BlueMonarch: A System for Evaluating Bluetooth Applications in the Wild

Timothy J. Smith
University of Toronto
tsmith@cs.toronto.edu

Stefan Saroiu
Microsoft Research
ssaroiu@microsoft.com

Alec Wolman
Microsoft Research
alecw@microsoft.com

ABSTRACT

Despite Bluetooth's popularity, low cost, and low power requirements, Bluetooth applications remain remarkably unsophisticated. Although the research community and industry have designed games, cell-phone backup, and contextual advertising systems with Bluetooth, few such applications have been prototyped or evaluated on a large scale. Evaluating Bluetooth applications requires recruiting devices in the wild and developing robust software that can adapt to the heterogeneity of these devices. These requirements have limited both the number and the magnitude of the experiments with Bluetooth applications.

This paper proposes BlueMonarch, a system for evaluating Bluetooth applications in the wild. BlueMonarch emulates a Bluetooth transfer to any device responding to Bluetooth Service Discovery requests; because many cell-phones, laptops, and PDAs in the wild respond to such probes, BlueMonarch enables quick prototyping of Bluetooth applications in the wild, to hundreds of unmodified Bluetooth devices. After we present the feasibility and accuracy of BlueMonarch, we use BlueMonarch to evaluate a content delivery system for Bluetooth. With BlueMonarch, we evaluated our system inside a mall and a subway system; we were able to send tens of megabytes of data to hundreds of Bluetooth devices in just a little over an hour.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – Wireless communication

General Terms

Experimentation,Measurement,Performance

Keywords

Bluetooth

1. INTRODUCTION

Bluetooth is more popular, cheaper, and consumes less power than Wi-Fi. Recent news reports estimate that one billion consumers own Bluetooth devices [7], and within a few years Bluetooth-enabled devices are predicted to outnumber Wi-Fi five to one [38].

Bluetooth interfaces consume an order of magnitude less power than Wi-Fi and are very inexpensive. The manufacturing cost of a Bluetooth radio is roughly one third of that of Wi-Fi [37].

Yet, despite its popularity, low cost, and low power requirements, Bluetooth is used today primarily to perform three simple tasks: synchronizing address books, connecting wireless headsets to cell-phones, and connecting laptops to the Internet via a cell-phone uplink. Both research and industry have developed more sophisticated Bluetooth applications, such as content delivery systems [12], cell-phone backup [23, 36], contextual advertising systems [1, 5, 2], games [30, 3, 15], and social software [19]. However, none of these systems has enjoyed widespread deployment.

We cannot claim to understand all the reasons for this lack of success. However, we believe that one significant factor is the enormous effort that developers must make to prototype and evaluate Bluetooth applications in the wild. First, recruiting a large number of participating devices poses a hardship. Second, Bluetooth devices have extremely heterogeneous hardware and software [33], which makes it difficult to develop robust application code. Third, performing a representative evaluation of a Bluetooth application requires repeating the experiment in different contexts because the application behavior is extremely sensitive to its execution context and environment. These challenges have significantly limited Bluetooth application experimentation. The absence of large-scale, in-the-wild deployments has in turn cast doubt over the viability of sophisticated Bluetooth applications.

This paper takes a modest step toward reducing the effort needed to prototype Bluetooth applications. We present BlueMonarch, a system for evaluating Bluetooth applications in the wild. BlueMonarch emulates a Bluetooth transfer to any device that is set to respond to Bluetooth inquiries. Because many cell-phones, PDAs, and laptops in the wild respond to such probes, researchers can use BlueMonarch to emulate transfers to a diverse set of devices. This functionality makes BlueMonarch useful for evaluating a large class of Bluetooth applications, those in which a local server under the experimenter's control sends data to any remote device answering Bluetooth inquiries. By requiring control of just one device of the two end points of a Bluetooth link, BlueMonarch enables evaluations of Bluetooth applications to tens or even hundreds of Bluetooth devices.

BlueMonarch's measurement technique is inspired by Monarch, a tool for emulating TCP transfers to Internet hosts without requiring their cooperation [14]. BlueMonarch shares Monarch's key observation about how transfer protocols work: a sender transfers data to a receiver in arbitrarily sized packets (typically large) that are answered with small-sized acknowledgment packets. BlueMonarch uses generic Bluetooth discovery probes and responses to emulate this packet exchange between a sending device under the experimenter's control and any remote device that is discoverable. Because devices need not cooperate, BlueMonarch significantly lowers the barrier for the evaluation of Bluetooth applications. To understand the extent of our system's practicality, we used BlueMonarch to evaluate a content delivery system prototype [1]: using a single laptop equipped with four Bluetooth radios, we transmitted tens of megabytes of data to hundreds of Bluetooth devices in the wild in just a little over an hour.

BlueMonarch's accuracy stems from its direct online measurements. For every packet transmitted in its emulated transfer, BlueMonarch sends an actual probe packet of the same size to the receiving device and interprets the response packet as an incoming acknowledgment. Thus, emulated transfers are subjected to the same wide range of conditions as real Bluetooth transfers, including interference, frame losses, and retransmission delays. However, because BlueMonarch controls only one device, it can estimate conditions for the round-trip link but not the one-way link. Despite this limitation, our evaluation shows that packet-level traces of transfers emulated with BlueMonarch closely match actual Bluetooth transfers.

BlueMonarch enhances the state of the art in evaluating Bluetooth applications. Researchers currently evaluate these applications by running them in simulators or in controlled environments [23, 1, 36]. In contrast, BlueMonarch provides live access to Bluetooth devices in the wild. This permits experimentation in realistic scenarios for which emulators are not widely available and controlled experiments are not representative. BlueMonarch captures the signal propagation characteristics, obstructions, and interference that exist in deployed systems. Additionally, software developers can test and debug the performance and reliability of Bluetooth applications to uncover bugs, performance bottlenecks, or poor application design decisions.

We organized this paper to meet the needs of readers who are unfamiliar with Bluetooth. Section 2 identifies several Bluetooth applications that could be successfully evaluated using BlueMonarch. Section 3 presents a short primer on Bluetooth that readers familiar with Bluetooth can skip. We present the design of BlueMonarch in Section 4, discuss implementation details in Section 5, and evaluate BlueMonarch's accuracy in Section 6. Section 7 shows the results obtained when we used BlueMonarch to evaluate a content delivery system prototype for Bluetooth. We describe the related work in Section 8 and summarize our conclusions in Section 9.

2. BLUETOOTH APPLICATIONS

BlueMonarch emulates only those Bluetooth transfers in which data is sent from a local device to any device in the wild. BlueMonarch cannot emulate upload transfers – those in which data is received from devices in the wild. Despite this limitation, BlueMonarch enables the evaluation of many classes of Bluetooth applications, three of which are now described.

2.1 Opportunistic Content Delivery

There is enormous interest in applications that deliver data opportunistically over Bluetooth. Examples include advertising systems [1, 12, 17, 5, 2], bulk-data content delivery [18], or systems that deliver contextual information, such as bus schedules [20] or environment monitoring information [26]. By placing servers in urban crowded environments, these applications could deliver data to many passers-by at low cost to both users and content producers.

Before deployment, advertisers and content producers could gauge the effectiveness of their application by answering several questions, such as:

- Where should content delivery servers be placed to be most effective? The answer depends on the application; for example, some applications are optimized to reach as many users as possible, while others must ensure that passers-by remain in range long enough to receive a full copy of the data.
- Once deployed, how much data will the average user receive from the server? For example, an advertiser should know if there is ample opportunity to deliver ads that include soundtracks.
- How should the server be provisioned? For example, how many Bluetooth radios should a server have?

Advertisers could use BlueMonarch to setup a server that would emulate the data transfer to passers-by and thereby help shed light on these questions prior to full system deployment.

2.2 Bluetooth Access Points

When used at home, modern handheld devices have two ways to connect to the Internet: using 3G or using Wi-Fi. Neither option is ideal: 3G has long latencies and its coverage is spotty; Wi-Fi is power hungry. Bluetooth provides an attractive Internet connectivity alternative for mobile devices because it consumes less power than Wi-Fi, has better latencies than 3G, and is simple to deploy.

Another potential scenario for the deployment of Bluetooth access points is bringing Internet connectivity to areas with no landline or cellular infrastructure. Typically, a long-range Wi-Fi link [28] operates as a backbone, and a rooftop Wi-Fi mesh network connects each building or house to the long distance link [35]. This approach works well for last mile connectivity. However, Wi-Fi use on "last meter" handsets has disadvantages in terms of power consumption and interference among mobile handsets, the mesh network, and the long distance link. Bluetooth access points [21] could serve as bridges between the Wi-Fi mesh network and mobile handsets.

BlueMonarch could help engineers answer several questions about the viability of Bluetooth access points before a full deployment:

- What are the bandwidth capabilities of such a system? Would Bluetooth access points provide adequate Internet connectivity?
- Where should the Bluetooth access points be located to minimize the possibility of Wi-Fi interference?
- What handover strategies work well between multiple Bluetooth access points?

