

Bluetooth-based P2P Content Distribution to Mobile Users

Uichin Lee, Sewook Jung, Alexander Chang, Dae-Ki Cho, Mario Gerla
 Department of Computer Science
 University of California, Los Angeles
 {uclee,sewookj,acmchang,pinecho,gerla}@cs.ucla.edu

Abstract—Using handheld devices such as PDAs, cellular phones and smart phones for personal entertainment has become commonplace in today’s lifestyle. Most of these devices are equipped with Bluetooth technology that can be used to distribute entertainment contents such as music, movie clips, and commercial films. However, content sharing faces several challenges such as short communication range, limited bandwidth, user mobility, performance variations between Bluetooth cards of different versions and from different manufacturers. This paper analyzes the characteristics of Bluetooth connection establishment and data phases and based on these evaluates the performance of Bluetooth Peer-to-Peer (P2P) communications in a mobile environment. The paper then investigates techniques to enhance Bluetooth content sharing performance. Simulation experiments document the improvements obtained with our proposed techniques.

I. INTRODUCTION

Bluetooth is an always-on, low-power, short-range radio technology suitable for Wireless Personal Area Networks (WPANs). Recent legislation banning drivers from using mobile phones without a hands-free kit has given a tremendous boost to Bluetooth-based mobile headset sales. This in turn has cleared the way for Bluetooth inclusion in all kinds of products, starting with the Bluetooth gradual replacement of cables between computers, keyboards, printers, and mouse.

With Bluetooth continuing proliferation, Bluetooth-based applications are well positioned to deliver new opportunities to all facets of the industry. One fast growing application is proximity advertising and marketing; namely, the wireless distribution of advertisements associated with a particular locality. In 2005, handset maker Nokia and music label EMI started a project to let coffee shop customers listen to music via Bluetooth [1]. Last year, CBS announced that it will make video clips from its prime-time fall program line-up available for free via CBS Outdoor billboards [6]. The digital billboards, initially located in NYC’s Grand Central Station, will pipe video clips directly to cell-phones and PDAs. In Europe, Viacom Outdoor installed a permanent network of Bluetooth-enabled posters on the London Underground to distribute content to subway riders. Channel 4, a public-service British television station uses posters to promote FourDocs, its broadband documentary channel which encourages the public to view and make their own four minute documentaries.

Such solutions are based on a client/server model: Bluetooth Access Points (BT-APs) push contents to mobile users. Pushing a relatively large file (5-10MB) to mobile users is

challenging, however. The mobile user is in contact with a BT-AP only for a short time due to short range (i.e., 10m) and to mobility. Moreover, the Bluetooth channel is error-prone, due to obstacles and WiFi interference. Finally, it is neither practical nor economical to install BT-APs every 10m. Under such circumstances, distributing contents larger than several mega bytes requires users to stop at the nearest BT-AP, unless they adopt P2P content sharing.

Given this, Jung et al. [12] proposed BlueTorrent, a cooperative content sharing protocol for Bluetooth that exploits sparsely distributed BT-APs and allows mobile users to cooperatively download relatively large files. To effectively share content in spite of short link duration, BlueTorrent uses BitTorrent-like file swarming. Content is divided into a number of small chunks (called pieces), and mobile users can exchange whatever pieces they store. BlueTorrent uses a cooperative carry and forward strategy: pieces are forwarded as soon as a connection is made in order to minimize download latency. Meta-data (e.g., unique file ID, title, media type) of a file is also spread opportunistically via mobility.

In this paper, we study the feasibility of Bluetooth-based Content Distribution (BCD), e.g., BlueTorrent. Our main focus is to measure the performance of Bluetooth communications in “mobile” environments. To this end, we emulate a mobile user with an Amigobot¹, a programmable robot that carries a laptop on its back and can travel at the speed up to 2.2m/s. To the best of our knowledge, this is the first experiment with Bluetooth in an externally controlled mobile environment. The measurements include peer discovery latency, connection setup latency and download bandwidth in an environment with variable mobility. In addition, the compatibility among different Bluetooth versions is examined, and is shown that it has a crucial impact on design/evaluation of a Bluetooth system. As shown in Figure 1, different Bluetooth versions populate the market, due to the slow evolution of Bluetooth standards and the rapid pre-standard adoption. Thus, our performance measurements consider the impact of different Bluetooth versions.

After characterizing Bluetooth characteristics and performance in a dynamic urban environment (i.e., mobility, obstacles, WiFi interference, etc.), we find strategies that can improve the system performance. To this end, we review the key aspects of the BlueTorrent system, namely channel uti-

¹<http://www.activrobots.com>

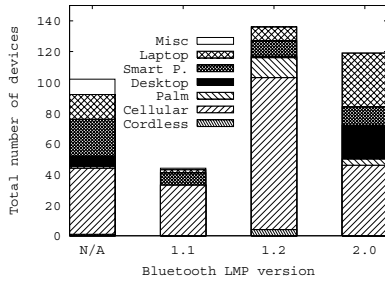


Fig. 1. Bluetooth LMP version distribution: The statistics were gathered in the ABC campus on Nov. 29/30, 2006. 75% of the total 402 devices allowed making connections, from which the Bluetooth Link Manger Protocol version information (that is almost equivalent to the Bluetooth version) was fetched.

lization, resource discovery, and resource availability. First, we propose a method for adaptive Bluetooth packet selection so as to fully utilize the channel. Secondly, we discuss solutions that can effectively reduce the resource discovery latency; thus, for given contact time, a node can spend more time for data transfer. Third, we evaluate coding techniques such as rateless codes and network codes that can potentially increase the “usefulness” of a contact. We also consider an energy efficient scheme that saves energy with minimal performance penalty. These schemes are validated with a combination of simulations and testbed experiments.

The following is a preview of the key results reported in the paper.

- From measurements we find that *i*) peer discovery and connection setup latency do not vary with distance between BT devices, whereas data rate varies significantly with distance, mainly due to the automatic rate control scheme in Bluetooth called Channel Quality Driven Data Rate (CQDDR); *ii*) when Bluetooth and WiFi coexist, they interfere with each other affecting their data throughputs; in particular, a long range Bluetooth device (i.e., 100m, Class 1) showed a significant impact on WiFi; *iii*) Advance features in Bluetooth (i.e., transmission power control and adaptive frequency hopping) do not work well, mainly because of their long response time in mobile scenarios; *iv*) a significant throughput loss is observed in the field test, i.e., 71% and 44% for Bluetooth v1.1 and v2.0 respectively.
- We found that the particular type of Bluetooth versions/chipset has a great impact on peer discovery, connection setup, and data throughput. In addition, our simulation results show that system performance heavily depends on the popularity of Bluetooth versions.
- We proposed and evaluated techniques that improves the performance of a BlueTorrent system: *i*) A receiver feedback scheme for L2CAP packet size control best utilizes the Bluetooth channel; *ii*) Adaptive inquiry mode where the inquiry mode is initiated by other peers conserves energy with minimal performance degradation; *iii*) network/rateless codes yield at most 15% improvement in the tested scenarios.

This paper is organized as follows. Section II introduces the basics of Bluetooth. Section III describes the experimental setup and presents our measurement results. Section IV describes various schemes to enhance BlueTorrent performance.

The following section evaluates the proposed schemes via extensive simulations. Related work is reviewed in Section VI. Finally, Section VII concludes the paper.

II. BLUETOOTH OVERVIEW

Bluetooth is defined as layered protocol architecture. Radio specifies details of the physical layer of Bluetooth. Baseband concerns with peer discovery, connection setup, addressing, packet format, power control, and timing etc. Link Manager Protocol (LMP) is responsible for link setup between Bluetooth devices and link management. This includes the control and negotiation of baseband packet size and transmission power. Logical link control and adaptation protocol (L2CAP) adapts upper-layer protocols (e.g., segmentation and reassembly, protocol multiplexing, flow control per L2CAP channel, error control and retransmissions, etc.) to the Baseband layer via Host Control Interface (HCI). In this section, we review Radio and Baseband of Bluetooth. Readers can find more in the specification [3].

In Bluetooth, the total bandwidth is divided into 79 channels (each 1 MHz). Frequency hopping (FH) occurs by jumping from one physical channel to another in a pseudorandom sequence. The hop rate is 1600 hops per second, so that each physical channel is occupied for $625\mu\text{s}$. This interval is referred to as a slot and is numbered sequentially. The basic unit of networking in Bluetooth is a *piconet*, consisting of a master and from one to seven active slave devices. The master determines a hopping sequence based on its Bluetooth ID (referred as an FH channel) that shall be used by all devices on this piconet. The FH channel is shared between a master and slaves using Time Division Duplex (TDD); i.e., data are transmitted in one direction at a time with transmission alternating both directions. Multiple slaves share the piconet medium using Time Division Multiple Access (TDMA). This FH channel is a form of Code Division Multiple Access (CDMA), and thus, several piconets can coexist with minimal interference. Two piconets will occasionally use the same physical channel during the same time slot, causing a collision and data loss, but this happens rarely, and it is recovered by *forward error correction (FEC)* and *error detection/ARQ techniques*.

Physical link and packet types: Baseband defines an asynchronous connectionless (ACL) link for a point-to-multipoint link between the master and all the slaves in the piconet. Achievable data rates on the ACL link vary depending on the number of slots per packet (1, 3, and 5 slots) and on the FEC strategy. FEC packets formats are DM1 (17B), DM3 (121B), and DM5 (224B), and the non-error coded formats are DH1 (27B), DH3 (183B), and DH5 (339B). The maximum data rate is 732.2Kbps with DH5. As of Bluetooth v2.0 Enhanced Data Rate (EDR), Phase Shift Keying (PSK) modulation has added. PSK increases coding bits per symbol to 2 and 3 bits, introducing EDR packet types *without FEC* as 2-DH1/2-DH3/2-DH5 (packet size= $2 \times \text{DH}x$, max rate=1448.5Kbps) and 3-DH1/3-DH3/3-DH5 (packet size= $3 \times \text{DH}x$, max rate=2178.1Kbps) respectively.

