A Measurement Study of Resource Utilization in Internet Mobile Streaming

Yao Liu[1] Fei Li[1] Lei Guo[2] 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

ABSTRACT

The pervasive usage of mobile devices and wireless networking support have enabled more and more Internet streaming services to all kinds of heterogeneous mobile devices. However, Internet mobile streaming services are challenged by the inherently limited on-device resources, device heterogeneity, and the bulk amount of streaming data.

In this paper, focusing on resource utilization and streaming quality on mobile devices, we investigate 10 deployed Internet mobile streaming services that employ client-server, client-proxy-server, and P2P architectures from a client's perspective. We find that (1) existing Internet mobile streaming services mainly use the client-server architecture and commonly adopt burst traffic delivery that can save battery power consumption on mobile devices; (2) to deal with device heterogeneity, some streaming services have already utilized intermediate nodes (often the user's home computer) for online transcoding with a client-proxy-server architecture, but currently they lack power-friendly design for mobile devices; (3) a mobile device in P2P streaming consumes significantly more battery power mainly due to the inevitable P2P control traffic and uploading traffic to other peers. These findings provide us new insights to further optimize Internet mobile streaming in the future.

Categories and Subject Descriptors

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

General Terms

Experimentation, Algorithms

Keywords

Mobile Streaming, Battery, Power, Resource Utilization, QoS

1. INTRODUCTION

Recent years have witnessed the rapid development and wide deployment of the 802.11 and the third generation wireless networks. For example, Jupiter Research estimated 65% of households in the United States have home-deployed WiFi access points (APs) [1]. Meanwhile, the smartphone market is also growing very fast. By September 2010, over 58.7 million people in the US owned smartphones [2]. With the most recent technology advancements, people today are more inclined to use their mobile devices to access the Internet.

In tandem with pervasive mobile Internet accesses, the Internet media streaming services are widely accessed with the rich and fast growing Internet bandwidth and streaming media content. For example, according to Comscore, 146.3 million viewers watched more than 2 billion videos on Youtube in October 2010 [3]. Many P2P/overlay based systems, such as PPLive [4] and PPStream [5], have enabled easy and highly scalable live streaming in practice and have attracted millions of users daily [6].

Naturally, under these two trends, there is a quickly increasing demand for Internet streaming to mobile devices (Internet mobile streaming hereafter). For this purpose, both iOS and Android have native support for YouTube. Targeting this market, more and more content providers now allow their customers to access multimedia content on their mobile devices via wireless connections. Most recently, Netflix [7] started to provide streaming services to subscribed iPad, iPhone, and iPod Touch users. However, delivering high quality Internet streaming to mobile devices faces several challenges due to inherent constraints of mobile devices.

First, compared to the traditional desktop computers, mobile devices commonly have limited resources, such as slower CPU speed, smaller memory and storage sizes, and limited battery power. Among these resources, the battery power poses a fundamental constraint. For Internet streaming, it often involves bulk and continuous data transmissions with stringent timing requirements. As a result, it demands continuous operations of the mobile device, including the wireless network interface card (WNIC) to receive the data, the CPU to decode the received data, and the screen to display the media content.

Second, mobile devices are very heterogeneous, differing from each other and from traditional desktop computers on not only the uploading/downloading bandwidth capacities, but also the screen sizes, the color depths, etc., which we refer to as *multi-dimensional heterogeneity.* Thus, most of the current popular Internet streaming content must be customized to the appropriate image resolution, size, frame rate, and bit rate for each mobile device. Such customizations could be done in advance or at runtime. For example, for pre-recorded video clips, it is possible to pre-code various versions of the same content. On the other side, if the customization is done at runtime, it demands a great deal of CPU cycles at either the server or the client side.