2.3 Decentralized Applications

Another class of Bluetooth applications are those with a decentralized design, where data exchange occurs between Bluetooth devices. Existing examples of such applications are multi-player games [15, 30], Bluetooth dating [19], or cell-phone backup [23, 36]. Today's evaluations of these prototypes follow the same four-step plan: (1) recruit Bluetooth devices, (2) instrument them with the new application code, (3) deploy them, and (4) collect data [11, 34]. Unfortunately, this methodology leads to very small scale evaluations unless the recruited devices could also gather data from their interactions with uninstrumented Bluetooth devices. These interactions are much more abundant: in an evaluation spanning several months, the number of interactions with uninstrumented devices was two orders of magnitude higher than those between participating devices only [15]. Despite the greater number of these interactions, they currently provide little useful data because of the inability to measure the Bluetooth environment.

BlueMonarch could enhance the richness of the data collected from interactions with uninstrumented devices by emulating downstream Bluetooth transfers. For example, BlueMonarch could help measure the duration of cell-phone backup transfers to instrumented or at-large devices, or BlueMonarch could measure interference when many people participate in multi-player games.

3. BLUETOOTH NETWORKING PRIMER

Bluetooth, a short-range wireless protocol designed to enable personal area networks, primarily connects laptops, mobile phones, cameras, GPS receivers, and headsets. The Bluetooth protocol specification covers all layers of a typical networking stack, from baseband to application. Bluetooth operates in the 2.4GHz license-free ISM radio band. Because this band is shared with many other radio transmitters – such as 802.11, car security systems, and microwave ovens – Bluetooth uses rapid frequency hopping (1.6 kHz) across 79 1 MHz channels to reduce the impact of interference.

Bluetooth is complex. Its most recent core specification exceeds 1200 pages [6]. BlueZ [8], the Bluetooth stack included in most Linux distributions, has over 25K lines of code yet supports only a partial implementation of the Bluetooth specification. We perform all experiments in this paper on Linux using the BlueZ stack.

3.1 Bluetooth Network Formation

Bluetooth networks are organized into groups called *piconets*, each of which is limited to eight active devices. A piconet consists of one *master* device and one or more *slave* devices. These devices share a frequency hopping sequence that is determined by the master's MAC address and its clock. Bluetooth devices use a two-step process to interconnect. First, *inquiry* detects nearby devices: nodes hop along a special inquiry sequence and broadcast inquiry messages; other nodes periodically enter inquiry scan mode to hop along the same sequence listening for inquiry messages.

Second, *paging* involves nodes asking to join an existing piconet or to form a new one. The node that initiates paging becomes master of the resulting piconet. Therefore, a *role switch* must occur to let new devices join an existing piconet. This works as follows: device N performs inquiry and paging with device M, master of an existing piconet. After paging, a new piconet exists with N as master and M as slave. These devices then perform a role switch so that N becomes a slave in the original piconet where M is master. Role switching also occurs for other purposes, such as balancing power consumption among devices participating in a piconet by periodically switching masters [29].

3.2 Bluetooth Pairing

The Bluetooth specification defines a simple authentication procedure based on the exchange of a shared secret, or "PIN". Authentication is pair-wise, i.e. between a master and slave. Authentication between slaves is not supported because piconet slaves cannot communicate directly with each other.

When either the master or a slave requests authentication, a hashed PIN is used to generate a *link key*. This key, used in future sessions, usually resides in each device's persistent storage. The entire PIN challenge-response procedure is called *pairing*. On most embedded devices, pairing requires user participation. Pairing is completely optional, and it occurs after a piconet has been formed and in response to a connection request at a higher protocol layer (e.g. by a file transfer application that requires authentication). The BlueMonarch mechanism for Bluetooth emulation, based on Service Discovery Protocol requests, does not usually require pairing because it precedes the establishment of a connection to a higher level service.

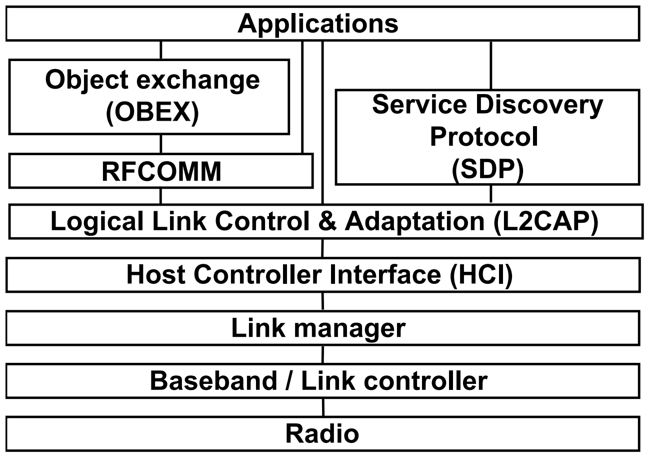

Applications
Object exchange (OBEX)
Service Discovery Protocol (SDP)
RFCOMM
Logical Link Control & Adaptation (L2CAP)
Host Controller Interface (HCI)
Link manager
Baseband / Link controller
Radio

Figure 1: ***The Bluetooth Protocol Stack. The SCO layer is not shown; SCO is used solely for voice transmission between Bluetooth devices.***

3.3 The Bluetooth Protocol Stack

The Bluetooth core specification [6] defines an entire network stack ranging from the operation of the baseband radio layer to application layer protocols(see Figure 1). This section briefly describes each layer emphasizing those most relevant to BlueMonarch.

3.3.1 The Baseband, Link Manager, and HCI Layers

The *baseband and link controller* layer performs medium access control for the *radio* layer. It establishes and maintains Bluetooth connections, including performing and responding to inquiries, forming piconets, and managing master/slave relationships. The *link manager* implements two types of logical links between Bluetooth devices: (1) synchronous and connection-oriented (SCO), and (2) asynchronous and connection-less (ACL). SCO links enable voice connections by supporting regular, periodic exchange of data with a pre-allocated level of bandwidth; ACL links communicate data with no real-time requirements. The *host controller interface (HCI)* is the API between the higher and lower layers of the protocol stack. It provides a uniform way to access hardware status and control registers across of variety of Bluetooth devices. This paper uses packet traces captured at the HCI layer to characterize the behavior of Bluetooth transfers. All layers above the HCI level are implemented by the host's Bluetooth stack (the BlueZ [8] stack in BlueMonarch); lower layers are implemented by the device driver and hardware for a specific Bluetooth network card.

3.3.2 The L2CAP Layer

The L2CAP layer multiplexes higher-level protocols and applications across a single ACL link. Unlike ACL, connection-oriented SCO links bypass the L2CAP layer and are directly accessible from the application layer. L2CAP lets each higher layer protocol set its own L2CAP maximum transmission unit (MTU), which can be set independently for each direction of the ACL connection. The receiver always sets the MTU and communicates its value to the sender during connection initiation. The default L2CAP MTU in Bluetooth is 672 bytes [6]; applications running on top of the BlueZ Bluetooth stack can adjust the MTU values using socket options.

Although the L2CAP specification [6] defines optional retransmission functionality, many Bluetooth stacks, including BlueZ, do not implement packet retransmissions [16]. Because Bluetooth is used primarily for single-hop links, retransmission functionality implemented at the baseband layer obviates the need for L2CAP retransmissions. At the baseband layer, Bluetooth senders expect a frame to be acknowledged during a single timeslot. If the frame is not acknowledged, the sender continues to retransmit it on the next channel hop until either a "flush timeout" or "link supervision timeout" occurs. Flush timeouts, used for time-sensitive data (e.g., real-time streaming media), expire when no acknowledgments have been received from a remote host after a specific amount of time; the BlueZ Bluetooth stack does not currently support these timeouts. Link supervision timeouts, which detect total link failures, expire when no packets of any type are received from a remote host for a specific amount of time. By default, BlueZ sets the link supervision timeout value to 32000 timeslots, or 20 seconds. When the timeout expires, the baseband layer declares the packet lost, and the L2CAP layer terminates the connection with an error. In addition to retransmissions, the L2CAP specification also describes an optional flow control mechanism that is not supported in many implementations, including BlueZ.

3.3.3 Service Discovery Protocol (SDP)

The SDP layer lets devices discover each others' services. Whenever a new service is installed on a device, the Bluetooth specification requires the application to register with the local SDP server. Other devices can then connect to this server and search through service records to determine which services this device supports.

Each transaction in SDP consists of one request protocol data unit (PDU) and one response PDU. The standard five-byte SDP header lists the message type field, the PDU's length, and the transaction ID, which matches response and request PDUs. If an SDP server receives a 'malformed' SDP request, it issues an Error Response PDU, that contains the standard header plus a two-byte error code, for a total of seven bytes. The transaction ID in the Error Response PDU is set to the transaction ID of the malformed request.