Peer Discovery: The first step in establishing a piconet is to identify devices in range that wish to participate in the

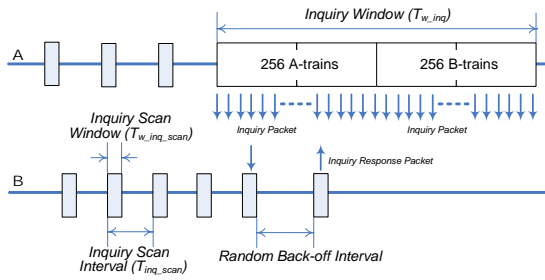


Fig. 2. Bluetooth inquiry procedure example: The size of the inquiry window is 5.12s ($=4 \times 1.28s$).

piconet. As shown in Figure 2, the inquiry procedure begins when the potential master transmits an ID packet with *Inquiry Hopping Sequence* (Tx slot) and waits for the response packets from the slaves with the *Inquiry Response Hopping Sequence* (Rx slot). The Inquiry Hopping Sequence is divided into two distinct sequences called A- and B-train. Each train contains 16 physical channels and must be repeated at least 256 times (i.e., $2.56s = 256 \times 10ms$) before switching to another train. During the inquiry window (T_{w_inq}), the master repeats the following: send two inquiry packets in each Tx slot, and then listen to response packets from other devices in the following Rx slot. Note that the host controller interface allows us to set the interval as a *multiple* of 1.28s through *hci-inquiry* host controller interface (HCI) command. The potential slaves periodically (i.e., T_{inq_scan}) enter the Inquiry scan state and stay there for inquiry scan window ($T_{w_inq_scan}$) to search for inquiry packets. When a node receives an inquiry packet, it enters the Inquiry Response state and returns an FHS packet after backing off a random number of slots to avoid collision. A FHS packet contains the device address, page scan repetition mode (PSRM) and clock offset (CO), required by the master to initiate a connection.

Connection setup: Once the master has found devices within its range, it can establish a connection to each device (i.e. paging). For a device to be paged, the master uses the slave device address to calculate the page hopping sequence. For the phase in the sequence, the master uses the estimate of the slave’s clock from the inquiry response, if available. To synchronize the phase, the master node performs the similar procedure as the inquiry procedure, but using the page hopping sequence. The potential slaves periodically (i.e., T_{page_scan}) enter the Inquiry scan state and stay there for page scan window ($T_{w_page_scan}$) to search for paging packets. The slave page scan mode can be either $R1$ ($T_{page_scan} \leq 1.28s$) or $R2$ ($T_{page_scan} \leq 2.56s$). The master repeats each train N_{page} times, specifically $N_{page} \leq 128$ for $R1$ and $N_{page} \leq 256$ for $R2$, which is specified in the inquiry response packet. The slave immediately returns the response packet after receiving the paging packet. The master then responds with its own FHS packet, and the slave sends back an acknowledgement. As a result, a connection is established and both master and slave begin to use the connection hopping sequence defined in the master’s FHS packet. Note that as of Bluetooth v1.2 the *interlaced inquiry/page scan* is introduced such that the slave scans two trains in a row. Thus, the probability of missing an inquiry/page packet is negligible [24].

Power class: A Bluetooth device is typically classified into

two power classes: Class 1 and Class 2 (100m and 10m radio range respectively). The output powers of Class 1 and Class 2 must be in range of [1mW (1dBm), 100mW (20dBm)] and [0.25mW (-6dBm), 2.5mW (4dBm)] respectively. The nominal output power is only defined in Class 2 as 1mW (0dBm). It is mandatory for Class 1 devices, but optional for Class 2 devices to implement power control. Based on Received Signal Strength Indication (RSSI), devices exchange messages (via LMP) to adjust output power.

III. MEASUREMENT STUDY OF BLUETOOTH-BASED CONTENT DISTRIBUTION

In this section, we evaluate the Bluetooth-based content sharing system through measurement. We are mainly interested in measuring *peer discovery/connection setup latency* and *data rate of a mobile user*. Peer discovery latency denotes the time to discover a random node. Connection latency is the time to make a connection to a discovered peer. The efficiency of the connection is measured through the download throughput. These performance metrics may depend on various factors; namely, a class of Bluetooth devices (i.e., Class 1 or 2) and versions (i.e., Bluetooth 1.1, 1.2, or 2.0 EDR), speed of users, distance between users, and WiFi interference. We measure the impacts of these variables on the metrics of interests. In this section, after describing the experimental setup we first show the results of peer discovery/connection latency. We then present the downloading throughput of a mobile user followed by the impact of WiFi interference. Finally, the downloading performance in a real environment is presented.

A. Experiment Setup

Measurement hardware: Evaluating mobile users is challenging since it is nontrivial to control the speed of users. Thus, the key ingredient is the ability to control the speed of mobile users so that we can repeat the experiment multiple times in order to accurately measure the performance. To this end, we use Amigobot, a programmable robot that can travel at the speed up to 2.2m/s. Amigobot can be controlled wirelessly in a remote machine using AmigoWireFree operating at 900Mhz. A mobile user is emulated by an Amigobot carrying a laptop on its top. In the experiment, the speeds of the Amigobot are set to 0.5, 1.0, and 1.5m/s. The laptop used is Dell Latitude D610 with a Pentium M 770 processor (2.0Ghz) and 512MB RAM.

The following Bluetooth dongles are used for experiments. In the case of class 2 devices, we use Belkin F8T003v (CSR chipset, Bluetooth v1.1), Bluetake BT009Si (Silicon Wave, Bluetooth v1.2), and Belkin F8T013V (Broadcom chipset, Bluetooth v2.0 EDR). Since most commercial BCD systems use the class 1 interfaces, we also experiment a class 1 device, Belkin F8T012V (Broadcom chipset, Bluetooth v2.0 EDR) in order to see the potential benefits of its high transmission power for data transfer to class 2 devices. Note that a long range transfer is only feasible between class 1 devices because Bluetooth requires a bidirectional link for handshaking.

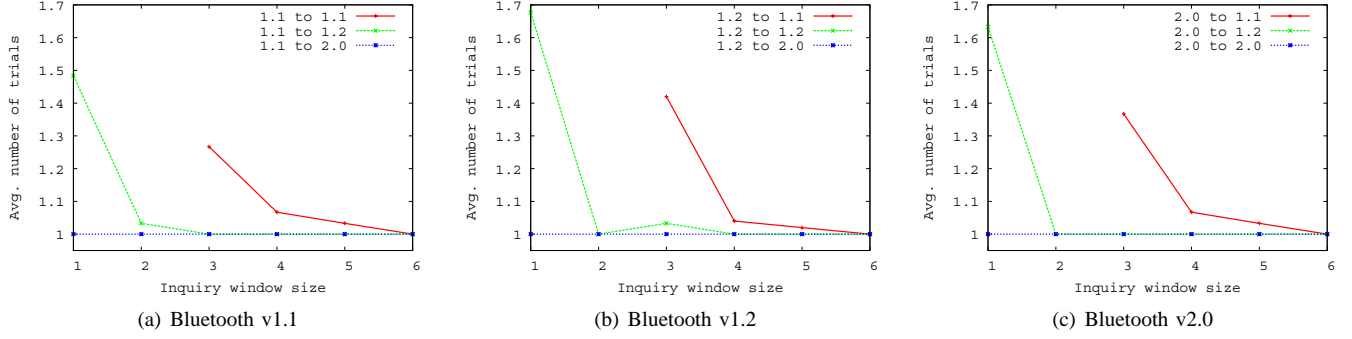


Fig. 3. Peer discovery as a function of inquiry window size with various Bluetooth versions

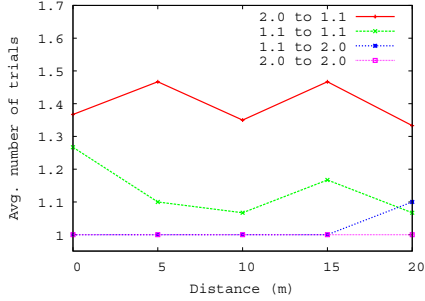


Fig. 4. Avg. number of trials as a function of distance

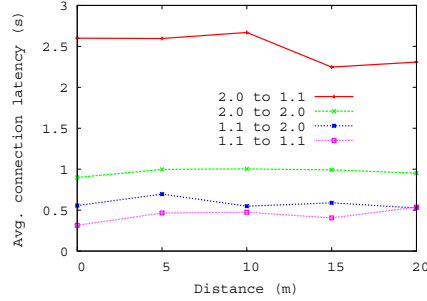


Fig. 5. Average connection latency as a function of distance

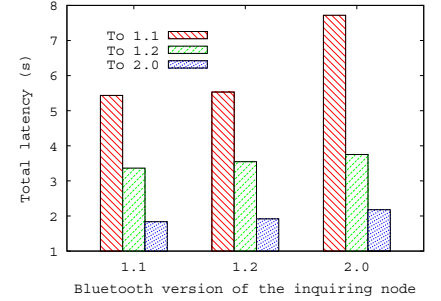


Fig. 6. Total latency: discovery latency + connection setup latency

Measurement Environment: Unless otherwise mentioned, the experiments were carried out in the second level of the underground parking lot to exclude external factors, such as WiFi interference and obstacles (i.e., human). There was no physical interference of human/vehicles because the experiments were performed late at night.

