Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network

Xian Chen, Ruofan Jin
University of Connecticut

Kyoungwon Suh
Illinois State University

Bing Wang, Wei Wei
University of Connecticut

ABSTRACT

Smart mobile handheld devices (MHDs) such as smartphones have been used for a wide range of applications. Despite the recent flurry of research on various aspects of smart MHDs, little is known about their network performance in WiFi networks. In this paper, we measure the network performance of smart MHDs inside a university campus WiFi network, and identify the dominant factors that affect the network performance. Specifically, we analyze 2.9TB of data collected over three days by a monitor that is located at a gateway router of the network, and make the following findings: (1) Compared to non-handheld devices (NHDs), MHDs use well provisioned Akamai and Google servers more heavily, which boosts the overall network performance of MHDs. Furthermore, MHD flows, particularly short flows, benefit from the large initial congestion window that has been adopted by Akamai and Google servers. (2) MHDs tend to have larger local delays inside the WiFi network and are more adversely affected by the number of concurrent flows. (3) Earlier versions of Android OS (before 4.X) cannot take advantage of the large initial congestion window adopted by many servers. On the other hand, the large receive window adopted by iOS is not fully utilized by most flows, potentially leading to waste of resources. (4) Some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks. Our observations provide valuable insights on content distribution, server provisioning, MHD system design, and application-level protocol design.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Measurements

General Terms

Performance, Measurement

IMC'12, November 14–16, 2012, Boston, Massachusetts, USA.

Keywords

Smart Mobile Handhelds, WiFi Networks, Performance Measurement

1. INTRODUCTION

Smart mobile handheld devices (MHDs) such as smartphones have been used for a wide range of applications including surfing web, checking email, watching video, accessing social networking services, and online games. Most MHDs are equipped with both cellular and WiFi interface cards. Whenever available, WiFi is still a preferred way for Internet connection due to its higher bandwidth, lower delay and lower energy consumption [9]. Although recently there is a flurry of studies on various aspects of smart MHDs (see Section 2), little is known about the network performance of smart MHDs in WiFi networks. Specifically, how is their network performance? What are the major factors that affect the network performance?

In this paper, we answer the above questions by measuring the performance of smart MHDs inside a university campus WiFi network. Specifically, our study is over a data set that is passively captured by a monitor placed at a gateway router in the University of Connecticut (UConn). The data set is collected over three days (2.9TB of data), containing traffic from various wireless devices, including MHDs such as iPhones, iPod touches, iPads, Android phones, Windows phones, and Blackberry phones, and wireless non-handheld devices (NHDs) such as Windows laptops and MacBooks. To understand the performance of MHDs, we first separate the traffic of MHDs from that of NHDs. Our focus is on MHDs; we describe the results on NHDs when necessary for comparison.

Analyzing the data set, we find HTTP is the dominant traffic type, accounting for over 92% of the TCP flows. We therefore focus on the performance of HTTP flows in this paper. The behavior of HTTP is complicated: we find many HTTP flows contain multiple HTTP requests, and a significant portion of an HTTP flow is idling (with no data packets). Hence, the traditional throughput metric (i.e., the amount of data downloaded in a flow divided by the total duration of the flow) may introduce bias in measuring the performance of HTTP flows. We therefore define a metric, *per-flow servicing rate*, i.e., the amount of data downloaded corresponding to HTTP requests in a flow divided by the downloading duration of the flow, to quantify the performance of an HTTP flow (see Section 4). This metric is interesting in its own right: it represents the network performance of an HTTP flow, while excluding the effect of various delays (e.g., client processing delays and user pause times) that are irrelevant to network performance. Our main findings are as follows.

- We observe that the best performance is achieved for the MHD flows served by an Akamai server cluster that is located close to UConn. Furthermore, 16% of the MHD flows are served by this server cluster, accounting for 35% of the bytes. Overall, a large percentage (38.4%) of MHD flows are served by well provisioned Akamai and Google servers, account for 62% of the bytes. Interestingly, these fractions are significantly larger than the corresponding values for NHDs, indicating that MHDs use Akamai and Google servers more heavily, which boosts their overall network performance. We also observe that Akamai and Google servers have adopted large initial congestion window, which contributes to the network performance of MHDs (particularly for short flows). On the other hand, we observe that most flows have low loss rates, indicating that loss rate is not a limiting factor in the campus WiFi network that we study.

- We find that MHDs tend to have longer local RTTs (i.e., delays within the university network) than NHDs. In addition, the number of concurrent flows has negative effect on performance, and the effect is more significant for MHDs than NHDs. These two differences between MHDs and NHDs might be caused by the inferior computation and I/O capabilities on MHDs. The effect of local RTT on network performance seems negligible due the fact that local RTT only takes a small portion of the RTT.

- We find that earlier versions of Android OS (before 4.X) cannot take advantage of the large initial congestion window adopted by many servers. While Android OS increases the receive window adaptively in every round trip time, it uses a very small initial receive window (much smaller than the initial congestion window adopted by Google and Akamai servers), which limits the performance of Andriod devices. In contrast, we observe that iOS uses a large static receive window, which fully exploits the benefit of large initial congestion window. On the other hand, most flows do not fully utilize the large receive window, potentially leading to unnecessary waste of resources on iOS devices.

- We find that some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks. One example is the native YouTube application on iOS devices, which can use a large number of TCP flows to serve a single video. We suspect this is a design optimization for cellular networks, which is not suitable for WiFi networks.

Our findings highlight the impact of TCP parameters and application-level design choices on MHD network performance, providing valuable insights on content distribution, server provisioning, mobile system design, and application-level protocol design.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes data collection and classification. Section 4 introduces the performance metric. Section 5 describes our methodology. Section 6 presents network performance of MHDs, and explores the impact of network and application layer factors on the performance. Section 7 discusses what findings are specific to UConn WiFi network and what can be applied to other networks. Last, Section 8 concludes the paper and presents future research directions.

2. RELATED WORK

Several recent studies characterize the usage and traffics of MHDs in 3G cellular networks, public WiFi or residential WiFi networks. Trestian et al. analyze a trace collected from the content billing system of a large 3G mobile service provider to understand the relationship among people, locations and interests in 3G mobile networks [31]. Maier et al. study packet traces collected from more than 20,000 residential DSL customers to characterize the traffic from MHDs in home WiFi networks [21]. Falaki et al. employ passive sniffers on individual devices (Android and Windows Mobile smartphones) to record sent and received traffic, and provide a detailed look at smartphone traffic [11]. This study is limited to a small user population (43 users), and the traces were not separately analyzed for the different network interfaces (i.e., 3G and WiFi) being used, which have different properties. The study in [12] uses a larger population of 255 users, and characterizes intentional user activities and the impact of these activities on network and energy usage. Gember et al. compare the content and flow characteristics of MHDs and NHDs in campus WiFi networks [14]. We focus on network performance of MHDs in a campus WiFi network and the impact of various factors on the network performance of MHDs, which are not investigated in the above studies.

Huang et al. anatomize the performance of smartphone applications in 3G networks using their widely-deployed active measurement tool, 3GTest [16]. Tso et al. study the performance of mobile HSPA (a 3.5G cellular standard) networks in Hong Kong using extensive field tests [32]. Our study differs from them in that we study the network performance of MHDs in WiFi networks (instead of cellular networks). Furthermore, our study is based on large-scale traces collected passively from a university campus network, while the study in [16] adopts an active measurement tool and the study in [32] uses customized applications. The studies of [13, 26] report that iOS native YouTube player generates multiple TCP flows to stream a single video. We expand these studies by presenting the degree of inefficiencies caused by the large number of TCP flows and the main reason for the design choice in the player.

Several measurement studies are on university WiFi networks. However, MHDs have only been widely adopted recently, and none of those studies explores the network performance of MHDs as in our study. Our study builds upon the rich literature on understanding the behavior of TCPs and content distribution in operational environments (e.g., [19, 23, 20, 15, 30, 10, 4, 22]), and our observations confirm some of the findings in those studies.

Last, several other aspects of MHDs have been studied recently, including battery use and charging behaviors [6, 25], energy consumption [29, 5], and performance enhancement techniques (e.g., [24, 34]). Our study differs in scope from them.