3.3.4 The RFCOMM and OBEX Layers

The RFCOMM layer emulates a serial communication channel similar to RS-232, so that RFCOMM can support legacy serial port applications running over Bluetooth. BlueZ's RFCOMM implementation supports a credit-based flow control scheme: each party regularly sends a five-byte packet advertising how many RFCOMM frames it can accept before filling up its buffers.

OBEX is an application-layer protocol for transferring files (e.g., an MP3 file, a photograph, or business card) over Bluetooth. Two devices that want to exchange a file must perform these three steps:

1. The sender must send an SDP request to make sure the receiver supports OBEX transfers.
2. The receiver answers with an OBEX service record. This record contains information used to establish an RFCOMM connection.
3. Once the RFCOMM communication is set up, the sender segments the file into OBEX packets, sending them one at a time. The receiver reassembles the incoming OBEX packets back into a file.

OBEX packets are fragmented into RFCOMM frames, which in turn are fragmented into L2CAP packets. RFCOMM's MTU typcially differs from the L2CAP layer's MTU for RFCOMM. In our experiments, we discovered that many devices set their RFCOMM MTU to 1024 and their RFCOMM L2CAP MTU to 1013 or lower. This MTU mismatch led to fragmentation, where large RFCOMM frames (1024 bytes) were split into two L2CAP packets. Each fragment also contained an L2CAP start or continuation header, so the actual RFCOMM payload transmitted in each fragment was 1008 bytes and 16 bytes, respectively. Although the preceding MTU values are common in practice, they are not universal. For example, many mobile phones use a 672-byte RFCOMM L2CAP MTU, and Apple's Bluetooth stack uses a 32KB SDP L2CAP MTU.

4. DESIGN

This section presents BlueMonarch's design. We first review how BlueMonarch emulates Bluetooth transfers at the L2CAP layer. We then discuss the interface that BlueMonarch provides to facilitate simple prototyping of Bluetooth applications. Finally, we describe the types of probes that BlueMonarch uses, the L2CAP protocol options that it can emulate, and factors affecting its accuracy.

4.1 BlueMonarch Data Transfers

A typical Bluetooth file transfer occurs when a sender on one device sends small control packets and large data packets to a receiver on another device, and the receiver responds with small control packets (Figure 2a). BlueMonarch emulates these exchanges by sending appropriately sized, malformed SDP packets to the remote device that elicit small SDP responses (Figure 2b). To emulate an L2CAP transfer, BlueMonarch creates both an L2CAP sender and an L2CAP receiver on the *same* local device, but interposes between them (see Figures 2c and 2d). BlueMonarch captures a sender's packet and sends in its place an identically sized SDP packet to the remote device. Upon receiving a response, BlueMonarch forwards the original captured packet to the local L2CAP receiver. Packets in the reverse direction, from the local receiver to the local sender, are forwarded directly.

BlueMonarch sets the sizes of the outgoing SDP requests to exactly match the sizes of the L2CAP transfer's actual outgoing data and control packets. In Section 6.2, we show that the SDP responses that BlueMonarch receives also closely match in size the L2CAP transfer's incoming packets (to within 96% for an OBEX transfer). As a result, the sender observes similar round-trip times, delays, and interference for its packet transmissions. Because BlueMonarch uses online measurements rather than analytical models of Bluetooth transfers, the characteristics of transfers emulated by BlueMonarch closely match those of real Bluetooth transfers.

BlueMonarch currently emulates Bluetooth data transfers in the downstream direction only, i.e. connections in which data flows from the BlueMonarch host device to the remote devices. This mimics the typical usage pattern in which a mobile Bluetooth device downloads content from a fixed infrastructure Bluetooth device. Emulating upstream data flows requires small probe packets to elicit large response packets; ideally, this would require BlueMonarch to accurately control response sizes. Although we can suggest ways to generate large upstream packets (e.g., by crafting SDP queries that produce large responses), we have not implemented upstream capability in our current version of BlueMonarch and we believe that doing so poses a significant challenge for future versions of BlueMonarch.

Two Bluetooth devices must be initially paired with each other to perform an OBEX transfer (see Section 3.2). However, BlueMonarch uses SDP to emulate transfers and thus the probing device need not be paired with the receiver to perform its emulated transfer. Except for pairing, all other stages of an OBEX Bluetooth transfer, such as inquiry, paging, and data exchange, are present in a BlueMonarch transfer.

For BlueMonarch to accurately perform data transfers, several assumptions must hold. For example, we assume that arbitrary

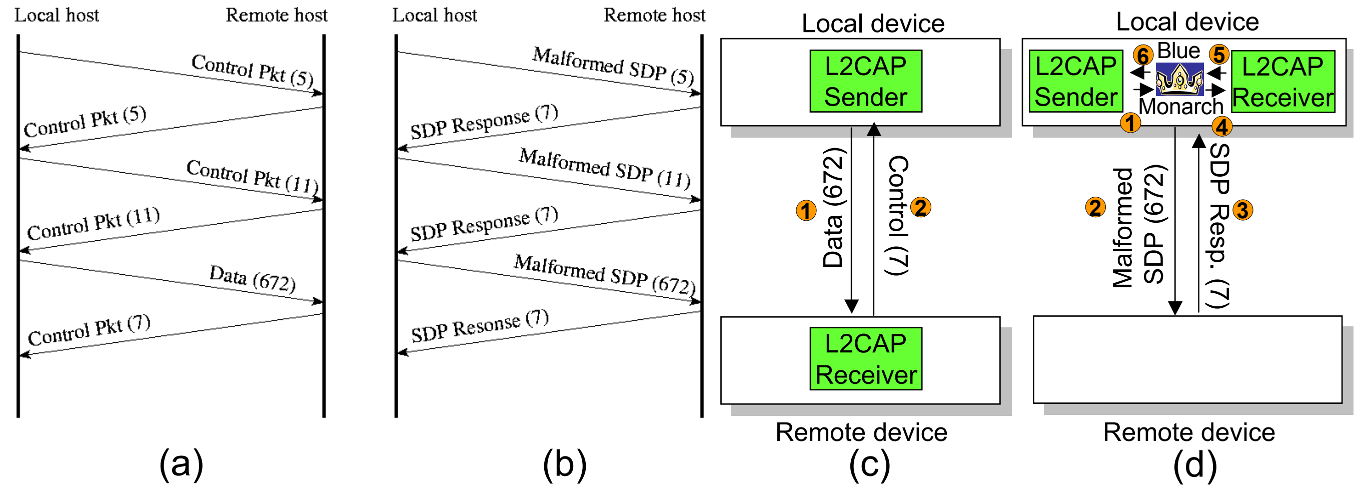


Figure 2: ***The BlueMonarch packet exchange. In a normal Bluetooth transfer, small control packets and large data packets flow in one direction, and small control packets flow in the other (shown in part (a)). BlueMonarch emulates this transfer by using malformed SDP packets of different sizes that elicit small SDP response packets (shown in part (b)). In a normal transfer, the sender and the receiver are on different devices (shown in part (c)); however, BlueMonarch places them on the same device and interposes between them (shown in part (d)). The numbers in parentheses show packet lengths in bytes. For simplicity, we do not show baseband layer acknowledgments.***

Bluetooth devices respond to service discovery requests, and that an accurate emulation of round-trip (rather than one-way) packet latencies and packet loss is sufficient for an accurate emulation of Bluetooth transfers. We discuss the factors that affect BlueMonarch's accuracy later in this section.

4.2 BlueMonarch Interface

BlueMonarch does not require applications to be modified in order to be evaluated. It requires only the ability to instantiate the application sender and receiver on the same device. Thus, BlueMonarch exposes an interface that supports all Bluetooth networking and systems calls. Additionally, it does not modify the semantics of these calls; applications need not be aware that BlueMonarch is emulating their flows.

4.3 BlueMonarch Probes

BlueMonarch uses a specially crafted "malformed" SDP packet to emulate a Bluetooth transfer. These SDP packets are addressed to a specific receiver and ask for information about its services. The SDP transaction ID monotonically increases with each packet sent. BlueMonarch uses this ID to match outgoing probes with incoming replies. It sets the SDP probe's request type to "0x06" which corresponds to an "SDP Search services and attributes" request. Normally, this request's payload should contain descriptions of the services under inquiry. Instead, BlueMonarch inserts null data in the packet's payload and carefully choses the payload size so that the SDP probe's size matches the BlueMonarch sender's packet size.

Upon receiving a malformed SDP packet from BlueMonarch, a remote device responds with an SDP error packet whose size is seven bytes, matching the size of acknowledgments received in an OBEX transfer. Figure 3 illustrates the format of the seven-byte SDP error packet.