Measurement software: To measure the metrics of interests, we develop a measurement tool using BlueZ v3.7.² BlueZ is one of the most popular Bluetooth host protocol stack implementations and is included in the official Linux kernel. BlueZ implements core protocols (e.g., L2CAP, RFCOMM, etc.) and provides the BSD socket interfaces. The software is used for peer discovery, connection, and data transfer. Peer discovery is carried out using *hci-inquiry* function by the master node. This returns a list of inquiry responses it receives during the period of inquiry, each of which includes the responder's Bluetooth address, page scan repetition mode (PSRM), clock offset (CO), etc.

On one side, the measurement software operates as master node, and on the other side, it operates as slave node. The actual data transfer is realized through L2CAP sockets. L2CAP is a connection-oriented protocol that sends individual datagram of fixed maximum length. The default transport policy is to retransmit until success or total connection failure, thus providing a *reliable* point-to-point connection. L2CAP uses Protocol Service Multiplex (PSM) ports to differentiate multiple applications on the same host. The slave node binds a PSM port and waits for a connection. The master node can initiate a connection to the slave node by calling L2CAP *connect* socket function. An ACL connection is used for data transfer, but this connection initiation routine (i.e., paging) is hidden beneath the L2CAP software layer in BlueZ kernel

module. Note that *connect* requires the timeout value, but this is nothing to do with the actual paging interval. As described in Section II, the paging interval is determined by PSRM value in the inquiry response. Our tested devices all use PSRM mode R2 in which the page scan interval is less than 2.56s. In this case, the specification recommends that N_{page} , the number of paging train repetitions, should be greater than 256, i.e., $> 2.56s$. During the period, if the paging node receives a page response, it immediately returns and starts exchanging additional packets to setup an L2CAP connection. Therefore, given that $N_{page} = 256$ is used, we can assume that the overall connection takes roughly less than 3s. In our experiment, this value is used as a connection timeout value. If the connection times out, the program initiates another connection.

The overall procedure can be summarized as follows. The master node first calls *hci-inquiry* to discover peers. The number of inquiry attempts is logged to measure the discovery latency. Upon finding a node, it tries to make an L2CAP socket connection to the peer. The latency of L2CAP *connect* function is logged to measure the connection latency. As described above, by setting connection timeout as 3s each connection attempt takes less than 3s. The number of connection attempts is also logged. Dummy data with a packet size of 2300B is continuously transmitted until the connection is lost. For every packet transfer, the master logs the transmission power level (*hci-read-transmit-power-level*). The slave node sets on inquiry and page scan, and listens to a specific port in order to accept an L2CAP connection. Once the connection is created, it starts receiving packets and logs link quality (*hci-read-link-quality*) and Receive Signal Strength Indication (RSSI) (*hci-read-rssi*) information. For every 500ms period, it logs the total amount of received data and the data throughput. In addition, the slave node logs HCI event messages using *hcidump*, a

²<http://www.bluez.org>

packet snooping program for Bluetooth. As we will see later, *hcidump* at the receiver side allows us to infer the current packet type for data transfer. This information is managed by Link Management Protocol (LMP) in Bluetooth and is not revealed to the upper layer.

B. Peer Discovery/Connection Setup Latency

Peer discovery and connection latencies are the key to BCD. Given limited contact duration, the above overheads determine the useful time for actual data transfer. In this section, we investigate various parameters impacting the performance of these metrics; namely, inquiry window size, and distance between two peers, Bluetooth versions. First, we show the impact of inquiry window size in the range of [1, 6] with different Bluetooth versions. Note again that the unit of the inquiry window is 1.28s, i.e., window size 1 is equal to 1.28s. If the inquiry fails, then the master node tries again. We measured the average number of such trials to discover a peer. Upon discovery we measured the connection setup latency. The distance between two peers was set to 0m to 20m with a gap of 5m. Bluetooth v1.1, v1.2, and v2.0 EDR were used. For each configuration, we ran experiments 50 times to get the average value.

Figure 3 shows the average number of trials for various Bluetooth versions. Bluetooth v2.0 devices can be successfully discovered with inquiry window size 1, and v1.2 devices take less than 2. This is mainly due to the use of interlaced scan operations introduced as of Bluetooth v1.2. In the interlaced inquiry scan, scanning one inquiry packet train is immediately followed by scanning the other train. The probability of missing an inquiry packet train is thus negligible [24], but it is still possible for the following reason. The specification recommends random back-off before a node returns an inquiry response, mainly to reduce the chances of collision when multiple devices have opened scan windows and an inquiring device begins transmitting inquiry packets. If the scanning interval is larger than 1.28s, the range of [0,1023] is used for back-off; otherwise, the range of [0,127] is used. Since the average back-off interval is one half of the maximum interval, the probability of failure with inquiry window size 1 is given as $P[\text{failure}|T_{w_inq} = 1, T_{max_bf} = 1024] = 512 \times 0.625ms/1.28s = 0.249$ and $P[\text{failure}|T_{w_inq} = 1, T_{max_bf} = 128] = 64 \times 0.625ms/1.28s = 0.03$. From the figures, we see that the implementation may vary by vendors: Bluetooth v1.2 may use $T_{max_bf} = 1024$ as the maximum back-off window size, and Bluetooth v2.0 does not implement any random back-off.

In contrast, Bluetooth v1.1 uses inquiry window size of at least 3. We try many times with the inquiry window size less than 3, but we are not able to succeed in most of the cases. Theoretically speaking, since window size 1 and 2 show about 0.36 and 0.48 of discovery probabilities according to [24], the discovery process can be modeled using geometric distribution assuming each trial is independent. On average, by repeating a trial 3 times with window size 1, we should be able to discover a peer. However, according to our finding this is not true with small window size. In practice, we found

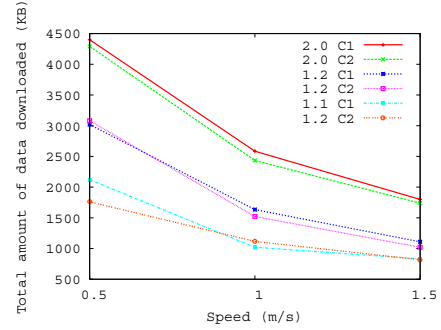


Fig. 7. Total amount of downloaded data while traveling 25m

that for Bluetooth v1.1 we need at least inquiry window size 3. Bluetooth v1.1 scans only a single train, and missing a train incurs 2.56s (i.e., window size 2) of penalty. We suspect that our tested Bluetooth device may always start with the same train, thus requiring at least window size 3. Then, should we repeat with small inquiry window size, say 3 or try with bigger window size? For the sake of analysis, we can assume the independence of each trial if the inquiry window size is greater or equal to 3. Then, from Figure 3, we can calculate the average latency for each Bluetooth version, i.e., the average number of trials is multiplied by the inquiry window size. Let us find the optimal window size to discover each Bluetooth version; i.e., by comparing the average latency of the plots with “* to v1.1” in Figure 3. For instance, for “v1.1 to v1.1” the latency with window size 3 and 4 is 4.83s ($1.26 \times 1.28 \times 3$) and 5.47s ($1.07 \times 1.28 \times 4$) respectively. As a result, we find that the optimal window size to discover Bluetooth v1.1 and v1.2 or higher is given as 3 and 1 respectively.

We then measure the impact of distance on discovery latency. Figure 4 shows the average number of trials as a function of distance. To discover Bluetooth v1.1 and v2.0 devices, we use window size of 3 and 1 respectively. The distance is not a critical factor for device discovery. It confirms the fact that the inquiry procedure per se is robust, because an inquiry packet is repeatedly sent and the size of an inquiry packet is small.

Finally, we present the results of connection latency. Again, connection latency is the time for a node to connect to a discovered peer. Figure 5 shows the results as a function of distance and Bluetooth versions. Surprisingly, the latency between Bluetooth v2.0 devices is twice larger than that between Bluetooth v1.1. The latency for a Bluetooth v2.0 device to connect to a Bluetooth v1.1 device takes 5 times longer than that for the other direction. Similar to inquiry, the connection latency is not sensitive to distance. Figure 6 shows the total latency for discovery and connection. In general, it takes much longer to discover and make a connection to a Bluetooth v1.1 device.

C. Download Throughput of Mobile Users

We measured the downloading throughput of a mobile user as follows. A static node as a master transferred data to a mobile user, initially located at the same place. As described before, the mobile user was emulated using an Amigobot. We programmed the robot to travel up to 25m at the speed of 0.5, 1.0, and 1.5m/s to mimic slow, moderate, and fast walking speeds respectively. The impact of various Bluetooth versions

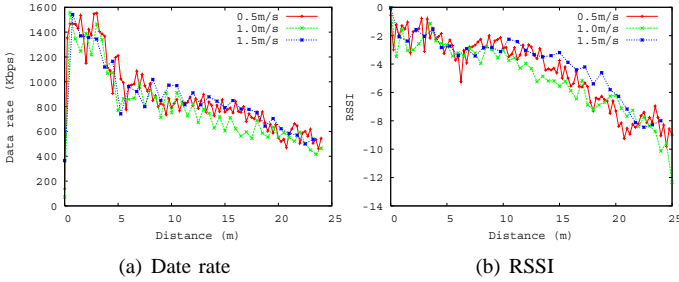


Fig. 8. Data rate and RSSI at a mobile user with Bluetooth v2.0 (C2 sender)

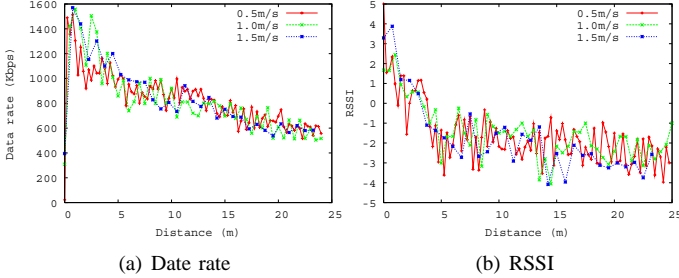


Fig. 9. Data rate and RSSI at a mobile user with Bluetooth v2.0 (C1 sender)

was explored by testing different Bluetooth versions at the mobile user side. We used a class 1 Bluetooth v2.0 device to investigate the benefits of high transmission power at the sender side. We ran each configuration 5 times to calculate the average.