3. DATA COLLECTION & CLASSIFICATION

We collect measurements at a monitoring point inside the University of Connecticut (UConn). The monitor is a commodity PC, equipped with a DAG card [2]. It is placed at a gateway router of UConn, capturing all incoming and outgoing packets through the router with accurate timestamps[1]. In particular, it captures up to 900 bytes of each packet, including application-level, TCP and IP headers. The campus network uses separate IP pools for Ethernet and Wireless LAN (WLAN). Since we are interested in the network performance of MHDs, which use the WLAN IP address pool, we use filters to capture only WLAN traffic.

We have collected two data sets. One data set is for three days, from 9am March 2 to 9am March 5, 2011. The other data set is for one day, from 9am April 23 to 9am April 24, 2012. Unless otherwise stated, we use the first data set (reasons for focusing on this data set and findings from the second data set are deferred to Section 6.9). In the following, we first provide a high-level view of the data, and then present methodology to separate MHD traffic from NHD traffic (the captured data set contains a mixture of traffic from both types of devices).

3.1 Data

Table 1 lists the number of packets captured during the three days. Overall, we collected over 5.8G packets (2.9 TB of data). Among them, 91.9% of the packets are carried by TCP. We only report the results obtained from the data collected on the first day; the statistical characteristics of the data on the other two days are similar[2].

Table 1: The number of packets captured during the three days.

	Day 1	Day 2	Day 3	Overall
incoming pkts	1.2G	1.4G	0.8G	3.5G
outgoing pkts	0.8G	1.0G	0.5G	2.3G
total pkts	2.0G	2.4G	1.4G	5.8G
% TCP pkts	91.8%	92.7%	90.8%	91.9%

We say a TCP flow is *valid* if it finishes three-way handshaking and does not contain a RESET packet[3]. Among the valid TCP flows, we identify the applications using destination port numbers (a recent study shows that this simple approach provides accurate results [20]). Table 2 lists the most commonly used applications, and the percentages of their traffic over all TCP traffic in terms of flows, packets and bytes, where the number of bytes in a packet is obtained from the IP header. We observe a predominant percentage (92.3%) of the TCP flows are HTTP flows. This is consistent with measurement results in other campus networks [14] and home environments [20]. In the rest of the paper, we focus on the network performance of HTTP flows; the performance of other protocols is left as future work.

Table 2: Percentage of the traffic from commonly used applications (in terms of flows, packets and bytes) over all the TCP traffic (Day 1).

Application	flows	packets	bytes
HTTP (80)	92.3%	82%	86.7%
HTTPS (443)	4.3%	8%	5.4%
IMAPS (993)	0.2%	0.28%	0.13%

3.2 Data classification

We refer to an HTTP flow that contains packets coming from and/or destinating to an MHD as an MHD flow (similar definition for an NHD flow). Since our captured data contains a mixture of MHD and NHD flows, we need to separate these two types of flows. The first approach we use to differentiate MHD and NHD flows is based on the keywords in the user-agent field in HTTP headers (MHDs and NHDs use different keywords) [21, 14]. Once the type of a flow is identified using the user-agent field, it provides us information on the type of the associated device (i.e., whether it is an MHD or NHD), which can be used to further identify the types of other flows. Specifically, if we know that a flow, f, from a device (the device is represented using an IP address) is an MHD flow, then we know all the flows that are concurrent with f and have the same source IP address as f are MHD flows[4]. If the type of an HTTP flow is not identified using the above two approaches, we use the knowledge that an IP address pool is dedicated to Apple mobile devices at UConn[5], and hence a flow using an IP address from this pool is an MHD flow. Using the above three approaches, we categorize 94.1% of the HTTP flows to be either MHD or NHD flows (91.9% of them are directly identified through the user-agent field, 6.7% are identified through the concurrent-flow approach, and the rest 1.4% are identified using the knowledge of the dedicated IP address pool). Specifically, we identify 0.7M MHD flows and 10.4M NHD flows, containing 38.8M and 821.1M packets, respectively. Further separation of the flows using other heuristics (e.g., TTL [21]) is left as future work.

We further identify the operating systems used by the MHDs and NHDs using the user-agent field. For MHDs, the dominant operating system is iOS, used in 96.2% of the MHD flows (iOS is used by iPhone, iPod touch, and iPad, with the percentages of flows as 65%, 26%, and 9%, respectively); and Android is the second most popular operating system, used in 3.2% of the MHD flows. For NHDs, the two dominant operating systems are OS X and Windows, taking 56.7% and 42.2% of the NHD flows, respectively; and Linux is at the third place, used by 1.1% of the NHD flows.

4. PERFORMANCE METRIC

As mentioned earlier, we mainly characterize the perfor-

[1] The campus network balances loads among two gateway routers. The load balancing strategy is set up so that data packets and ACKs in a TCP flow are through the same router.

[2] The amount of traffic on Day 3 is less than that of the other two days since Day 3 is a Friday.

[3] 11.9% of the flows cotain a RESET packet.

[4] We can only classify the flows that are concurrent with f due to IP reassignment which can assign the same IP address to another device at another point of time.

[5] UConn network administrators set aside this dedicated pool to ease the management of IP addresses. Specifically, a device is assigned an IP address from this dedicated pool if its host name indicates that it is an Apple mobile device during the DHCP request phase.

mance of HTTP flows since HTTP is the predominant traffic type. The behavior of HTTP is complicated. For instance, modern browsers often open multiple TCP connections when browsing a web page [10]. Furthermore, an HTTP flow can send and receive multiple HTTP requests/responses. In our passive measurements, without any knowledge of the accessed content, it is difficult to correlate multiple TCP connections. We therefore focus on per-flow network performance, specifically, the performance that is the result of the complex interactions of myriad network and application related factors. In the following, we first describe our measurement results on HTTP flows, and then define a performance metric that we will use in the rest of the paper.

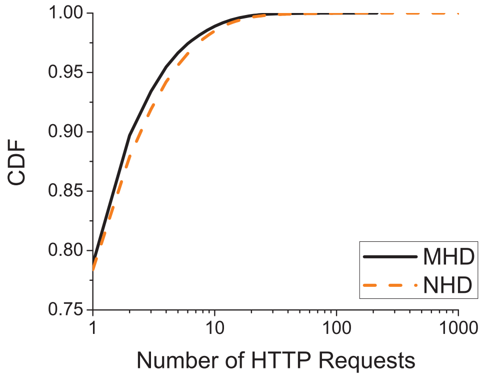


Figure 1: Distribution of the number of HTTP GET requests in an HTTP flow.

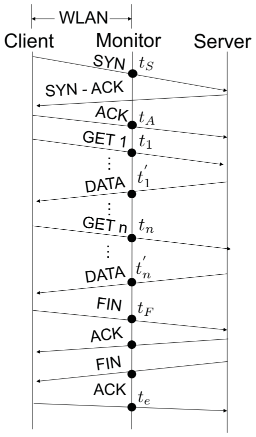
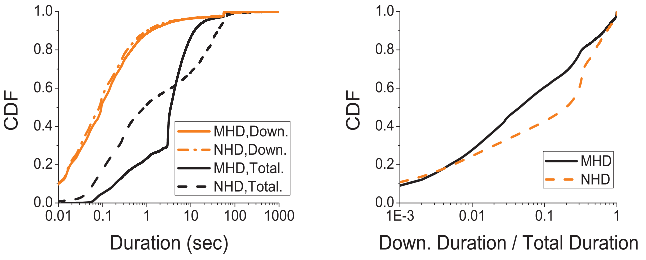


Figure 2: Illustration of a persistent HTTP flow.

We find that over 99% of the HTTP flows use HTTP 1.1, where TCP connections are persistent, i.e., each TCP connection allows multiple GET and POST commands. On the other hand, we observe only 6.5% of the HTTP flows contain POST commands. This is not surprising since most of the users in the campus network are content consumers. In the following, we ignore the HTTP flows with POST commands. Fig. 1 plots the distribution of the number of HTTP GET requests within an HTTP flow. Observe that over 20% of the HTTP flows contain at least two requests. The number of requests in a flow can be as large as 100. In addition, NHD flows tend to contain a larger number of requests than MHD flows. This might be because regular web pages accessed by NHDs contain more objects than their mobile versions that are often accessed by MHDs [33].