PDU type (1 byte)	Transaction ID (2 bytes)	Length (2 bytes)	Error Code (2 bytes)

Figure 3: ***The format of the SDP error packet sent by a remote device to a BlueMonarch probe. The first byte is set to 0x01 – "SDP Error code". The last two bytes are set to 0x003 – "Invalid request syntax".***

4.4 Bluetooth Devices that Respond to BlueMonarch Probes

To be accurate, BlueMonarch requires the remote host to respond to Bluetooth inquiries and to every SDP probe it receives. In a previous experiment, we found that many devices in public spaces were discoverable (i.e., responsive to Bluetooth inquiries) [33]. The exact percentage of such devices is extremely difficult to determine experimentally: a Bluetooth-capable device with its radio turned off is by definition undetectable; one would have to rely instead on a survey of device owners, which to our knowledge has not been published. In our experience, Motorola is the only large manufacturer that sets their cell-phones to be non-discoverable by default.

We also conducted a simple experiment to verify what fraction of discoverable Bluetooth devices answer every SDP probe they receive. We sent three SDP probes back-to-back to all discoverable Bluetooth devices found in one of our University's cafeterias. The first and last SDP probes were regular SDP request packets. The middle probe was the BlueMonarch malformed SDP probe packet. To determine how many Bluetooth devices respond to these probes, we measured how many devices responded to the first and last SDP packets without responding to the middle malformed probe. We could not include devices responding to the initial SDP request but not to the last one because we could not distinguish between the case where the device does not answer BlueMonarch's malformed probe and the case where the device moves outside our probing device's range during the measurement.

Using this methodology, we conducted a short experiment to measure 34 devices. Based on their MAC addresses, we inferred that these devices were made by 16 different manufacturers. Only BlackBerry devices do not answer SDP probes; instead they first require their owner to pair. We therefore decided to filter them out in future experiments; whenever BlueMonarch discovers a BlackBerry device, it does not attempt to form an SDP connection.

4.5 Factors that Affect BlueMonarch's Accuracy

As noted previously, BlueMonarch is based on round-trip (rather than one-way) estimates of packet losses. It cannot distinguish between losses occurring on the downstream link, i.e., from the sending device to the remote device, and those occurring on the upstream link, i.e., from the remote device to the sender. While in theory this could cause BlueMonarch to behave differently than regular Bluetooth transfers, in practice BlueMonarch is not affected by loss: Bluetooth stack implementations often do not implement retransmissions instead, as Section 3 presented, losses are handled by the baseband layer in these Bluetooth stack implementations.

One potential source of BlueMonarch inaccuracy derives from differences in incoming packet sizes. When data is transferred, all incoming acknowledgment packet sizes are seven bytes, which BlueMonarch can match. However, Bluetooth transfers also exchange a few small control packets of different sizes (e.g., 4, 5, 8, 11, and 14 bytes) for connection startup and tear-down; BlueMonarch cannot match these sizes. Section 6 shows that the small difference in packet sizes does not have a significant effect on BlueMonarch's accuracy.

As mentioned in Section 3.3.4, a Bluetooth receiver's RFCOMM layer occasionally sends flow-control packets. Because these packets are never solicited by the sender, a BlueMonarch transfer misses them. However, our evaluation shows that this factor also has negligible effect on BlueMonarch's accuracy.

Another source of potential inaccuracy results from the way BlueMonarch and Bluetooth receivers process incoming data. As we show in Section 6, the behavior of a regular Bluetooth transfer is influenced by the application running on the receiver. For example, some receivers store incoming data only in RAM (e.g., an RSS feed or a streaming video application); others store it in stable storage (e.g., a song download application). In our experiments, we found that a transfer to a receiver that stores incoming data on a flash card one packet at a time takes 8.2% longer than a transfer to a receiver that buffers incoming data in RAM and stores it to disk only at the end of the transfer. Because remote devices answer BlueMonarch's probes immediately, BlueMonarch transfers closely match the latter. Although these differences do not directly affect BlueMonarch's accuracy in emulating the wireless link, they do affect its accuracy in modeling the end-to-end behavior of Bluetooth applications.

Finally, BlueMonarch emulates a Bluetooth transfer by sending specially formatted SDP requests that elicit small responses from a remote device. Thus, BlueMonarch can accurately emulate applications that send out packets and receive small responses. Because BlueMonarch can control the size of outgoing packets but not of incoming responses, it cannot be used for applications that primarily "pull" data from client devices.

5. IMPLEMENTATION

This section present the details of BlueMonarch's implementation. We also describe BlueMonarch's limitations that arise from Bluetooth protocol features that are not implemented by the BlueZ Linux Bluetooth stack. We then discuss the potential security concerns raised by our system and how we addressed them.

5.1 Application Interface to BlueMonarch

We implemented BlueMonarch to facilitate the evaluation of Bluetooth applications. Because the interface BlueMonarch exposes is identical to that of the standard Bluetooth stack, applications need no modifications to run over BlueMonarch; they can be evaluated "out of the box". Since application-level Bluetooth communications are ultimately routed through the L2CAP socket layer, BlueMonarch can intercept and re-route network flows from any Bluetooth application. In fact, we have already ran multiple applications written in different languages (e.g, C and Python) on top of BlueMonarch without modification.

5.2 Bluetooth Transfer Emulation

BlueMonarch is implemented as a Linux 2.6 kernel module. To emulate a flow to a remote device, the BlueMonarch kernel module is inserted before the sender and receiver are started in user space. This module replaces the BlueZ L2CAP layer with BlueMonarch's L2CAP, which contains modifications to three L2CAP socket interface calls: *connect()*, *sendmsg()*, and *close()*.

5.2.1 Connect

The L2CAP *connect()* call creates an L2CAP socket connection to a remote device. This call takes as parameters the MAC address and "Port/Service Multiplexer" (PSM) value of the remote device. The PSM acts as an endpoint identifier: its value reflects the remote service to which the sender is trying to connect. For example, in an OBEX transfer, the sender is trying to connect to the default RFCOMM server (the layer under OBEX) on a remote device, in which case the PSM value should, by convention, be three.

BlueMonarch modifies the connect call by changing the PSM to initiate an L2CAP connection to the remote device's SDP server, whose PSM value is one. Once the connection is established, BlueMonarch also creates a loopback connection to an L2CAP receiver on the local device. The implementation maintains a mapping between each L2CAP connection to a remote device and its associated L2CAP loopback connection.

5.2.2 Sendmsg

Our two modifications to *sendmsg()* involve how we handle the local sender's and receiver's calls. For senders, BlueMonarch queues L2CAP packets in a local buffer instead of forwarding them. For each queued packet, BlueMonarch creates a malformed SDP request of the same size and forwards it to the remote device. Each SDP request has a unique transaction ID; these IDs match incoming SDP replies to corresponding requests.

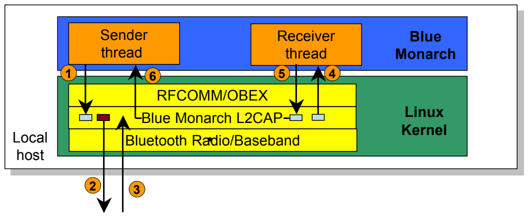
Upon receiving an SDP reply, BlueMonarch dequeues the corresponding L2CAP packet and forwards it to the local receiver. When the local receiver calls *sendmsg()*, BlueMonarch forwards the L2CAP packets immediately to the local sender. Figure 4 illustrates BlueMonarch's sequence of packet exchanges.

5.2.3 Close

When the local sender calls *close()*, BlueMonarch discards all locally queued data before notifying the local receiver about the closed connection. We have not altered *close()* behavior for calls made by the local receiver.

5.3 Unsupported L2CAP Options

BlueMonarch does not support four options described by the L2CAP Bluetooth specification: (1) packet retransmission; (2) flow control; (3) connection authorization; and (4) connection encryption. While fully documented in the Bluetooth Specification, packet retransmission and flow control are optional features not implemented by the BlueZ Linux stack. These options may be useful in applications where low-level Bluetooth retransmissions and high-level RFCOMM flow control operate in an undesirable way. For example, when a client device constantly moves in and out of radio range, it may be preferable for an application to attempt retransmis-



#	Action	Packet	Source	Destination
1	Sender transmits packet	OBEX Data	local sender	remote receiver
2	BlueMonarch L2CAP intercepts packet, saves it, and transmits a probe	Malformed SDP	BlueMonarch	remote SDP
3	Remote responds	Resp. SDP	remote SDP	BlueMonarch
4	BlueMonarch L2CAP forwards saved packet to the receiver	OBEX Data	BlueMonarch	local receiver
5	Receiver sends OBEX ACK	OBEX ACK	local receiver	BlueMonarch
6	Proxy forwards OBEX ACK directly to the sender	OBEX ACK	BlueMonarch	local sender

Figure 4: ***Sequence of packet exchanges in the BlueMonarch implementation: BlueMonarch consists of a local sender and a local receiver. It replaces the Bluetooth L2CAP layer in the kernel to interpose between the sender and receiver. BlueMonarch also replaces OBEX packets with similarly sized malformed SDP probes to create the illusion that the remote host is the other endpoint of the Bluetooth transfer.***

sions less frequently than the 1.6kHz rate used by the baseband retransmission mechanism, thus freeing up the radio to service other clients between retransmission attempts. In future work, we plan to implement these two options in BlueZ and add support for them in BlueMonarch. We do not anticipate any problems incorporating packet retransmission and flow control in BlueMonarch.

