Smartphones vs. Laptops: Comparing Web Browsing Behavior and the Implications for Caching

Ioannis Papapanagiotou
Electrical & Computer Engineering
NC State University
Raleigh, NC
ipapapa@ncsu.edu

Erich Nahum
IBM TJ Watson Research Center
Hawthorne, NY
nahum@us.ibm.com

Vasileios Pappas
IBM TJ Watson Research Center
Hawthorne, NY
vpappas@us.ibm.com

ABSTRACT

As more and more users access the web through mobile devices, traffic patterns for these devices need to be better understood. Characterizing their traffic is of primary importance both to wireless network carriers and to corporate network administrators, as they prepare their networks for the explosive growth of smartphones and tablets.

In this paper, we have collected and analyzed the wireless data traffic of a large corporate network, of around 2200 laptops and 400 smartphones. Using this data set, we devise a methodology to distinguish different types of wireless devices (smartphone vs laptops) as well as operating system instances (iOS, Android, BlackBerry etc.). We analyze the mobile web traffic behavior and identify the similarities and differences across the most common mobile platforms. For example, we observe that streaming content to iOS devices is different compared to other Operating Systems. That is because iOS devices have a media component that handles the downloaded traffic, and it has some unique properties, such as requesting objects with a specific size. We also observe that laptop devices have more intelligent browser caching capabilities.

Hence, we investigate the impact of a browser cache in all devices. We determine that a 10MB browser cache which is able to handle partial downloads, in smartphones would be enough to handle the majority of the savings. Finally, we showcase that caching policies need to be amended to attain the maximum possible savings in proxy caches. Based on those optimizations the emulated proxy cache provides $10\% - 20\%$ in bandwidth savings.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Management*

General Terms

Measurements, Design

Keywords

Measurements, Mobile Traffic, Network Usage

1. INTRODUCTION

As smartphones reach the 1 billion threshold, they are already contributing more than 10% of Internet traffic [10]. Recently a number of studies have been conducted on smartphone usage [17, 19, 25, 22]. Yet many aspects of smartphone traffic behavior are still not well understood. Their behavior may have implications for network design and provisioning, particularly if their traffic is significantly different from other devices, such as PCs.

This paper exposes some of those differences by comparing the Web browsing behavior of smartphones with a control group, namely laptops. We do this by capturing a full-packet trace in a wireless enterprise environment, composed of the traffic from over 3000 unique devices over 3 weeks. We use this trace to evaluate the Web browsing behavior of the devices, as that makes up the bulk of the traffic. We examine standard metrics such as object size, popularity, and content type. We focus mainly on those areas where the traffic of the two device categories differ. In particular, we focus on issues related to video and how it is delivered over HTTP, via progressive downloads with range requests. We also examine the effectiveness of caching in detail, both at the browser and at a proxy.

We summarize our contributions as follows:

- **Device Identification**: We show that earlier approaches to device identification which rely on the User-Agent header are insufficient. We introduce a data-mining technique based on DHCP information. Our approach identifies 98.5% of the devices in the network, as compared to 84% for earlier approaches.
- **Web Traffic across Devices**: We examine Web traffic across both smartphones (iOS, Android, BlackBerry) and laptops (Windows and Mac OS X). We show that smartphones make less use of conditional GET requests, and that iOS users consume more video in terms of bytes. We highlight the significance of partial downloads, showing that

larger transfers are more likely to be aborted, particularly if the content is streaming video.

- **Browser Caching**: We examine browser caching for each device type and show that smartphone browser caches are not as effective as laptop caches. Adding a small browser cache to each device is sufficient to capture most of the achievable savings. Browser cache savings vary widely across users, with benefits from 0 to 80% in terms of both request hits and byte savings.
- **Proxy Caching**: We evaluate the effectiveness of a Web proxy cache for our environment. We show request hit rates from 8 to 40 percent, and bandwidth savings from 5 to 25 percent. We break out caching results per content type, and show that video is not very amenable to caching. In addition, it is important to handle partial downloads properly, otherwise a cache can actually increase bandwidth consumption by a factor of 3.

The remainder of our paper is organized as follows: In Section 2 we describe our proposed device identification methodology and compare it with other approaches. Section 3 presents the Web traffic characteristics from our trace. In Section 4 we examine browser caching in more detail, and in Section 5 the effectiveness of proxy caching. Section 6 surveys related work and we conclude in Section 7.

2. DEVICE CLASSIFICATION

In this Section we describe how we identify the types of devices in our network using packet information and data mining rules.

In a typical wireless enterprise network, DHCP servers are used to dynamically assign IP address to devices. This presents a challenge in identifying devices on the network, since an IP address can be re-used by multiple devices over separate, non-overlapping periods of time. Thus, a device can not uniquely identified solely by an IP address. While a MAC address usually uniquely identifies a device, the MAC address seen in a trace may not correspond to the device that originated the packet, depending on where the sniffer is situated. In our case, the sniffer sits between two gateways, thus the MAC addresses are always that of the next hop routers.

Instead, we additionally capture all traffic from the two DHCP servers that provide IP addresses to the enterprise network. We feed our classification algorithm with data derived from those packets to we perform our identification. Specifically, we perform the following actions: *a)* calculate the DHCP lease to determine a time period that a device is associated with a specific IP address, *b)* calculate the session lengths during which devices actively generate traffic, *c)* uniquely identify the device based on the client MAC address available in the DHCP headers when the device requests an IP address, *d)* classify the device.

		DHCP		HTTP & OUI [22]	
Device	**OS**	#	%	#	%
Laptop	All	2478	79.9	2182	73.6
	Windows	2107	67.9		
	Mac OS X	333	10.7		
	Linux	38	1.2		
Smarphone	All	534	17.2	398	13.4
	iOS	440	14.2		
	Android	54	1.7		
	BlackBerry	37	1.2		
	Windows	3	0.1		
Other	All	52	1.7	484	16.3
	Cisco VoIP	15	0.5	0	0
	Unclassified	37	1.2	65	2.2
	Unknown	-	-	419	14.1
All		3064	100.0	3064	100.0

Table 1: Distribution of Devices in the Trace

We create a set of classification rules based on the arguments that are included in the options of the DHCP Request:

- *Host-Name*: We see if it contains a device specific string. Some cellphones set their host-name to a string that can identify the type of the device. For example, many iOS cellphones have names that follow the pattern of '*-iPhone', where * usually corresponds to a string related to the user.
- *Vendor-Name*: We see if it contains an operating-system specific string. Some OSs include, in their vendor-name, a string that can uniquely identify the OS. For example, most versions of Microsoft Windows include the string 'MSFT'[5].
- *Parameter-Request-List*: We see if it contains specific DHCP options. A DHCP request contains a list of parameters indicating the set of DHCP options that a client is interested in. Some of these options (as well as their combinations and ordering) are unique for each device, as they typically indicate its auto-configuration capabilities.
- *MAC-Address*: We see if the first three bytes of the device MAC address can be associated with a specific vendor, also called the Organization Unique Identifier (OUI). Using the IANA Ethernet assignments [3], we determine the vendor of the interface and then we identify if that vendor can be directly mapped to a specific type of device.

We use association rules to determine regularities between certain of the above values. For example, an association rule can be expressed as follows: given that we have seen a host-name containing the string 'BlackBerry', what is the probability that the vendor-name will contain the string 'BlackBerry' (2-itemset). We select only few of all possible association rules by using breadth-first search, and prune those that are infre-

quent (low support) or have low confidence (similar to *a-priori* algorithm [13]).