(a)

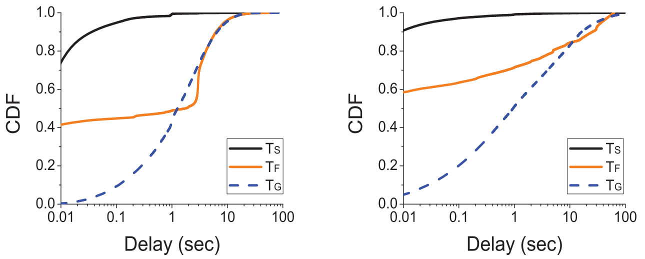
(b)

Figure 3: (a) Total and downloading durations of MHD and NHD flows. (b) The ratio of downloading duration over total duration.

Fig. 2 illustrates a persistent HTTP flow with n requests (the requests are sequential since we do not observe any pipelined requests in our trace where a request is sent out without waiting for the response of the previous request[6]). The measurement point sits between the client and server, capturing the packets with accurate timestamps. In particular, let t_S denote the timestamp of the SYN packet, t_A denote the timestamp of the ACK packet in the three-way handshaking, t_i denote the timestamp of the ith HTTP request, t_i' denote the timestamp of the last data packet corresponding to the ith HTTP request, t_F denote the timestamp of the first FIN packet, and t_e denote the timestamp indicating the end of a TCP flow. Then the measured duration of the HTTP flow at the measurement point is $t_e - t_S$, while the duration for data downloading is $\sum_{i=1}^{n}(t_i^{'} - t_i)$. We refer to the former as *total duration* and the latter as *downloading duration*. Fig. 3(a) plots the distributions of the total and downloading durations for MHD and NHD flows. We observe that total durations can be significantly longer than downloading durations: the median downloading durations are 0.63s and 0.93s for MHD and NHD flows, respectively, while the median total durations are 5.06s and 15.45s for MHD and NHD flows, respectively. As shown in Fig. 3(b), the downloading duration is less than 10% of the total duration for respectively 60% and 42% of the MHD and NHD flows.

The difference between the downloading and total duration contains various delays inside an HTTP flow. Parts of the delays are for the three-way handshaking and TCP connection termination, and the rest of the delays are application idle/processing time. We divide the application idle/processing time into three parts. The first is the delay from finishing the three-way handshaking to the first GET request, defined as $T_S = t_1 - t_A$. The second is the sum of the delays from finishing one HTTP request to starting the next HTTP request, i.e., $T_G = \sum_{i=1}^{n-1}(t_{i+1} - t_i')$, and the last is the delay from finishing downloading the last object to start to close the TCP connection, i.e., $T_F = t_F - t_n'$. Fig. 4

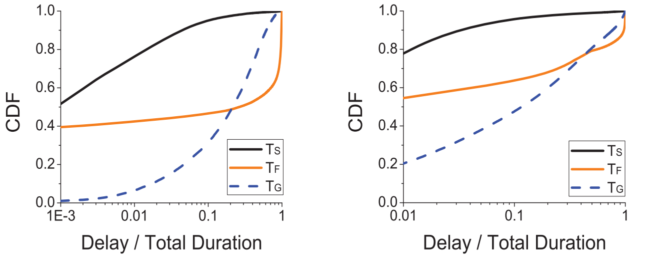
[6] HTTP pipeline is not recommended and disabled in most browsers [3].



(a) MHD

(b) NHD

Figure 4: Distribution of the various delays in an HTTP flow.



(a) MHD

(b) NHD

Figure 5: Distribution of the ratios of the various delay over the total duration in an HTTP flow.

plots the distributions of the various delays. We observe that T_S is small for both MHD and NHD flows. Although a large fraction of the T_F values is small, it can be as large as several seconds for MHD flows and tens of seconds for NHD flows. The values of T_G are in a wide range from milliseconds to tens of seconds. Fig. 5 plots the CDF of the ratios of the various delay over the total duration.

In summary, the drastic difference between total duration and downloading duration highlights the importance of defining performance metric carefully: a traditional throughput metric, defined as the total amount of data downloaded divided by the total duration, can lead to biased results on the performance of HTTP flows. To exclude the effects of application-level delays on network performance, we define a performance metric, *per-flow servicing rate*, i.e., the total number of data bytes that are downloaded corresponding to HTTP requests divided by the downloading duration, to represent network performance. This metric represents the rate at which data are being fetched in an HTTP flow from a server to a client, while excluding the effect of various delays (e.g., client processing delays and user pause times) that are irrelevant to network performance.

5. METHODOLOGY

In the traces that we collected, MHDs are typically clients, requesting contents from servers outside UConn campus network. To understand the performance of MHD flows, we first group them according to their destinations. The rationale is that since the clients (i.e., MHDs) are at the same geographic location, the destinations (corresponding to the content servers) directly determine the network path. For the flows that are served by the same group of servers, we further divide the flows according to their lengths (in terms of number of packets inside the flow), since flow length affects the impact of the various TCP parameters and network path conditions on servicing rate. Specifically, short flows may terminate before finishing the slow-start phase in TCP, while long flows are more likely to reach congestion avoidance phase, and hence achieve more steady throughput.

In the following, we first describe server and flow classification, and then describe how we measure the various TCP and application-level characteristics.

5.1 Server Classification

We use two steps to study each flow destination IP address. First, we use reverse DNS lookup to find its domain name, and then query the IP registration information to determine the organization that registered the IP address. For the IP addresses that do not have a published domain name, we use the registration information only.

Table 3: Top five domain names for MHD and NHD flows (percentage is in terms of flows).

MHD	NHD
Akamai.com (26.1%)	Akamai.com (20.0%)
Google.com (12.3%)	Google.com (8.9%)
Facebook.com (10.0%)	Facebook.com (8.3%)
Amazon.com (6.8%)	Yahoo.com (5.9%)
Apple.com (5.0%)	Amazon.com (5.3%)

Table 3 presents the top five domain names of MHD flows that are obtained using reverse DNS lookup. We see that the largest percentage of MHD flows are served by Akamai CDN. Indeed, many widely used applications on MHDs are served by Akamai. For instance, when an iOS device downloads an application from Apple App Store, the data are actually downloaded from Akamai. In addition, some popular destinations, e.g., "fbcdn.net", are served by Akamai (we find 12% of MHD flows are destined to "fbcdn.net"). Following Akamai.com, Google.com is the second most frequently accessed domain name. This is not surprising: Google.com is one of the most frequently accessed web sites [1], and many MHDs have embedded Google applications (e.g., Google Map, Google Voice, Gmail). The third one is Facebook.com, which uses Akamai to serve many static contents, and use its own servers to serve many dynamic contents directly. The fourth one is Amazon.com, whose cloud services serve many popular applications running on iOS (e.g., Foursquare).

Since Akamai.com and Google.com are the top two domain names, we detail the network performance of the MHD flows that are served by Akamai and Google servers, as well as those served by the rest of the servers in Section 6. We also observe from Table 3 that 38.4% of MHD flows are served by Akamai CDN and Google servers. This percentage is significantly larger than the corresponding value for NHD flows (i.e., 28.9%). We believe the reason why MHD flows use Akamai and Google servers more heavily is that the destination hosts accessed by MHDs are less diverse than those accessed by NHDs [14]. Based on the host field in the HTTP request headers, we find around 6k distinct destination host domains from MHDs and 51k distinct destination host domains from NHDs. In addition, Table 4 lists the top 10 destination host domains accessed by MHD and NHD flows. We see for MHDs, accesses to the top ten host domains account for 39.2% of the MHD flows, while the percentage for NHDs (32%) is much lower. Last, we also observe from Table 4 that for MHDs, except for facebook.com and pandora.com, the other top eight host domains are all served by Akamai and Google, confirming the heavy usage of Akamai and Google servers by MHDs.

