BlueStreaming: Towards Power-Efficient Internet P2P Streaming to Mobile Devices[*]

Yao Liu[1] Fei Li[1] Lei Guo[2] Yang Guo[3] Songqing Chen[1]
[1]Department of Computer Science, George Mason University, {yliud, lifei, sqchen}@cs.gmu.edu
[2]Microsoft Corporation, Mountain View, CA, USA, leguo@microsoft.com
[3]Bell Labs, Alcatel-Lucent, yang.guo@alcatel-lucent.com

ABSTRACT

P2P streaming applications are very popular on the Internet today. However, a mobile device in P2P streaming not only needs to continuously receive streaming data from other peers for its playback, but also needs to continuously exchange control information (e.g., buffermaps and file chunk requests) with neighboring peers and upload the downloaded streaming data to them. These lead to excessive battery power consumption on the mobile device.

In this paper, we first conduct Internet experiments to study in-depth the impact of control traffic and uploading traffic on battery power consumption with several popular Internet P2P streaming applications. Motivated by measurement results, we design and implement a system called BlueStreaming that effectively utilizes the commonly existing Bluetooth interface on mobile devices. Instead of activating WiFi and Bluetooth interfaces alternatively, BlueStreaming keeps Bluetooth active all the time to transmit delay-sensitive control traffic while using WiFi for streaming data traffic. BlueStreaming trades Bluetooth's power consumption for much more significant energy saving from shaped WiFi traffic. To evaluate the performance of BlueStreaming, we have implemented prototypes on both Windows and Mac to access existing popular Internet P2P streaming services. The experimental results show that BlueStreaming can save up to 46% battery power compared to the commodity PSM scheme.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Measurement, Design, Experimentation

Keywords

P2P, Internet Mobile Streaming, Bluetooth, Power Saving

1. INTRODUCTION

Internet Peer-to-Peer (P2P) streaming applications are very popular today. For example, PPTV [1], PPStream [2], SopCast [3], and QQLive [4] are all Internet-scale P2P streaming systems that provide hundreds of TV channels and serve millions of Internet users every day. As a result, CNN has started to use P2P technology provided by Octoshape to deliver high quality live video to its users [5]. According to Cisco's report [6], today, global P2P TV systems have generated over 280 petabytes (and 6% of) Internet video traffic per month, and is growing at a rate of 47% annually.

In tandem with the fast-growing Internet streaming services, mobile devices are pervasively used today. For example, it has been reported that iPad users generated 2.5 times more Internet traffic than iPhone users for accessing popular services such as USAToday, Google Maps, Bloomberg, eBay after its release [7]. To support mobile Internet accesses, 802.11 and the third generation (3G) wireless networks are also quickly developing. For example, WiFi operates in more than 595,000 public hotspots [8]. Jupiter Research also estimated that 65% households in the United States have home-deployed WiFi access points (APs) [9].

However, using mobile devices to receive Internet streaming services can quickly exhaust the limited battery supply because receiving Internet streaming services often involve continuous and bulk data transmissions, decoding, rendering, and displaying. Among them, wireless data transmissions consume a significant amount of energy. For example, it was reported that the WiFi radio on HTC Tilt 8900 series consumes up to 5x more power than the base energy consumption of the phone [10]. A lot of efforts have been made to save battery power consumed by the wireless network interface cards (WNICs) by exploiting sleep opportunities via proxying [11], batching [12], PSM-throttling [13], etc. However, existing power saving strategies mainly focus on client/server (C/S) based streaming applications. Compared to C/S based streaming, Internet P2P streaming can further aggravate the battery power consumption on mobile devices due to the following unique characteristics.

First, different from C/S based streaming, a client participating in P2P streaming needs to continuously exchange extra control traffic, such as buffermaps and fine-grained data chunk requests, with neighbors. Such control traffic, on the one hand, greatly facilitates the streaming content distribution in a scalable manner. On the other hand, it also remarkably increases the total number of IP packets transmitted and reduces the inter-packet delays, which further reduces the sleep opportunities of the WNIC.

Second, critical to the scalability of a P2P system, a client participating in P2P streaming has to upload the downloaded content to other neighboring peers. Such uploading requirement does not exist in C/S based streaming. The uploading traffic increases the total traffic volume for a P2P client, resulting in more battery power consumption.

Third, a client participating in P2P streaming receives streaming content from multiple sources instead of a single server. Further-

[*]Area Chair: Surender Chandra

more, these streaming sources are changing dynamically along time since a P2P client often needs to frequently connect to new neighbors to download missing file chunks. In addition, the dynamics of the streaming sources result in different round-trip times to the client, making data receiving at the client side totally random, and making it difficult for the WNIC to switch into the power saving mode (PSM).

In this paper, we first conduct Internet measurements to study in-depth the impact of Internet P2P streaming on battery consumption on mobile devices with several popular Internet P2P streaming applications. Our measurement results show that: (1) a client in P2P streaming needs to transmit an extremely large number of small packets for control traffic (e.g., twice as many as streaming data packets). Such a large number of frequent and delay-sensitive control packets not only increase the traffic volume, but also significantly reduce the inter-packet delay and the potential sleep time of the WNIC on the mobile device; (2) the amount of uploading traffic a mobile client needs to transmit to its neighbors changes from time to time and from application to application, ranging from 10 Kbps to over 1.5 Mbps. In addition to increasing the transmission load on the WNIC, such highly un-predictable uploading further reduces the inter-packet delay and the sleep opportunities of the WNIC for power saving; and (3) a client in P2P streaming receives packets from dynamically changing sources (e.g., 3 to 20 peers), making it difficult to employ server or client-centric traffic shaping techniques to save power consumption on mobile devices.

Motivated by the measurement results, we seek to address these challenges for power-efficient Internet mobile P2P streaming services. Accordingly, we propose and design the BlueStreaming system by leveraging the commonly existing Bluetooth interface on mobile devices. In BlueStreaming, instead of having WiFi and Bluetooth interfaces to work alternatively, we always keep Bluetooth active to transmit delay-sensitive control traffic. With intelligent traffic shaping techniques applied on the WiFi interface for data traffic, the extra battery power consumed by the Bluetooth interface can be over-compensated by the greater power saving on the WiFi interface. Furthermore, the uploading traffic from the client is opportunistically scheduled on both interfaces to minimize battery power consumption incurred.

To evaluate the performance of BlueStreaming, we have implemented prototypes on both Windows and Mac OS to participate in several popular Internet P2P streaming services. Our experimental results show that BlueStreaming can effectively reduce battery power consumption by up to 46% without affecting streaming quality. In summary, this paper makes the following contributions:

- Through Internet measurement and analysis, we show that when a mobile device participates in Internet P2P streaming services, its battery power consumption is highly affected by its control traffic, uploading traffic, and the dynamics of neighboring peers.
- We design and implement BlueStreaming, a power-efficient Internet P2P streaming system that takes into account the unique characteristics of Internet P2P streaming and trades Bluetooth's power consumption for greater power saving on the WiFi interface via intelligent traffic shaping.
- We evaluate our BlueStreaming prototype with popular Internet P2P streaming applications in both infrastructure and hybrid modes, and show that BlueStreaming can improve energy saving by up to 46% compared to the commodity PSM scheme.

The remainder of the paper is organized as follows. Section 2 describes some related work. We present our measurement results in Section 3 and the design of BlueStreaming in Section 4. The implementation is described in Section 5 and the evaluation is discussed in Section 6. We make concluding remarks in Section 7.