The rules that have high confidence in at least one direction ($conf(X \Rightarrow Y)$ and $conf(Y \Rightarrow X)$), and are not contradictory, are broken into their corresponding itemsets X and Y. Those rules are then used for potential classification. For example, [host-name contains Android'] $\Leftrightarrow$ [Parameter-Request-List contains '1 121 33 3 6 28 51 58 59'] happens with confidence 100%. The reverse direction [Parameter-Request-List contains '1 121 33 3 6 28 51 58 59'] $\Leftrightarrow$ [host-name contains Android'] happens with confidence 82.35%, and the remaining 17.63% are related to a device that neither has 'Android' in the host name (e.g., when the user has modified the default host-name) or any other name from another device type. Now a host-name that contains 'Android' or a Parameter-Request-List that contains '1 121 33 3 6 28 51 58 59', can be used to classify Android devices. In other words, we assume no ground-truth but quantify every rule. The appendix includes additional examples as well as some common association rules.

However, there are contradictory rules that contain useful information. For example, we have observed different manufacturers of 'Android' devices contain some unique fields. For these rules, we apply a Bayesian Classifier to identify the device. We classify into the following categories: Windows, Linux, Mac OS X, Other Laptop, Android, Symbian, Blackberry, iOS, Windows Mobile, Other Mobile, Cisco VoIP, and Uncategorized. In addition, one could use our methodology to derive sub-categories of those (iOS version, Android manufacturer, etc.). Table 1 shows the resulting classification from our trace. We use the first identified device, as some of the laptops may run multiple OSs (e.g., multiple boot clients or those running virtual machines).

Table 1 also includes the classification resulting by using the algorithm from [22], which is effectively based on the combination of HTTP User-Agents (UA) and an audit database of OUIs. For comparison, we reconstruct the database using the *client IP address*, the *client MAC address* and the *IP Lease Time* from the DHCP requests. We see that the approach from [22] has a high number of unclassified devices. The reason for this behavior is that classification based on the User-Agent header can cause errors. We have observed software development kids installed on laptops that emulate iPhone browsers (e.g. Mozilla/5.0 (iPhone Simulator; U; CPU iPhone OS 4-2 like Mac OS X; en-us) AppleWebKit/533.17.9), Virtual Machines running on top of a laptop, user-agent strings (e.g. 'Mobile','LG') that are related to applications in laptops and not devices (MobileSyncClient...), or encrypted user agents (e.g., INj**LG**Mo...). All of these can lead to misclassifications.

Our methodology classifies almost 98.3% of devices, compared to 83.7% using the approach from [22]. Those that are listed as "Unclassified" effectively do not belong to any of the above categories. The ones listed as "unknown" are from devices that [22] could not identify due to use of multiple conflicting user agents.

Dates	April 14-May 5, 2011
Client MAC Addresses	2,748
Client IP Addresses	1,895
Total Packets	1,067,846,981
Total Bytes	673 GB
DHCP Packets	1,629,537
HTTP Transfers	30,302,203
HTTP Downloaded Bytes	505 GB
HTTP Accessed Servers	115,977

Table 2: Dataset Properties

Response	iOS	Android	RIM	Windows	MAC OS
HTTP	76.17	85.16	71.38	83.99	77.13
HTTPS	6.39	9.31	28.57	10.24	17.51
Email	1.88	2.26	0	0.28	1.65
SSH	0	0	0	0.04	0.09
Other	14.85	3.19	0.05	5.48	3.62

Table 3: TCP/IP Application Breakdown (%)

3. WEB TRAFFIC

We use traffic from a corporate network collected from the last-hop gateway before the firewall to the Internet. Both internal and external IP addresses are visible. A switch is configured with port mirroring to echo the Internet traffic to the interface of the sniffer. All interfaces are copper Gigabit Ethernet. The gateway is rate limited to 100 Mbps, and the maximum bandwidth observed by the wireless was 30 Mbps. Thus we believe our system has sufficient capacity to capture all wireless traffic[1]. We collect traces from roughly 3000 unique wireless users divided in 5 subnets. Table 2 summarizes the details of the trace we captured.

We observe that almost 90% of the packets are from TCP, and the majority of those are related to HTTP protocol. Table 3 shows the breakdown of different TCP services for our trace. We see across all devices, at least 75% of TCP traffic is HTTP. Since the focus of our paper is on web behavior, we begin by briefly describing how we extract HTTP behavior from raw captured packets.

3.1 Methodology

Our overall approach is to first reconstruct the TCP streams from the component packets, and if the stream is HTTP, parse the stream for the relevant HTTP information (e.g., methods, URLs, request headers, response

[1]Our sniffer machine is an 2.33 GHz 8-processor Intel Xeon with 10 GB of RAM and 11 TB of disk. It runs Linux RHEL 5.5 with RedHat's patched 2.6.18 kernel. Our software was developed using the libpcap that comes with RHEL 5.5, version 0.9.4-15.

Response	iOS	Android	RIM	Windows	MAC OS
GET	92.75	96.21	95.30	95.47	95.55
POST	7.10	3.77	4.77	4.44	4.37
HEAD	0.14	0.01	0.23	0.09	0.07
PUT	0.01	0.01	0.00	0.00	0.02

Table 4: HTTP Request Method (%)

headers, etc.).

For TCP reassembly, we chose to implement our own routines based on the RFC specification [7], because some of the available tools could not satisfy our requirements. For example, Libnids [4] allows only a fixed number of simultaneous active TCP streams, whereas our code is limited only by available memory. Moreover, Libnids produces flow information only only if a complete connection sequence is observed (*SYN* and *FIN* or *RST* packets have been exchanged). However, in our trace of mobile devices, a full connection sequence is not always observed. This happens in 11.3% of the connections in our trace. We deal with this by timing out connections that do not generate any packets for over two minutes. If a connection is idle for more than that period, we consider it closed.

At the HTTP layer, we reconstruct the Web object by parsing the stream and calculate metrics such as the actual bytes transferred. In fact, the actual bytes transferred is not always available in the HTTP request or response headers for the following reasons. First, many HTTP connections are aborted before the whole object gets completely downloaded [20]. Second, HTTP 1.1 introduced a special type of object encoding, called *Chunked-Encoding*, in which response headers do not include the content length of the object (this is useful for transfers where the server does not know in advance the full size of the object). We thus calculate bytes transferred based on the actual unique (non-duplicate) packets seen. The Bro IDS [2] is another engine that can perform HTTP object reconstruction. Bro does not report the object transferred sizes, but the actual object size (after decompression). Hence, we developed our own HTTP object reconstruction process.

3.2 HTTP Traffic

The approach we follow for analyzing the HTTP data is to first present our main observations and then dig deeper to elaborate as appropriate.

3.2.1 Requests and Responses

Table 4 presents the breakdown of HTTP request methods in the trace. Only methods that exceed a certain threshold ($> 0.01\%$) are listed. Methods that fall below that threshold (e.g., OPTION, DELETE, etc.) are omitted. While the distribution of request methods is broadly similar across devices, iOS devices use a larger percentage of POST requests than the other devices do. POST requests are used when the user sends information to the server.

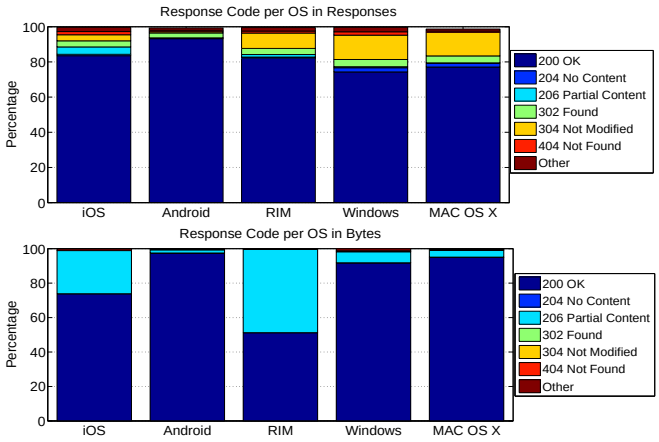

Figure 1: Response Code per Operating System

Response	iOS	Android	RIM	Windows	MAC OS
If-Modified	3.97	36.49	0.37	16.07	16.77
If-None-Match	1.62	28.40	0.05	8.53	8.49
If-Match	0	0	0	0.03	0
If-Range	0	0.07	0	0.08	0

Table 5: Conditional GET Requests (%)

We classify applications in iOS by using the HTTP Request *User-Agent* header. We observe that the top 10 applications in iOS generate 80% of the POST messages. Of those 10, 6 of them come pre-installed in iOS devices (Apple Maps, Stocks, Weather, iTunes and Browser).