Table 4: Top ten destination host domains for MHD and NHD flows (percentage is in terms of flows).

MHD	NHD
fbcdn.net (12.0%)	fbcdn.net (12.9%)
facebook.com (10%)	facebook.com (6.5%)
apple.com (3.9%)	google.com (2.4%)
googlevideo.com (3.8%)	doubleclick.net (2.0%)
google.com (2.5%)	ytimg.com (1.7%)
admob.com (1.6%)	quantservec.om (1.6%)
doubleclick.net (1.5%)	yieldmanager.com (1.5%)
youtube.com (1.4%)	tumblr.com (1.2%)
google-analytics.com (1.3%)	twitter.com (1.1%)
pandora.com (1.2%)	yahoo.com (1.1%)

5.2 Flow Length Classification

For the flows that are served by the same server category, we divide the flows according to their lengths into three groups, denoted as G_1, G_2, and G_3. Specifically, G_1 contains short flows, with one to ten data packets, G_2 and G_3 contain longer flows, with at least 10 and 50 data packets, respectively. We make the above grouping since many web pages belong to G_1 [10], while long video streaming sessions may belong to G_3[7]. Table 5 lists the number of flows, packets and bytes (calculated from packet size in IP header) of the various groups. Observe that around 75% of the flows are short G_1 flows.

Table 5: Information of the various groups of MHD flows (for the data collected on Day 1).

	G_1	G_2	G_3
flows	0.5M	0.2M	0.03M
packets	2.9M	25.1M	19.0M
bytes	2.3GB	22.8GB	19.4GB

5.3 TCP flow characteristics

We now describe how we estimate the various TCP flow characteristics, including RTT, loss rate, local RTT and loss rate, server's initial congestion window, client's advertised receive window, and the maximum window size.

We use the techniques in [19] to obtain a sequence of RTT samples for a flow, and use the median to represent the flow's RTT. Packet losses are detected when observing triple duplicate ACKs, or a retransmission that is caused by timeout[8]. Local RTT represents the delay that a TCP flow experiences locally, inside a local-area network (i.e., the UConn campus network in our context). Similarly, local loss rate represents the loss rate that a TCP flow experiences inside a local-area network. In our study, both local RTT and local loss rate can be directly obtained from the measurements at the monitoring point since it is at the edge of the local-area network (see details in [7]).

The server's initial congestion window size (icwnd) has a significant impact on the flow downloading rate. It determines the amount of data that the server can send in the first window. If the icwnd is too small, the sender needs to pause and waits for ACKs before transmitting again; on the other hand, if it is too large, the sender pushes too much data into the network, and congestion could happen [10]. We infer the icwnd based on the packet arrival times at the monitoring point. Specifically, we obtain an estimate of the icwnd as n when observing the first window of n packets arriving close in time at the the monitoring point. The first window is dynamically estimated based on RTTs using the technique in [19]. This approach will underestimate the icwnd if the application does not have sufficient amount of data to send in the first window. To avoid potential underestimation, we only apply the approach to HTTP flows that contain at least one GET request and the data corresponding to the first GET request contain at least 20 packets. The constraint of at least 20 data packets (corresponding to at least 20KB when each packet is at least 1000 bytes) is based on the literature that servers can choose a large icwnd, e.g., between 10KB to 20KB [10, 23]. To obtain a server's icwnd, we infer a series of icwnd values from all of the flows that are served by this server and satisfy the requirements stated earlier, and then obtain the average, median, and maximum from these inferred values (our measurements indicate that a server's icwnd can vary among the flows). In Section 6.4, we only report the results from servers that have at least 10 estimates.

Client's advertised receive window, constantly reported from the client to the server, states the size of the available receive buffer, i.e., the maximum size of data that a sender transmits to the receiver before the data is acknowledged by the receiver. The receive window field in the TCP header is 16 bits, limiting the maximum size of receive window to 64KB. However, if both TCP server and client support window scaling option, they can agree upon a window scaling factor, which is the multiplier to the 16-bit receive window field. For example, if window scaling factor is 4, the maximum receive window that a client can specify is 256KB.

At each time point, the server's available window is the minimum of the client's advertised receive window and server congestion window. We also measure the maximum window of each flow during its lifetime. The maximum window has impact on the network performance as well: it determines the maximum amount of data that the sender can transmit in one RTT, which affects the maximum downloading rate.

5.4 Application layer characteristics

We consider the following three application-level characteristics. The first is the design of application-level proto-

[7] We also consider another group, G_4, which contains flows with at least 100 packets. The results of G_4 are similar to those of G_3, and hence are not reported in this paper.

[8] These are inferred losses at the TCP level. The loss rate thus estimated is an overestimate since the inferred losses may be due to long delays, not actual losses.

cols. In particular, we consider the protocol for YouTube videos. This is motivated by the popularity of this application and the large amount of video data in our trace (38% of the data are video contents). Secondly, we examine how servers respond to requests for different types of contents (e.g., video, image and texts). For this purpose, we define *application response time* as the delay from the first GET message to the first responding data packet for an HTTP request (the delay is measured at the monitoring point). Last, we quantify the relationship between the number of concurrent TCP flows and per-flow servicing rate. The rationale behind this is that the number of concurrent TCP flows might be an indicator of the amount of CPU and I/O activities on an MHD device. We calculate the amount of concurrent TCP flows for a flow as follows. Consider a flow f. Since the number of its concurrent flows can vary over time, we divide time into 0.3s bins, and count the number of concurrent flows in each bin, and then obtain the average number of concurrent flows during f's life-time as its number of concurrent flows. We only consider concurrent flows for long flows, specifically, the flows in G_3.

6. NETWORK PERFORMANCE OF MHDS AND RELATED FACTORS

In this section, we first present network performance of MHDs and then investigate how network and application layer factors affect the performance. Unless otherwise stated, we mainly report the results of iOS devices (i.e., iPhone, iPod touch, iPad) that account for 96.2% of the MHD flows.

As shown in Table 4, a large percentage (38.4%) of MHD flows are served by Akamai and Google servers. Using a commercial database [17], we identify that the Akamai servers are located in four main clusters, respectively 40km, 113km, 517km, and 966km away from UConn. Furthermore, consistent with the Akamai DNS design policy that gets servers close to end users [15], we find 90% of the MHD Akamai flows are served by the first two close-by server clusters. IP registration information reveals that all Akamai servers in the 40km range belong to the University of Hartford (only one hop away from UConn network), while it does not reveal any detailed information for the servers in the other distance ranges. We therefore refer to the first two server clusters as Akamai-Hartford and Akamai-113km, respectively. In the rest of the paper, we classify the servers into four clusters: Akamai-Hartford, Akamai-113km, Google[9], and other servers.

Figures 6(a), (b), and (c) plot per-flow servicing rate distributions of G_1, G_2 and G_3 flows, respectively. The results for all four server clusters are plotted in the figure. For the same server cluster, we observe larger per-flow servicing rate for longer flows because longer flows allow the congestion window to ramp up. Overall, Akamai-Hartford servers provide the best performance, with median servicing rates of 4.7Mbps, 5.9Mbps, and 6.5Mbps for G_1, G_2, and G_3 flows, respectively. For G_2 and G_3 flows, we observe a clear stochastic order: the performance achieved by Akamai-Hartford servers is the best, followed by Akamai-113km, Google, and the other servers. For G_1 flows, the performance achieved by Google servers is superior to that of Akamai-113km servers, and is superior to that of Akamai-Hartford servers at low percentiles, a point that we will return later.

Existing studies (e.g., [28, 26]) have shown that video servers may control the sending rate of video flows, which can affect per-flow servicing rate of those flows. In the traces that we collected, we confirm that YouTube flash videos to NHDs are indeed rate controlled. On the other hand, videos to NHDs are predominantly mp4 videos, which are not rate controlled. Therefore, per-flow servicing rates of MHD flows presented in Fig. 6 are not affected by rate control mechanisms. We next investigate the impact of various network and application-layer factors on MHD network performance.