2. BACKGROUND AND RELATED WORK

With pervasive wireless Internet accesses, power saving has been considered on the commodity WNIC. Today most WNIC can operate in two power modes: Constantly Awake Mode (CAM) and PSM. It has been reported in [10] that a mobile smartphone in CAM uses as much as 1120 mW power, while it consumes much less power of 72 mW in PSM. In practice, commodity WiFi devices often use adaptive PSM (PSM-A) instead of static PSM. PSM-A switches between CAM and PSM modes. If there is no network activity (idle) for a pre-determined time period, called *idle timeout interval*, the WiFi interface would switch to PSM by sending a NULL data frame with the `Pwr Mgt` field set to 1. It only wakes up to check the Traffic Indication Map (TIM) per beacon interval (e.g., 100 ms). The WiFi interface can notify AP that it is ready to receive buffered data by sending another NULL data frame with the `Pwr Mgt` field set to 0. To further improve the energy efficiency, a lot of prior work has sought to exploit idle opportunities for the WNIC. For example, μPM [14] allows the WiFi interface to exploit short intervals in terms of microseconds between MAC frames to enter the low power mode. Catnap [12], on the other hand, exploits large sleep intervals by scheduling TCP transfers at the granularity of application data units with the help of workload hints.

Meanwhile, since mobile devices commonly have equipped with multiple network interfaces that have different power consumption rates, research has been conducted on alternatively activating different interfaces for battery power saving, focusing on substituting a high-power consumption interface with a low-power consumption interface whenever possible. For example, Agarwal et al. proposed to wake up the WiFi interface for incoming calls using GSM radio [15]. In CoolSpots [16], policies were proposed to intelligently switch between the Bluetooth and WiFi interfaces. However, CoolSpots is not appropriate for today's Internet streaming delivery to mobile devices because Bluetooth alone cannot afford the constant bandwidth demanded by modern streaming applications (see Section 4.1). In contrast to CoolSpots that uses the two interfaces alternatively, BlueStreaming uses both WiFi and Bluetooth interfaces simultaneously, and relies on the greater power saving on the WiFi interface to overcompensate the extra power consumption incurred by the Bluetooth interface.

Focusing on Internet streaming to mobile devices, Bertozzi et al. proposed to switch off the WiFi interface during playback and turn it on when running out of streaming buffer [17]. Linear prediction was proposed to select sleep time intervals for the WNIC [18]. PSM-throttling [13] utilizes client-side reshaping of TCP traffic to improve energy saving. Compared to existing work, this study shows that a client in P2P streaming not only generates extra uploading traffic, but also transmits highly frequent and delay-sensitive control traffic, making direct adoption of existing solutions ineffective.

3. MEASUREMENT AND ANALYSIS

To investigate the battery power consumption on mobile devices in popular Internet P2P streaming applications, in this section, we conduct measurements with several existing Internet P2P streaming applications. We then analyze the traffic in-depth in order to study underlying factors that impact the battery power consumption.

3.1 Methodology and Result Overview

To study the battery power consumption on mobile devices in receiving P2P streaming services, we first use a 2nd generation iPod Touch (iTouch) running iOS 3.1.2 to receive streaming services. We conduct experiments with TVUPlayer [19], and Justin.tv [20]. To the best of our knowledge, TVUPlayer is the only P2P based

Table 1: Summary of Statistics on Laptop

Name	Archi-tecture	Encoding Rate (Kbps)	Total # of IP Packets	Total # of Contro Packets	Total # of Streaming Packets	Sleep Time (%)	Average# of Neighbors
PPTV	P2P	400	191716	116819	70873	4.42	12
PPS	P2P	396	433935	322779	85180	0.09	20
SopCast	P2P	530	305826	202240	92457	0.99	3
QQLive	P2P	500	293474	183814	108827	7.12	4
J.tv	C/S	433	123655	56689	66966	21.44	N/A

streaming application available on iTouch at the time of this measurement, while Justin.tv is a C/S based Internet streaming service.

In these experiments, we set up Wireshark to listen on the same channel to capture all incoming/outgoing packets. Since we log packets at the data-link layer, we are able to get frame control information from IEEE 802.11 headers. We use `Pwr Mgt` flag in the frame control field to determine the Power Management mode that the WiFi interface will switch to after the transmission of the current frame. All experiments are run for 30 minutes and are conducted with a dedicated AP running 802.11g in order to minimize traffic contention.

We set both TVUPlayer and Justin.tv to watch channels with the same streaming rate (281 Kbps), and examine the potential power consumed by the WiFi interface. Our results show that during 30 minutes test, C/S based Justin.tv allows the WNIC to sleep for 83% time, while in P2P based TVUPlayer, the WNIC can only sleep for 26% time, meaning 3 times more power consumption in receiving P2P streaming services. Given that iOS uses an extremely aggressive PSM-A policy (idle timeout interval is only about 20-25 ms [10]), the energy performance of TVUPlayer on other mobile devices could be even worse.

Although the above results provide an overview of the power inefficiency P2P streaming services for mobile devices, TVUPlayer is not as widely used as other P2P based Internet streaming services such as PPTV (PPLive is renamed as PPTV), PPStream (PPS), SopCast and QQLive, each of which serves millions of Internet users daily. To study in-depth the battery power consumption of mobile devices in these applications, we further conduct experiments with a laptop computer since these applications do not have a version for mobile devices. We again compare their results against C/S based Justin.tv (J.tv) [21]. In these experiments, the laptop computer connects to a dedicated AP running 802.11n and watches different channels of these popular applications. The laptop runs Windows 7 with `Maximum WiFi Power Saving` enabled. Again, Wireshark is used to capture all incoming and outgoing packets to/from the laptop at the data-link layer. Because each peer in P2P streaming is required to upload to its neighbors and such uploading throughput can be highly un-predictable (we will show later in Section 3.3), we thus conduct experiments at various times of a day and first present the results of tests that involve the least amount of uploading traffic. This makes the comparison against C/S based J.tv more meaningful because otherwise the additional high uploading from a mobile device undoubtedly increases the battery power consumption. Note that these tests are not necessarily accessing unpopular channels, since uploading of a peer is also affected by a number of other factors.

Table 1 summarizes some raw data based on the traces we have collected in the experiments. Note that *Total # of Streaming Packets* refers to the downloaded streaming data packets only. Although the video encoding rates are different across four P2P applications, it is not necessarily correlated to the amount of time that the WiFi interface has spent in the PSM: both PPTV and PPS stream at similar rates, but they have different power efficiency; SopCast allows the WiFi interface to sleep for less than 1% of the session time with a streaming rate of 530 Kbps, but at a similar rate of 500 Kbps in QQLive, the WiFi interface operates in the PSM for 7% of the session time, and is the most power efficient.

Nevertheless, compared to C/S based J.tv in which the WiFi operates in the PSM for 21% of the session time, none of these four popular P2P streaming applications is power efficient. Note that all above experiments involve a small amount of uploading traffic. If the laptop has to upload more to neighboring peers, the power efficiency of P2P streaming applications would be even worse (we will show soon in Section 3.3).