Despite these technical difficulties, Internet mobile streaming services are booming. For example, CTV [8], CCTV [9], WTV [10], ImgoTV [11], and NHKworld [12] all allow iOS users to access their live TV programming content via 3G or WiFi network. SPBtv [13] supports more diverse platforms including iOS, Android, and WebOS. It also provides a great variety of live TV channels from different countries. Orb [14] and AirVideo [15] allow users to access media content stored on their home computers. Justin.tv [16] even enables users to watch live streaming content broadcasted by other users. While these services become more and more popular on the Internet and potentially contribute a significant portion of Internet traffic, it remains unclear how they have addressed the aforementioned problems and how effective their schemes are in practice.

In this paper, we investigate these Internet mobile streaming services from an end user's perspective. In particular, we focus on the resource consumption on mobile devices and the streaming quality received at the client side. For this purpose, we conduct Internet measurements on 10 different Internet mobile streaming services using client-server, client-proxy-server, and P2P architectures. Our measurement and analysis show that (1) existing Internet mobile streaming services mainly use the client-server architecture and commonly adopt burst traffic delivery techniques that can save the power consumption on mobile devices; (2) to deal with device heterogeneity, some streaming services have already utilized intermediate nodes (often the user's home computer) for online transcoding with a client-proxy-server architecture. While the intermediate proxies provide great convenience for heterogeneous mobile devices, currently they lack power-friendly design for mobile devices; (3) a mobile device in P2P streaming consumes significantly more battery power mainly due to the inevitable P2P control traffic and uploading traffic to other peers, although the P2P architecture is most scalable. Based on these findings, we can further optimize existing Internet mobile streaming systems.

2. INTERNET MOBILE STREAMING AND MEASUREMENT METHODOLOGY

In this section, we briefly illustrate the three typical architectures of existing Internet mobile streaming applications, and then describe our measurement methodology.

2.1 Internet Mobile Streaming

Today Internet mobile streaming applications mainly use three different architectures: client-server (C/S), client-proxy-server (C/P/S), and peer-to-peer (P2P).

2.1.1 Client-Server (C/S) Architecture

C/S is the most traditional and popular architecture. Typically, within this architecture, a mobile device requests streaming data from a dedicated server using standard HTTP protocols. HTTP live streaming was made available on iOS 3.0 in July 2009 [17].

In C/S based streaming applications, we evaluate *CTV*, *CCTV*, *W.TV*, *Imgo.TV*, *Justin.tv*, *SPBtv*, and *NHKworld*. In these applications, the media content is encoded as MPEG-4 audio and video, and transported via TCP using MPEG-2 Transport Stream (MPEG2-TS). A `.ts` file typically contains 10 second media content. For streaming access, the client queries the server for streaming content about every 10 seconds and the server replies with links to several `.ts` files, each of which contains media content for the next a few (typically 10) seconds in MPEG2-TS format. These files are reassembled after being downloaded and fed into the Media Player for playback.

2.1.2 Client-Proxy-Server (C/P/S) Architecture

Recently, placeshifting streaming services are getting popular. A placeshifting streaming system generally employs a C/P/S architecture, where an intermediary unit, referred to as a transcoding proxy, is needed for content customization. In addition, the placeshifting service provider may also set up a relay proxy to relay the content between the placeshifting server and the mobile device.

In our study, we examine *Orb* and *AirVideo*, both of which use personal computers as the placeshifting server and the transcoding proxy. In Orb, a user can set up the placeshifting server at home and register its available multimedia content with the Orb relay proxy. To access streaming services, the placeshifting server with a built-in transcoding proxy would transcode the video content based on its uploading bandwidth, the downloading bandwidth of the mobile device, and the supported codec on the mobile device. Unlike Orb that has been initially released in 2005, AirVideo does not have good support for remote accesses.

2.1.3 Peer-to-Peer (P2P) Architecture

Lots of Internet streaming systems today have adopted P2P techniques. While different protocols have been extensively studied, in practice, most systems use a pull based mesh structure, in which peers need to frequently exchange control messages [6], such as buffermaps, in order to determine the data chunks to exchange with each other.