Figure 1 provides an overview of the response codes per device type. We observe that Windows and MAC OS X laptops generate a higher fraction of 304 (Not Modified) responses compared to the smartphones. This code is typically returned in response to a client generating a conditional GET request, which validates a client's cached copy of a document which may not be current. If the copy is up-to-date, a 304 response is returned, with no actual data transferred; if not, the full object is downloaded and a 200 response is generated. Thus, 304 responses are an indication of the browser caching activity at each of the devices. Given the lack of 304 response in the smartphones, Figure 1 suggests that browser caches on the smartphones are not as effective or sophisticated as those on the laptops.

We also identify the conditional GET requests that are responsible for the *304 Not Modified* responses. Our results are summarized in Table 5. We see that laptops make significantly more use of these headers than smartphones do. This indicates that laptops have more sophisticated browser caching features, despite the fact that smartphones may be ostensibly using the same browser code base. We examine this in more detail in the next section.

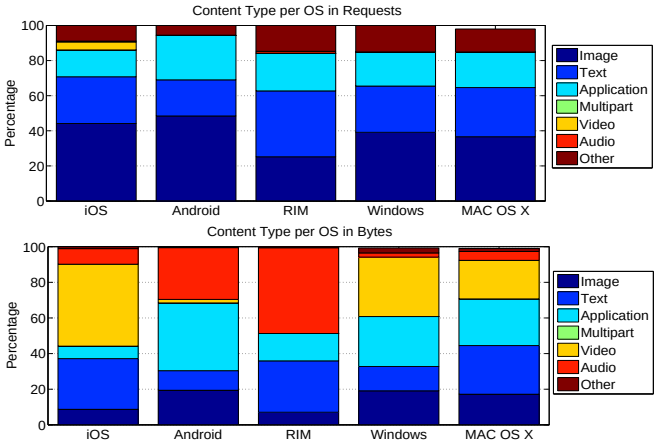

Figure 2: Content Types per Operating System

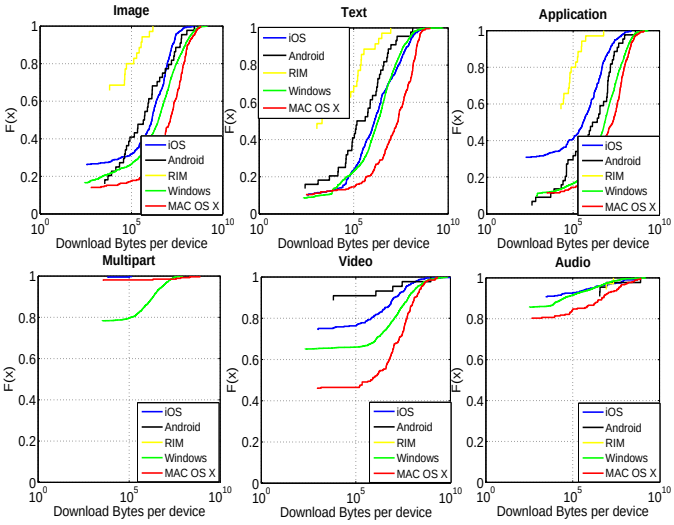

Figure 3: CDF of downloaded bytes per content type

In addition, a significant fraction of bytes downloaded by iOS and BlackBerrys are returned using 206 (Partial Content) responses. This code is used for large objects that must be chunked. Most of those objects have the same URL. In these cases, identifying the unique requests requires processing also the *Range:* field of the HTTP request.

We next examine what kinds of content are downloaded by smartphones and laptops. Figure 2 shows the relative percentages of the various content types requested in the trace, both in terms of requests and in bytes. Across all platforms, a minority of requests for multimedia content (video and audio) generate the majority of the bytes transferred. Nonetheless, among the iOS requests, a higher number of video requests is associated with video objects. In addition, 55% of the iOS bytes downloaded are multimedia, which is higher than seen on the other devices. There is also a significant portion of BlackBerry traffic that is associated with Audio. Finally, the distribution of content types on Windows and MAC OS X laptops are broadly similar.

We next identify, for each content type, how many devices download that content type. We do this to determine if the differences observed above are due to outliers or if it is a behavior broadly characteristic of each device type. Figure 3 shows the cumulative distribution of downloaded bytes for each content type on a per-client basis. The starting point of each CDF indicates the relative percentage of the devices that *do not use* that content type. For example, there is a unique outlier in the BlackBerry trace for audio traffic, which requests 40% of the total HTTP traffic across *all* BlackBerries. Moreover, video traffic is more popular across laptops (Windows and MAC OS X) than it is for smartphones. For example, 55% of the Windows laptops download at least one video object, whereas this happens only for 20% of the iOS devices. The median laptop device downloads almost 100 times more video bytes than an iOS device, and significantly more than the median Android device. Thus, iOS users access video content relatively more frequently than Android users.

3.2.2 *Multimedia Analysis*

Figures 1 and 2 showed that, respectively, there are a large fraction of 206 responses in our trace and that a large number of bytes are due to multimedia traffic. Here we examine the interaction of these two factors to further understand the role of multimedia content.

The stacked bars of Figure 4 breaks down the distribution of response codes across the devices for video and audio in terms of number of requests and bytes transferred. Only the 200, 204, 206, and 304 response codes are given since they are the only ones seen for the multimedia content. In addition, BlackBerry devices did not generate any video traffic. The 304 code was rarely returned in video requests generated by laptops. This suggests that there is little locality in video content, since the browser caches do not re-validate them using if-modified-since requests. Moreover, the majority of the responses in iOS and Androids are 206 for multimedia content. This is a unique behavior especially for iOS devices, since around 20% of the users download video traffic.

To further understand the role of 206 responses, we examine the User-Agents headers in iOS requests. This header, while not completely reliable, generally indicates the application that is requesting content. We observe that two User-Agents, namely *AppleCoreMedia* and *iTunes*, generate almost 98% of the video traffic. We also observe that there is a small portion of video traffic from MAC OS X that belongs to a similar User-Agent, namely *CoreMedia*. Figure 5 shows the breakdown of the application usage in iOS and MAC OS X

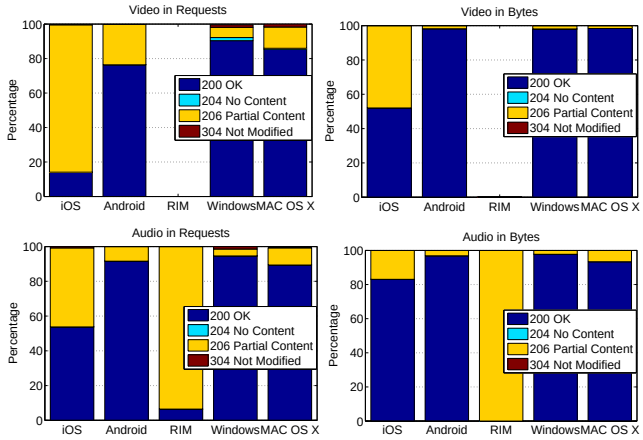

Figure 4: Response Codes for Multimedia Traffic across all types of devices

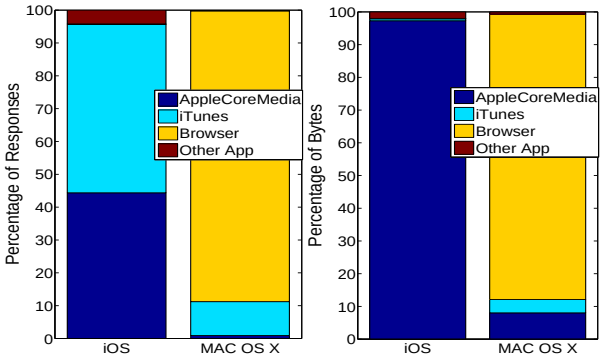

Figure 5: Applications that generate multimedia traffic in iOS vs MAC OS X (requests, bytes)

for multimedia traffic. Note that the total video content downloaded is 23GB and 31 GB, for iOS and MAC OS X respectively.