Bluetooth devices use connection authorization and encryption for pairing. As part of the pairing protocol, devices exchange a session key used to encrypt all content exchanged over any future connection formed between them. Because it requires the cooperation of remote devices, BlueMonarch does not implement pairing.

5.4 Usage Concerns and Best Practices

As is often the case with using active measurement tools, BlueMonarch experiments can raise potential privacy and intrusiveness concerns. Bluetooth devices in the wild could perceive BlueMonarch's traffic as hostile and intrusive. Because many of these devices are power constrained, handling and responding to BlueMonarch's probes could cause them to unnecessarily deplete power. To address these concerns, we intentionally limited the behavior of our system running BlueMonarch to limit its impact on other devices. We believe the guidelines described below are appropriate for other small-scale public deployments of BlueMonarch.

We limited the amount of communication that BlueMonarch imposes on any one device. BlueMonarch never transfers more than 1MB of data to the same device over the course of an experiment. The power consumed during a 1MB transfer is the same as that consumed using a wireless Bluetooth headset for just over two minutes. In practice, however, we found that we transferred less than 1MB of data to most devices.

Another cause of concern is the amount of RF interference BlueMonarch's measurements cause to nearby devices. Bluetooth shares its frequency band with other consumer devices, including Wi-Fi, microwave ovens, and cordless phones. To mitigate this concern, we restricted the duration of our BlueMonarch experiments to no

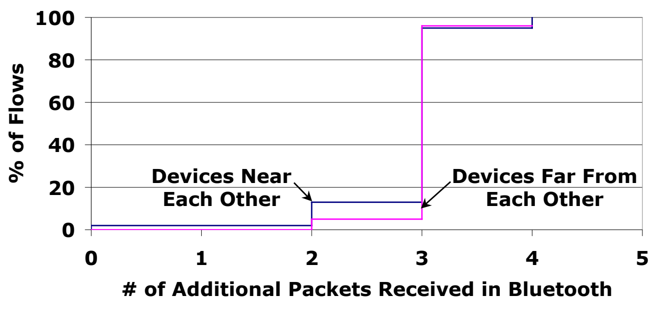


Figure 5: ***Distribution of the number of extra packets received by a regular Bluetooth transfer relative to a BlueMonarch transfer. In almost all transfers, there are between 1 to 4 extra packets of RFCOMM flow control messages sent by the receiver. Because they are unsolicited by the sender, they do not occur in BlueMonarch's transfers.***

more than four hours per day at a single location. We relegate to future work an investigation of interference effects from BlueMonarch experiments.

6. EVALUATION

This section presents two experiments designed to evaluate BlueMonarch's ability to emulate Bluetooth flows. First, we investigate network flow-level characteristics to assess BlueMonarch's flow-level accuracy. Second, we evaluate BlueMonarch when running multiple transfers to multiple receivers to assess its accuracy in piconet mode.

6.1 Methodology

We evaluated BlueMonarch's accuracy by comparing its flows to actual Bluetooth flows between the same devices. Performing this evaluation on a large-scale is difficult because it requires control over the software running at both end-points of a Bluetooth flow. We performed our evaluation by instrumenting the Bluetooth stack of two classes of devices: laptops and PDAs. Our laptops are Sony X505s running the BlueZ Bluetooth stack[1] in the Linux kernel version 2.6.23.1. Our PDAs are Nokia N800s running the Linux kernel version 2.6.18. On the laptops, we instrumented the default Linux OBEX package called *OBEXFtp*; on the PDAs, we experimented with two different OBEX packages: the one supplied by Nokia and another Linux package called *sobexsrv* [10] that we ported to the PDAs.

We ran BlueMonarch only on the laptops; we have not yet ported BlueMonarch to the PDAs. All flows presented in this evaluation were initiated from one laptop to one or many PDAs. We also evaluated BlueMonarch using transfers between laptops only, but brevity precludes our presenting these results; all our laptop-to-laptop BlueMonarch flows matched their Bluetooth counterparts perfectly.

Each experiment was split into rounds. In each round, we performed one regular OBEX transfer followed by a BlueMonarch transfer; we sent 1MB of data in each transfer. We used two different setups: "near" and "far". In the former, we placed the sender and receiver near each other on a desk. In the latter, we placed the sender and receiver about 10 meters apart, without line of sight, in two different rooms in our lab. For each flow, we gathered packet logs at the HCI layer (i.e., low in the Bluetooth stack); these logs captured the packets' sizes, payloads, and timestamps. We repeated

[1]BlueZ is the default Bluetooth stack in Linux.

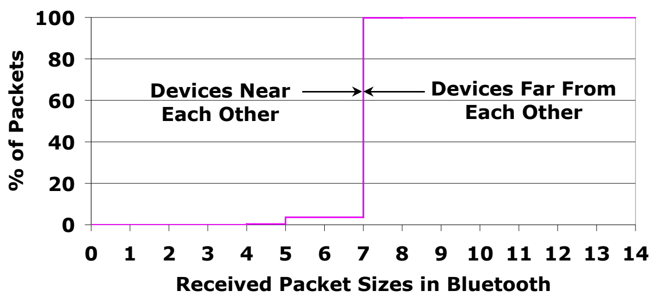


Figure 6: ***Distribution of the sizes of received packets in regular Bluetooth transfers. Over 96% of received packets are seven-bytes long, matching the sizes of received packets in BlueMonarch transfers. The remaining packets are connection setup and teardown control packets of between 4 and 14 bytes.***

each experiment 100 times and used all the data to plot our distributions.

6.2 Flow-level Accuracy

Number of Packets. We start by examining the differences between the number of packets exchanged by a BlueMonarch and a regular Bluetooth flow. The flows always have an identical number of sent packets: BlueMonarch is designed to send a probe for each packet sent by a regular Bluetooth stack. However, the numbers of received packets from the remote device (i.e., the Bluetooth slave) differ between the two flows. In each round, we measured the number of additional packets received in a regular Bluetooth transfer relative to a BlueMonarch transfer. Figure 5 presents the cumulative distributions of these extra packets for both the near and far setups.

Bluetooth transfers receive one to four additional packets in almost all our rounds. Our examinations revealed that all these packets were unsolicited RFCOMM control messages that implemented the RFCOMM flow-control mechanism. Because they are unsolicited by the sender, they never appear in BlueMonarch's emulated flows. The absence of unsolicited packets is a fundamental limitation of BlueMonarch.

Packet Sizes. We also examine whether L2CAP packets of BlueMonarch flows are the same size as their Bluetooth counterparts. As before, we do not present the sizes of the packets being sent because they match perfectly between BlueMonarch and Bluetooth. Instead, we present the sizes of the received packets. All BlueMonarch received packets are identical because they are replies to BlueMonarch's probes; their size is always seven bytes. For regular Bluetooth flows, Figure 6 shows the cumulative distributions of packet sizes for both the near and far setups.

For both setups, 96% of the received packets are seven bytes in size, identical to the packet sizes of the BlueMonarch flows. The remaining packets are either five-byte flow-control RFCOMM packets or 4 to 14 byte control packets used for OBEX connection setup and teardown.

Flow Dynamics. We compare the dynamics of BlueMonarch to Bluetooth flows by plotting the number of bytes transfered by the flow over its lifetime. Figure 7 (left side) illustrates the dynamics of two such flows that were run in one of our rounds chosen at random; the remaining 99 rounds show results consistent with these.

Our results show that Bluetooth's transfer rate is slower than BlueMonarch's: the BlueMonarch flow finishes a full two seconds earlier. We investigated the causes of this difference by examining the Nokia OBEX package running on the N800 PDA. One difference we found was that the PDA performed a *synchronous write* as soon as new data was received by the OBEX layer. In contrast, BlueMonarch flows never trigger any write operations by the remote device.

Unfortunately, we could not change this behavior because we did not have access to the source code for Nokia's OBEX package. Instead, we ported a Linux OBEX package (*sobexsrv* [10]) to the PDA. This package buffers all received packets and performs one single *write* operation at the end of the Bluetooth transfer.

Figure 7 (right side) compares the BlueMonarch flow to the Bluetooth flow running the ported OBEX package. These results show that the flow dynamics of BlueMonarch and Bluetooth are almost identical. The BlueMonarch flow still finishes earlier but only by 120 milliseconds. We believe that this difference is attributable to the additional packet processing performed by a full-blown Bluetooth receiver; in contrast, much less packet processing is done by a BlueMonarch receiver.

6.3 Piconet Accuracy

When sending data to multiple receivers, a Bluetooth master must form a piconet. In a piconet, the master sends data to each receiver one packet at a time in a round-robin fashion. To evaluate BlueMonarch's accuracy in piconet mode, we performed the following experiment. We set up a sending device to transmit a very large file to each slave joining its piconet. We set up six PDAs as slaves to join the piconet one by one, every 90 seconds. We measure the transfer rate of the initial flow established between the master and the first joining slave and we plot how this rate changes over time in Figure 8. The PDAs ran the Linux OBEX package that produced accurate results in earlier experiments.