In this study, we investigate *TVUPlayer* [18], which uses a peer-to-peer architecture to distribute live streaming content. To the best of our knowledge, TVUPlayer is the only P2P based live streaming application available on iOS at the time of this measurement. Instead of using TCP, it uses UDP for streaming delivery.

2.2 Methodology

In the experiments, we use the second generation iPod Touch (iTouch) running firmware version 3.1.2 to receive streaming services. The device is jailbroken to install essential tools for logging performance statistics. The iTouch is instructed to access 10 Internet mobile streaming services.

In the measurement, we record all incoming/outgoing data to/from our testing device at the data link layer by setting up Wireshark to listen on the same channel as the iTouch in promiscuous mode. For performance analysis, we also have logged both statistics of battery power consumption and CPU usage. We take a snapshot of battery state directly from battery every 30 seconds and we log kernel I/O statistics and CPU usage every 15 seconds. In order to minimize disturbance during the measurement, we mute the speaker on the device. We also set the screen backlight to 35 (max 127) and disable auto-adjustment.

As we do not have access to the server logs, we run our devices to access these streaming services repetitively. The reported results in this paper are based on experiments conducted from 03/20/2010 – 05/10/2010. Since we run experiments without plugging the device to a power source, for a single run, the longest session lasts for the lifetime of a fully charged battery. As we mainly focus on the relationship between resource consumption and perceived quality at the client side, we conduct all C/S and C/P/S based measurements at non-peak time, mostly from midnight to 6 AM in the morning unless noted otherwise. For P2P, we have conducted experiments at both peak time (8:00 PM in the evening) and non-peak time (midnight to 6 AM in the morning). All experiments are conducted with a dedicated AP running 802.11g deployed in a lab on a university campus in order to minimize traffic contention. We disable all other connections (e.g., bluetooth) on the iTouch.

For accessing each of these streaming services, two types of tests have been conducted. The first is *Stress Test* to examine how long video streaming can last. Basically, we start the streaming with a fully charged battery and keep the application running until the device battery is exhausted. The second is *1-Hour Test* for more detailed analysis as presented below, where we log all statistics for one hour immediately after the application is started.

3. MEASUREMENT RESULTS

In this section, we present our measurement results of Stress Tests. Among the 10 services we have investigated, we present the representative ones for brevity, namely SPBtv (a client-server (C/S) architecture), Orb (a client-proxy-server (C/P/S) architecture), and TVUPlayer (a P2P architecture). Some results of other services are collectively presented or presented for comparisons.

3.1 Overview

Figure 1(a) shows the CPU usage and battery consumption while the iTouch runs SPBtv to watch a video channel with a fully charged battery. In this figure, the left-y axis shows the CPU usage breakdown of user, system, and idle. The right-y axis shows the remaining battery (in terms of percentage) along the playback. The streaming rate of this video channel is 464 Kbps. All data are received from a dedicated server in Oregon, USA. As is shown in this figure, starting from a fully charged battery, the streaming lasted for about 6.4 hours. Note that Apple announced that the battery on 2nd generation iPod Touch can last 6 hours for video watching [19]. Our test here confirms this. On the other hand, we also notice that the CPU is mostly (about 80%) idle in this session.

Figure 1(b) shows the result of iTouch running Orb to watch episodes stored on a personal computer at home. With a fully charged battery, it can watch the program of 297 Kbps for a total of 4.3 hours. Compared to C/S based SPBtv, the watched video has a lower quality (297 Kbps vs. 464 Kbps), meaning 36% lower streaming rate. But a fully charged battery only works for about 67% of time duration in the C/S based SPBtv. It is also surprising that the user level CPU usage is as high as 50%. Given that the CPU also consumes a lot of battery power, this may explain the earlier exhaustion of battery power in C/P/S based Orb with a lower streaming rate. We will further analyze this later.

