. To encourage adoption and development the source code is open source and can be downloaded from the project home page [19].

As experiments indicate that setting up RFCOMM connections is costly, implementing L2CAP protocol might reveal some performance gains, but as of this writing, the Android SDK has no supports for L2CAP.

Beddernet currently supports the DSDV routing algorithm, but the loosely coupled design allows for easy implementation of different routing algorithms. B.A.T.M.A.N has been identified as a promising routing protocol [21] and it would be interesting if a larger real world comparison would be made, not only measuring overhead and bandwidth, but also the practical use of such an algorithm for features such as service discovery and multicasting.

Lastly, Beddernet's usefulness could be increased by adding more transmission protocols e.g. Wi-Fi to the datalink layer. The standard, (802.11a/b/g/n) is very widely deployed and is getting more common in mobile devices, providing long communication range and high transfer speeds [20].

8 Bibliography

1. Michael Nielsen, Arne John Glenstrup, Frederik Skytte and Arnar Guðnason, "Real-world Bluetooth MANET Java Middleware". Technical report TR-2009-120, IT-University of Copenhagen , 2008.
2. Scattercom, http://sourceforge.net/projects/scattercom/
3. Ibraheem, "Development of Routing Algorithm Based on Bluetooth Technology". Thesis, University of Technology, Iraq, December 2006.

4. B. Donegan, D. Doolan, S. Tabirca, "Mobile Message Passing using a Scatternet Framework", International Journal of Communications & Control 3(1), 2008.
5. Bluetooth. Wikipedia, http://en.wikipedia.org/wiki/Bluetooth
6. Bluetooth specifications: Core Specification v2.0 + EDR, Bluetooth SIG, 1994.
7. Roger M. Whitaker, Leigh Hodge and Imrich Chlamtac, "Bluetooth scatternet formation: A survey", Ad Hoc Networks 3, 2005.
8. Azzedine Boukerche, "Performance Evaluation of Routing Protocols for Ad Hoc Wireless Networks", Mobile Networks and Applications 9(4), 2004.
9. Michael Nielsen, Arne John Glenstrup, Frederik Skytte and Arnar Guðnason, “Bluetooth Enabled Device ad-hoc NETwork”, 2009
10. Lushend Ji, M. Scott Corson, "Explicit Multicasting for Mobile Ad Hoc Networks", Mobile Networks and Applications 8(5), 2003
11. J.C. Haartsen, S. Mattisson "Bluetooth–a new lowpower radio interference providing short-range connectivity", Proceedings of the IEEE 88, 2000
12. Android guide, http://developer.android.com/guide/basics/what-is-android.html
13. Android documentation: Bluetooth, http://developer.android.com/guide/topics/wireless/Bluetooth.html
14. R. Gohs, S.R. Gunnarsson "Bluetooth Scatternet Framework For Mobile Devices (Beddernet) ", IT - University of Copenhagen, 2010
15. R. Gohs, S.R. Gunnarsson "Beddernet Protocol Specifications 0.1 ", IT - University of Copenhagen, 2010
16. Birthday problem, http://en.wikipedia.org/wiki/Birthday_problem
17. Dubhashi et. a, "Blue pleiades, a new solution for device discovery and scatternetformation in multi-hop Bluetooth networks", Kluwer Academic Publishers , 8 May 2006
18. J.C. Haartsen, S. Mattisson "Bluetooth—a new lowpower radio interference providing short-range connectivity", Proceedings of the IEEE 88(10), 2000
19. The Beddernet project homepage, http://code.google.com/p/beddernet/
20. Wi-Fi Alliance: Wi-FI Direct, http://www.wi-fi.org/Wi-Fi_Direct.php
21. S. Annese, C. Casetti, C. Chiasserini, P. Cipollone, A. Ghittino, M. Reineri "Assessing Mobility Support in Mesh Networks", WiNTECH’09, September 21, 2009