Our results show that the transfer rate to the first slave PDA decreases over time; the rates measured (in Kbps) are 320, 220, 166, 152, 124, and 103 for piconets with 1, 2, 3, 4, 5, and 6 slaves, respectively. Although the transfer rates change over time, they are very similar between BlueMonarch and Bluetooth flows.

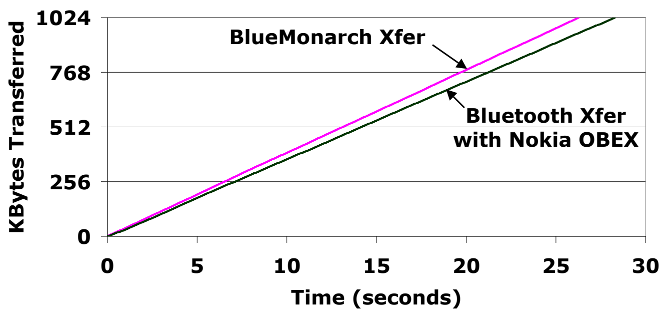
6.4 Summary

Our results demonstrate that BlueMonarch is accurate: its emulated flows behave similarly to Bluetooth flows with respect to the number of packets exchanged and packet sizes. We also found that the flow dynamics are similar, too, but only when the Bluetooth receiver does not have to perform synchronous writes to a slow storage device, such as a flash card. Finally, we found that BlueMonarch is accurate when running in piconet mode.

7. A CONTENT DELIVERY SYSTEM PROTOTYPE

We used BlueMonarch to implement a prototype of a Bluetooth-based content delivery system that could have many uses. For example, vendors and advertisers could use it to deliver ads or coupons for nearby stores to attract potential customers. Airlines could use it to inform their passengers about delays or deliver boarding passes and receipts directly to passengers. News companies could deliver RSS feeds of news articles and weather forecasts using such a system. And instructors could use it to deliver course materials and assignment solutions to their students without needing to post them on password-protected websites.

Our system's design is very simple. We use a dedicated server located in a busy place to deliver content over Bluetooth to all nearby receiving devices. Despite its simplicity, remarkably little is known



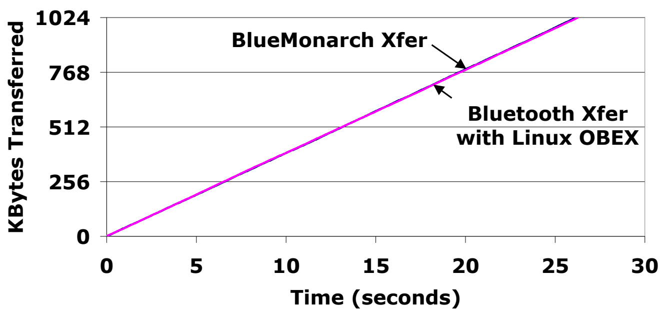


Figure 7: ***Flow dynamics: the number of KBytes transferred over the lifetime of a flow. On the left, a BlueMonarch flow finishes two seconds earlier than a regular Bluetooth flow running the Nokia OBEX package. On the right, the BlueMonarch and Bluetooth flow dynamics are very similar. The loss of accuracy on the left results from the Nokia OBEX package implementation: additional received data is immediately written to the N800 flash card.***

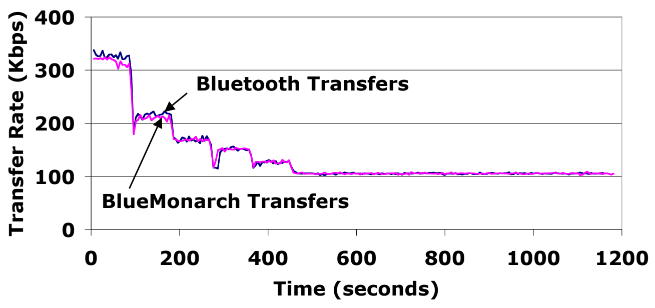


Figure 8: ***Rates for a Bluetooth and BlueMonarch transfer in a piconet scenario. In this experiment, six Nokia N800 devices joined the piconet one device at a time every 90 seconds. The transfer rate to the first joining slave decreases as each subsequent slave device joins the piconet. BlueMonarch and Bluetooth transfer rates match very closely.***

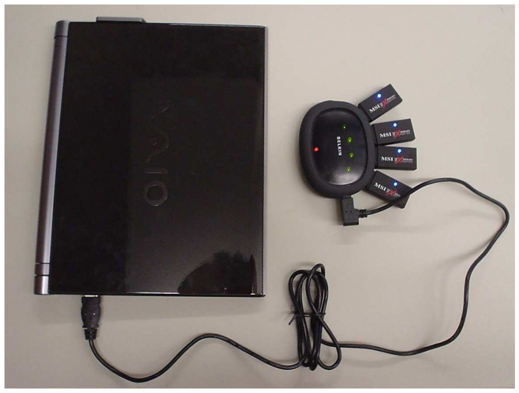


Figure 9: ***The Bluetooth content delivery server: We used a Sony Vaio X505 laptop equipped with four Bluetooth radios to serve content to all passing Bluetooth devices. One Bluetooth radio was responsible for discovering new devices, while the remaining three used BlueMonarch to serve content to all discovered devices.***

about how such a system would behave in practice. The difficulty of evaluating Bluetooth systems in the wild has left many important questions unanswered:

1. How many people can receive data from such a system? Can a stationary server reach more people than a mobile one?
2. How much data can a Bluetooth content delivery system send opportunistically to nearby devices, such as cell-phones, PDAs, and laptops? Could this system deliver large-sized data, such as songs?
3. How important is the location of servers to the delivery of large amounts of data to many people?

BlueMonarch shed light on these questions quickly and with little effort, proving the benefits of using BlueMonarch to prototype Bluetooth systems in the wild. We now describe our experiments' methodology and present a set of preliminary answers to the preceding questions. We leave a long-term evaluation of a Bluetooth content delivery system as future work.

7.1 Methodology

We implemented a multi-threaded Bluetooth server in Python and reserved one thread to discover nearby Bluetooth devices. Our system issues Bluetooth inquiries periodically, performing an inquiry scan for 10 seconds with an interval of 20 to 30 seconds between scans. Making the inquiry period random rather than static is a deliberate design choice to reduce the likelihood of colliding inquiries from multiple inquiring devices. When a device is discovered, our application forks a new thread that runs BlueMonarch to it. We configured BlueMonarch to emulate sending 1MB of data to each discovered device. We did not experiment with different settings for the inquiry scan duration and periodicity. The values used in our experiments were selected to be similar to the ones used by previous experiments with Bluetooth [11] and to meet our energy constraints.

Our application uses a Sony X505 laptop running the BlueZ Bluetooth stack in Linux kernel 2.6.23.1. Our laptop is equipped with four Class 1 (i.e., with a range of 100 meters) Bluetooth radios. We reserve one of the radios to the thread running Bluetooth discovery. We emulate BlueMonarch transfers on the remaining three radios, allocating them in a round-robin manner. Figure 9 shows our Bluetooth content delivery server.

We deployed our Bluetooth content delivery server in two different locations: inside of a mall (Eaton Centre in downtown Toronto) and in a subway system (the Toronto Subway System). We collected two traces at each location: a stationary one and a mobile one. In the mall, we gathered the stationary trace by placing our server in the center of the mall's food court. We collected the mobile trace by carrying the server and walking casually without en-

	Traces Collected in the Mall		*Trace Collected in the Subway*	
	Stationary Server	***Mobile Server***	***Stationary Server***	***Mobile Server***
Start Time	07/11/2008 15:06 EST	07/09/2008 15:20 EST	07/30/2008 16:34 EST	07/28/2008 16:08 EST
End Time	07/11/2008 16:32 EST	07/09/2008 16:44 EST	07/30/2008 18:10 EST	07/28/2008 17:44 EST
Duration	1 hour, 22 minutes	1 hour, 22 minutes	1 hour, 36 minutes	1 hour, 36 minutes
# of Devices Discovered	68	81	490	137
# of Successful Connections	58	71	184	51
# of Full Transfers (1MB)	30	27	5	9
Total Data Sent	37MB	40MB	29MB	14MB
Average Data Sent per Device	526KB	506KB	61KB	107KB
Average Connection Duration	65.28 seconds	40.85 seconds	27.23 seconds	39.57 seconds

Table 1: ***Summary statistics for the four traces: We prototyped our system in two locations, inside Eaton Centre (a mall in downtown Toronto) and in the Toronto subway system. We created a "stationary" scenario by placing the server in the center of the mall's food court and in a busy subway station, and a "mobile" scenario by carrying the server around the mall and riding the subway.***

tering any stores. On the subway, we gathered the stationary trace by placing the server in a busy corridor near the subway platform at Union Station, the largest commuter hub in Toronto's downtown core. We collected the mobile trace by riding a crowded subway car but remaining still inside the car. Each trace spanned about 90 minutes and was collected at roughly the same time of day, but on different weekdays. Table 1 shows high-level statistics of all our four traces.