The majority of the multimedia requests in iOS devices are generated by *AppleCoreMedia* and those requests contribute almost half of the bytes. Moreover, a very small portion of *iTunes* requests contribute the other half of the bytes, indicating that the size of each object is larger. However, *iTunes* is used by only 2% of the users. On the other hand, in laptops such as MAC OS X, most of the video traffic is generated by the Web browser, and only 5% of the requests are generated by *CoreMedia* and even fewer are related to *iTunes*.

To investigate this unique behavior we isolated the DNS names of the *AppleCoreMedia* in order to get a view on the services that use this iOS component. We observe at least 82 services. The most used service was googlevideo (at 65%). Thus, our results indicate that *AppleCoreMedia* is a component that plays the role of the media player in iOS devices [1].

Given that the majority of the requests are from *AppleCodeMedia*, but contribute only half of the bytes,

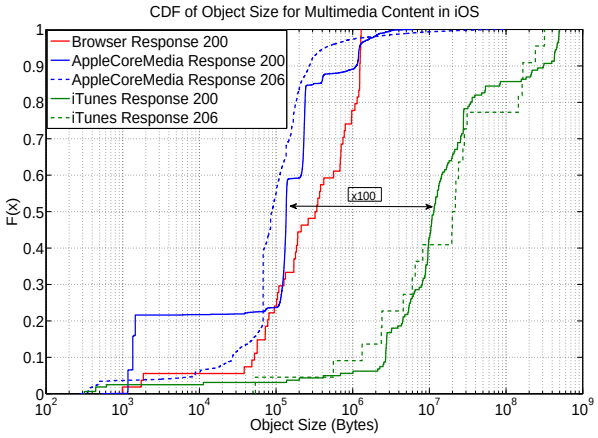

Figure 6: Cumulative Distribution of the iOS multimedia traffic per object size

and that there are a high number of 206 responses, we turn our attention to the unique properties of this iOS component. Figure 6 shows the cumulative distribution per object size of the multimedia content in iOS. We observe that the median object handled by *AppleCoreMedia* is 100 times smaller than the median object for *iTunes*. Moreover, almost 95% of the *AppleCoreMedia* objects are $<$ 1MB. Finally, we see no relationship between 200 and 206 responses and object size. This suggests the iOS component will chunk the videos using the 206 response, but only if they exceed a specific size.

Thus, we see that the AppleCoreMedia iOS component plays a significant role in smartphone video streaming. We finally check whether this is related to the *Adaptive Streaming* technique that Apple has introduced in RFC [26]. We break the video content type to the sub-content types, this is because the there are only few MIME video types that are used for *Adaptive Streaming*. We observe that only 13% of the video requests and 7% of the video is generated by MP2T (MPEG-2 Transport stream) video type, which is supported by the *Adaptive Streaming* RFC. Hence the properties that we have observed for the iOS media player is a generalized phenomenon and not limited to a specific technology.

3.2.3 *Aborted Transfers*

In this subsection, we investigate the behavior of the objects in relation to the size of the object. For presentation purposes, we group the data in laptops vs smartphones.

Figure 7 shows two scatter plots of document size versus how many bytes were actually downloaded, for laptops and smartphones respectively. Each point is a download from our trace. Note both axes are in log scale. If downloads were never aborted, each graph

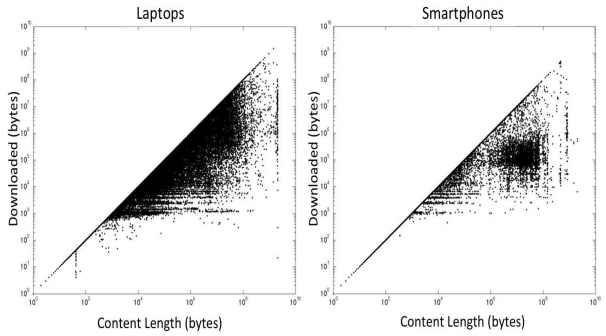

Figure 7: Transfer size versus content length

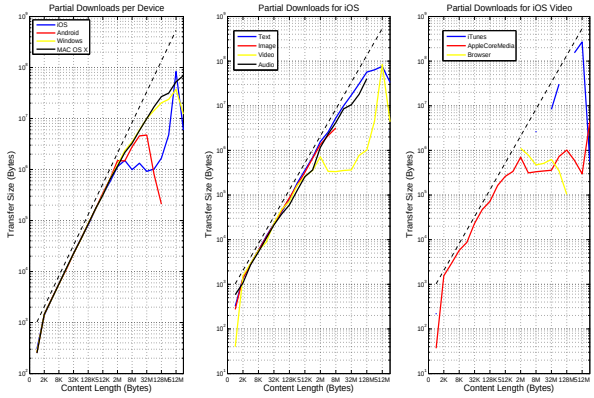

Figure 8: HTTP downloads per device type, iOS content type, and iOS application

would show a diagonal with a slope of one. However, we see that many transfers are significantly less than the corresponding document's content length, indicating that they are frequently aborted. The scatter plots visually show a distinct difference in behavior between smartphones and laptops. For large object sizes, around 5MB to 100MB, the scatter plots indicate that smartphones tend to abort the download much more frequently than laptops.

While interesting, it is difficult to discern the relative behavior of the devices from the scatter plots, particularly since there are many more laptops than smartphones, and consequently more data points. To illustrate this issue further, we normalize based on the average transfer size M for a document of size N, based on all transfers for documents of size N. We call this the *download ratio*. The results are shown in Figure 8. Again, if all downloads were completed, one would see a diagonal $Y = X$, which here we include for convenience. Again, note the log scales. We see that, on average, the full document is not downloaded, and that for sizes greater than 4 MB, the behavior of the iOS devices diverges from the other devices. iOS devices are significantly less likely to download full objects than other devices. The second subplot (8 (b)) looks at the download ratio for iOS devices broken down by content type. We see here that videos are much less likely to be fully downloaded than other content types. In the third subplot, we look exclusively at video downloads on iOS and distinguish by the application generating the request. We see that streaming content, requested by the *AppleCoreMedia* component, is frequently aborted. While there very few *iTunes* transactions, most of them are downloaded fully.

These results indicate that aborted downloads happen mainly for streaming video requests. This also suggests that proxy or browser caching policies [27] that would cache the object only if it gets fully downloaded would probably fail to provide savings for video traffic in iOS devices. We examine this issue in more detail in Sections 4 and 5.

	iOS	Android	RIM	Windows	MAC OS
Requests (K)	283.0	70.9	1.1	10087.7	3145.3
Objects (K)	261.4	70.6	1.0	9978.0	3137.2
Difference	7.7%	0.4%	9%	1%	0.2%

Table 6: Differences between Requests and Objects

3.2.4 *Popularity*

Popularity is frequently calculated from Web traces as a means of illustrating document access frequency and its implications for caching of Web objects. While most prior works have focused on the distributions of accesses to *objects*, we see a large proportion of requests that are returned as chunks with the *206 partial content* response code. We thus believe the appropriate way to view Web metrics is to include this feature and account for its behavior.

For example, in the context of popularity, consider the case of two successive requests for content to the same video but for successive distinct chunks, as is commonly the case. Ignoring the chunk ranges would count those requests as two references to the same object, rather than one request, thus over-estimating the popularity of that object. Since media traffic in smartphones are mostly served by 206 responses, this issue is becoming more prominent. Here we show its implications for popularity.