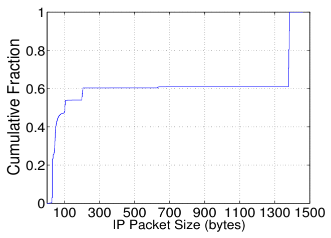
While the sleep time varies in different P2P streaming applications, above results on both the mobile device and the laptop show that compared to C/S based streaming, a mobile client in existing P2P streaming applications is much less power efficient. As aforementioned, this is likely due to the uploading and control traffic. To quantify their impact, next we further analyze the incoming and outgoing traffic to/from our testing client.

3.2 Impact of Control Traffic

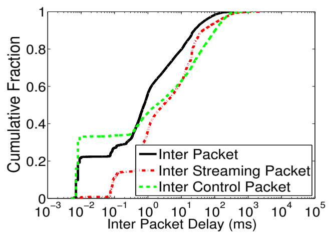
Figures 1, 2, 3, and 4 show more detailed traffic analysis in the above experiments for PPTV, PPS, SopCast, and QQLive, respectively. Figure 1(a) shows the packet size distribution of PPTV. As indicated by this figure, the packet size distribution is highly bimodal. Via reverse-engineering, we find that only about 37% packets are streaming packets with the size around 1408 bytes while about 60% packets have a size less than 300 bytes. Similarly, Figures 2(a), 3(a), and 4(a) show that there are more than 74%, 66%, and 62% small packets in the streaming sessions of PPS, SopCast, and QQLive, respectively.

Recall that existing P2P streaming systems mainly use a pull based mesh structure, in which neighboring peers need to frequently exchange control information. For example, peers exchange `buffermaps` to determine data chunk availability among neighbors. Peers also need to send out `file chunk requests` to retrieve missing streaming data chunks. In addition, PPTV, PPS, Sopcast, and QQLive all use UDP to transmit packets. Thus, these small packets can only be control traffic of the P2P protocol. Via reverse-engineering, it is confirmed that these packets include both buffermaps exchanged with neighbors and file chunk requests sent to neighbors.

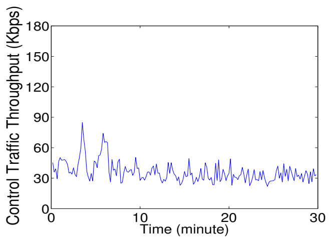
Table 1 shows that in all these P2P streaming applications, there are more control packets (the 5th column) than streaming data packets (the 6th column). PPS even generated about 3 times more control packets than data packets. Regardless of the total traffic amount of these control packets, such a large number of control packets can significantly reduce the the inter-packet delay and thus idle time of the WiFi interface. This would result in more battery power consumption. Figures 1(b), 2(b), 3(b), and 4(b) show the inter packet delay distribution in these experiments. In these figures, `Inter Streaming Packet` delay considers only ingress streaming data packets. `Inter Control Packet` delay considers both incoming and outgoing control packets, and `Inter`



(a) Packet Size (CDF)

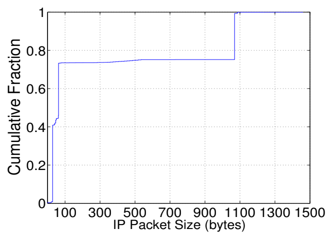


(b) Inter Packet Delay (CDF)

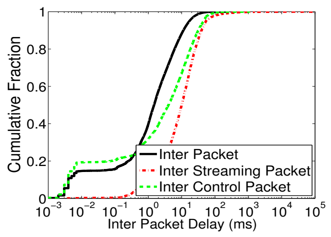


(c) Control Traffic Throughput

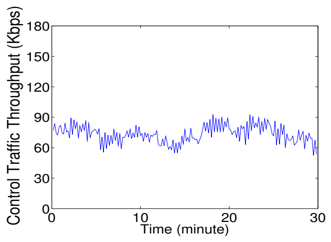
Figure 1: PPTV (400Kbps)



(a) Packet Size (CDF)

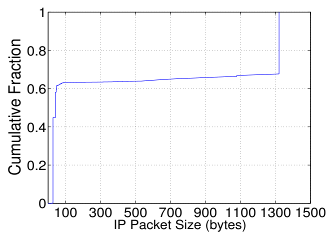


(b) Inter Packet Delay (CDF)

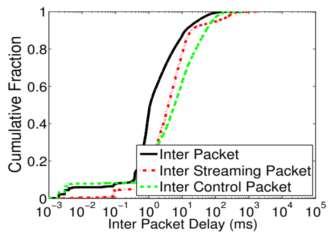


(c) Control Traffic Throughput

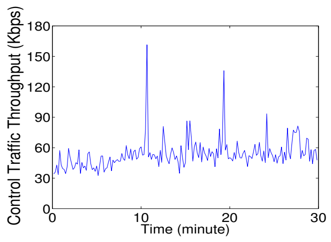
Figure 2: PPS (396Kbps)



(a) Packet Size (CDF)

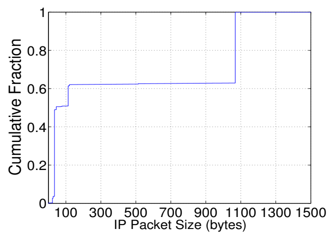


(b) Inter Packet Delay (CDF)

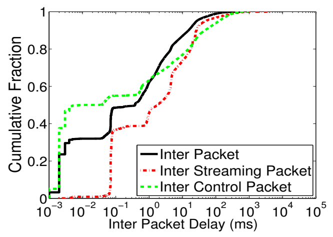


(c) Control Traffic Throughput

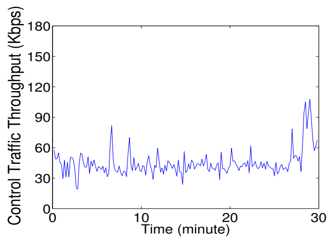
Figure 3: SopCast (530Kbps)



(a) Packet Size (CDF)



(b) Inter Packet Delay (CDF)



(c) Control Traffic Throughput

Figure 4: QQLive (500Kbps)

Table 3: Comparison of Sleep Time (%)

	PPTV	PPS	SopCast	QQLive
MAX	4.42	0.09	0.99	7.12
MIN	0.08	0.00	0.22	4.33

Packet delay takes all packets transmitted into account. As shown in these figures, while 10% and 3% successive streaming packets of PPTV and PPS arrive with a delay larger than 100 ms (meaning opportunities for power saving), about 60% control packets in PPTV and 42% in PPS are sent/received within 1 ms from the preceding one. This causes the overall Inter Packet delay to significantly deviate from Inter Streaming Packet delay patterns.

Although control packets are much more frequent than streaming data packets, the throughput of control traffic is relatively small. As shown in Figures 1(c), 2(c), 3(c), and 4(c), the throughput of control traffic of the four P2P streaming applications is less than 100 Kbps for most of the time. In these figures, the throughput is averaged over 10 seconds. The relatively small control traffic throughput is mainly due to the small size of control packets. This is a very important feature that we could explore to save power consumption in BlueStreaming in Section 4. Although control traffic may increase with the increase of uploading, typically the control traffic throughput is still much smaller than the streaming data rate.