7.2 Results

We use the devices' MAC addresses as a rough approximation of the number of people who receive data from our system. As Table 1 shows, our server sent data to 58 people inside the mall when stationary, and 71 people when mobile. However, fewer people received the entire 1MB of data: 30 people (51%) when stationary and 27 people (38%) when mobile.

Our findings are even more interesting when riding the subway. Our server discovered 490 people when placed on a subway platform. However, we sent data to only 184 people (38%). This low rate of success is most likely due to the environment; when a subway train arrives in the station, many people get off and quickly exit the station. In such a crowded environment, our server could discover a large client population but had little opportunity to send data to many of them.

Table 1 shows that the location and the stationarity of the server are very important to the success of the system. Our system sent more data when placed in a mall (37MB stationary and 40MB mobile) than when placed in the subway (29MB stationary and 14MB mobile). While being stationary or mobile made little difference inside the mall, placing our server in the subway station rather than inside a subway car drastically increased the number of people reached and the amount of data delivered.

Figure 10 provides a breakdown of Bluetooth devices discovered in each trace based on their manufacturer. To plot this graph, we used an online site that maps MAC addresses to device manufacturers [9] and excluded those devices whose manufacturers were not found by this online site. While Sony Ericsson and Nokia together account for about two-thirds of all devices found, our client population is very heterogeneous: we found between 9 and 15 manufacturers in each trace. As described in Section 4.4, our BlueMonarch implementation filters out BlackBerry devices because they require pairing before answering our SDP probes.

We also found different SDP MTU settings for the devices we discovered. Figure 11 presents the distribution of the MTUs found in each of the four traces. An MTU of 672 bytes was the most popular setting used by more than 80% of all discovered devices.

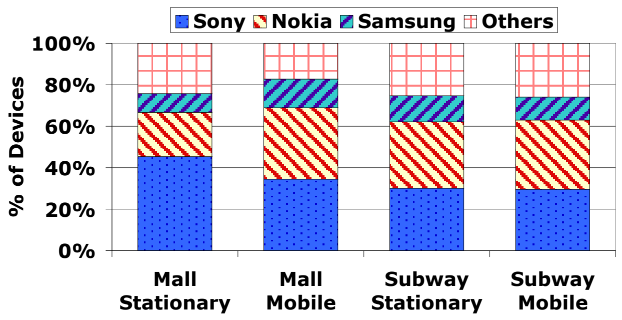


Figure 10: ***Breakdown of devices by their manufacturer: There are 12 manufacturers in the Mall Stationary trace, 9 in Mall Mobile, 15 in Subway Stationary, and 10 in Subway Mobile. Sony Ericsson and Nokia together account for about two-thirds of all devices discovered.***

The second most popular MTU setting was much smaller, only 256 bytes. These results suggest that a content delivery system must be able to handle a diverse client population with very heterogeneous hardware and software characteristics.

To examine in-depth how much data our system delivered, Figure 12 plots the cumulative distribution of the amount of data sent in each trace. We can see that our two environments produced very different results. Our system could send at least 100KB to 66% of the devices discovered in the mall when either stationary or mobile. However, it could send the same amount of data to only 15% of devices discovered in the subway. This suggests that a Bluetooth content delivery system intended to deliver tens of KBytes (e.g., a RSS feed delivery system[2]) could be successful when placed in a mall but not in the subway.

We also examined the data-rates of the Bluetooth connections in all four traces. We divided each flow into one second intervals, computed the data-rate for each interval, and averaged these numbers to measure the average flow data-rate. Figure 13 presents the results. On average, the data-rates inside the mall were more than a factor of two higher than those in the subway. These findings suggest that the subway was a much worse delivery medium for Bluetooth than the mall. Unlike the mall, it is crowded and a tight space. We further hypothesize that our server did not have line of sight to most of the clients discovered in the subway, unlike in the mall.

[2]100KB of data is sufficient to send about 17 RSS feed updates [22].

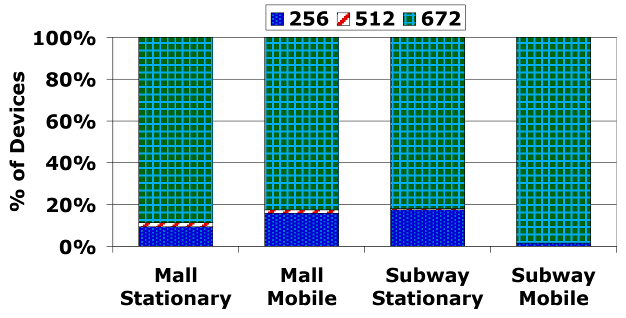


Figure 11: ***Breakdown of devices by their SDP MTU: While more than 80% of devices have an MTU of 672 bytes, 2 to 18% of devices have a much smaller MTU of only 256 bytes.***

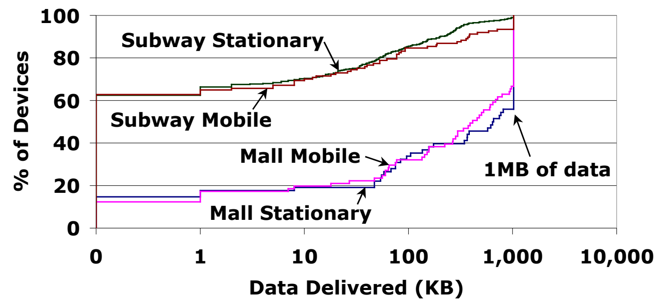


Figure 12: ***The amount of data delivered by our content delivery system prototype: Cumulative distribution function of the number of KBytes sent to the discovered devices for each of the four BlueMonarch traces.***

7.3 Summary

Our BlueMonarch-based evaluation shed light on the performance of a content delivery system for Bluetooth. We found that:

1. A Bluetooth content delivery server can deliver tens of megabytes of data to tens to hundreds of people in just over an hour. To a single user, such a system can deliver tens to hundreds of KBytes, making it very difficult to distribute large-sized data, such as songs.

2. The server's location is hugely important to the needs of the system. For example, for delivering RSS feeds, the server should be placed inside of a mall rather than in a subway system. In contrast, for an event notification system that needs to deliver very little data, it is more important to increase the number of people reached by our system; in such a case, a subway system is a better venue than a mall.

3. The client population of such a system has very heterogeneous hardware and software characteristics. Client devices are made by many different manufacturers with different Bluetooth software settings, such as different MTUs. A content delivery system therefore needs to be robust in the face of this large degree of heterogeneity.

8. RELATED WORK

The first part of this section describes related work pertaining to the development of measurement techniques similar to those used in BlueMonarch. The second part describes prior work on developing Bluetooth applications.

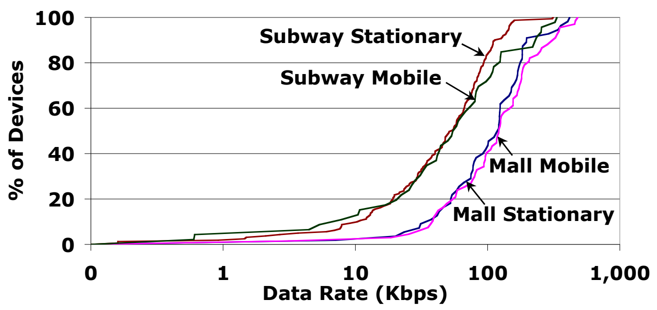


Figure 13: ***The average data-rate of the Bluetooth flows in each of the four traces: To measure the average data-rate of each flow we divide the flow in one second intervals, we compute the data-rate for each interval, and we average these numbers.***

8.1 Measurement Techniques

BlueMonarch's measurement techniques are inspired by those used in Monarch, a tool for emulating TCP transfers to Internet hosts without the need for their cooperation [14]. Monarch uses large TCP probes, such as TCP ACK packets, that are deliberately enlarged by appending a dummy payload. These probes often elicit TCP RST responses. By sending large packets to a remote host that are answered with short TCP RST packets, Monarch accurately emulates a TCP flow that transfers data to a remote host.

Other Internet measurement tools use existing protocols in unanticipated ways to perform measurements that were previously intractable. Sting [32] manipulates the TCP protocol to measure packet loss. T-BIT [27, 25] exploits the TCP protocol to characterize Web servers' TCP behavior. King [13] uses DNS queries to measure latencies between two arbitrary DNS servers. SProbe [31] sends packet pairs of TCP SYN packets to measure bottleneck bandwidth to uncooperative Internet hosts. Like BlueMonarch, these tools send carefully crafted packet probes to remote hosts to measure network properties.

Several previous measurement studies gathered traces to study Bluetooth devices' mobility patterns and interactions [11, 34]. BlueMonarch could augment this work by providing more in-depth measurements of path properties to these devices.