Table 6 shows the difference between calculating requests versus objects. Since iOS and BlackBerry devices receive a lot of media with 206 responses, there is a much bigger number of requests compared to objects. These extra requests are due to chunked objects, where ranges must be taken into account to identify accesses to different parts of the object.

Figure 9 depicts the object and request distributions. We perform a statistical fit using several distributions

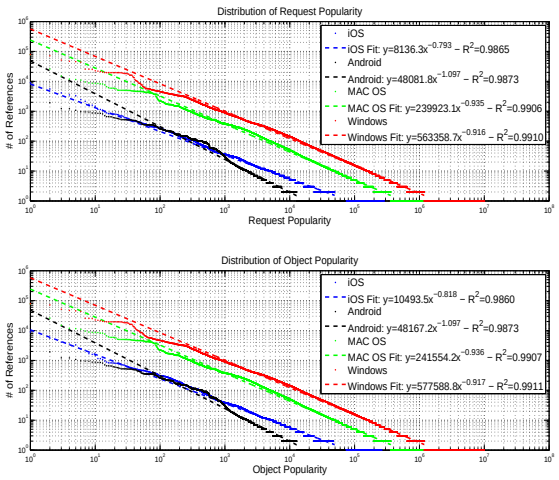

Figure 9: Distribution of objects ranked in number of references

(Zipf Law, Stretched Exponential, Zipf Law with exponential cutoff). We find that none of them are able to satisfactory fit the entire range of requests or objects in our dataset. This is an artifact of the very long tail of requests with just only reference For example, in iOS, 70% of requests are to a unique object range, implying a hit rate no greater than 30%, whereas ignoring the range requests gives a set of 60% of objects, implying a 40% upper bound.

Nonetheless, the head of the distribution, follows a Zipf Law. Figure 9 shows that the fit has $R^2 > 0.987$ across all devices. This indicates that caching the most most common objects would be enough to capture most of the request hit rate.

4. BROWSER CACHE EMULATION

Virtually all browsers use local caches to improve the latency of requests that can hit in the cache and to reduce the network bandwidth consumed. In the mobile environment, this can also reduce energy consumption and improve battery life. In Section 3.2.1, Figure 1 and Table 5 provided evidence that suggested that the browser caches on the smartphones were not as sophisticated as those on the laptops. In this Section, we explore this area further.

To examine the effectiveness of a local browser cache, we evaluate them by replaying the trace for each device through an additional simulated cache dedicated solely to that device. If the addition of such a cache does not provide any benefits, it suggests the the browser cache is performing well. If it does provide benefits, it suggests that the browser cache is either too small or not exploiting all the available caching features of HTTP. Our emulated browser cache follows the cacheability re-

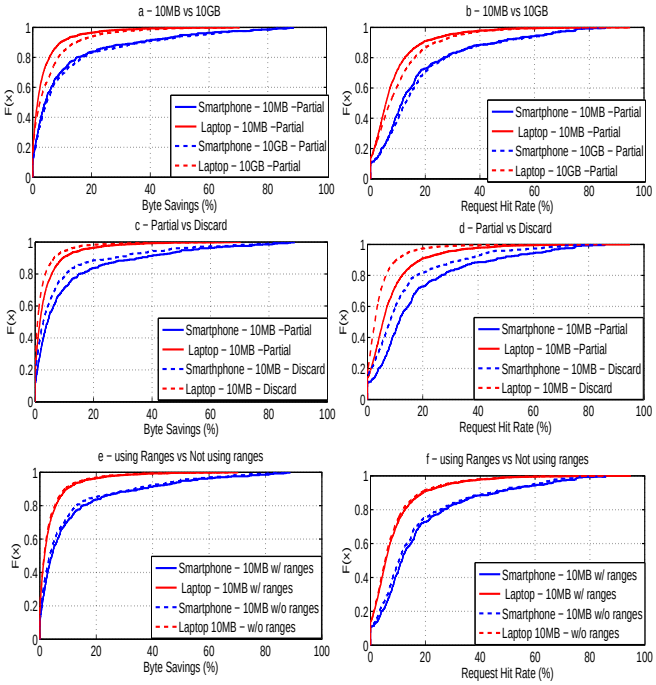

Figure 10: A comparison of browser cache hit rate in different dimensions: (a) persistent storage space, (b) caching policy, (c) handling ranges.

quirements from RFCs 2616 [8] and 2965 [9]. We use an LRU replacement policy, both for simplicity and because it is the default in the well-known Squid [27] proxy cache. We describe our cache implementation in more detail in the appendix, including several optimizations for video caching and partial downloads.

Since smartphones experience ranged requests, aborted transfers, and partial downloads, we consider several policies in order to evaluate the performance of a a dedicated browser cache:

- *Discard:* Partially downloaded files are excluded from reuse. The cache simply discards all data that has been already downloaded. This is the default policy for Squid.
- *Full Download:* The cache continues downloading the entire object, even when the user aborts the transfer.
- *Conditional Download:* The entire object is only downloaded when X percentage of the object has already been requested by the user. This is available in Squid as a configurable option.
- *Partial Download:* The cache stores only the portion of the object that has already been requested by the user.

Note that the *Cache-Control* header in an HTTP response is handled differently in browser caches, as docu-

Location	Cacheability in	iOS	Android	RIM	Windows	MAC OS
Browser	**Bytes**	86.74	70.27	51.83	81.75	82.43
	Requests	70.25	68.85	55.87	63.43	65.51
Proxy	**Bytes**	85.20	41.10	48.42	54.74	67.21
	Requests	64.70	65.20	47.86	56.20	57.42

Table 7: Cacheable Bytes and Requests per OS

ments characterized as *private* are considered cacheable. Table 7 shows the differences in cacheability across platforms, from the point of view of the browser cache and a proxy cache. We observe that the majority of the traffic in iOS, Windows and MAC OS X is considered cacheable. Interestingly, the percentage of content that is cacheable decreases significantly across most platforms when proxy caching rules are used, with the exception of iPhones.

Since some users generate significantly more traffic than others, simply reporting an average benefit for browser caches would not sufficiently describe their behavior. Instead, we show the cumulative distribution of the benefits.

We begin with Figures 10 (a) and 10 (b), which show the distribution of the improvement provided by an additional browser cache in terms of bytes and requests respectively. In these graphs, the partial download policy is used, and comparison is made between a 10 MB cache and a 10 GB cache. We see several points from the graphs. First, the distribution of benefit varies widely across the devices, based on their individual access behavior. Second, the browser cache consistently provides more benefit to the smartphones than to the laptops, adding to the evidence that the smartphone caches as not as effective as the laptop caches. Third, we see little difference between the 10 MB and 10 GB curves. This suggests that the reason the smartphone caches do worse is because the caches are not sophisticated enough, rather than because they are too small.

Figures 10 (c) and 10 (d) compare the partial and discard policies, again for bytes and requests respectively. We see the partial policy consistently outperforms the discard policy, sometimes by up to 50%. Thus a browser cache should not discard the object even if it is only partially downloaded.

Figures 10 (e) and 10(f) compare a browser cache that supports range offsets with one that does not. The RFC states that if the cache cannot process ranges, then it must not make any effort to cache range requests. We observe only a very slight improvement in performance for those caches that process ranges. Thus, if the implementation complexity to add the functionality that processes ranges is high, then it may not worth caching range requests.

We summarize our conclusions as follows: *a)* laptop browser cache implementations are more sophisticated than smartphones; *b)* A small increase in the browser cache is sufficient to capture most of the savings; *c)* A browser cache should be able to handle partially downloaded objects; *d)* A browser cache does not necessarily need to handle ranges, as the difference in savings is small; *e)* Looking only at average savings across all devices does not provide sufficient detail, as traffic distributions are skewed.