Figure 7 shows the average of the total amount of data received while an Amigobot travels 25 meters. It shows that a user moving at a moderate speed (i.e., 1m/s) can download several mega bytes. As speed increases, the download data size decreases. Note that the decrement is nonlinear since for given fixed distance the traveling time is inversely proportional to speed (i.e., 0.5m/s: 50s, 1m/s: 25s, 1.5m/s: 16.6s). Figure 8 shows the data rate and RSSI as a function of distance with different speeds. The overall trend of data rate is independent of speed and is a function of distance. Bluetooth v2.0 class 1 results in Figure 9 show that high transmission power improves the performance less than 5%. This validates the importance of a *symmetric* link in Bluetooth. However, later we will show that a class 1 device may bring improvement in presence of WiFi interference, at the cost of increased interference with WiFi.

Figure 8 shows that the RSSI value gradually decreases with distance. On the other hand, the data rate sharply drops after passing by around the 5 meter mark. There is another drop around 10-15m marks. It is counter intuitive if we think of this in terms of packet error rate with distance. We find that this is mainly due to Channel Quality Driven Data Rate (CQDDR), the auto rate control protocol in Bluetooth. The specification states that quality measurement of a link at the receiver side can be used to dynamically control the packet type transmitted from the remote device to improve throughput (closely related to FEC and ARQ). However, the incorrect choice of a packet type may result in a significant performance loss as noted in [7]. Therefore, the CQDDR policy plays a key role in determining the throughput, especially in mobile environments. The results of Bluetooth v1.1 and v1.2 in Figure 7 show that different policies may bring considerable throughput difference.

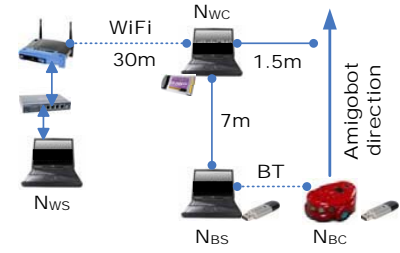


Fig. 10. WiFi interference test setup

Let us further investigate the behavior of CQDDR in our tested Bluetooth devices. CQDDR is implemented in the LMP. The receiver side LMP can ask the sender side LMP to transmit packets using a preferred packet type. Although there is *no* HCI function that can access current packet type being used, the current packet type can be *inferred* at the receiver side as follows. L2CAP layer segments an L2CAP packet by *ACL_MTU* size, which can be read from the Bluetooth device. For example, given 1017B of *ACL_MTU* (the default value of Broadcom chipset), 2300B of an L2CAP packet (+4 bytes for L2CAP header) is segmented into two 1017B and one 270B packets. Each segmented packet is attached with a 4byte ACL header (i.e., 1021B and 274B packets). The list of L2CAP segment packets are then sent to the lower layer. Bluetooth baseband processes those packets to maximize throughput. If the size of an incoming packet is larger than that of the current packet type, a node fragments the incoming packet. For instance, in the above example if the current baseband packet type is 3-DH5, which can load up to 1021B, the segmented packet with size 1021B is sent directly using a 3-DH5 packet. If the current packet type is 2-DH5 (max 367B), the baseband layer segments the payload, i.e., by generating two 367B packets and one 283B packet. The L2CAP layer of the receiver will reassemble the payload. Thus, by reading HCI event messages at the L2CAP layer, we can determine the current packet type. In our experiment, we use *hcidump* to log events. Due to the lack of space, we only present our main findings. First, the tested Bluetooth v2.0 devices start with 3-DH5 and then changes to 2-DH5 and 2-DH3 as link quality degrades. Second, the tested Bluetooth v1.1 devices use DH5 by default and as the distance increases (i.e., after passing on average 10m), it changes the packet type to DM5. Third, the tested Bluetooth v1.2 devices continue to use DH5 regardless of the distance.

D. Impact of WiFi Interference

Both Bluetooth and IEEE 802.11 WiFi operate in the same frequency spectrum, i.e., the 2.4GHz ISM band. Thus, it is expected that the performance of these devices is adversely affected in the presence of one other. In view of this, the Coexistence Task Group 2 (TG2) of IEEE 802.15 has adapted an adaptive frequency hopping (AFH) mechanism. AFH considers the channel condition and changes the hopping frequency dynamically, thus enabling coexistence with other devices in the 2.4GHz ISM band. AFH consists of two steps: channel classification and adaptive control protocol. Channel classification keeps the list of “good” and “bad” channels based on the channel quality. This information is then exchanged through an adaptive control protocol. Given

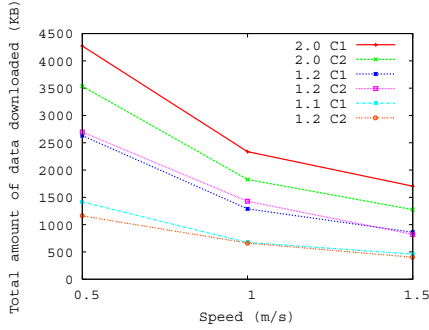


Fig. 11. Total amount of downloaded data while traveling 25m with WiFi interference

this, AFH kernel chooses a set of hop frequencies to use by avoiding as many bad channels as possible. Note that if the number of “good” channels are less than 15 (i.e., the minimum number of channels that FCC requires Bluetooth to hop over), some bad channels would still be used.

We evaluate the throughput of mobile Bluetooth users in presence of WiFi interference. The system configuration is presented in Figure 10. We use a Linksys Wireless-G router which is configured with channel 8. Node N_{WC} with Orinoco Gold 802.11b PCMCIA card is configured as a WiFi client located 30 meters apart from the router. Both Wireless Router and N_{WS} are directly attached to the hub. WiFi interference is induced using *iperf*, a bandwidth measuring tool³, by generating 4Mbps of UDP traffic from N_{WC} to N_{WS} . For each experiment, the bandwidth of N_{WS} is logged to show the impact of Bluetooth on WiFi. Node N_{BS} is located 7 meters away from the N_{WC} and sends dummy data using a Bluetooth connection to the mobile node N_{BC} .

Figure 11 shows the results of total amount of downloaded data. Compared to Figure 7, WiFi interference incurs throughput loss up to 30%. AFH is supposed to effectively circumvent this situation in both Bluetooth v1.2 and v2.0. Since the loss is prevalent in both cases, we conclude that AFH does not work well in a mobile environment. We suspect that this is due to a response time delay between the presence of interference and the realization of its existence. Although the choice of channel estimation method is vendor specific, it is generally believed that some common methods like packet error rate (PER) or bit error rate (BER) are used. However, these methods generally are on a channel-by-channel basis, and thus require a longer response time. The AFH channel map can be read by calling *hci-read-afh-map*. It returns 79 1-bit fields that represent the state of the hop sequence specified by the most recent AFH message exchange by LMP. If channel n is used, the corresponding bit is 1; otherwise it is 0. Our tested devices used all available channels even with WiFi interference. Thus, AFH is not effective in mobile environments.

From the figure, we also see that unlike Figure 7, a class 1 device brings more than 20% of throughput gain for Bluetooth v2.0 whereas other versions have mere improvements. We plot the rate over distance for Bluetooth v2.0 in Figure 12. It shows that the impact of WiFi on class 2 devices is visible enough to observe a drastic throughput decrement nearby N_{WC} . On the other hand, class 1 is fairly resilient to WiFi interference.

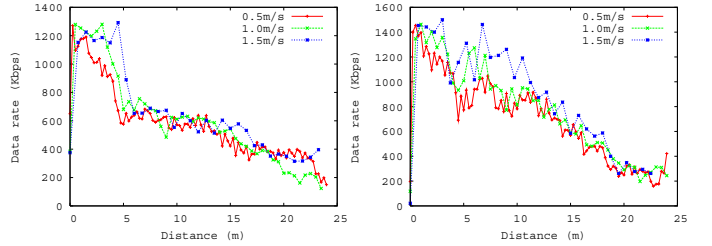


Fig. 12. Bluetooth data rate over distance with WiFi interference.

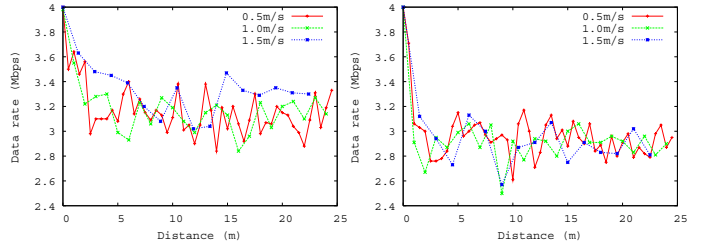


Fig. 13. WiFi throughput over distance with Bluetooth interference

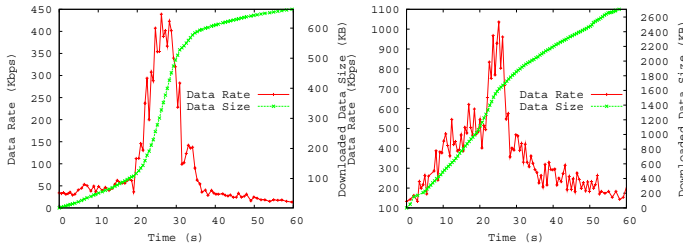
Figure 13 shows the results of WiFi throughput over distance with Bluetooth interference. With a class 2 sender, the throughput drops to around 3.2Mbps. On the other hand, with a class 1 sender, it drops to around 2.9Mbps (additional 10% drop). Figure 13(b) shows that the data rate drops at the very beginning due to its high transmission power unlike Figure 13(a). The high transmission power has a longer interference range. Thus, it is detrimental to the performance of WiFi users.