6.1 RTT

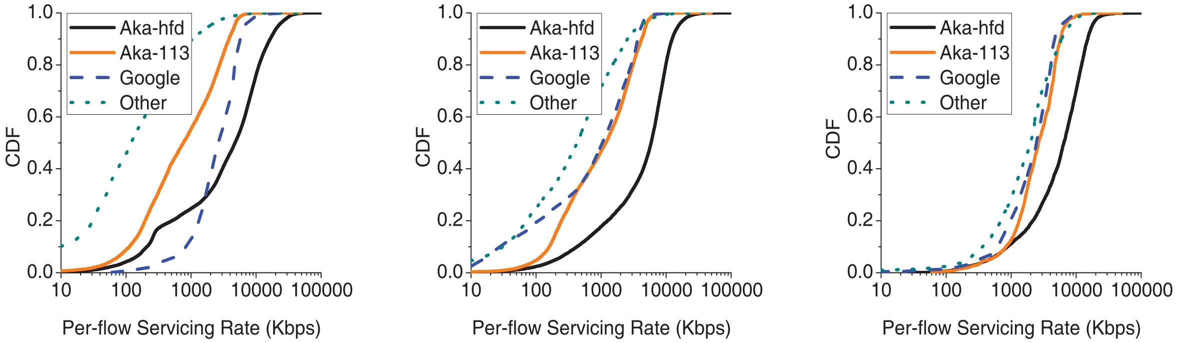
We observe that RTTs to Akamai-Hartford servers tend to be the lowest, followed by Akamai-113km, Google, and the other servers. The main reason for the lowest RTTs to Akamai-Hartford servers is the close geographic distance and good network provisioning (both the University of Hartford where the servers are deployed and UConn belong to Connecticut education network). Fig. 7 plots the RTT distributions of G_2 MHD flows served by the four server clusters (results for G_1 and G_3 flows are similar). The median RTT of the flows served by Akamai-Hartford servers is 8.5ms, while for Akamai-113km, Google, and other servers, the median RTTs are respectively 2.3, 4.3, and 10.3 times larger. For the four server clusters, their relative order in terms of RTT is consistent with their relative performance as shown in Fig. 6. That is, a server cluster with lower RTTs tends to provide better performance. This is not surprising since RTT is a major factor that affects network performance.

What is more interesting is perhaps the large percentage of data that is served by the first three server clusters. Specifically, 58% of the MHD data bytes are served by the first three server clusters, much larger than the corresponding value (41%) for NHDs. Furthermore, 16% of the MHDs flows (accounting for 35% of the data) are served by Akamai-Hartford servers, the server cluster that provides the best performance. As described earlier, we believe that these large percentages originate from the fact that the most popular services accessed by MHDs are provided by major companies such as Facebook, Google, and Apple that use CDN services heavily. The large percentage of data served by Akamai and Google servers boost the overall network performance of MHDs.

6.2 Local RTT

We observe that long MHD flows tend to experience larger local RTTs than short MHD flows. As an example, Fig. 8(a) plots local RTT distributions of G_1, G_2, and G_3 MHD flows served by Akamai-Hartford server cluster (results for the other three server clusters are similar). We see that local RTT of G_1 flows is smaller than that of G_2 flows, which is in turn smaller than that of G_3 flows. In addition, we observe that local RTT of MHDs tends to be larger than that of NHDs. One example is shown in Fig. 8(b), which plots local RTT distributions of G_3 MHD and NHD flows served by Akamai-Hartford servers. Using *t-test* [18], we confirm that

[9] Google does not publish the location information of its data centers. Therefore we cannot simply use the database [17] to determine the geographic locations of Google servers. Hence we present the aggregate results of all Google servers. It is possible to determine the geographic location of the servers using RTT [30]. Dividing the servers into clusters according to their geographic locations and investigating their respective performance are left as future work.



(a) G_1 (b) G_2 (c) G_3

Figure 6: Per-flow servicing rate of MHD flows served by the four server clusters.

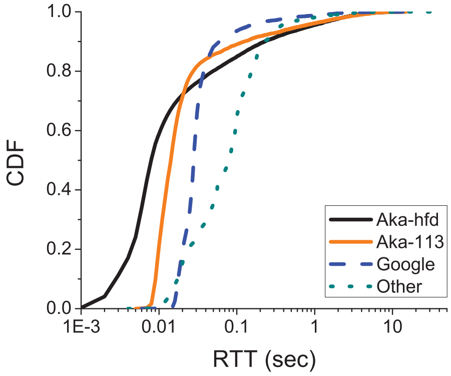
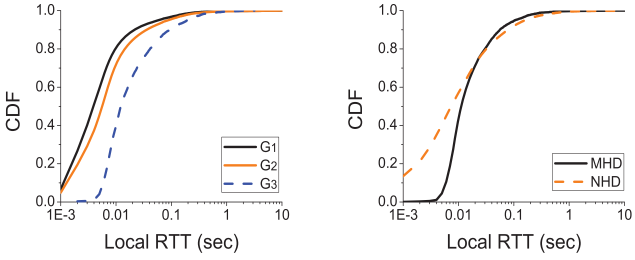


Figure 7: RTT distributions of G_2 flows served by the four server clusters.



(a) (b)

Figure 8: (a) Local RTT for the MHD flows served by Akamai-Hartford servers. (b) Local RTT of G_3 MHD and NHD flows served by Akamai-Hartford servers.

the above two observations indeed hold statistically. Specifically, for the MHD flows served by the same server cluster, the average local RTT of G_i flows is statistically smaller than that of G_j flows, $i < j$; and for G_i flows, the average local RTT of MHD flows is statistically larger than that of NHD flows, $i = 1, 2, 3$. The reason might be limited computation and I/O capabilities on MHDs, which lead to longer processing time for longer flows, as well as longer processing time than that on NHDs (a more in-depth study using active measurements is left as future work). On the other hand, except for Akamai-Hartford servers, local RTT is only a small portion of RTT (the ratio of local RTT over RTT is below 0.2 for 10% to 56% of the flows). Therefore, the impact of local RTT on per-flow servicing rate is negligible. We also observe a small fraction of local RTTs that are longer than 100ms, a point we will return to in Section 6.3.

6.3 Loss Rate

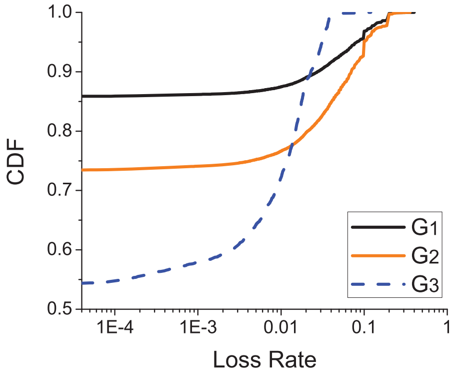


Figure 9: Loss rate distribution for the MHD flows served by Akamai-Hartford servers.

We find that most flows have very low loss rates, indicating that loss rate is not a limiting factor in the campus WiFi network that we study. An example is shown in Fig. 9, which plots loss rate distribution of the MHD flows served by Akamai-Hartford servers (results for the other three server clusters have similar trend). We observe that, for G_1, G_2 and G_3 flows, respectively 86%, 74% and 54% of the flows have zero loss. Furthermore, 90% of G_3 flows have loss rates below 0.02. The loss rate of a short flow can be large, which is however due to the artifact of small flow size (i.e., even a small number of losses can lead to high loss rate). We also find that all of the losses occur outside UConn (i.e., local loss rate is zero). This, however, does not mean that there is no loss at the MAC layer. We observe that a small percentage (around 3%) of local RTTs are longer than 100ms (see Fig. 8(a)). These long local RTTs might be caused by MAC-layer retransmissions and back-off times. Our measurements captured at the monitoring point does not contain MAC layer information; validating this conjecture through measurements captured at MAC layer is left as future work.