Figure 1(c) shows the result of using TVUPlayer, which uses P2P architecture for streaming delivery. The streaming only lasts for about 3 hours watching a 281 Kbps channel. We also notice that the CPU usage at user level is even higher than in the C/P/S based Orb, fluctuating between 60% and 80%.

The results above shows a mobile device in C/S based SPBtv and C/P/S based Orb consume much less energy compared to in P2P based TVUPlayer, even without considering the streaming rate difference. We next examine the potential reasons, focusing on two major power consumption sources: CPU and the network interface card.

3.2 CPU Usage Study

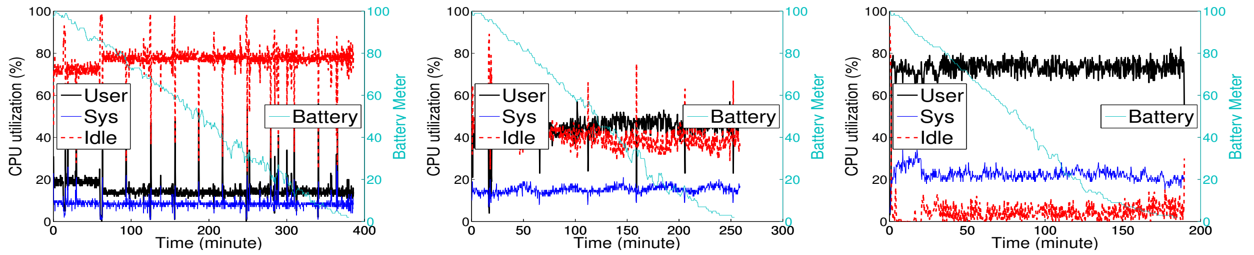
Since iOS 3.2.1 does not support multi-tasking, the CPU cycles in a streaming session could mainly be used for two purposes: decoding the received data and transcoding for the mobile device. Decoding is to decode the received data in the proper format for the MediaPlayer. The CPU cycles used for decoding depend on the video format, supported codec, the bit rate, the frame rate, the resolution, etc.

Transcoding is different and often consumes much more CPU cycles. In mobile streaming, mobile devices have different screen size, color depth, resolution rate, etc. from desktop computers, so not all streaming content can be directly played on mobile devices. An intuitive solution to address this heterogeneity issue is to provide different versions of the same video content for different platform/devices. Youtube uses this approach. Another more flexible solution is called online transcoding, which conducts content adaptation at runtime based on the device type, downloading bandwidth etc. This process, although very desirable, consumes a lot of CPU cycles at runtime.

We first investigate in SPBtv how the CPU cycles on our iTouch were spent. Via reverse-engineering, we find that CPU cycles are mainly spent on decoding the video content. For SPBtv, the video is encoded in H.264 standard, which can be decoded by hardware. H.264 has the native support on iTouch and iPhone as we discussed in section 2. This is confirmed by Figure 1(a) as about 80% CPU cycles are idle.

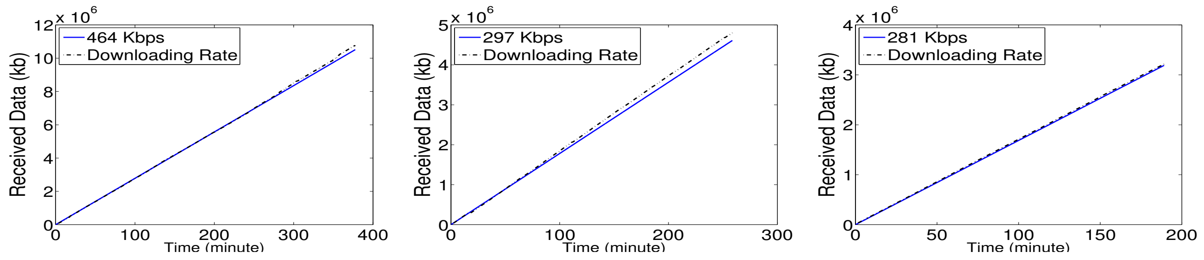
On the other hand, using a software decoder could consume more CPU cycles. Figure 1(b) shows that about 40% more CPU cycles are spent in Orb on decoding the media data for playback. The video content of Orb is encoded into `flv` format. Decoding flv on iTouch simply relies on the software embedded in the MediaPlayer on iTouch. Note that here the extra CPU cycles are not used for transcoding on iTouch, since transcoding is done by the transcoding proxy, co-located with the placeshifting server. The data received by iTouch has been transcoded.