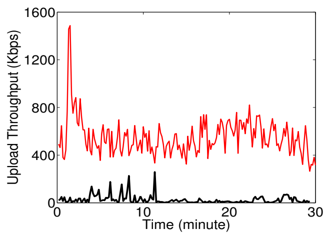
3.3 Impact of Uploading Traffic

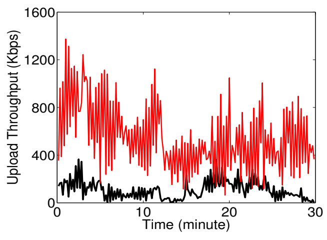
The measurement results we have presented above involve a minimum amount of uploading traffic. In practice, a P2P client may upload more and the uploading throughput may also change from time to time. Figure 5 shows the uploading variations of our client with a minimum uploading average and a maximum uploading average in our over fifty 30-minute tests at various time. As shown in this figure, the throughput of uploading traffic could reach as high as over 1 Mbps in PPTV, PPS, and SopCast. Such a high uploading throughput clearly consumes more battery power. As shown in Table 3, with a larger amount of uploading traffic transmitted, the sleep time (%) of the WiFi interface in all four applications decreases significantly, and in PPS it even could not switch into the PSM at all.

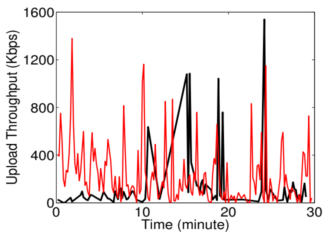
3.4 Impact of the Number of Neighbors

Both control traffic and uploading traffic are affected by the number of neighbors a peer communicates with at a time. A larger number of neighbors not only increase the control information exchanged with different peers and the uploading traffic amount, but also divide large idle time intervals into smaller ones because the response time of different peers to our client varies significantly. During the experiments, we have kept track of the number of peers that our streaming client downloaded data from. As shown in the last column of Table 1, during the 30-minute test, our client in QQLive exchanges streaming data with 4 neighboring peers on average every 10 seconds, while it involves 12 neighbors on average in PPTV.

Summary of Measurement Results: Our study shows that in Internet P2P streaming (1) although of small throughput, the control traffic is highly frequent, which remarkably reduces the potential sleep time of the WiFi interface; (2) the uploading required from a peer changes dynamically and could reach a very high throughput, which directly affects the battery power consumption and worsens the client side traffic pattern; (3) the dynamics of peers' neighbors directly affect both control and uploading traffic, and the variance


(a) PPTV


(b) PPS


(c) SopCast

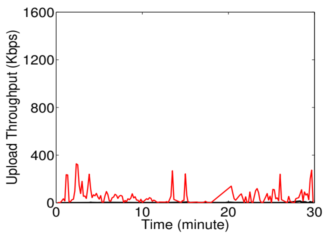

(d) QQLive

Figure 5: Variation of Upload Throughput (averaged over 10 seconds)

Table 2: Simple Strategies Do Not Work Well

Name	Total # of Streaming	Sleep Time (%)	Distortion/Freeze Time (%)
PSM-A	109800	5.24	0
Bluetooth only	37228	N/A	96
WiFi with traffic shaping	121598	26.89	38

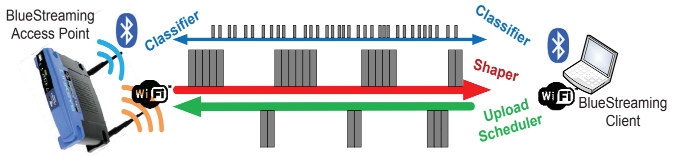


Figure 6: Overview of BlueStreaming Architecture

of response time among neighbors further shortens inter-packet delay.

4. DESIGN OF BLUESTREAMING

As discussed in the last section, P2P streaming differs from traditional client-server based streaming on the additional control traffic, uploading traffic, and data receiving from multiple peers. These can lead to excessive battery power consumption on mobile devices.

4.1 Motivation

To save power consumed for data transmission, an intuitive solution is to always switch to a low power interface, e.g., Bluetooth connection when available. Another solution is to use WiFi interface only, but aggressively shapes the traffic by brutally buffering all packets for long enough and transmit them in a very large MAC frame periodically.

We implement both of these schemes. Table 2 shows the results when accessing a QQLive channel of 500 Kbps for 30 minutes. `Bluetooth only` saves energy consumption by only using Bluetooth for all transmissions. However, Bluetooth alone cannot afford the constant bandwidth demanded by the application. As a result, Table 2 shows that the client with `Bluetooth only` receives about only 34% streaming data packets compared to PSM-A. This causes the playback to be frozen all the time during the entire streaming session.

On the other hand, in WiFi with traffic shaping, the control packets are also buffered and sent in burst periodically via WiFi. Table 2 shows that such an approach can make the WiFi interface to sleep for 26% of the session time. However, the delayed control packets have caused the client to receive 10% more streaming packets, indicating the delay-sensitivity of control traffic. Moreover, by analyzing the captured frames of the playback and comparing with those captured in PSM-A (since we do not have access to the original video), we find that quality degradation (playback freezing or distortion) happens for a total of 684 seconds, which is 38% of the entire streaming session.

The above results demonstrate that these intuitive strategies using either interface may cause problems in both user perceived quality and the efficiency of transmission. We next present our design of BlueStreaming to intelligently utilize both WiFi and commonly existing Bluetooth interfaces on mobile devices for Internet P2P streaming accesses.

4.2 BlueStreaming Overview

To efficiently deal with control traffic in P2P streaming, BlueStreaming leverages the Bluetooth interface on a mobile device to transmit control traffic. For this purpose, BlueStreaming always classifies the incoming and outgoing traffic to/from a client into two classes: ingress/egress control traffic and ingress/egress data traffic. BlueStreaming uses Bluetooth to transmit both ingress and egress control traffic. In addition, since uploading is inevitable and Bluetooth is always active in BlueStreaming, Bluetooth also uploads to neighbors whenever there is available bandwidth. The WiFi interface is mainly used for streaming data downloading, as well as uploading opportunistically upon excessive uploading.

Figure 6 depicts the architecture of BlueStreaming. Logically, BlueStreaming consists of three main components: the traffic classifier at both the AP and the client sides to decouple control traffic from streaming data traffic, the traffic shaper at the client side for shaping data traffic to save battery power, and the uploading scheduler at the client side to handle uploading traffic. The successful design of these three components, however, needs to address the following challenges:

- *traffic classifier*: given that in most P2P streaming, control traffic and data traffic use the same channel (port number), how can control traffic be decoupled from data traffic transparently (transparent to the application)?
- *traffic shaper*: since most existing P2P streaming traffic is delivered over UDP and a client often dynamically communicates with a set of different peers at a time, how shall BlueStreaming shape the downstream traffic to a client?
- *uploading scheduler*: because uploading from a peer is necessary and critical to the scalability of a P2P system, how does a client perform uploading while minimizing the corresponding battery power consumption?

4.3 Traffic Classifier: Diverting Control Traffic to Bluetooth

Since Bluetooth operates with about 20% power compared to WiFi in an active state, with BlueStreaming, a mobile device seeks to use Bluetooth to transmit the large number of small delay-sensitive control packets in order to put the WiFi interface into sleep mode for a longer time. This requires Bluetooth to provide sufficient bandwidth to transmit control traffic. Commodity Bluetooth interfaces today typically offer a low data rate of 1 Mbps or so. Although we have shown in Section 3 that control traffic may fluctuate from time to time depending on a number of factors such as the number of neighbors, the control traffic rate is often very low. Using Bluetooth to transmit control traffic also needs to consider the time sensitivity of control packets. A previous study shows that the delay of the control traffic would cause repetitive requests of the same data chunks [22]. Thus, to ensure timely delivery of control packets, in BlueStreaming, the Bluetooth interface stays active all the time so that control packets are transmitted without any delay.