Let us briefly note that LMP also uses a power control scheme: if the received signal strength differs too much from the preferred value, then it may request an increase or a decrease of the TX power. Most products implement power control to save energy although it is not mandatory to implement the function. The current transmission power can be read by *hci-read-transmit-power-level*. Interestingly, we were not able to observe any power control working during the experiments. We examined tested devices to see whether power control is working or not: we located two laptops close by and initiated data transfer. After passing more than 30 seconds the transmission power level gradually decreased. Note that the RSSI values were quite stable. We conclude that in a *mobile* environment, it is less likely that a Bluetooth device performs power control.

E. Real-world Measurement Results

So far we have shown the results in the controlled environment, i.e., an underground parking lot. In reality, the proximity-based marketing targets for an area or streets with many people carrying their mobile devices. Thus, the performance is affected not only by the obstacles (e.g., human, walls, etc.), but also by the ubiquitous existence of WiFi interference as observed in [4]. As a pilot study, we measured the downloading bandwidth in a real environment, i.e., the ABC student union. The student union has a 70m long corridor that leads to all different kinds of restaurants. We carried out the experiment at noon, the busiest period of a day. A master node was located in the middle of a 60 meter long corridor segment. We marked the corridor for every 5 meters. A mobile user

³<http://dast.nlanr.net/Projects/Iperf>



(a) Bluetooth v1.1 to v1.1 (b) Bluetooth v2.0 to v2.0
Fig. 14. Downloading data rate and data size distribution

(human) carried the laptop and moved at the speed of 1m/s approximately. A mobile user controlled the speed with a stop watch by checking the marks on the corridor. We measured the downloading bandwidth over among Bluetooth v1.1 devices and among v2.0 devices. The experimented repeated 5 times and the overall results were fairly consistent with different runs. The average data rate and the cumulative distribution of downloaded data size are shown in Figure 14.

In the previous experiment, while traveling 25m without any interference, a mobile user can download 1.1MB and 2.4MB for Bluetooth v1.1 and v2.0 respectively. Since the user in this experiment traveled more than twice longer in distance and the AP was located in the middle, it is expected that the download data size is expected to be at least 2.2MB and 4.8MB for Bluetooth v1.1 and v2.0 respectively. Download data size for Bluetooth v1.1 and v2.0 is approximately 650KB (29%) and 2700KB (56%) respectively. The figure shows a spike in the middle (30s mark is the point where the master node is located) and thus, faster data transfer is only possible when two devices are physically close, i.e., within 10 meter range. Since downloading between Bluetooth v1.1 devices shows a steeper spike than that between Bluetooth v2.0, it shows a larger throughput drop than Bluetooth v2.0. In fact, this is because of CQDDR of Bluetooth v1.1. The tested Bluetooth v1.1 device uses DM packets in presence of interference. It uses DM1 till 20s with occasional use of DM3 packets. As the mobile user is getting closer to the sender, the link quality increases (based on the logged link quality and RSSI information), and thus, we observe that the packet type has changed to DM5. As the distance is getting larger, it then switches back to DM3 and finally to DM1. In the case of Bluetooth v2.0, it starts off with 3-DH3, to 2-DH5, and to 2-DH3. The overall change happens in less than 5 seconds. It seems that after detecting EDR functionality, LMP starts off using 3-DH3 and then switches to smaller size packet types based on the channel quality. Once the tested Bluetooth v2.0 device starts using 2-DH3, it continues to use it for the rest of connection. It may be the case the tested Bluetooth v2.0 devices only uses 2-DH3 packets if the link quality is below a certain threshold. Also, this behavior persists even in the case the sender is located nearby. It is possible that downgrading the packet type happens quickly, but upgrading may take time. Since the link quality changes dynamically, it may not be stable enough to upgrade the packet type.

F. Summary

The experimental results can be summarized as follows.

- Inquiry and connection latencies vary widely among different versions and chipmakers. Distance is not critical for peer discovery and connection latencies. The optimal inquiry window sizes for v1.1 and v1.2 or above are given as 3 and 1 respectively.
- The data throughput of a mobile user is mainly determined by CQDDR. Transferring data with high power (using a class 1 device) results in mere improvements (less than 5% most of the cases). For an efficient connection management, the data rate/RSSI dynamics can be used to estimate the distance between peers.
- Both WiFi and Bluetooth affect the throughput of each other. Bluetooth v2.0 EDR is more vulnerable to WiFi interference (35% throughput loss for some case) than the other versions. Bluetooth class 1 improves the throughput, but it adversely affects the WiFi throughput (additional 10% drop). A long response time makes power control/AFH not work well in our tested devices, especially in a *mobile* environment.
- In the real environment with obstacles and WiFi interference, the average data rate is peaked when a mobile node is within 10m range. We observe a significant throughput loss, i.e., 71% and 44% for Bluetooth v1.1 and v2.0 respectively, showing that Bluetooth is extremely vulnerable to ambient interference. The performance mainly depends on the behavior of CQDDR and AFH.

IV. PERFORMANCE ENHANCEMENT FEATURES

We have characterized the system by extensive evaluation of discovery/connection latency and download throughput, thus showing the feasibility of Bluetooth-based P2P networking. In addition, we have shown that given a dynamic nature urban environment (i.e., due to mobility, obstacles, WiFi interference, etc.), a mobile user may experience a considerable throughput drop. Given this, improving the performance (i.e., by increasing the data throughput, yet efficiently managing power consumption of a mobile user) will be a holy grail of a BCD system. To this end, we review the key aspects of the Bluetooth-based content distribution, namely channel utilization, resource discovery, and resource availability. First, we try to maximize the throughput by increasing the utilization of the physical channel. Second, we use coding techniques to increase the “usefulness” of a contact. Third, we propose an energy efficient peer discovery protocol, namely the adaptive inquiry mode. Finally, we discuss other simple techniques that can enhance the performance.

A. Receiver Feedback for L2CAP Packet Size Selection

Figure 15 shows the overall flow of the packet transmission in Bluetooth. The application data is sent to the L2CAP layer and loaded into an L2CAP packet. The size of an L2CAP packet header is 4 bytes and it contains the following fields: length and channel ID. The channel ID is used to identify a logical channel endpoint of a device. The L2CAP packet is then loaded into an HCI ACL packet. If it is larger than *ACL MTU*, it is fragmented into multiple HCI ACL packets. There is a 4-byte HCI ACL packet header, containing a connection

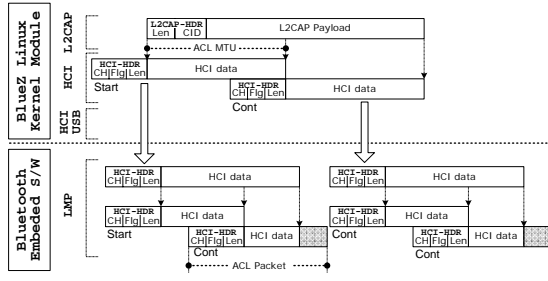


Fig. 15. Bluetooth stack overview: An L2CAP packet is fragmented into multiple HCI ACL packets. These packets are then transferred to the Bluetooth device via HCI-USB. The packet is sent without further fragmentation if the packet size is less than or equal to the size of the current HCI ACL packet; otherwise it is fragmented into multiple packets.

handle (CH), flag, and length. The CH is used to identify a connection between two Bluetooth devices. The flag is used to denote start/continuation of L2CAP packet fragmentation.

The main problem is that Bluetooth does not reveal any information of the current ACL packet type (i.e., any information of CQDDR in LMP). If the ACL MTU size matches with the size of the current ACL packet type, an L2CAP packet will be transmitted without any further fragmentation. Otherwise, it will be further fragmented into multiple ACL packets as shown in Figure 15. Let us take a look at the following example. An application sends a series of 1013B packets. The L2CAP packet size is 1017B (1013B + 4B L2CAP header). Assuming that ACL MTU is 1017B, there is no L2CAP level fragmentation. Given that LMP starts with the packet type of 3-DH5 (max 1021B), an L2CAP packet is fully loaded into a 3-DH5 packet (1017B+4B ACL header). Let us now assume that the packet type has changed to 2-DH5 (max 679B). A 1017B L2CAP packet cannot be fitted into a 2-DH5 packet. At the baseband layer, the packet is fragmented and loaded into two 2-DH5 packets, i.e., 679B+338B. Since the second packet cannot be fully utilized, this results in 24% throughput loss, i.e., 341B out of 1358B ($=2 \times 679B$). In Section III, we show that the receiver can know the sender’s packet type by reading HCI event messages at the L2CAP layer just as *hcidump*. As distance increases, the packet type gradually changes to smaller size due to CQDDR (e.g., 3-DH5 to 2-DH5). Thus, we propose that the receiver notifies the packet type change to the sender so that it can adjust the L2CAP payload size. The feedback overhead is minimal because the response packet is small.

To realize this we modify the BlueZ stack in Linux Kernel v2.6.15.1. During the L2CAP connection setup (*l2cap_do_connect*), *mtu* field of the L2CAP connection data structure (*l2cap_conn*) is initialized with *ACL_MTU*. This value is used to fragment the L2CAP packet in *l2cap_do_send()* (defined in *l2cap.c*). Thus, whenever receiving a feedback a node updates *mtu* field (via a newly defined socket option). In Figure 16, we compare the performance with the receiver feedback (Bluetooth v2.0 with class 2 moving at the speed of 0.5m/s). It shows that the receiver feedback improves the overall throughput more than 20%. Note that one can further maximize the channel utilization by feeding a large L2CAP packet (max 64KB), consequently reducing the frequency of *send* system call.