5. PROXY CACHE EMULATION

Given the predominance of Web traffic in our trace, it is useful to determine how effective a Web proxy cache would be in our environment, both in terms of bandwidth savings and in reduced response time. We use our cache emulation software described earlier in Section 4 with the following configurations: (a) we implement the proxy cache emulation across all traffic, and not per device and (b) we follow the cacheability rules for proxy caching rather than for the browser. We examine the same caching policies described in Section 4.

One challenge in properly emulating a proxy cache's behavior is how to account for objects in the cache that are not fresh. Even if an object is not fresh, it could still be the same as the object on the origin server. Thus when the proxy tries to retrieve it via a conditional GET request, the object could be returned either in full via a 200 response or simply re-validated via a 304 response. To quantify the impact of a proxy cache, both cases need to be considered, thus we take the following approach. In one configuration, *always revalidated*, a cached object is always assumed valid with the object at the server. In that case, the cache only revalidates the object and receives a 304 response. In the second configuration, *never revalidated*, the object is always assumed stale and the full object is retrieved via a 200 response. The first approach is the most optimistic case in terms of savings, while the second is the least optimistic one. In reality, of course, the actual behavior will lie between the two. To be conservative, we always evaluate using the second approach, unless otherwise stated.

Figure 11 shows the byte savings and request hit rates of laptops and smartphones, using the two assumptions, over a range of cache sizes. We note several observations. First, laptops tend to have better object and byte hit rates than smartphones. This may be a result of our trace, which has many more laptops than smartphones. Second, benefits achieve diminishing returns at around 100 GB, a relatively small size. Thus size, and consequently replacement policy, is not a significant issue. Finally, the validation assumption makes a large difference in the estimate of the byte savings. This is

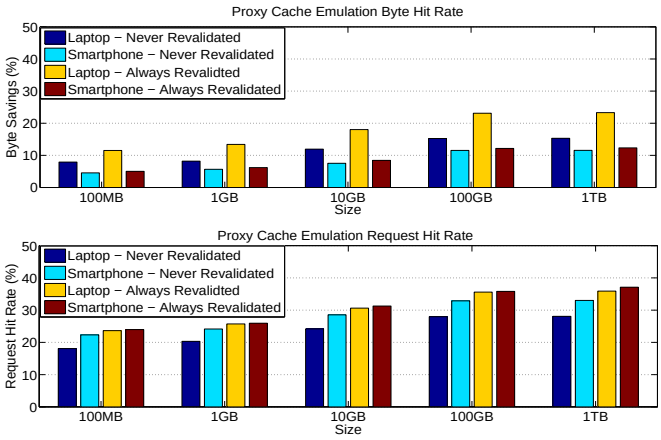

Figure 11: Proxy cache hit rate with several revalidation strategies and persistent storage sizes.

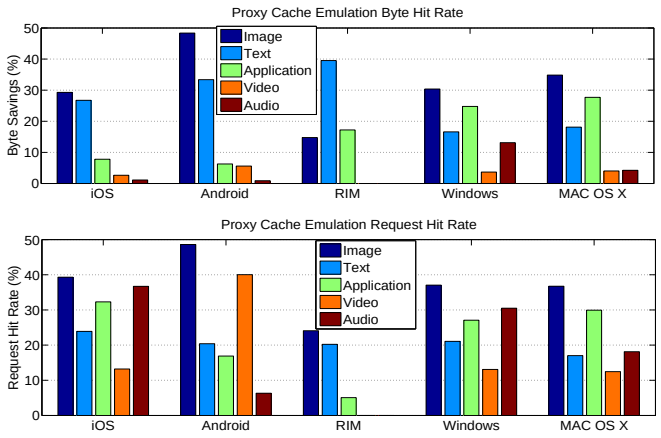

Figure 12: Proxy cache savings for infinite sized cache per content type for each operating system

Policy	Hit Rate	Smartphones	Laptops
Partial with URL & Range	**Byte**	11.34%	17.28%
	Request	38.42%	30.55%
Partial with URL & No Range	**Byte**	10.78%	16.32%
	Request	34.55%	27.87%

Table 8: Proxy cache hit rate for infinite sized storage space with and without handling Range Offsets

particularly true for laptops, where it can make as much of a difference as 50%. This happens because the object freshness for laptops is typically smaller than for smartphones, as observed by the *max-age* header values, which tend to be larger in smartphone responses.

Table 8 shows the benefits of a proxy cache comparing when range offsets are handled or not. The savings for both cases are similar, indicating that caching the range requests does not add a significant amount of savings.

We deepens our analysis by looking at the content-type of the Web traffic and evaluate how proxy caching benefits each content-type. Figure 12 shows the benefits of a proxy cache for each content type across the range of devices, for both requests and byte savings.

Images and text have reasonably good byte hit rates, usually $> 30\%$ across most of the devices. However, we observe video hit rates are very small. This may be an artifact of the few requests for video content in our trace, which means little redundancy can be identified at the object level. However, it suggests caching videos may not yield much savings.

Figure 13 (a) shows the byte savings for the various policies described in Section 4 that deal with partially downloaded objects. A partially downloaded object is cached in its entirety if more than $X\%$ of its size has been already downloaded by a user. The 0% case corresponds to the *full-download* policy, i.e., the cache downloads the full object independently of how much it has been already downloaded. The 100% case is the *discard* policy, i.e., any partially downloaded object is not cached. Any value in-between corresponds to the *conditional-download* policy. The graph shows that the full-download policy is actually detrimental, requesting content that is never viewed, particularly for smartphones.

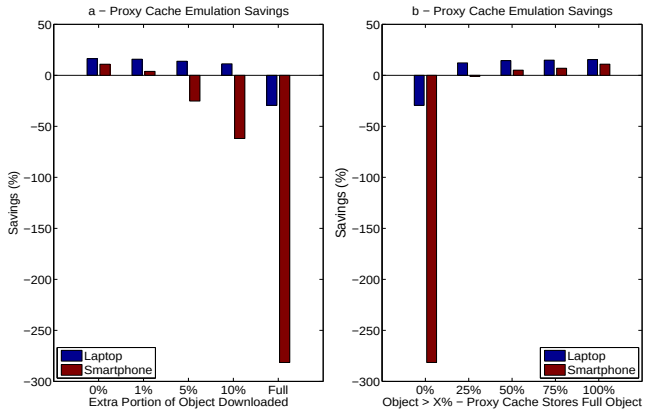

Figure 13: (a) Conditional Caching for Partial Downloads and (b) Effect of Cache Buffering on Savings

Moreover, savings for smartphone traffic is more sensitive to the actual percentage of downloaded bytes after which an object becomes cacheable. This is because large objects are more likely to be partially downloaded by smartphones, as shown in Figure 13 (b), which results in more downloaded data for the full-download and the conditional-download policies. The discard policy provides savings very close to the optimal one, indicating that partially downloaded objects are not likely to be accessed more than once.

Figure 13 (b) shows the impact that additionally downloaded data, over what users ask for, has on bytes savings. An HTTP proxy cache, conceptually, acts as a buffer between the clients and the servers. It usually

downloads more data than users request, typically because it is better connected with the servers. This is especially true for wireless clients, where packet losses at the wireless interface make TCP connections between the clients and the cache slower than the connection between the cache and the servers. To evaluate this effect, we consider the case where the cache downloads $X\%$ more data than the data downloaded by the clients (up to the maximum size of the object). The first set of bars correspond to the optimal case, i.e., when the cache stores only the amount of bytes that the user is requesting, while the last effectively becomes the full-download policy. We see that the smartphone traffic is especially sensitive to these buffering issues. Again, partially downloaded objects are the main reason that these effects are more prominent in smartphone traffic than for laptops.