To transmit control packets via Bluetooth, control traffic must be separated from data traffic at runtime. In BlueStreaming, the traffic classifier is responsible for decoupling control packets from data packets. Since there are both incoming and outgoing control packets, the traffic classifier works at both the AP and the client sides. Because (1) the current design of BlueStreaming aims to be application transparent, (2) both control traffic and data traffic in typical P2P streaming services use a same port, and (3) the classification must be real time, BlueStreaming has to avoid deep packet inspection for any application specific information.

Thus, in BlueStreaming, we leverage a feature that we have found via our measurement for runtime packet decoupling. In Section 3, Figures 1(a), 2(a), 3(a), and 4(a) show the packet size distribution. As shown in these figures, the control packets are of a much smaller size compared to that of streaming data packets. Thus, in BlueStreaming, we leverage the packet size to differentiate different types of packets. Although the classification is based on our observation in the measurements, we have further conducted measurements with these P2P streaming services for prolonged time in order to validate its accuracy.

Once the control traffic is decoupled, it is easy to divert to the Bluetooth interface. Upon packet arrival, merging received packets from different interfaces is straightforward since these P2P streaming applications use UDP.

4.4 Traffic Shaper: Shaping Ingress Streaming Traffic

After all control packets are diverted to the Bluetooth channel, only data traffic is transmitted through WiFi. Thus, more energy could be saved by exploiting the idle time of the WiFi interface.

Figures 1(b), 2(b), 3(b), and 4(b) show the inter-packet delays. As shown in these figures, although the Inter Packet delay distribution of WiFi (without control traffic) is similar to the distribution of Inter Streaming Packet delay, the increase of sleep opportunity due to control traffic elimination is still small. This is likely due to the multiple uploading peers instead of only a single server. Multiple uploading peers have different response time to this client, resulting in different ingress streaming packet arrival time as observed in our measurement.

Thus, after diverting control traffic, BlueStreaming still needs to handle incoming traffic from different neighboring peers. A natural scheme is to do traffic shaping at the AP. Since BlueStreaming mainly deals with UDP based streaming, the traditional TCP based traffic shaping in the client-server based architecture, such as PSM-throttling, does not work. In addition, due to the QoS requirements of Internet streaming services, such streaming packets cannot be buffered for too long.

To this end, in the traffic shaper, BlueStreaming utilizes the buffer maintained by the WiFi AP: the streaming packets arrived over time are combined into a single MAC frame burst. As a result, the interval between bursts is significantly enlarged and the WiFi interface can exploit these opportunities to switch into the PSM.

To minimize the change to AP, the traffic shaper in BlueStreaming works by not sending the NULL or PS-Poll frame even when the TIM indicates that there are buffered frames for the client. This forces the AP to continue to buffer frames destinated to this client. A subtle issue here is to determine when to wake up and retrieve buffered data. In BlueStreaming, we set it based on: (1) streaming bitrate, (2) effective WiFi bandwidth, and (3) QoS requirements. The first two are easy to determine. The third one, however, is highly application dependent. On the one hand, it impacts the client perceived streaming quality directly, although the playout buffer can absorb most of such delay. On the other hand, P2P applications usually use a timer to determine if any data chunk needs to be re-requested. For certain chunks that are not received in time, they will either be forfeited if their deadline has already passed or they will be requested again [22]. Hence, if streaming packets are buffered at the AP for too long, the application would believe such packets are lost, and request them again.

On the other hand, if the WiFi interface keeps staying in the sleep mode without responding to AP's beacon messages, the AP would believe that the client is no longer associated. After a pre-determined timeout interval, the AP would start dropping its buffered frames. Thus, how long the data packets should be buffered is critical, which directly determines the sleep duration of the WiFi interface.

Taking all the above considerations into account, we set the packet buffering duration as follows:

$$T_{buf} = min\Big(T_{re-req} \times (1 - \frac{Rate}{BW_{WiFi}}), T_{AP-timeout}\Big) \quad (1)$$

where T_{re-req} is the re-request timer of P2P applications, $Rate$ is the bitrate of the streaming channel, BW_{WiFi} is the estimated bandwidth of the WiFi link, and $T_{AP-timeout}$ is the timeout duration after that the AP starts to drop buffered data for the client.

4.5 Uploading Scheduler: Scheduling Upload Wisely

In P2P streaming, peers should upload data to neighbors. This is the key to the scalability of any P2P application. But such uploading brings new complexities. First, uploading to other peers of the downloaded data directly incurs the consumption of the uploading bandwidth and battery power. Second, uploading incurs a comparable number of control packets. Third, uploading is very dynamic and hard to predict in P2P streaming since it depends on how many peers the client is serving at a time. As a result, uploading may seriously offset our energy saving efforts.

Aiming to solve these issues efficiently so that the minimum battery power is consumed for such uploading, we design an uploading scheduler working at two levels: *Priority-based Bluetooth Uploading* and *Opportunistic WiFi Uploading*. In short, BlueStreaming tends to fully utilize the always-on Bluetooth first, and then opportunistically utilize WiFi for uploading if needed.

4.5.1 Priority-based Bluetooth Uploading

In the previous measurement, we have shown that control traffic, although resulting in a large number of packets, occupies a small portion in the traffic volume. Utilizing Bluetooth for control traffic transmission typically consumes less than 20% of the available Bluetooth bandwidth. This leaves a lot of Bluetooth bandwidth unused because the Bluetooth interface in BlueStreaming is always active. Thus, when there are uploading packets pending, un-used Bluetooth bandwidth is firstly exploited to transmit the uploading traffic. Since Bluetooth is kept active to transmit frequent control traffic, diverting uploading packets to it does not incur much more power consumption.

While the above idea works intuitively, a design pitfall is the delay of control packets due to the streaming data uploading to neighbors. In order to ensure timely delivery of control packets, upstream data packets should not be diverted to Bluetooth when the bandwidth utilization of Bluetooth reaches a threshold. For this purpose, different priorities should be assigned to control traffic and data traffic. A higher priority should always be given to the control packets when data packets also present.

4.5.2 *Opportunistic WiFi Uploading*

Since WiFi provides a much higher data rate compared to Bluetooth, when high uploading is demanded and it cannot be completed by Bluetooth, BlueStreaming considers to opportunistically use WiFi to deliver uploading traffic in burst. For this purpose, similar to downstream/incoming packets, intermittent upstream/outgoing packets are combined and sent at a higher data rate by buffering outgoing streaming packets for a certain amount of time at the network layer at the client side.

The timing in the above process is critical. Although it is appealing to buffer the packets for long enough and then transmit them in a very large MAC frame, it would impact the performance of the requesting peer and may cause our BlueStreaming client to receive excessive repetitive requests. This can further increase the number of control packets and negatively impact the throughput of the Bluetooth channel. In order for WiFi to upload with a minimum consumption of extra battery power, the scheduler should work seamlessly with the PSM mechanism on the client.

Thus, to opportunistically utilize WiFi for uploading when needed, the uploading scheduler schedules to transmit burst uploading traffic before or right after “shaped” downstream traffic based on the PSM configuration.

- If the idle timeout interval used in PSM is only based on outgoing traffic activities, BlueStreaming starts to upload while the WiFi interface is in PSM. It will switch from PSM to CAM and the transmission is scheduled to finish right before the start of downstream packet transmissions. Note that the end-timing of uploading is not stringent, because the AP will not start to send buffered downstream packets until it receives a NULL wake-up or PS-Poll frame from the client. Under such a scheduling policy, the time spent for downloading streaming packets would be counted toward the idle timeout interval.
- If the idle timeout interval for PSM is based on both incoming and outgoing traffic activities, the scheduler can schedule the uploading via WiFi after being notified by the AP that it has no more buffered data packets.