6.4 Initial Congestion Window

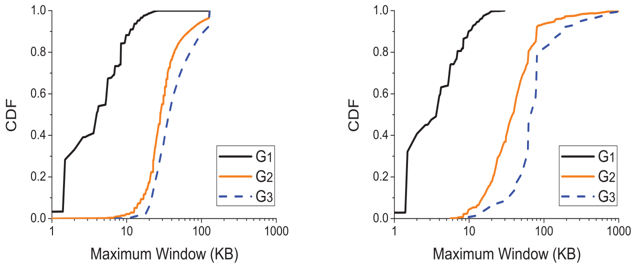
We find that Akamai and Google servers have adopted large initial congestion window. Specifically, the median initial congestion window of Akamai-Hartford servers is between 5 and 6KB, and the maximum initial congestion window is between 10 and 12KB. For Akamai-113km servers, 79% and 19% of the servers have median initial congestion window of 8KB and 5.5KB, respectively; the maximum initial congestion window is as large as 15KB. Google's adoption of large initial congestion window is even more aggressive: 98% of the Google servers use a median initial congestion window of 14KB (consistent with the advocated value of 15KB [10]), and the largest initial congestion window can be as large as 25KB. The rest of the servers (i.e., those other than Akamai and Google servers) have not adopted large initial congestion window as aggressively. Specifically, 61% of them use initial congestion window smaller than 4KB.

Large initial congestion window is particularly beneficial to short flows that can be completed in one RTT when initial congestion window is large. The more aggressive initial congestion window adopted by Google servers leads to its superior performance in serving G_1 flows: we observe from Fig. 6(a) better performance from Google servers than Akamai-113km servers at low percentiles despite that RTT to Google servers tends to be larger than that to Akamai-113km (RTT distributions for G_1 flows are similar to those for G_2 flows, which is plotted in Fig. 7). Furthermore, a smaller fraction of G_1 flows served by Google servers has very low servicing rate, as shown in Fig. 6(a). Even for G_2 flows, we observe similar servicing rates between flows served by Akamai-113km and Google servers (Fig. 6(b)) despite of the more dramatic differences in RTT (Fig. 7).

On the other hand, adopting larger initial congestion window can lead to congestion in the network. The study in [10] shows that this is only true for slow network connections. From our results, we do not observe significantly different loss rates from the server clusters that adopt different initial congestion windows.

6.5 Advertised Receive Window

We find that both iOS and Andriod operating systems support window scaling. When connecting to servers that support window scaling (e.g., we find all Akamai and Google servers support window scaling), the receive window advertised by iOS devices is 128KB[10], implying that the maximum window of a flow is 128KB. In contrast to the static large receive window advertised by iOS devices, Andriod devices dynamically adjust receive window every round trip time, starting from a fairly small size of 5KB. Furthermore, since the receive window can grow dynamically, the maximum window size can be larger than 128KB.



(a) iOS

(b) Android

Figure 10: Distribution of maximum window size for iOS and Android flows served by Google servers.

The different design choices adopted by iOS and Andriod devices have the following implications. First, we find the large receive window of iOS devices cannot be fully utilized by most flows. To illustrate this, we plot the distribution of maximum window size of iOS flows that are served by Google servers, as shown in Fig. 10(a). The reason for choosing Google server cluster is that it serves the highest percentage of video flows among all the four server clusters (the percentage is 30.5%), and video flows tend to be longer than other types of flows, and hence are more likely to reach large window size. From Fig. 10(a), we see that only 8% of G_3 flows reach the maximum window limit of 128KB, and the percentages for G_1 and G_2 flows are even lower, indicating that the large receive window of 128KB could potentially cause unnecessary waste of resources on iOS devices (whether it wastes resources or not depends on the kernel memory allocation mechanism: If the iOS kernel preallocates the memory for the socket, this causes waste of resources; otherwise, it does not waste resources).

Secondly, the very small receive window adopted by Andriod devices, while conserving resources, making Andriod devices unable to take advantage of large initial congestion window adopted by many servers (recall TCP congestion window is the minimum of the sender and receiver window), and hence can lead to inferior performance, particularly for short flows. As we shall see in Section 6.9, this is indeed the case (Section 6.9 reports findings from the data set collected in 2012, which contains more Andriod traffic and hence allows us to draw more convincing statistical conclusions regarding Andriod traffic). The adverse effect of small receive window may be even more dramatic in cellular networks where the round trip time can be significantly larger than that in WiFi networks.

Thirdly, since the receive window of Andriod devices is adjusted dynamically, it can grow to large values. Fig. 10(b) plots the distribution of maximum window size for Android flows that are served by Google servers. Indeed, we observe 9% of G_3 flows reach maximum window sizes larger than 128KB, while the maximum window size is bounded below 128KB for iOS flows. We suspect that since the receive window of Andriod devices can grow to large values, Android devices can achieve better performance for very long flows.

[10] The windows scaling factor is 4, leading to receive window of 256KB. However, we observe that the receive window is reduced to 128KB at the end of three-way handshake.

Our dataset, however, does not contain sufficient number of very long flows from Android devices (even in the data set that was collected in April 2012, which contains much more traffic from Android devices) to statistically verify the above conjecture.

Summarizing the above, we believe dynamic receive window adopted by Andriod devices is a suitable choice for resource limited MHDs. On the other hand, using very small initial receive window (e.g., 5KB) is too conservative, which can lead to inferior performance, particularly for short flows. How to choose receive window for MHDs to be both resource efficient and performance enhancing is beyond the scope of this paper, but is an interesting problem that we leave as future work.

6.6 Application-level Protocol

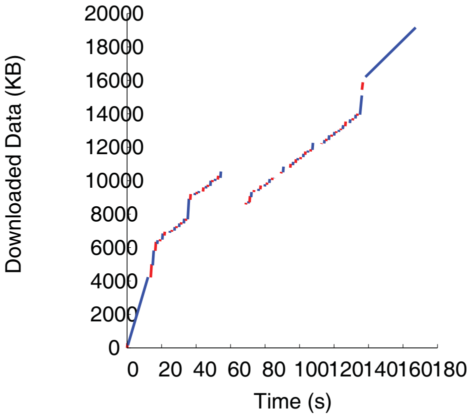
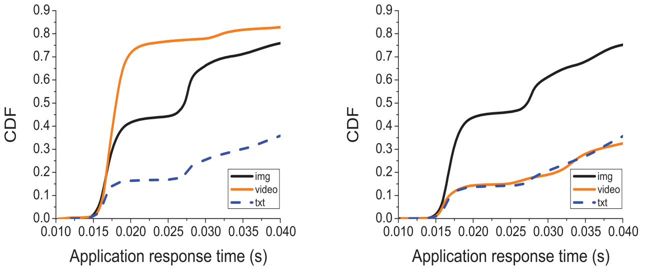


Figure 11: A series of 75 TCP flows is used to download one YouTube video when using the native YouTube application on an iOS device. The figure shows the starting and ending times of each TCP flow with the requested range. For better clarity, we use different colors (red and blue) to represent two adjacent TCP flows.

We find that some application-level protocols for MHDs are highly optimized for cellular networks, which may cause inefficient use of network and operating system resources of MHDs in WiFi networks. The native YouTube player in iOS is an example. When using this player, video contents are downloaded in segments; each segment is requested in a separate TCP connection, using the HTTP Range header field to specify the requested portion of the video [13]. While it has been reported in literature that multiple TCP connections are used to serve a single YouTube video [13, 26], we find, surprisingly, the number of TCP connections can be extremely large. Fig. 11 shows an example where a series of 75 TCP flows (most of them very short-lived) are used to download a single YouTube video. Considering the overhead of creating new TCP connections and the slow-start phase at the beginning of a TCP connection, using so many short-lived TCP flows to serve a single video does not seem to be sensible. We suspect this is a design choice that tries to cope with TCP timeouts (e.g., caused by long delays originated from handoffs, disconnections, and MAC-level retransmission) in cellular networks [8]. For example, the handoff in cellular network can take more than one second, which might cause TCP timeout and reduced congestion window (in the worst case, the congestion window might be reduced to one). Therefore, it might make sense to open a new TCP connection to take advantage of large initial congestion window to overcome the impact of the significantly degraded throughput.



(a) MHD (b) NHD

Figure 12: Application response time for MHD and NHD flows that are served by Google servers.