6. RELATED WORK

The following papers [18, 19, 17, 22, 25] have studied the network traffic generated by smartphone devices. More specifically, Falaki *et al.* [18, 19] characterized smartphone usage by installing custom logging software in 255 mobile devices. They focus on the diversity among smartphone users in terms of session times, traffic generated, energy consumed and application usage. Maier *et al.* [25] identified devices using a combination of IP TTL and *User-Agents*, and investigated smartphone traffic characteristics using full traces. Erman *et al.* [17] identified devices according to the *User-Agents* string only, and captured TCP flow properties. They provided an overview of the video delivery in smartphone devices. Finally, Gember *et al.* [22] cross-validated the *User-Agent* outcome with the Organization Unique Identifier from the packet's MAC address.

In our work, we use a DHCP OS fingerprinting algorithm to identify the devices. The proposed algorithm uses some of the fields reported in [24] and identifies association rules. Since the algorithm is based on the DHCP traffic, we are able to classify a device as soon as it receives an IP address; hence before sending any other packet. Moreover, we do not assume any ground truth in terms of classification. We identify association rules and quantify the classification outcome. We show that our methodlogy can identify 98% of the devices, compared to [22], which could identify 83% of the devices in our trace. The difference in the classification outcome is because UAs (a) can be encrypted, (b) may be related to an emulation enviroment (e.g. iPhone emulator), or (c) can be application specific. The later is more apparent in smartphone devices.

In terms of the actual traffic, our work also goes beyond a smartphone versus laptop analysis for HTTP content [22, 25], or video delivery study in smartphone devices [17, 21]. We break the network based on device type (iOS, Android, BlackBerry, Windows, MAC OS X), and showcase that each device has unique networking patterns. We study the impact of all applications in the network and extend our work into caching. We show that caching in mobile devices needs to take into account some new properties, which were not apparent in the earlier wired literature [11, 20, 14, 23]. For instance, storing the full object, as per the Squid default policies, may result in a networking overload. Finally, we investigate the impact of downloading policies in browser caches.

7. CONCLUSION

The explosive growth of smartphones is expected to have significant impact on future network architectures and the technologies that will be deployed to deal with the increased traffic demands. Understanding the material differences between smartphone and laptop traffic patterns is important both to enterprise network administrators and to cellular network providers.

In this paper we investigate the diverse traffic characteristics of mobile devices and the effectiveness of browser and proxy caching. Our results show that smartphone manufacturers should consider adding a small-sized persistent browser cache. Moreover, conventional caching rules may be reasonable for laptop traffic, but may result in adverse bandwidth savings when smartphones are used. smartphone devices. We show that the optimal policy for partially downloaded objects, which downloads and caches only the portion of the object that has been requested by the clients, is robust to both smartphone and laptop generated traffic. Therefore, implementing an optimal policy, while not trivial, should be a high priority of any HTTP traffic optimization product for wireless Internet access.

8. REFERENCES

[1] Applecoremedia. `http://developer.apple.com/library/IOs/#documentation/CoreMedia/Reference/CoreMediaFramework/_index.html`.
[2] The bro network security monitor. `http://bro-ids.org/`.
[3] Ethernet number reqistration. `http://www.iana.org/assignments/ethernet-numbers`.
[4] Libnids. `http://libnids.sourceforge.net/`.
[5] Microsoft DHCP Vendor and User Classes. `http://support.microsoft.com/kb/266675`.
[6] Video cache squid plugin. `cachevideos.com/`.
[7] RFC 793 TCP transmission control protocol. `http://www.ietf.org/rfc/rfc793.txt`, Sep 1991.
[8] RFC 2616 hypertext transfer protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, June 1999.
[9] RFC 2965 HTTP state management mechanism. `http://www.ietf.org/rfc/rfc2965.txt`, Oct 2000.
[10] *Sandvine Global Internet Phenomena Report*. Spring, 2011.
[11] M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, and S. Williams. Removal policies in network caches for world-wide Web documents. In *SIGCOMM*, 1996.
[12] B. Ager, F. Schneider, J. Kim, and A. Feldmann. Revisiting cacheability in times of user generated content. In

INFOCOM. IEEE, 2010.
[13] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, 1994.
[14] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *INFOCOM*. IEEE, 2002.
[15] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck. To Cache or not to Cache: The 3G case. *Internet Computing, IEEE*, 15(2):27–34, 2011.
[16] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, and O. Spatscheck. Network-aware forward caching. In *WWW*. ACM, 2009.
[17] J. Erman, A. Gerber, K. Ramakrishnan, S. Sen, and O. Spatscheck. Over the top video: The gorilla in cellular networks. In *IMC*. ACM, 2011.
[18] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *IMC*. ACM, 2010.
[19] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *MobiSys*. ACM, 2010.
[20] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *INFOCOM*. IEEE, 1999.
[21] A. Finamore, M. Mellia, M. Munafo, R. Torres, and S. Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *IMC*, 2011.
[22] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus wi-fi networks. In *Passive and Active Measurement*, pages 173–183. Springer, 2011.
[23] S. Jin, A. Bestavros, and A. Iyengar. Network-aware partial caching for Internet streaming media. *Multimedia Systems*, 9(4):386–396, 2003.
[24] E. Kollman. Chatter on the wire: A look at DHCP traffc. http://myweb.cableone.net/xnih/download/chatter-dhcp.pdf, 2007.
[25] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Passive and Active Measurement*. Springer, 2010.
[26] P. R. and M. W. Ed. HTTP live streaming. `http://tools.ietf.org/html/draft-pantos-http-live-streaming-07`.
[27] The Squid Project. Web proxy caching. `www.squid-cache.org`.
[28] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *IMC*. ACM, 2011.

9. APPENDIX

9.1 Classification

Table 9 summarizes some common association rules produced with the DHCP classification process. To quantify the confidence of the rules, we used standard data mining metrics: Support $supp(X)$ is defined as the portion of all devices that satisfy the rule x. Confidence $conf(X \Rightarrow Y)$ of an association rule $X \Rightarrow Y$ is defined as $supp(X \cap Y)/supp(X)$, where $supp(X \cap Y)$ is the support of rule $X \wedge Y$, namely, the portion of all devices that satisfy both rule X and Y.

Note that all the association rules that were produced happen to be non conflicting for our data. In other words, two independently produced association rules, when given the same input, will always agree on the OS and device type classification. The main reason is that most of the devices, given their OS, tend to have unique Parameter-Request-List. For example, the difference between MAC OS X and iOS devices is in the last three options of the parameter-list, which are options 44, 46 and 47. In fact, these three options correspond to NetBIOS options, and they are present in all the Windows, Mac OS X and Linux laptops, but not in any of the smartphone devices. Similarly, option 150, relevant to TFTP server information, is only present in DHCP request generated by Cisco VoIP phones.

9.1.1 Multiple OS Machines

In a multiple boot client, the same MAC address maybe related to different OS. That is because DHCP protocol is OS dependent the physical address may remain the same when the user is booting the system with different OSs. However, some of the DHCP options will be OS dependent and the end host will request a new IP address once it boots with a different OS. We identified several users that load at different time instances in different OSs. This was done by continuously observing the DHCP requests.

Moreover, we identified several Virtual Machines (VM). That is OSs that have been assigned an IP address and whose MAC address OUI is related to the VM Hypervisor.

9.2 HTTP Web Site Popularity

We also identified the top web sites that the clients visited. We present the top 20 sites in tables 10 and 11, in bytes and requests respectively. There are two observations regarding the user behavior. First, we see that smartphones make relatively more references to the top 20 sites than laptops, suggesting a tighter pattern of reference. The second, is that in smartphones we see that the top 80% of the websites in terms of bytes are related to the top 20 applications. Those observations could be relevant, for example, for a CDN service targeted for smartphones.