Compared to SPBtv, Figure 1(c) shows that TVUPlayer consumes even more CPU cycles. To investigate how TVUPlayer use these CPU cycles, first, we set to examine video codec of TVUPlayer streaming content. Given that on the same P2P overlay, peers on Mac OS and iOS receive the same streaming data, we start a TVUPlayer client on Mac OS. As the playback starts, we examine all active local TCP connections and locate the port that the MediaPlayer downloads streaming data from. We then start a HTTP connection that downloads data from that port to a file. A closer examination of this file shows that it is encoded with Advanced Systems Format (ASF). However, neither Mac OS nor iOS supports such a video format. Therefore, it is clear



(a) SPBtv – 464 Kbps (b) Orb – 297 Kbps (c) TVUPlayer – 281 Kbps

Figure 1: Stress Test: CPU and Battery Usage



(a) SPBtv (b) Orb (c) TVUPlayer

Figure 2: Stress Test: Data Receiving vs. Playback

that TVUPlayer performs online transcoding from ASF to the supported codec at the client side.

The results in this section show that the hardware- and software-based decoding and transcoding can lead to significant difference in the CPU usage, and thus incur different amount of battery power consumption.

3.3 Battery Power Consumption By Transmission

Aside from the battery power consumed by the CPU, it is believed that the wireless network interface card (WNIC) is one of the largest drains of the battery power on mobile devices and the power saving mode (PSM) is commonly used in practice. In this subsection, we further study how the battery power is consumed by the WNIC on the iTouch in these streaming sessions.

Since we log packets at the data-link layer, we are able to extract frame control information from IEEE 802.11 header. One of the flags in the frame control field is `Pwr Mgt`. It indicates the Power Management mode that the device will switch to after this frame is transmitted. Basically, the mobile device would either sleep or stay active.

First, we plot the total duration of the mobile device in the power saving mode based on its power management activity. For each application, we present the median of the runs we had conducted (we ran 1-Hour Test for each application between 5 to 12 times). Figure 3 shows the percentage of time of the WNIC in the power saving mode. As shown in this figure, the WNIC in C/S based streaming commonly has the longest sleeping time: it spends about 80% of total streaming time in the power saving mode. In C/P/S based Orb, the WNIC also spends about 70% of time in the power saving mode, while in AirVideo, it can sleep more with the more efficient packetization scheme (details are omitted due to page limit). However, in P2P based TVUPlayer, the WNIC operates in the power saving mode for as little as 38% of the session time.

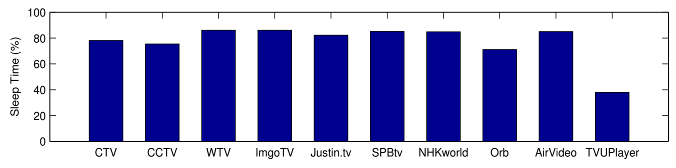


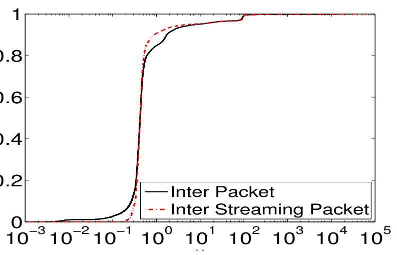
Figure 3: Sleep Time (%) of the WNIC

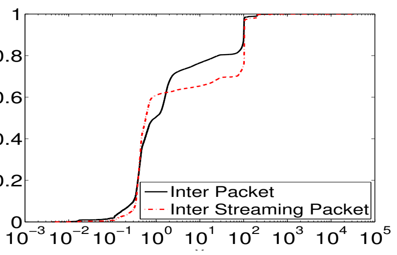
While the above results show that the power saving mode does take effect on iTouch in all the three Internet mobile streaming delivery architectures, the results also indicate the sleep time of the WNIC varies, which may be affected by the number of packets transmitted, packet size, inter packet delay, etc. Next, we investigate from these aspects in order to understand the underlying reasons.