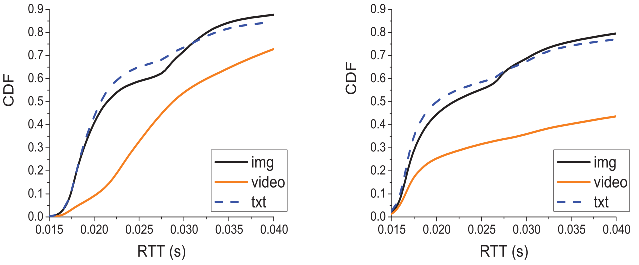
We again find the design choices taken by iOS and Android devices are different: Android devices use only one TCP flow to download a single YouTube video. How to optimize application-level protocols for MHDs, considering the characteristics of both cellular and WiFi networks, is an interesting problem, but is beyond the scope of this paper.

6.7 Application Response Time

We investigate application response time for different types of content to understand whether servers use different mechanisms based on content type (recall application response time represents the delay from the first GET message to the first responding data packet for an HTTP request). Fig. 12(a) plots application response time distribution for three types of contents, video, image and text, that are served by Google servers. Most of the requested videos are YouTube videos, which are destined to Google servers because Google has migrated YouTube servers to its own network infrastructure after acquiring YouTube [30]. We observe from Fig. 12(a) that, perhaps surprisingly, videos have the lowest response time, followed by images and text. The reason why videos have the lowest response time might be that users tend to watch popular videos (note that YouTube iOS application has "Featured" and "Most Viewed" categories, convenient for users to select and watch those popular videos) and servers cache popular videos, leading to low response time. The slowest response time for texts might be because most of the requests for texts query search engines, which can take a longer time to respond.

Interestingly, the results for NHDs (see Fig. 12(b)) differ from those for MHDs: for NHDs, the response times for videos and texts are similar. We do not know the exact reasons that cause the difference between MHDs and NHDs. One observation is that the user interface for NHDs differ from that in MHDs: for NHDs, YouTube suggests related videos based on a user's preference (such as browsing history, local cache, etc.) while for MHDs, YouTube suggests featured and most viewed videos. Further investigation is left as future work.

In Fig. 12, we observe that the distribution of the response times for image flows that are served by Google contains



(a) MHD (b) NHD

Figure 13: RTT for MHD and NHD flows that are served by Google servers.

two modes. We conjecture that this is because the images are served by two clusters of servers, one closer to UConn than the other. Since Google data center information is not disclosed to the public, we try to infer the Google server locations through the RTT distribution [30]. Fig. 13 presents the RTT distributions for the flows with different content types. We indeed observe that the RTT distribution for image flows contains two modes, implying that they may be served by two clusters of servers at different geographic locations.

6.8 Number of Concurrent TCP Flows

As stated in Section 5.4, the number of concurrent TCP flows might be an indicator of the amount of CPU and I/O activities on an MHD device. We next present the relationship between the number of concurrent TCP flows and per-flow servicing rate.

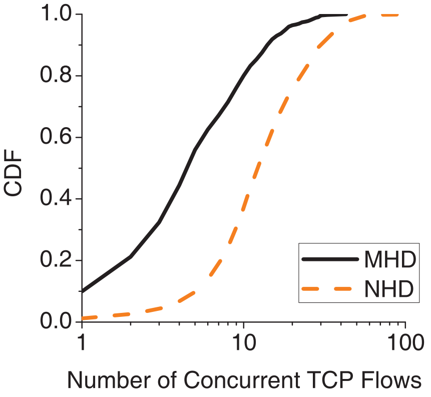


Figure 14: Average number of concurrent TCP flows for G_3 flows served by Akamai-113km servers.

We find that many G_3 flows have more than one concurrent TCP flow. Fig. 14 plots the distribution of the average number of concurrent TCP flows for G_3 flows served by Akamai-113km servers (for comparison, the results for both MHDs and NHDs are plotted in the figure). For MHDs, over 90% of the G_3 flows have more than one concurrent TCP flow, and there can be tens of concurrent flows. The results for the other three server clusters are similar (figure omitted). Intuitively, a larger number of concurrent flows on an MHD may lead to less resources to each flow, and hence lower per-flow servicing rate. To verify this conjecture, we obtain the correlation coefficient between the average number of TCP flows that are concurrent with a G_3 video flow and the per-flow servicing rate of the G_3 video flow. We find that the correlation coefficients are -0.03, -0.4 and -0.29 for G_3 video flows served by Akamai-Hartford, Akamai-113km and Google servers, respectively. The negative values indicate that indeed more concurrent flows can lead to lower per-flow servicing rates. The correlation is more significant for video flows served by Akamai-113km and Google. For the flows served by Akamai-Harford, the less significant correlation might be due to the superior performance achieved by this server cluster, which makes the effect of other factors less visible.

For comparison, let us look at the distribution of the average number of concurrent TCP flows for G_3 NHD flows in Fig. 14. Not surprisingly, the number of concurrent TCP flows on NHDs can be much larger than that on MHDs. On the other hand, we observe less significant correlation between the number of concurrent TCP flows and per-flow servicing rates on NHDs: in general, the negative correlation coefficient varies between -0.12 and -0.10 for the NHD flows served by the four server clusters. This less significant correlation may be due to superior computation and I/O capabilities on NHDs.

6.9 Findings from the 2012 Data Set

In the previous sections, we have presented the results from the data set collected in March 2011. We next report the results from the data set collected in April 2012.

In this data set, we find that only 64.7% of the TCP flows use HTTP, compared to 92.3% in the 2011 data set. On the other hand, the percentage of HTTPS flows has increased from 4.3% to 25.3%. This sharp increase is caused by the fact that many popular web sites such as Google, Facebook, and Hotmail, add the functionality to access their services using HTTPS, which protects users' privacy especially when they use public WiFi access points. For the rest of the TCP flows (11%), we do not observe any dominant application protocol.

Since the percentage of HTTP flows in the 2012 data set is much lower than that in the 2011 data set, and our approach to classifying MHD and NHD flows cannot be applied to the non-HTTP flows (since it uses User-Agent field in HTTP headers), leaving a large percentage (36.3%) of the TCP flows in the 2012 data set not analyzed, we have focused on the 2011 data set in this paper. We next describe the main results from analyzing the HTTP flows in the 2012 data set (developing new approaches to identify the device types for the non-HTTP flows in the 2012 data set, and analyzing their characteristics are left as future work). In the interests of space, we only present the main findings that differ from those from the 2011 data set.

- We observe that more MHD and NHD flows are served by Akamai and Google servers. Specifically, 43.5% of MHD flows and 38.4% of NHD flows are served by these two sets of servers, respectively. The number of distinct destination host domains accessed by MHDs has increased significantly to 9.3k (compared to 6k in the 2011 data set), while for NHDs, the increase is less dramatic (from 51k to 53k). The destination host domains accessed by MHDs are still less diverse than those accessed by NHDs. This is also evidenced by the observation that for MHDs, 38% of the flows are destined to the top ten host domains, while for NHDs, the percentage is 29%.

- We find that Google has deployed a set of servers (containing 12 IP addresses) very close to UConn, in the Connecticut Education Network at Hartford. Similar to the Akamai-Hartford server, this set of close-by Google servers provide high per-flow servicing rate due to small RTTs. We find 2.8% of MHD flows are served by these servers. In addition, analyzing the content type, we find that they mainly serve Youtube traffic (account for 45% of the MHD flows from these servers).

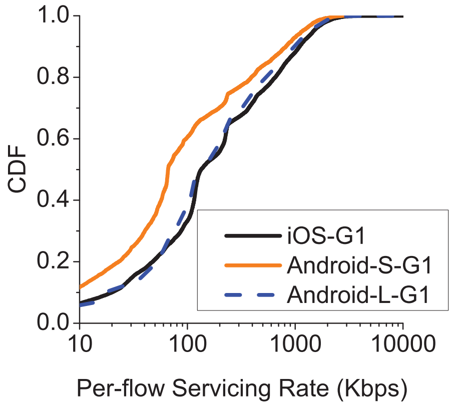


Figure 15: Distribution of per-flow servicing rate of G_1 iOS and Android flows that are served by Google servers from the 2012 data set, where Android-S-G1 corresponds to Andriod flows using small initial receive window (5KB) and Android-L-G1 corresponds to Andriod flows using large initial receive window (14KB).