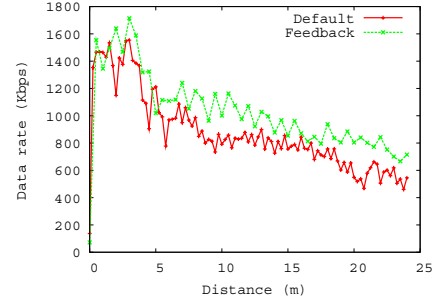


Fig. 16. Performance comparison with receiver feedback

B. Coding Techniques to Increase Resource Availability

Coding techniques such as Rateless and Network codes are assumed to increase the availability of resources in the network [20], [8], [11]. This is also true in our contact-based opportunistic content sharing with churning. These techniques mitigate the problem of the truncated downloads. In the network, it is likely that there are more users with partial downloads than ones with a complete file. Upon contacting these users the partial downloads tend to have overlapping information (known as a coupon collection problem). We describe the coding techniques as follows:

Rateless Codes: In rateless codes (e.g., Online Code, Raptor Code, etc.), a file with n blocks is used to generate an infinite series of encoded blocks at the APs [20]. Each block is attached with its ID and is disseminated through the lossy “contact”-based Bluetooth network. A user upon collection $(1 + \epsilon)n$ blocks can decode the blocks with high probability where ϵ is a system parameter.

Network Code: Network coding involves a coding operation in any intermediate nodes in the network. At the AP, the infinite series of encoded blocks are generated through random linear combination of the original blocks; i.e., $\mathbf{c} = \sum_{k=1}^n e_k \mathbf{p}_k$ where e_k is drawn randomly over a finite field \mathbb{F} and \mathbf{p}_k denotes the k -th original block. The corresponding code vector is the set of coefficients used for coding, i.e., $\mathbf{e} = (e_1, \dots, e_n)^T$. The AP distributes the coded block with its encoding vector, i.e., $\hat{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{e} \end{bmatrix}$. Similarly, an intermediate node with m coded blocks (denoted \mathbf{c}'_k) in its local memory can generate a re-encoded block on-the-fly, i.e., $\sum_{k=1}^m e_k \mathbf{c}'_k$ where e_k is drawn randomly over a finite field \mathbb{F} . The code vector of the re-encoded block is used to find a “helpful” node that has at least one linearly independent coded block. If helpful, this re-encoded block is transferred. After collecting n linearly independent coded blocks, a node can decode the blocks. Readers can find the details in [8], [11].

C. Energy Efficient Peer Discovery

Power consumption in a mobile device is a critical issue, and it is preferable to design an energy efficient protocol, yet preserving the performance. [21] shows the statistics of power consumption per unit time in each Bluetooth operation, i.e., $C_{Idle} = 20\text{mA}$, $C_{Inquiry} = 38\text{mA}$, $C_{Scan} = 49\text{mA}$, $C_{Data} = 35\text{mA}$. Note that although the scan operation is more expensive than an inquiry operation, the inquiry has to be repeated for a longer period of time, and thus, the total amount of consumed energy is much higher than scan.

Bluetooth-based content sharing requires nodes to periodically alternate their role as master and slave so that they can discover each other (i.e., P2P mode). Let us see how much energy a node spends in an hour to perform the P2P mode. For the sake of analysis, let us assume that the time spent for inquiry is the same as that for inquiry scan (i.e., 30 minutes). Thus, it spends $38mA \times 30m \times \frac{1h}{60m} = 19mAh$ for the inquiry and $[30m \times \frac{60s}{1m} \times \frac{1}{1.28s}] \times 49mA \times 11.25ms \times 10^{-3}s + [30m - [30m \times \frac{60s}{1m} \times \frac{1}{1.28s}]] \times 11.25ms \times 10^{-3}s] \times 20mA = 36458.7mAs \approx 10.13mAh$ for the inquiry scan. During the idle period of inquiry scan (99.2%), most devices can change their state to the standby state (which consumes 2mAs) [5]. Therefore, the actual energy consumption of the inquiry scan is approximately 1.21mAh – that is why most devices (e.g., cell phones) typically stay in the inquiry scan mode. The cost of inquiry is significant, considering the fact that recent cell phones such as LG chocolate and Motorola MOTOKRZR K1m are equipped with less than 900mAh battery. Thus, we propose a simple solution to reduce the frequency of inquiry.

To this end, we use the concept of sociological orbits [10]. Sociological orbits are probabilistic mobility models where nodes move between a set of hubs, e.g., subway stations or bus stops. In Bluetooth-based content distribution, such hubs become information exchange bazaar [22] such that people with the shared interests cooperatively carry and forward the content. Since it is less likely that information exchange happens in transit to hubs, we propose a *adaptive inquiry mode*: a node continues to stay in the *inquiry scan* state unless some other nodes in the *inquiry* mode wake it up by creating an actual connection. To be precise, in the very beginning nobody is in the inquiry mode. Upon passing by an AP, the node will be activated. Any node in the inquiry mode can then wake up others. If a node fails to create a connection or to find any nodes over a certain threshold, it then goes into the inquiry scan state to save energy. Instead of switching back to the scan state abruptly, the inquiry interval can be dynamically adjusted using the contact frequency (or recent activity) as proposed in [9].

D. Discussion

We discuss several other simple methods that can achieve better performance, namely fast resource discovery and distance-aware connection management. For resource discovery, the Bluetooth stack includes a service discovery protocol (SDP), proposed by the Bluetooth Special Interest Group (SIG). SDP provides a simple discovery mechanism based on requesting service classes or service attributes (i.e., file ID) successively from all devices in range. However, the overall procedure is time consuming. Since Bluetooth does not support broadcast, one has to connect to each node. In Bluetooth, the only broadcast message is the inquiry packet. Sedov et al. [26] proposed that fast resource discovery can be done via using the Class of Device (CoD) field of an inquiry response. CoD field is composed of 22bit class information with 2bit format type. Since each file is uniquely identified by its hash value, 22bits of the hash value are put into the CoD field. In Feb. 2007 the Bluetooth v2.1 standard draft was released. The

major enhancement is the Extended Inquiry Response (EIR) mode. The potential slave node can immediately send an EIR packet (max 240B) after returning the inquiry response. Thus, the EIR packet can be used for fast resource discovery as well. Next is the distance-ware connection management. In Section III, we notice that RSSI and data rate can be a good measure of distance estimation. Our results also show that owing to CQDDR nodes experience a sudden throughput drop after they are apart more than a certain threshold distance. In this case, assuming that there are enough neighbors one may want to disconnect the current connection and find others. However, it is nontrivial to correctly estimate the mobility of users without localization. One good solution would be periodically running *inquiry-with-rssi* and cooperatively sharing this information in order to roughly coordinate neighbors [18], [29]. Developing an efficient heuristic is part of our future work.

V. SIMULATION

In this section, we simulate a Bluetooth based content sharing application, namely BlueTorrent [12] to evaluate the proposed features: *i*) data encoding (options: no encoding; network coding; end-to-end rateless coding, and; *ii*) normal and adaptive inquiry mode. Since coding strategy and inquiry mode are “orthogonal” options, we will test both options jointly in the same experiment. Finally, we show the performance of the system in presence of different Bluetooth versions.

A. Simulation Setup

We implemented BlueTorrent algorithms in the UCBT ns-2 based Bluetooth simulator, a publicly available open source Bluetooth simulator.⁴ UCBT implements the majority of Bluetooth protocols, such as baseband, LMP, L2CAP, and BNEP.

Mobility: We assume Bluetooth devices (AP’s and mobiles) are placed in a long corridor with fixed boundary (for example, an airport alley, or a shopping mall). Users are moving with waypoint motion from East to West or from West to East. Waypoints are randomly chosen in the initial stage. To mimic realistic pedestrian motion, speeds are selected based on the *normal* distribution with the mean speed (0.8, 1.0, 1.2, or 1.4m/s) and the variance (*mean_speed*/3). Mean speeds are selected based on the mobility model in [23]. To add a random factor, direction is changed periodically with an offset in the range [-10, 10] degrees with respect to the original direction. When a node reaches the North or South boundary, it is reflected back in to the working area. When a node reaches East or West bound, it moves out of the area. A new node is then generated at the other boundary with same speed and direction as the eliminated node.

Metrics: We measure the progress using download percentage of all the nodes that have passed the simulated area during the simulation. By dividing the summation of the downloaded blocks by the nodes, we can calculate the expected number of downloaded blocks. As time tends to infinity, by the strong law of large number, this is equal to the ensemble average of the number of blocks that a random node can download while passing by the simulated region.

⁴<https://www.eecs.uc.edu/cdmc/ucbt>

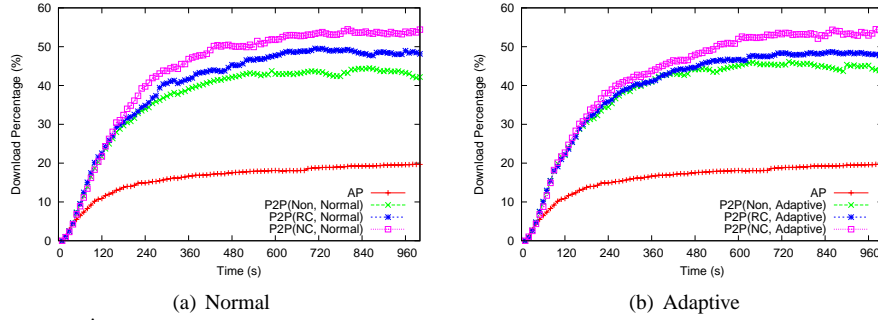


Fig. 17. Download percentage vs. time

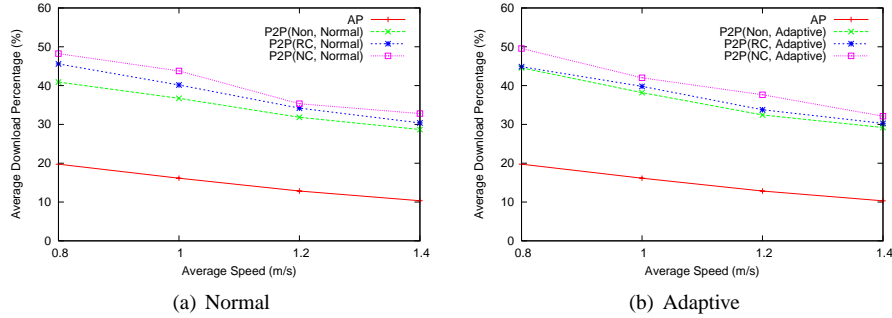


Fig. 18. Download percentage vs. speed

Inquiry methods: Every Bluetooth node has two stages: Inquiry and Connection stages. In the inquiry stage, nodes perform peer discovery by alternating inquiry and inquiry scan based on the selected inquiry mode. The Connection stage begins with after a connection is established. If there is no useful data to transfer in both directions, the connection is terminated. It is maintained until all useful data are transferred in both directions. After this, they enter the Inquiry stage.