In either case, the client can sleep for:

$$T_{sleep} = min\left(T_{re-req} \times (1 - \frac{Rate}{BW_{WiFi}}), T_{APtimeout}\right) - \frac{SIZE_{Up-Buf}}{BW_{WiFi}} \quad (2)$$

where $SIZE_{Up-Buf}$ denotes the volume of uploading packets buffered in the outgoing buffer at the client side.

5. IMPLEMENTATION

To evaluate the performance of BlueStreaming, we have implemented prototype systems on both Mac OS and Windows. In this section, we discuss some implementation and deployment issues.

5.1 Prototype Implementation

Our current prototypes run on laptops since laptops have implemented more complete Bluetooth profiles including Personal Area Network (PAN) as desired by BlueStreaming. BlueStreaming could be totally AP-transparent if control traffic can be separated at the transport layer. However, as we mentioned before, existing P2P streaming applications do not differentiate control traffic from data traffic at the transport layer. Therefore, we need a traffic classifier at both the AP and the client sides. We prototyped BlueStreaming AP using a MacBook as AP for both WiFi and Bluetooth. It shares its wired Ethernet connection via WiFi and Bluetooth interfaces. In a nutshell, the traffic classifier works at the IP layer. It intercepts packets at the IP layer, modifies destination addresses, and re-injects the packets to the corresponding interfaces.

The traffic shaper utilizes the buffer maintained at the WiFi AP by allowing the client not to wake up even if it has buffered packets. For this purpose, we implement a buffer at the network layer of our AP, which buffers downstream data packets and then forwards to the link layer in burst periodically. The buffering time is set according to Equation (1).

The upload scheduler is implemented at the client side. In our experiments, the idle timeout interval of PSM set by the driver is solely based on outgoing network activities. Therefore, we schedule outgoing uploading traffic before the bursty downloading of incoming traffic. To implement this, we set up a client side buffer at the network layer to buffer its outgoing packets. It also logs the timestamp of the first arrived packet of the latest incoming burst via WiFi, and determines when to start transmitting bursty uploading traffic based on Equation (2).

5.2 Infrastructure vs. Hybrid Mode

BlueStreaming can be deployed in two modes in practice: (1) an Infrastructure mode where a dedicated AP provides both WiFi and Bluetooth connections, and (2) a Hybrid mode where an intermediate device is used.

A BlueStreaming client needs both WiFi and Bluetooth interfaces, so that both interfaces can be used in parallel. In an infrastructure mode, the AP also needs to have both interfaces so that a BlueStreaming client and the AP can direct different types of traffic to different subnets (e.g. wireless LAN and bluetooth PAN).

In practice, however, a Bluetooth-enabled AP is not so common compared to a WiFi AP. If WiFi AP does not support Bluetooth, we allow a BlueStreaming client to operate in a hybrid mode by utilizing other Bluetooth-enabled network devices nearby. The intermediate devices should have a Bluetooth interface as well as a connection to the AP (wired or wireless). In practice, the hybrid mode can be easily deployed by using a plug-and-play Bluetooth dongle. In this mode, a BlueStreaming client can form an ad-hoc network with the intermediate device, and ask it to relay its traffic.

Another natural concern is the range mismatch problem of WiFi and Bluetooth. The expected communication range of WiFi is about 100 meters, while Bluetooth often covers only about 10 meters. In BlueStreaming, the hybrid mode can help deal with this issue. That is, with a relay in the middle, the communication distance of Bluetooth can double. Multi-hop relay could also be leveraged if necessary. We will study the impact of relay in the hybrid mode in the next section. If there is no relay possible at all, the user has to use WiFi as before.

On the other hand, the new Bluetooth standard may shed light on this range mismatch issue. For example, with the development of Bluetooth 4 [23], Bluetooth can improve its coverage to 50 meters or more. Better yet, with new low energy techniques, such an improved communication range may not necessarily increase the energy consumption.

5.3 Channel Selection and Competition Avoidance

Co-existence of Bluetooth and WiFi has attracted a lot attention during the past years, because Bluetooth and WiFi may operate in the same spectrum. When a Bluetooth transmission and a WiFi transmission happen at the same frequency, interferences could happen and packets may be lost.

Recent advances of wireless technology have made significant progress to address this problem. For example, hardware vendors today commonly use specific co-existence algorithms to handle Bluetooth and WiFi traffic intelligently when both interfaces are in operation. Operating systems such as Windows 7 also implement Adaptive Frequency Hopping (AFH) to minimize interferences. In our experiments, our laptop is equipped with the latest Bluetooth/WiFi combo chipset from Broadcom and runs Windows 7. We will study retransmissions rates in Section 6.4.

6. PERFORMANCE EVALUATION

6.1 Experimental Setup

To evaluate BlueStreaming, we run our Windows prototype to access PPTV, PPS, SopCast, and QQLive. A laptop (MacBook) is set as the BlueStreaming Access Point for both WiFi and Bluetooth, and WiFi runs 802.11n at 2.4GHz. A second laptop runs Windows 7 and acts as the BlueStreaming client to run the four applications. The third laptop associates with the AP and captures WiFi traffic in a promiscuous mode. In a hybrid mode, a fourth laptop is used to relay the control traffic between our testing client and the AP by associating with the AP via WiFi, while the AP turns off its Bluetooth interface and only serves as AP for WiFi. In all experiments, laptops are placed 3 meters from each other except that the traffic sniffer (the third laptop) and our testing client (the second laptop) are placed close to each other.

6.2 Energy Saving in the Infrastructure Mode

We first examine the effectiveness of BlueStreaming in infrastructure mode. For each of the four applications, three tests are conducted: (1) `PSM-A` (without BlueStreaming), (2) BlueStreaming with only the traffic classifier enabled, and (3) BlueStreaming with all three components enabled. Because there is always uploading traffic from our testing client, the traffic shaper has to work with the uploading scheduler in order to maximize power saving. So we did not test BlueStreaming with only the traffic classifier and the shaper enabled. Each of these tests is conducted for 30 minutes. For each application, three tests are carried out consecutively in the hope that the Internet conditions and dynamics of these P2P services have not remarkably changed.

Table 4: PPTV at 400 Kbps

Name	Total # of Streaming	Total # of Control	Sleep Time (%)	Consumed Energy(J)
PSM-A	66465	147554	0.56	2005
Classifier only	66121	121512	25.82	1745
BlueStreaming	70622	132466	60.50	1090

Table 4 gives an overview of the power saving on our testing client when accessing a channel of 400 Kbps in PPTV. During all three tests, our testing client receives smooth playback and captured frames show no quality degradation when compared to that in PSM-A. However, the amount of the battery power consumed by the WiFi interface for receiving streaming data is vastly different. With `PSM-A`, the WiFi interface spent only 10 seconds in the PSM and more than 99% of the session time in the active mode. When the traffic classifier is enabled, the amount of power saving is greatly improved: the WiFi interface spent more than 25% of session time in the PSM. When BlueStreaming is fully enabled, the WiFi interface spent over 60% time in the PSM, a 134% improvement over the case with only the traffic classifier enabled.