- The amount of Android MHD flows has increased to 10% (compared to 3.2% in the 2011 data set), indicating the increasing popularity of Android devices. In addition, we observe 12% of the Android flows start with a significantly larger advertised receive window of 14KB, compared to the small initial receive window of 5KB that is observed predominantly in the 2011 data set. This larger initial receive window was adopted by newer versions of Android OS (starting from Android OS 4.X), and allows Android devices to better utilize the larger initial congestion window adopted by many servers. Fig. 15 plots the distribution of per-flow servicing rate of iOS and Andriod flows that are served by Google servers. We separate the Android flows into two groups, one using initial receive window of 5KB, and the other using initial receive window of 14KB. We only plot the results for G_1 flows; the results for G_2 and G_3 flows are consistent. Fig. 15 shows that iOS devices outperform Android devices that use small initial receive window, a point that we made in Section 6.5. On the other hand, the performance of Android devices that have adopted large initial receive window is comparable to that of iOS devices.

7. DISCUSSION

Our study has been conducted in a specific WiFi network, UConn campus WiFi network. A natural question is what findings from our study are specific to UConn network and what are applicable to other WiFi networks.

- Our findings related to hardware and software of MHDs and application-level protocols are not specific to UConn network, and hence we believe that they are equally applicable to other WiFi networks.
- The content delivery infrastructures in other networks may differ significantly from that perceived by UConn network. On the other hand, we believe MHDs in other networks also use Akamai and Google servers heavily because the most popular services accessed by MHDs are provided by major companies such as Facebook, Google, and Apple that use Akamai and Google servers heavily. The extent of usage and whether the usage by MHDs is heavier than that by NHDs in other networks, however, depend on the popularity of the applications in other networks.
- The use of Akamai and Google servers can also boost the network performance of MHDs in other networks in the US due to small RTTs provided by these servers. Specifically, the study of [15] reports that Akamai tends to deploy CDN sites close to the end users, and among all of the Akamai services, the 10th percentile and median of delays to Akamai are around 10ms and 20ms, respectively (these delays are comparable to the median delays of 8.5ms and 20ms from UConn campus to the two closest Akamai sites); Google is reported to provide similar RTT performance to most of the users in the US [27].
- The loss rates and RTTs in other networks may be significantly larger than those in UConn network. This implies that local losses and RTTs may play a significant role in network performance of MHDs in other networks. In addition, large initial congestion window adopted by many servers may not be strictly beneficial in other networks.

8. CONCLUSION AND FUTURE WORK

In this paper, we have studied the network performance of MHDs inside UConn campus network. We find that, compared to NHDs, MHDs use well provisioned Akamai and Google servers more heavily, which boosts the overall network performance of MHDs. Furthermore, MHD flows, particularly short flows, benefit from the large initial congestion window that has been adopted by Akamai and Google servers. Secondly, MHDs tend to have longer local delays inside the WiFi network and are more adversely affected by the number of concurrent flows. Thirdly, Android OS cannot take advantage of the large initial congestion window adopted by many servers, while the large receive window adopted by iOS is not fully utilized by most flows, leading to waste of resources. Last, some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks. Our observations provide valuable insights on content distribution, server provisioning, MHD system design, and application-level protocol design.

As future work, we plan to use active controlled experiments to understand why local RTTs of MHDs tend to be larger than those on NHDs, and to understand the bottleneck(s) of MHD flows. We also plan to study network performance of MHDs in other public WiFi networks. For instance, we believe that WiFi hotspots (e.g., hotspots in Starbucks) and other campus WiFi networks might have different network characteristics (e.g., higher loss rates and higher RTTs) as well as different content popularity among users. Furthermore, the content delivery infrastructure may differ significantly from that perceived by UConn network. Quantifying the network performance of MHDs and identifying the performance limiting factors in a wide range of settings will provide us better insights on designing network services for MHDs. Last, we plan to study how the performance of 3G cellular network differs from that of WiFi in our campus network and when/why users on campus switch from one network interface to another network interface.

Acknowledgement

The work was supported in part by NSF CAREER Award 0746841. We thank J. Farese, R. Kocsondy, J. Pufahl, and S. Maresca (UConn) for their help with the monitoring equipment and their assistance in gathering the data. We thank Ilhwan Kim and Jaewan Jang (NHN) and D. Xuan (Ohio State University) for helpful discussions. We also thank the anonymous reviewers for their insightful comments, and our shepherd A. Balasubramanian for many helpful suggestions.

9. REFERENCES

[1] Alexa. http://www.alexa.com.

[2] DAG card. http://www.endace.com.

[3] HTTP Pipelining. http://en.wikipedia.org/wiki/HTTP_pipelining.

[4] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window, 2002. RFC 3390.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of ACM IMC*, 2009.

[6] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *Proc. of ACM Ubicomp*, 2007.

[7] X. Chen, B. Wang, K. Suh, and W. Wei. Passive online wireless LAN health monitoring from a single measurement point. *ACM Mobile Computer Comm. Review*, 14, November 2010.

[8] C. M. Choon and R. Ram. Improving TCP/IP performance over third-generation wireless networks. *IEEE Transactions on Mobile Computing*, 2008.

[9] P. Deshpande, X. Hou, and S. R. Das. Performance comparison of 3G and metro-scale WiFi for vehicular network access. In *Proc. of ACM IMC*, 2010.

[10] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM CCR*, 40:26–33, June 2010.

[11] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proc. of ACM IMC*, 2010.

[12] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proc. of ACM MobiSys*, 2010.

[13] A. Finamore, M. Mellia, M. Munafo, and S. G. Rao. YouTube everywhere: Impact of device and infrastructure synergies on user experience. In *Proc. of ACM IMC*, 2011.

[14] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus WiFi networks. In *Proc. of PAM*, 2011.

[15] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and evaluating large-scale CDNs. Technical Report MSR-TR-2008-106, Microsoft Research, 2008.

[16] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proc. of ACM Mobisys*, 2010.

[17] IP2Location. http://www.ip2location.com.

[18] R. Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[19] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. of IEEE INFOCOM*, 2004.

[20] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband Internet traffic. In *Proc. of ACM IMC*, 2009.

[21] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Proc. of PAM*, 2010.

[22] A.-F. Mohammad, E. Khaled, R. Benjamin, and G. Igor. Overclocking the Yahoo!: CDN for faster web page loads. In *Proc. of ACM IMC*, 2011.

[23] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. In *Proc. of ACM IMC*, 2009.

[24] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proc. of ACM MobiSys*, 2010.

[25] A. Rahmati and L. Zhong. Human battery interaction on mobile phones. *Elsevier Pervasive and Mobile Computing Journal*, 5(5), 2009.

[26] A. Rao, Y.-S. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. of ACM CoNext*, 2011.

[27] K. Rupa, M. H. V., S. Sridhar, J. Sushant, K. Arvind, A. Thomas, and G. Jie. Moving beyond end-to-end path information to optimize CDN performance. In *Proc. of ACM IMC*, 2009.

[28] A. Shane and N. Richard. Application flow control in YouTube video streams. *SIGCOMM Computer Comm. Review*, 41, 2011.

[29] A. Shye, B. Sholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimization for mobile architectures. In *Proc. of IEEE/ACM MICRO*, 2009.

[30] R. Torres, A. Finamore, J. Kim, M. Mellia,

M. Munafo, and S. Rao. Dissecting video server selection strategies in the YouTube CDN. In *Proc. IEEE ICDCS*, 2011.

[31] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Measuring serendipity: connecting people, locations and interests in a mobile 3G network. In *Proc. of ACM IMC*, 2009.

[32] F. P. Tso, J. Teng, W. Jia, and D. Xuan. Mobility: a double-edged sword for HSPA networks: a large-scale test on Hong Kong mobile HSPA networks. In *Proc. of ACM MobiHoc*, 2010.

[33] D. Zhang. Web content adaptation for mobile handheld devices. *Communications of the ACM*, 50(2), February 2007.

[34] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *Proc. of ACM MobiSys*, 2010.