9.3 Proxy Cache Details

9.3.1 Caching Metrics

We use the following three metrics to assess cache effectiveness. The first metric is the *request hit rate*, i.e., the percentage of requests that are served by the cache. It captures the expected reductions in request response times. The second metric is the *byte hit rate*, defined as the percentage of the bytes that are served by the cache. It shows the amount of traffic that can be served locally without going to the origin server. Finally, the third metric is the actual *savings*, i.e., the traffic reduction achieved through caching. Note that savings can be negative, when caching policies generate more traffic

		Association Rule ($Rule1 \Rightarrow Rule2$)		
Device	**OS**	**Rule 1**	**Rule 2**	**Confidence**
Laptop	Windows	Vendor contains 'MSFT'	List: 1 15 3 6 44 46 47 31 33 249 43	99.18%
		List: 1 15 3 6 44 46 47 31 33 249 43	Vendor contains 'MSFT'	93.00%
	Mac OS X	Host contains 'Macbook'	List: 1 3 6 15 119 252 44 46 47	100.00%
		Host contains 'Macbook'	MAC-Vendor = Apple	100.00%
		List: 1 3 6 15 119 95 252 44 46 47	MAC-Vendor = Apple	100.00%
	Linux	Host contains 'Ubuntu'	List: 1 28 2 3 15 6 119 12 44 47 26 121 42	100.00%
Smartphone	iOS	Host contains 'iPhone|iPad|iPod'	List:1 3 6 15 119 252	100.00%
		Host contains 'iPhone|iPad|iPod'	MAC-Vendor = Apple	100.00%
		List: 1 3 6 15 119 252	MAC-Vendor = Apple	100.00%
	Android	Host contains 'Android'	Vendor contains 'dhcpcd'	100.00%
		Host contains 'Android'	List: 1 121 33 3 6 28 51 58 59	100.00%
		List: 1 121 33 3 6 28 51 58 59	Vendor contains 'dhcpcd'	100.00%
		List: 1 121 33 3 6 28 51 58 59	Host contains 'Android'	82.35%
		Vendor contains 'dhcpcd'	Host contains 'Android'	60.87%
	BlackBerry	Host contains 'BlackBerry'	Vendor contains 'BlackBerry'	100.00%
		Host contains 'BlackBerry'	List: 1 3 6 15	100.00%
		List: 1 3 6 15	Vendor contains 'BlackBerry'	100.00%
		Host contains 'BlackBerry'	MAC-Vendor = RIM	100.00%
Other	Cisco VoIP	Vendor contains 'Cisco Wireless Phone'	List: 1 3 6 12 15 28 42 66 149 150	100.00%
		List: 1 3 6 12 15 28 42 66 149 150	Vendor contains 'Cisco Wireless Phone'	100.00%

Table 9: Common Association Rules for Device Classification

Laptops	%	**Smartphones**	%
Search Engine	8.51	Online Journal 2	10.30
Social Networking 1	5.89	Search Engine	5.58
Advertising	3.45	Social Networking 1	4.73
Online Journal 1	2.44	Computer Vendor	3.97
CDN 1	1.82	Online Journal 1	3.52
Images Search	1.33	Tracking 2	3.32
News Site	1.12	Advertising	1.67
Analytics	1.11	Undefined IP	1.53
Antivirus Update	1.09	Feeds	1.49
Game	1.04	Video Site 3	1.29
Retail	0.98	Analytics	1.25
Video Site 1	0.89	CDN 1	1.16
Social Networking 2	0.88	Video Site 1	0.85
Tracking 1	0.85	News 3	0.83
Flash Service	0.83	News 4	0.80
News 4	0.81	Social Networking 2	0.79
Advertising 2	0.80	Retail	0.75
Computer Vendor	0.71	Advertising	0.71
Adds	0.66	Images Search	0.64
Online Journal 3	0.56	International	0.62
Total Top 20	35.73	**Total Top 20**	45.81

Table 10: Top 20 Web Sites (Requests)

Laptops	%	**Smartphones**	%
Video Site 1	20.15	Computer Vendor	23.8
Computer Vendor	4.54	Video Site 4	16.34
Search Engine	3.99	Video Site 3	11.03
Antivirus Update	3.89	Video Site 2	4.34
Social Networking 1	2.07	Video Site 1	3.42
Online Radio 1	1.93	Search Engine	3.35
Video Site 2	1.86	Online Radio 1	3.09
Software Update 1	1.79	Online Journal 1	2.27
Software Update 2	1.67	Undefined IP	1.60
Images Search	0.87	TV Streaming	1.46
Advertising 2	0.86	Tracking 2	1.42
OS Download 1	0.79	News 1	1.41
University Media	0.78	File Hosting	1.23
Online Journal 2	0.61	Podcast 2	1.22
File Host	0.49	News 2	0.88
Undefined IP 2	0.47	Social Networking 1	0.82
Video Site 4	0.47	Broadcasting	0.73
News Site	0.43	CDN 1	0.6
OS Download 2	0.42	CDN 2	0.57
Retail	0.41	Online Radio	0.48
Total Top 20	47.82	**Total Top 20**	80.1

Table 11: Top 20 Web Sites (Bytes)

that requested by the users.

9.3.2 Proxy Cache Rules

At a high level, an HTTP proxy caches responses, usually by storing them in a non-volatile memory, and uses them potentially in the future to generate other responses. Cached responses are generated from the locally stored content either without interacting with the origin server or after consulting with it.

RFC 2616 makes explicit: *a)* what HTTP responses cannot be cached, *b)* how to validate the cached HTTP responses, and *c)* how to revalidate cached HTTP responses that are no longer valid. Note that the cacheability rules in the RFC are not inclusive, i.e., HTTP content can be cached even if it is not explicitly defined in RFC 2616, as far as it is not in conflict with any of the rules that define non cacheable content. For example, a later RFC (RFC 2965 [9]) clarifies how requests and responses that contain cookies can be cached. RFC 2616 defines that only responses to requests with *GET*, *POST* or *HEAD* methods are potentially cacheable. All other HTTP methods are not cacheable.

In the absence of *cache-control* headers and the *expires* header, an HTTP 1.1 compliant cache can implement its own caching algorithm for *GET*, *POST* and *HEAD* requests. For example, most popular caches [27] follows an HTTP 1.0 compliant caching algorithm. In other words, in the absence of *cache-control* and *expires*, they will consider requests with '?', 'cgi-bin', in the URL as uncacheable. Similarly for the POST messages. On the other hand, in the presence of *cache-control* and *expires*, cacheability is dictated by those arguments only. There are also some additional rules that are left open for implementation. In order to derive the maximum potential savings: *a)* responses with no validator or expiration time are cacheable (RFC 2616 Sec. 13.4 defines that they may not be cached), and *b)* responses with codes 200, 203, 206 that include *range* or *content-range* headers, 300, 301 or 410 are cacheable (RFC 2616 Sec. 13.4 defines that they may be cached).

Cookies and Caching: Our emulator does not consider the presence of cookies when deciding whether a response can be cached and whether a request can be served with cached content. This is based on the RFC 2965 which states that *set-cookie* headers can be used in public as well as private objects. It is the server's responsibility to include additional caching directives in the HTTP headers in order to enable or disable caching of HTTP objects with cookies.

Range Requests and Caching: Although HTTP 1.1. defines that is optimal to cache request *content-range* headers, our emulator determines differences in requests based on the reported ranges. Thus a cache hit means that both the URL and the byte ranges need to match.

Exceptions for Video Caching: Currently, many popular video streaming Web sites make their video non-cacheable across multiple users by attaching user-specific information in their video file URLs. For example, YouTube video file URLs include various metadata related to user preferences as well as keys unique to each user. Using the full URL as an object name prevents effective caching given that the same video file results in different URLs when accessed by different users or even when it is downloaded from different CDN servers. For this reason, we have developed a number of exceptions when dealing with major video contributors (YouTube, Google Video, Daily Motion) that allow caching of video content. This feature is available in other proxies such as Squid [6, 27]. More specifically, rather than referring to each video using its full URL, we use only the part of the URL that uniquely identifies a video file. In addition, for video file responses that have *cache-control:private* headers, we ignore those headers and we assume that the video file can be cached publicly.