- *Normal Inquiry Mode* Every Bluetooth node does inquiry and inquiry scan alternately. This alternately changing inquiry and inquiry scan during inquiry stage maintained throughout simulation time.
- *Adaptive Inquiry Mode* All nodes except Access Point (AP) initially do inquiry scan only, to save energy. A node reactively starts peer discovery after the first connection. If there is no incoming connection for 10s, it switches back to the inquiry scan only mode.

Data transfer methods: There are two types of nodes in our system: APs and mobile nodes. APs have the complete file. They are static and are randomly distributed in the area. Mobile nodes initially have no file. They can receive data blocks from APs or other mobile peers. They are also randomly distributed in the moving area and they move based on selected speed.

- *Access Point (AP) only Data Transfer* This option corresponds to downloading from AP only (ie, no P2P content sharing via Bluetooth). The AP option is tested as a reference, to show the relative improvement introduced by P2P sharing.
- *Non-coded Data Transfer* After the connection is made between two nodes, block bitmaps are exchanged to show availability of blocks in each node [12]. Each node thus can make a “helpful” data list, i.e., a list of blocks that can help the peer. If the helpful data list is non empty, one block is randomly selected from the list and transferred

to the peer.

- *Network Coding Data Transfer* For AP-to-Peer connections, an AP distributed coded blocks by encoding blocks via network coding. Similarly, for P2P connections each node generates a coded block by linearly combining all the blocks. Note that the code vector is first transferred to the other party to check the helpfulness; thus, actual data transfer happens only if it is helpful. The 2^8 Galois field is used for network coding.
- *Rateless Coding Data Transfer* In an AP, data is encoded using Online Code [20], cached in background and then transmitted. Each encoded block has its unique ID. After a connection with the peer is made, the peers exchange data block ID lists (instead of data block bitmaps as in normal data transfer). The data block list contains all block IDs that a certain node has. After exchanging data block list, each node constructs the helpful data list for the peer. If the list is non empty, one block is randomly selected and transferred to the peer.

Parameter Summary: The area is a corridor with 100m length and 5m width where we deploy 50 nodes. We set the number of APs to 1 to keep the system simple. Mean speeds are selected between 0.8 and 1.4 m/s to simulate pedestrian mobility. Unless otherwise mentioned, we use the mobility of 1m/s and the DH5 packet by default. The AP distribute 4.8MB file. The size of a block is 24KB (total 200 blocks). Rateless and network codes use the same block size for coding. Note that due to the space limitation, we do not present the impact of the block size, but the overall performance is consistent with the default configuration.

B. Simulation Results

Download Progress: Figure 17 shows average download percentage at 10 second intervals. For Figure 17(a), normal inquiry mode is used and for Figure 17(b), adaptive inquiry

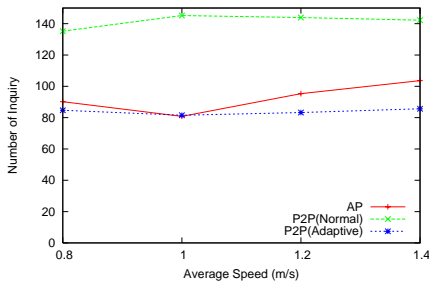


Fig. 19. Number of inquiry vs. speed

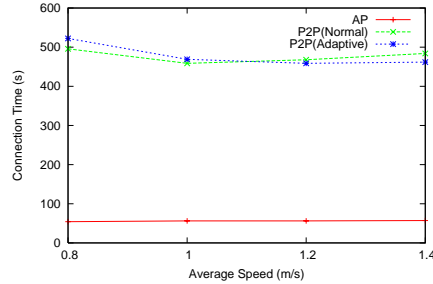


Fig. 20. Connection time vs. speed

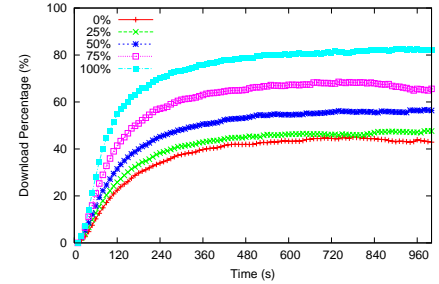


Fig. 21. Download percentage vs. the fraction of Bluetooth v2.0 nodes

mode is used. In this experiment, as mentioned earlier, we evaluate both inquiry modes and coding strategies. Starting with normal vs. adaptive inquiries, network coding and rateless coding data transfer, normal and adaptive inquiry mode shows almost same performance. But, non-coded data transfer with adaptive inquiry mode shows better performance than that with normal inquiry mode. When adaptive inquiry mode is used, an AP node transfers data and also wakes up mobile nodes. Other nodes in the adaptive inquiry mode do the same. In this case, those inquiring nodes always have data and therefore, this decreases unhelpful connections. As a result, overhead for data transfer is reduced; thus, download percentage increases. For coded data transfers, their data blending feature already helps to increase download percentage, therefore it makes the adaptive inquiry mode less effective.

Moving to the data transfer options, P2P transfers have consistently twice the download percentage as AP transfer. As for the P2P coding options, rateless coding and network coding methods show better performance than the non-coding method after the download percentage reaches approximately 30%. These coding methods reduce overlapping of data blocks between two connected nodes; thus, they increase download percentage. Rateless coding encodes data only in AP or at nodes that are acting as AP's after recovering the entire file; whereas network coding encodes blocks in all nodes after receiving data blocks from others and therefore data are more well blended. This *blending* feature of network coding increases the download percentage much more than rateless coding.

Figure 18 shows average download percentage during 1000 seconds. For Figure 18(a), the normal inquiry mode is used, and for Figure 18(b), the adaptive inquiry mode is used. As speed increases, download percentage decreases for all cases. In fact, as speed increases, nodes move out of communication range faster and therefore packet drops happen more frequently. Also, lifetime of nodes decreases. This results in more frequent regenerations of nodes. Note that when a node is regenerated, all received data are deleted. This decreases average download percentage.

All P2P data transfers show twice download percentage than AP data transfer. Among P2P data transfers, download percentages can be ranked in the following order; network coding, rateless coding, and non-coding because of overlapping of blocks and data blending feature that mentioned previous section. Adaptive inquiry mode has almost same download percentage as normal inquiry mode.

Number of Inquiry and Connection Time: Figure 19 shows the number of inquiries used during 1000 seconds. In AP mode, only AP performs inquiry, so the number of inquiries is calculated from AP node. In P2P mode, this number is calculated by averaging total number of inquiries from all nodes. The adaptive inquiry mode has 60% of inquiries compared to normal inquiry mode. Inquiry consumes more energy than inquiry scan. So, by using adaptive inquiry mode, nodes can save energy. Figure 20 shows connection time during 1000 seconds. Normal and adaptive inquiry modes show almost same connection times. By using the adaptive inquiry mode, nodes can save energy, while having same discovery efficiency and data download percentage. The AP mode shows much lower connection time than all P2P modes because there is up to one connection possible among 20 nodes.

Presence of Different Bluetooth Versions: We show the impact of different Bluetooth versions in presence. In the simulation, we use Bluetooth v2.0 users as well as Bluetooth v1.2 users. Bluetooth v2.0 uses the 3-DH5 packet by default and Bluetooth v1.2 uses the DH5 packet by default. We increase the fraction of Bluetooth v2.0 users with a gap of 25%. Figure 21 shows the results in presence of different Bluetooth versions. The graph clearly shows that the presence of different Bluetooth versions has a critical impact on the system performance.

VI. RELATED WORK

Bluetooth-based content distribution (BCD) uses Bluetooth Access Points (BT-AP) to distribute content to the mobile users. It can be classified based on whether users are cooperative or not. Non-cooperative BCD is based on a client server model; BT-APs are the servers, and clients download files only from the APs. BlueCasting⁵ and BlueBlitz Magic Beamer⁶ are a widely accepted proximity marketing system such that they can identify Bluetooth users and deliver tailored messages. LeBrun et al. [14] proposed BlueSpot, a public-transit based content distribution system accessible such that riders can opportunistically access data during their travel time via Bluetooth. In contrast, cooperative BCD is based on a P2P model such that users can share the downloaded files from APs. Farkas et al. [17] proposed a push-based P2P content distribution model: A file is divided into blocks, and blocks

⁵<http://www.bluecasting.com>

⁶<http://www.blueblitz.com>

are multicast through Bluetooth piconets. Given that nodes are synchronized and static, a ring overlay is formed to schedule parallel piconets to efficiently distribute blocks. Unfortunately, the scheme does not consider network dynamics such as mobility, churning, lack of synchrony, and intermittent connectivity. Jung et al. [12] made the first step towards *realistic* content distribution among “mobile” Bluetooth users. P2P peer discovery in Bluetooth is optimized to best utilize the short contact duration among mobile users. Since the Bluetooth sustains a low data rate and the channel is error-prone mobile users cannot download a full file from the access point; thus, BlueTorrent uses a *delay-tolerant cooperative file swarming*. In the Huggle Project⁷, Leguay et al. [15] proposed various content distribution strategies to distribute a *small size* file to a group of users in a large scale urban network. Users can cooperatively relay a file to other interested users, and a set of mobile bridges (regardless of interests) can be used to further expedite delivery. Note that BlueTorrent can be equipped with this “intelligent” file dissemination. However, none of the above works attempt to consider the actual characteristic of P2P Bluetooth communications in realistic environments (with mobility, heterogeneous Bluetooth versions, WiFi interference, etc.), although it has a significant impact on the BCD system performance. In this paper, we first evaluate Bluetooth communications in such environments. The evaluation results are then used to examine various aspects of a BCD system, e.g., peer discovery, resource discovery, and resource availability. Based on this, we propose techniques for efficient file swarming.