3.3.1 Bursty traffic delivery saves battery power consumption

In the 802.11 power saving mode, the WNIC on a mobile device would wake up periodically and listen for Traffic Indication Map (TIM). If it does not have data buffered at the Access Point (AP), it will go to sleep. Otherwise, it will retrieve buffered data. In the PSM adaptive mode, if no data has arrived during last beacon interval (typically 100 ms), the WNIC can operate in the power saving mode until the next scheduled beacon interval. Thus we first study inter packet delay, and analyze its impact on energy consumption.

Figure 4 shows the corresponding inter packet delay distribution of the *Stress Test* we have studied. While `Inter Packet` considers all incoming/outgoing traffic to/from the mobile device, `Inter Streaming Packet` considers only ingress streaming data packets. In this figure, for SPBtv, we observe that both the Inter Packet delay and the Inter Streaming Packet delay show a clear bimodal pattern: about 5% packets arrive (both ingress and egress) over 20 ms after the previous packet, most of them are even over


(a) SPBtv


(b) Orb

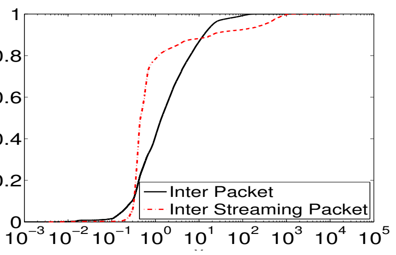
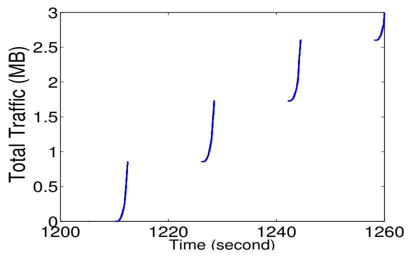
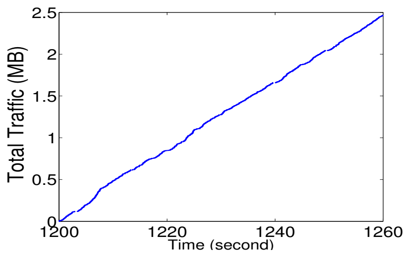

(c) TVUPlayer

Figure 4: Stress Test: Inter Packet Delay (ms) (CDF)


(a) SPBtv


(b) Orb

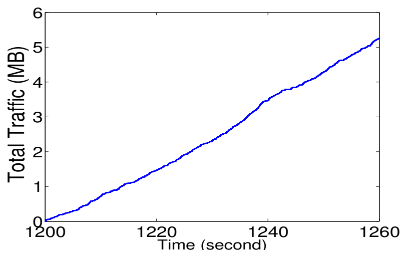

(c) TVUPlayer

Figure 5: Stress Test: Traffic Pattern

100 ms, leaving great opportunity for WNIC to sleep for power saving. On the other hand, the majority (over 85%) of packets arrive within very short time interval (less than 5 ms), resulting in bursty traffic as shown in Figure 5(a). Note Figure 5 shows the traffic pattern in 60 seconds only for clear visual effect. The traffic pattern in other time durations is similar. This indicates that the SPBtv server has considered the power saving on mobile devices by adopting the traffic shaping and delivery techniques.