8.2 Bluetooth Applications

Bluetooth location-aware applications make use of positional cues to adapt their functionality based on users' locations. Existing examples of Bluetooth location-aware applications developed by both research and industry include advertising systems [1, 12, 17, 5, 2], games [15, 30], and dating services [19]. BlueMonarch could help these application designers evaluate their applications in the wild.

Another class of previous work has incorporated Bluetooth to enhanced the accuracy of localization algorithms because of its short range. For example, [24] evaluated a fine-grained Bluetooth localization system and found it to be accurate to within 3 centimeters 95% of the time. More recently, [4] designed a localization system that uses Bluetooth in addition to other wireless interfaces, such as Wi-Fi and 3G. As before, BlueMonarch could help in evaluating these Bluetooth-based localization schemes. For example, these systems could use BlueMonarch to shed light on the Bluetooth network environment for devices participating in localization.

9. CONCLUSIONS

This paper presents BlueMonarch, a system for evaluating last meter applications running over Bluetooth in the wild. BlueMonarch offers a key abstraction: the ability to emulate a Bluetooth transfer to any devices that respond to Bluetooth inquiries. BlueMonarch is highly accurate: our experiments show that its transfers match regular Bluetooth transfers with respect to the number of packets exchanged, their sizes, transfer rates, and behavior when operating in a multi-device piconet.

With BlueMonarch, we prototyped a content delivery system for Bluetooth. With a single laptop equipped with four Bluetooth radios, we transmitted tens of megabytes of data to hundreds of Bluetooth devices in just over an hour. Our evaluation also shed light on a number of previously open issues regarding Bluetooth content delivery capabilities and requirements.

As a final note, we hope that BlueMonarch's future users will carefully construct their studies in a way that minimizes any side-effects arising from its use. Our safeguards (presented in Section 5.4) worked well for our limited deployment of BlueMonarch. For larger deployments, we recommend consulting organizations that handle the ethical issues arising from performing research experiments, such as an academic ethical review board.

Acknowledgments

We thank Victor Bahl, Nigel Davies (our shepherd), and the blind reviewers for their insightful comments on our work. BlueMonarch's measurement technique is inspired by our previous work on Monarch together with Krishna Gummadi, Andreas Haberlen, and Marcel Dischinger.

10. REFERENCES

[1] L. Aalto, N. Göthlin, J. Korhonen, and T. Ojala. Bluetooth and WAP push based location-aware mobile advertising system. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Boston, MA, June 2004.

[2] Ad-Pods. Bluetooth marketing, 2008. `http://www.a-p-marketing.co.uk/`.

[3] C. Baber and O. Westmancott. Social networks and mobile games: The use of bluetooth for a multiplayer card game. In *Proceedings of Mobile HCI*, Glasgow, UK, 2004.

[4] N. Banerjee, S. Agarwal, V. Bahl, R. Chandra, A. Wolman, and M. Corner. Virtual compass: Relative positioning to sense mobile social interactions. In *Under Submission*, 2009.

[5] bloozy. Bluetooth advertising, 2008. `http://www.bluetoothadvertising.com/`.

[6] Bluetooth. Specification of the Bluetooth System, 2006. `http://www.bluetooth.org/foundry/adopters/document/Core_v2.0_EDR/en/1/Core_v2.0_EDR.zip`.

[7] Bluetooth SIG. Bluetooth wireless technology surpasses one billion devices, 2006. `http://bluetooth.com/Bluetooth/Press/SIG/BLUETOOTH_WIRELESS_TECHNOLOGY_SURPASSES_ONE_BILLION_DEVICES.htm`.

[8] BlueZ. Official bluetooth linux protocol stack, 2008. `http://www.bluez.org/`.

[9] coffer.com. Vendor/ethernet/bluetooth mac address lookup and search, 2008. `http://coffer.com/mac_find`.

[10] Collin Richard Mulliner. sobexsrv - scripting/secure obex server (for bluez linux), 2008. `http://www.mulliner.org/bluetooth/sobexsrv.php`.

[11] N. Eagle and A. S. Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4):255–268, 2006.

[12] Filter UK Ltd. Bluecasting: The proximity marketing system, 2006. `http://www.bluecasting.com`.

[13] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Internet Measurement Workshop (IMW)*, Marseille, France, November 2002.

[14] A. Haeberlen, M. Dischinger, K. P. Gummadi, and S. Saroiu. Monarch: A tool to emulate transport protocol flows over the internet at large. In *Proceedings of the Internet Measurement Conference (IMC)*, Rio de Janeiro, Brazil, 2006.

[15] M. Handy, F. Golatowski, and D. Timmermann. Lessons learned from developing a bluetooth multiplayer-game. In *Proceedings of the Workshop on Gaming Applications in Pervasive Computing Environments*, Linz/Vienna, Austria, 2004.

[16] M. Holtmann. Bluetooth L2CAP device specific information, 2008. `http://www.holtmann.org/linux/bluetooth/l2cap.html`.

[17] Hypertags. Hypertags, 2008. `http://www.hypertag.com/`.

[18] S. Jung, U. Lee, A. Chang, D.-K. Cho, and M. Gerla. Bluetorrent: Cooperative content sharing for bluetooth users. *Pervasive and Mobile Computing*, 3(6):609–634, 2007.

[19] Kangourouge. Proxidating, the first ever Bluetooth dating software for mobile phones, 2008. `http://www.proxidating.com`.

[20] J. LeBrun and C.-N. Chuah. Bluetooth content distribution stations on public transit. In *Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare)*, Los Angeles, CA, 2006.

[21] D. J. Y. Lee and W. C. Y. Lee. Ricocheting bluetooth. In *Proceedings of the 2nd International Conference on Microwave and Millimeter Wave Technology*, Beijing, China, 2000.

[22] H. Liu, V. Ramasubramanian, and E. G. Sirer. A measurement study of rss, a publish-subscribe system for web micronews. In *Proceedings of the Internet Measurement Conference (IMC)*, Berkeley, CA, 2005.

[23] B. T. Loo, A. LaMarca, and G. Borriello. Peer-to-peer backup for personal area networks. *Intel Research Technical Report IRS-TR-02-015*, October 2002.

[24] A. Madhavapeddy and A. Tse. A study of bluetooth propagation using accurate indoor location mapping. In *Ubicomp*, pages 105–122, 2005.

[25] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet. *Computer Communication Review*, 35(2):37–52, 2005.

[26] C. Mulliner. btChat. `http://www.mulliner.org/bluetooth/btchat/`.

[27] J. Padhye and S. Floyd. Identifying the TCP behavior of web servers. In *Proceedings of the ACM SIGCOMM Conference*, San Diego, CA, USA, June 2001.

[28] R. Patra, S. Nedevschi, S. Surama, A. Sheth, L. Subramanian, and E. Brewer. Wildnet: Design and implementation of high performance wifi based long distance networks. In *In Proc. of USENIX NSDI*, pages 87–100, 2007.

[29] B. Prabhu and A. Chockalingam. A routing protocol and energy efficient techniques in bluetooth scatternets. *Communications, 2002. ICC 2002. IEEE International Conference on*, 5:3336–3340 vol.5, 2002.

[30] H. Ritter, T. Voigt, M. Tian, and J. Schiller. Experiences using a dual wireless technology infrastructure to support ad-hoc multiplayer games. In *Proceedings of the 2nd workshop on Network and system support for games (NetGames)*, Redwood City, CA, 2003.

[31] S. Saroiu, K. P. Gummadi, and S. D. Gribble. SProbe: A fast tool for measuring bottleneck bandwidth in uncooperative environments, 2002. `http://sprobe.cs.washington.edu`.

[32] S. Savage. Sting: a TCP-based network measurement tool. In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, USA, October 1999.

[33] J. Su, K. K. W. Chan, A. G. Miklas, K. Po, A. Akhavan,

S. Saroiu, E. de Lara, and A. Goel. A preliminary investigation of worm infections in a bluetooth environment. In *4th Workshop of Recurring Malcode (WORM)*, Fairfax, VA, 2006.

[34] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton. Haggle: Seamless networking for mobile applications. In *Ubicomp*, pages 391–408, 2007.

[35] S. Surama, R. Patra, S. Nedevschi, M. Ramos, L. Subramanian, Y. Ben-David, and E. Brewer. Beyond pilots: Keeping rural wireless networks alive. In *In Proc. of USENIX NSDI*, pages 119–132, 2008.

[36] F. Tan, S. Ardon, and M. Ott. Ubistore: Ubiquitous and opportunistic backup architecture. In *Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, White Plains, NY, 2007.

[37] Techworld. Why Bluetooth version 2 matters, 2005. `http://www.techworld.com/mobility/features/index.cfm?featureid=1198`.

[38] The Register. Bluetooth to Outship Wi-Fi Five to One, 2003. `http://www.theregister.co.uk/2003/06/18/bluetooth_to_outship_wifi_five/`.