In spite of its popularity, the performance measurements of Bluetooth devices have not been thoroughly explored. Kasten et al. [13] evaluated their customized Bluetooth-based sensor device as a part of the Smart-its project and reported the issues of the power consumption and the inquiry latency. Similarly, Siegemund et al. [29] measured the performance the inquiry procedure and then proposed a cooperative peer discover protocol. The measurement study by Leopold [16] is close to our work. The inquiry procedure and the throughput were evaluated using Bluetooth v1.1 devices. The author found that the inquiry is not sensitive to distance and the latency distribution is centered on the train length. The measured throughput varied widely over distance and was much lower than their simulation results, which the author was not able to explain. Our work differs in that *i*) we explore the interoperability issues of various Bluetooth versions (v1.1, v1.2, and v2.0 EDR), *ii*) we measure the throughput of a “mobile” user, *iii*) we find the reason behind the throughput variation over distance, *iv*) we explore the impact of interferences (e.g., WiFi/Obstacles). Note that our inquiry results are consistent with [16] over all Bluetooth versions. Beaufour et al. [2] showed that the connection latency follows a long-tail distribution, i.e., quite a few take longer than 3 seconds. In contrast, we find that the connection rarely takes more than 3 seconds.

WiFi and Bluetooth coexistence issues are well documented in [27], [25], [28]. Shoemake [27] performed coexistence testing and showed that interference results in considerable throughput loss. Punnoose et al. [25] identified that effective

bandwidth degrades more rapidly than the packet loss rate owing to deferred transmissions caused by carrier sensing in WiFi. In contrast, Bluetooth performance starts to degrade rapidly when the interfering WiFi signal is comparable to the desired signal level since Bluetooth does not use “carrier sensing.” Shuaib et al. [28] evaluated the coexistence of 802.11g with Bluetooth. They showed that the shorter the distance between Bluetooth node and WiFi node, the less is the impact on the throughput, mainly due to the power control in Bluetooth. Mander et al. [19] evaluated the adaptive frequency hopping (AFH). They showed that the AFH improves the throughput more than 30%. Unlike [16], they claimed that average throughput does not vary significantly with distance. In this paper, we evaluate the impact of WiFi interference on “mobile” Bluetooth users. Contrary to the previous results [28], [19], we find that in the mobile environment *i*) power control/AFH may not work well potentially due to dynamics of RSSI, and *ii*) the average throughput is dependent on the distance due to CQDDR.

VII. CONCLUSION

In this paper, we studied the feasibility of Bluetooth based P2P content distribution. We first performed extensive measurements to better understand the Bluetooth communications in an environment with mobility, heterogeneous Bluetooth versions/chipsets and WiFi interference, etc. We found that the data rate varies widely with distance, and the overall behavior is mainly dependent on the Bluetooth version/chipset. Our field test results showed that Bluetooth experiences a significant throughput loss (e.g., 71% for Bluetooth v1.1). To compensate this, we then introduced performance enhancement features, namely the receiver feedback for packet size selection, adaptive inquiry mode, and coding-based file swarming. We validated the schemes via experiments and simulations. We found that *i*) the receiver feedback achieved 20% throughput increment; *ii*) the adaptive inquiry mode significantly reduced the frequency of inquiry without any performance loss; and *iii*) Coding-based file swarming improved the system performance at most 15%.

REFERENCES

- [1] BBC NEWS: Music trial taps into Bluetooth. <http://news.bbc.co.uk/1/hi/technology/4392534.stm>.
- [2] A. Beaufour, M. Leopold, and P. Bonne. Smart-Tag Based Data Dissemination. In *WSNA'02*, Atlanta, Georgia, Sept. 2002.
- [3] Bluetooth SIG. Bluetooth Specification v2.0, 2004.
- [4] V. Bychkovsky, B. Hull, A. K. Miu, H. Balakrishnan, and S. Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *MobiCom'06*, Los Angeles, CA, September 2006.
- [5] J.-C. Cano, J.-M. Cano, E. Gonzalez, C. Calafate, and P. Manzoni. Power Characterization of a Bluetooth-based Wireless Node for Ubiquitous Computing. In *ICWMC'06*, Bucharest, Romania, July 2006.
- [6] CBS Goes Bluetooth To Promote Fall TV Line-Up. <http://www.telecomweb.com/tnd/18847.html>.
- [7] L.-J. Chen, R. Kapoor, M. Y. Sanadidi, and M. Gerla. Enhancing Bluetooth TCP Throughput via Link Layer Packet Adaptation. In *ICC'02*, Anchorage, AK, May 2002.
- [8] P. Chou, Y. Wu, and K. Jain. Practical Network Coding. In *Allerton'03*, Allerton, Oct. 2003.
- [9] C. Drulă, C. Amza, F. Rousseau, , and A. Duda. Adaptive Energy Conserving Algorithms for Neighbor Discovery in Opportunistic Bluetooth Networks. *IEEE JSAC*, 25(1):96–107, Jan. 2007.

⁷<http://www.huggleproject.org>

- [10] J. Ghosh, S. Yoon, H. Q. Ngo, and C. Qiao. Sociological Orbit for Efficient Routing in Intermittently Connected Mobile Ad Hoc Networks. In *Technical Report TR-2005-19*, University at Buffalo, Apr. 2005.
- [11] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *INFOCOM'05*, Miami, FL, USA, Mar. 2005.
- [12] S. Jung, U. Lee, A. Chang, D. Cho, and M. Gerla. BlueTorrent: Cooperative Content Sharing for Bluetooth Users. In *PerCom'07*, White Plains, NY, Mar. 2007.
- [13] O. Kasten and M. Langheinrich. First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network. In *PACT'01*, Barcelona, Spain, Oct. 2001.
- [14] J. LeBrun and C.-N. Chuah. Feasibility Study of Bluetooth-Based Content Distribution Stations on Public Transit Systems. In *ACM MobiShare*, Los Angeles, CA, Sept. 2006.
- [15] J. Leguay, A. Lindgren, J. Scott, T. Friedman, and J. Crowcroft. Opportunistic Content Distribution in an Urban Setting. In *CHANTS'06*, Pisa, Italy, Sept. 2006.
- [16] M. Leopold. Evaluation of Bluetooth Communication: Simulation and Experiments. Technical report, Dept. of Computer Science, Univ. of Copenhagen, Spring 2002.
- [17] P. S. Lőránt Farkas, Balázs Bakos. A Practical Approach to Multicasting in Bluetooth Piconets. In *WCNC'06*, Las Vegas, USA, Apr. 2006.
- [18] A. Madhavapeddy and A. Tse. A Study of Bluetooth Propagation Using Accurate Indoor Location Mapping. In *UbiComp'05*, Tokyo, Japan, Sept. 2005.
- [19] S. Mander, D. Reading-Picopoulos, and C. Todd. Evaluating the Adaptive Frequency Hopping Mechanism to Enable Bluetooth - WLAN Coexistence. In *LCS'03*, London, UK, Sept. 2003.
- [20] P. Maymounkov and D. Mazieres. Rateless Codes and Big Downloads. In *IPTPS'03*, Berkeley, CA, Feb. 2003.
- [21] L. Meier, P. Ferrari, and L. Thiele. Energy-efficient bluetooth networks. In *Technical Report 204*, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Jan. 2005.
- [22] M. Motani, V. Srinivasan, and P. S. Nuggehalli. PeopleNet: Engineering A Wireless Virtual Social Network. In *MobiCom'05*, Cologne, Germany, Sept. 2005.
- [23] New York City Pedestrian Level of Service Study - Phase I, 2006.
- [24] B. S. Peterson, R. O. Baldwin, and J. P. Kharoufeh. Bluetooth Inquiry Time Characterization and Selection. *IEEE TOMC*, 5(9):1173–1187, Sep. 2006.
- [25] R. J. Punnoose, R. S. Tseng, and D. D. Stancil. Experimental Results for Interference between Bluetooth and IEEE 802.11b DSSS Systems. In *VTC'01*, Rhodes, Greece, May 2001.
- [26] I. Sedov, S. Preuss, C. Cap, M. Haase, and D. Timmermann. Time and energy efficient service discovery in Bluetooth. In *VTC'03*, Jeju, Korea, Apr. 2003.
- [27] M. B. Shoemaker. Wi-Fi (IEEE 802.11b) and Bluetooth coexistence issues and solutions for the 2.4 GHz ISM Band. Technical report, Texas Instruments, Feb. 2001.
- [28] K. Shuaib, M. Boulmalf, F. Sallabi, and A. Lakas. Performance analysis: Co-existence of IEEE 802.11g with Bluetooth. In *WOCN'05*, Dubai, United Arab Emirates UAE, Mar. 2005.
- [29] F. Siegemund and M. Rohs. Rendezvous Layer Protocols for Bluetooth-Enabled Smart Devices. *Personal and Ubiquitous Computing Journal*, 7(2):91–101, Jul. 2003.