On the other side, in C/P/S based Orb, the bimodal distribution of the inter packet delay is less pronounced. Compared to SPBtv, Figure 5(b) shows the traffic in Orb is almost at the constant streaming rate delivered by the proxy. This implies two things: 1) the sender in Orb does not use any traffic shaping technique; 2) the battery power saving simply relies on if the inter packet delay is larger than 100 ms or not. As shown in Figure 4(b), there are still 30% packets falling into this range, leading to battery power saving.

Similarly, Figure 4(c) shows that under P2P, only about 2% of packets have an inter packet delay larger than 100 ms, resulting in even less battery power saving during streaming. Figure 5(c) confirms that in P2P streaming, the streaming rate is almost constant and the streaming traffic is not delivered in bursts.

3.3.2 *Inefficient packetization increases battery power consumption*

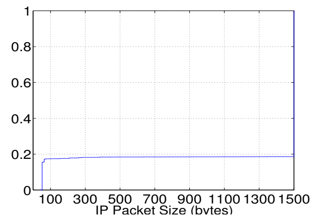
Both SPBtv and Orb use TCP to transmit streaming data. However, their traffic patterns are significantly different due to the adoption of traffic shaping techniques in SPBtv. In this subsection, we further examine if other factors have contributed to the less power saving in Orb.

Figures 6(a) and 6(b) show the packet size distribution for SPBtv and Orb. As shown in the figures, nearly 20% packets in SPBtv are TCP ACKs. They are constantly 52 bytes, and contain no payload. The rest (about 80%) packets are streaming packets, whose size is the same as maximum transmission unit (MTU) in Ethernet (1500 bytes). In Orb, however, only about 50% packets are streaming packets with a size of 1500 bytes. Besides the 35% control packets (TCP ACKs), the rest 15% packets are streaming packets with smaller sizes: varying from about 100 bytes to 1500 bytes. This increases the number of packets transmitted as such small packets could have been combined to large packets. Potentially, this further reduces the inter packet delay, dis-allowing the WNIC from switching to the power saving mode. Given high speed connection at both the first-mile and the last-mile, this is less likely due to the congestion. Rather, it means the deficiency in Orb design without considering the power consumption on the mobile device.

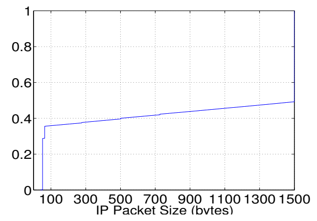
3.3.3 *P2P control and uploading traffic leads to excessive battery power consumption*

For P2P based TVUPlayer, Figure 6(c) shows that only less than 30% packets are streaming packets with a size of 1300 bytes, with about 70% of packets having a size less than 100 bytes. Note that most P2P streaming systems today use UDP to transmit packets. Thus, these small packets can only be the control traffic of P2P protocol, such as buffermap information exchanged with other peers. Since P2P streaming systems require peers to exchange control messages with neighbors frequently, this is not a surprising result. However, these control packets significantly change the traffic pattern, resulting in less opportunities for the WNIC switching into the power saving mode.

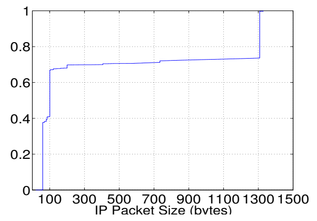
In P2P streaming, besides the impact of the control traffic, the amount of uploading traffic is also important since a peer is required to upload streaming data to other peers during streaming. This can further aggravate the power consumption on a mobile device. For example, Figure 7 shows the uploading traffic in the Stress Test of TVUPlayer. In this figure, we can see that the uploading rate could reach as high as 1000 Kbps, 3 times more than the downloading rate of streaming data (281 Kbps) in the first half an hour.