The additional sleep time of WiFi does not necessarily mean additional energy saving since BlueStreaming keeps Bluetooth active all the time and thus constantly consumes some battery power. We need to take that into account to evaluate the total power consumption of BlueStreaming. Since we do not use an instrument to measure the absolute battery consumption, we rely on the specification given in [16] and [10] to estimate the total power consumed. That is, the WiFi interface consumes 1120 mW in the CAM and 72 mW in the PSM; the Bluetooth interface consumes 120 mW in the active mode, and 25 mW in the idle mode. Note that we consider Bluetooth to be always active, which is conservative for the total power saving. The last column of Table 4 shows that considering the total power consumption of both the WiFi and Bluetooth interfaces, the classifier only and BlueStreaming can save over 13% and 46% energy, respectively, when compared to `PSM-A`.

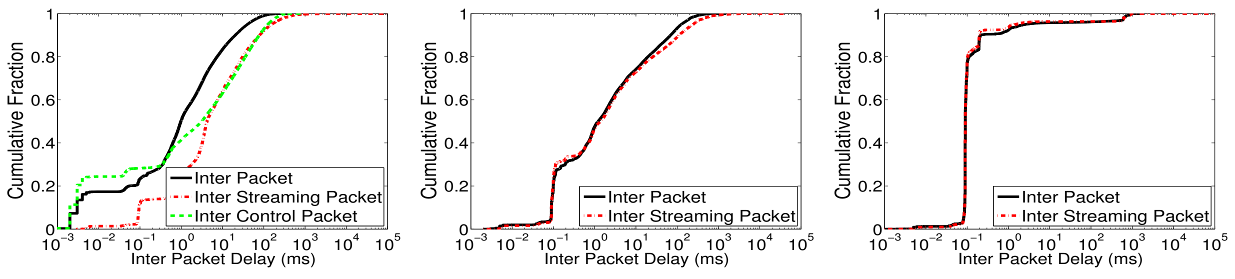
Figure 7 further shows the Inter Packet Delay distribution of WiFi traffic in these three tests. Compared to Figure 7(a), Figures 7(b) and 7(c) show that the Inter Packet delay distribution and Inter Streaming Packet delay are similar to each other, indicating the effectiveness of `Traffic Classifier` in diverting control traffic and exploiting sleep opportunities for the WiFi interface. In particular, Figure 7(c) shows a very pronounced bimodal pattern when all BlueStreaming components are enabled: about 95% packets arrive within 1 ms of the preceding one, while the rest arrive over 600 ms of the preceding packet, demonstrating the effectiveness of downstream traffic shaping.

Figure 8 further plots a snapshot (10 seconds) of the traffic pattern on the WiFi interface of our testing client in these three tests. Both incoming and outgoing packets are included. Figure 8(a) shows the traffic pattern when `PSM-A` is used. The traffic pattern shows a stair-case shape, and the small intervals between successive packets imply few opportunities for the WiFi interface to switch into PSM. Streaming with the classifier only can divert the frequent control packets to Bluetooth, resulting in more sleep opportunities as shown in Figure 8(b). However, due to the variation of response time from different neighbors, the traffic pattern is still not bursty enough. With the help of `Traffic Shaper` in BlueStreaming, Figure 8(c) shows that bursty traffic is sent more regularly and periodically, given that the interval between bursts is determined by Equation (1).

Table 5: QQLive at 500 Kbps

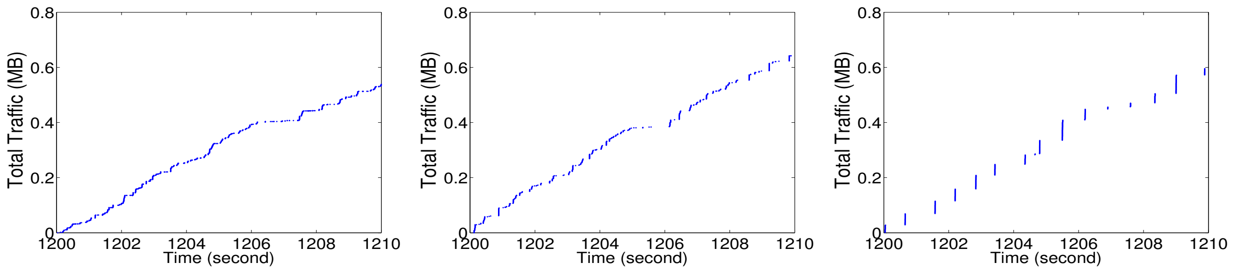
Name	Total # of Streaming	Total # of Control	Sleep Time (%)	Consumed Energy(J)
PSM-A	109800	206259	5.24	1917
Classifier only	105702	190428	34.37	1584
BlueStreaming	103600	187311	52.89	1234

Table 5 shows the results of our testing client accessing a 500 Kbps channel in QQLive. The table shows that although QQLive transmits 50% more packets than PPTV, it has more power saving in `PSM-A` and `Classifier-only` than in PPTV. This is likely due to the smaller variation of response time from a different number of neighboring peers. However, the request time-out T_{re-req} of QQLive is smaller than that of PPTV (based on our reverse en-



(a) PSM-A (b) Classifier only (c) BlueStreaming

Figure 7: PPTV 30-min Tests: Inter Packet Delay of WiFi (CDF)



(a) PSM-A (b) Classifier only (c) BlueStreaming

Figure 8: Traffic Pattern of PPTV

gineering), and thus traffic shaping is less effective for QQLive. Even so, the extra amount of battery power saved by applying traffic shaping and upload scheduling is still prominent: the WiFi interface spent more than 52% of the session time in the PSM.

While omitting detailed results of PPS and SopCast due to page limit, we also show the overall energy saving for them in Figure 9, where *Blue-I* means BlueStreaming under the infrastructure mode and *Blue-H* means BlueStreaming under the hybrid mode. Because PPS has a very small request time-out value, our client saves about 10% energy in BlueStreaming compared to PSM-A. SopCast, on the other hand, has a larger T_{re-req} smiliar to that of PPTV, and thus our client can save about 45% energy with BlueStreaming.

6.3 Hybrid Mode

We have shown that BlueStreaming is effective in saving power in the infrastructure mode where the AP supports both Bluetooth and WiFi. In practice, if Bluetooth is out of range or the AP does not support Bluetooth, we propose to use the hybrid mode. In this section, we evaluate BlueStreaming in hybrid mode with a relay node in the middle. The intermediate node forms a Bluetooth Adhoc network with our BlueStreaming client, and joins the subnet of BlueStreaming AP via the WiFi connection.

Since control traffic in P2P streaming is delay-sensitive, a particular concern on the hybrid mode is the delay of relaying, which could increase the response time at the application layer and deteriorate streaming quality. Because of the highly dynamic nature of P2P streaming overlay, we could not directly use application layer response time to compare the delay in different scenarios. Instead, we use `Ping` to measure the approximate round trip time from our BlueStreaming client to the AP via different paths, and estimate the increased control traffic delay. In short, the control traffic could be delivered under three scenarios: (1) `PSM-A` via a direct WiFi connection when BlueStreaming is not enabled; (2) `BlueStreaming-Infrastructure` via a direct Bluetooth connection; and (3) `BlueStreaming-Hybrid` via a relayed Bluetooth device. For `PSM-A` and `BlueStreaming-Infrastructure`, we use standard `Ping` to measure the RTT. For `BlueStreaming-Hybrid`, we run a customized `Ping` at our BlueStreaming client that measures the RTT of the *relayed* connection to AP.