(a) SPBtv



(b) Orb



(c) TVUPlayer

Figure 6: Stress Test: Packet Size (CDF)

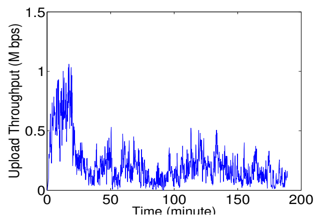


Figure 7: Stress Test: Upload Traffic in TVU-Player

Our measurement results show that some C/S and C/P/S based Internet mobile streaming services have adopted techniques to deal with battery power constraints and device heterogeneity. However, the existing P2P streaming service lacks power-efficient design and device heterogeneity handling capabilities, and deserves further optimization.

4. OTHER RELATED WORK

For media applications over wireless connections, Chandra studied typical Internet streaming services in 2002 and concluded that 802.11 PSM does not save energy if the streaming is over 56 Kbps [20]. Accordingly, history-based prediction [20] and linear prediction [21] schemes were proposed to schedule the WNIC into sleep or active modes. PSM-throttling leverages traffic shaping from the client side in order to allow the WNIC to sleep for longer time [22]. Targeting Internet P2P streaming, Tan et al. designed SCAP [23] in order to reduce the power consumption on wireless devices by caching the uploading traffic at the AP. Xiao et al. studied the power consumption of mobile Youtube [24].

5. CONCLUSION

Despite the fundamental constraints of limited available resources, device heterogeneity, and bulk data transmissions, Internet mobile streaming is becoming more and more popular. In this paper, we examine the resource utilization with a focus on battery power consumption on mobile devices in receiving such streaming services from a client's perspective. Our measurement and analysis show that mobile devices have different power efficiency in receiving Internet streaming services under existing architectures. Our study provides some new insights into effective power saving on mobile devices in Internet streaming and calls for future systems that can minimize battery power consumption while be scalable and efficient in heterogeneity handling.

6. ACKNOWLEDGEMENT

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

7. REFERENCES

[1] 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. ACM MOBICOM*, 2006.

[2] "Sep. 2010 U.S. Mobile Subscriber Market Share," `http://www.comscore.com/Press_Events/Press_Releases/2010/11/comScore_Reports_September_2010_U.S._Mobile_Subscriber_Market_Share`.

[3] "Top U.S. Online Video Content Properties by Unique Viewers," `http://www.comscore.com/Press_Events/Press_Releases/2010/11/comScore_Releases_October_2010_U.S._Online_Video_Rankings`.

[4] "PPlive," http://www.pplive.com.

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

[6] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "Insights into PPLive: A measurement study of a large-scale P2P IPTV system," in *Proc. of IPTV Workshop*, May 2006.

[7] "Netflix," http://itunes.apple.com/app/id363590051.

[8] "CTV," http://itunes.apple.com/app/id340381556.

[9] "CCTV," http://itunes.apple.com/app/id331259725.

[10] "W.TV," http://itunes.apple.com/app/id318676629.

[11] "ImgoTV," http://itunes.apple.com/app/id349448995.

[12] "NHKworld," http://itunes.apple.com/app/id350732480.

[13] "SPBtv," http://itunes.apple.com/app/id356830174.

[14] "OrbLive," http://itunes.apple.com/app/id290195003.

[15] "Air Video," http://itunes.apple.com/app/id306550020.

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

[17] "HTTP Live Streaming Draft," `http://tools.ietf.org/html/draft-pantos-http-live-streaming`.

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

[19] "iPod touch (2nd generation) - Technical Specifications," http://support.apple.com/kb/SP496.

[20] S. Chandra, "Wireless network interface energy consumption implications of popular streaming formats," in *Proc. of MMCN*, 2002.

[21] 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.

[22] 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*, October 2007.

[23] E. Tan, L. Guo, S. Chen, and X. Zhang, "Scap: Smart caching in wireless access points to improve p2p streaming," in *Proc. of IEEE ICDCS*, June 2007.

[24] Y. Xiao, R. Kalyanaraman, and A. Yla-Jaaski, "Energy consumption of mobile youtube: Quantitative measurement and analysis," in *Proc. of NGMAST*, September 2008.