We send out one `Ping` request every 1 second with 32 bytes payload, which is the payload of a typical control packet. For each scenario, we take the average of 400 RTT results. Table 6 shows that with a direct 802.11n WiFi connection, the RTT is about 3 ms; with a direct Bluetooth connection in the infrastructure mode, the RTT increases to about 22 ms on average, which is likely due to the smaller throughput of Bluetooth; with a relayed Bluetooth connection in the hybrid mode, the RTT slightly increases to 25 ms, which is due to the additional WiFi transfer occurred between the relay node and the WiFi AP. Apart from the potentially increased control traffic response time, the streaming quality during the tests has no degradation in either the infrastructure or hybrid mode.

Figure 9 further summarizes the average energy consumption across four applications in these tests. The figure shows that the estimated energy consumption is comparable in both modes.

Due to high dynamics of P2P streaming overlay, it is not appropriate to say if one mode is better/worse than the other in terms of power saving even if they have demonstrated slightly different performance on energy consumption. Nevertheless, it is reasonable to conclude that BlueStreaming running in both modes can achieve good power saving.

6.4 Retransmission Rate

Throughout our experiments, we have been using 802.11n at 2.4 GHz band (instead of 5 GHz), which is the same band that Bluetooth operates on. In addition to that, using `airport scan` on the MacBook to scan all available WiFi APs nearby, we find that all non-overlapping channels (channel 1, 6, and 11) of 2.4 GHz are occupied by more than 6 APs each during our experiments because our experiments are done in a lab covered by the AP hotspots. Because we run our experiments during 1pm to 5pm local time, there are a lot of wireless users around, meaning lots of background traf-

Table 6: RTT based on customized Ping

Name	RTT (ms)
PSM-A	3
BlueStreaming-Infrastructure	22
BlueStreaming-Hybrid	25

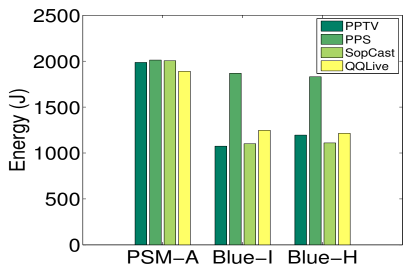


Figure 9: Energy Consumption Comparisons

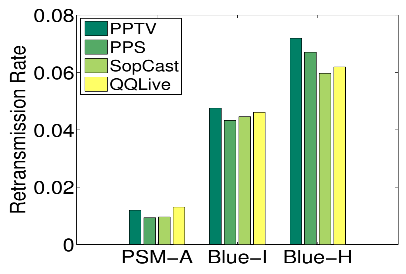


Figure 10: Retransmission Rates

fic. Both sharing 2.4 GHz band and significant background traffic may cause collisions and interferences.

To examine the situation of such intereferences, we study the re-transmission rate during our experiments. Figure 10 depicts the average retransmission rate of WiFi traffic throughout our experiments. The result shown in the figure is the mean of five runs. When the Bluetooth interface is not used, the retransmission rate under `PSM-A` across four applications is about 0.01. When we turn on BlueStreaming and use the infrastructure mode to transmit P2P streaming traffic, the retransmission rate increases to above 0.04. This shows in our environment, actively using Bluetooth does mildly increase retransmissions. When BlueStreaming uses an intermediate node to relay control traffic in the hybrid mode, the retransmission rate increases to about 0.06, possibly because the relay node uses WiFi to communicate with our BlueStreaming AP and deliver control traffic. Even so, the streaming quality in the experiments is not degraded, demonstrating the practicability of BlueStreaming. In an environment with less background traffic, the retransmission rate could further decrease.

7. CONCLUSION

Recently the Internet has witnessed the rapid increase of P2P streaming traffic and the quick shift of using mobile devices for Internet accesses. Unfortunately, accessing delay-sensitive streaming services on a mobile device is hindered by the limited battery power supply, because a mobile client in P2P streaming needs to transmit more traffic, including uploading to neighbors as well as exchanging a large number of control packets with neighbors. In order to enable power-efficient P2P streaming to mobile devices, in this work, we have designed and implemented BlueStreaming, a system that simultaneously utilizes WiFi and commonly existing Bluetooth interfaces. The energy consumed by the Bluetooth interface is effectively over-compensated by the greater power saving on the WiFi interface. Extensive experiments with our implemented prototypes demonstrate the effectiveness and practicability of BlueStreaming.

8. ACKNOWLEDGEMENT

We appreciate constructive comments from our shepherd Surender Chandra and anonymous referees. The work is partially supported by NSF under grants CNS-0746649, CNS-1117300, CCF-0915681, CCF-1146578, and AFOSR under grant FA9550-09-1-0071.

9. REFERENCES

[1] "PPTV (previously PPLive)," http://www.pptv.com/.

[2] "PPStream," http://www.ppstream.com/.

[3] "SopCast," http://www.sopcast.com/.

[4] "QQLive," http://live.qq.com/.

[5] "CNN Live Video," http://www.cnn.com/help/liveflash.html.

[6] "Cisco VNI," `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`.

[7] "Apple's iPad Generates 2.5x the Wireless Data Traffic Generated by the iPhone," `http://bytemobile.com/news-events/2010/archive_260410.html`.

[8] "WiFi," http://v4.jiwire.com/search-hotspot-locations.htm.

[9] 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 *Proc. of ACM MOBICOM*, 2006.

[10] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, "NAPman: Network-Assisted Power management for WiFi Devices," in *Proc. of ACM MobiSys*, 2010.

[11] S. Chandra and A. Vahdat, "Application-specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats," in *Proc. of USENIX Annual Technical Conference*, 2002.

[12] F. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices," in *Proc. of ACM MobiSys*, 2010.

[13] E. Tan, L. Guo, S. Chen, and X. Zhang, "PSM-throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs," in *Proc. of IEEE ICNP*, 2007.

[14] J. Liu and L. Zhong, "Micro Power Management of Active 802.11 Interfaces," in *Proc. of ACM MobiSys*, 2008.

[15] Y. Agarwal, R. Chandra, A Wolman, P. Bahl, K. Chin, and R. Gupta, "Wireless Wakeups Revisited: Energy Management for Voip over Wi-Fi Smartphones," in *Proc. of ACM MobiSys*, 2007.

[16] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio interfaces," in *Proc. of ACM MobiSys*, 2006.

[17] D. Bertozzi, L. Benini, and B. Ricco, "Power Aware Network Interface Management for Streaming Multimedia," in *Proc. of IEEE WCNC*, 2002.

[18] Y. Wei, S. M. Bhandarkar, and S. Chandra, "A Client-side Statistical Prediction Scheme for Energy Aware Multimedia Data Streaming," *IEEE Transactions on Multimedia*, vol. 8, no. 4, 2006.

[19] "TVUPlayer," http://itunes.apple.com/us/app/id323640984.

[20] "Justin.tv," http://itunes.apple.com/us/app/id358612216.

[21] "Justin.tv," http://www.justin.tv.

[22] Y. Liu, F. Li, L. Guo, and S. Chen, "Reducing Data Request Contentions for Improved Streaming Quality," in *Proc. of ACM NOSSDAV*, 2010.

[23] "Bluetooth 4," http://www.gizmodo.com.au/2010/04/bluetooth-4-0-uses-less-power-while-extending-range/.