On Traffic Patterns of HTTP Applications

Brice Augustin, Abdelhamid Mellouk

On Traffic Patterns of HTTP Applications

Brice Augustin and Abdelhamid Mellouk
University of Paris-Est Créteil Val de Marne (UPEC)
Image, Signal and Intelligent Systems Lab-LiSSi Lab and Netw. & Telecoms Dept, IUT C/V
Transport Infrastructure and Network Control Group - TINC
122 rue Paul Armangot, 94400 Vitry sur Seine, France
{brice.augustin, mellouk}@u-pec.fr

***Abstract*—HTTP has been the most popular internet protocol for 30 years. Until recently, its role has been limited to a traditional transfer of hypertext documents. However, its flexibility and interoperability cause it to be progressively involved in a much wider range of applications, from video and audio streaming to email, chat and documents editing. Understanding the behavior of modern Web applications is a crucial step to apply QoS or security policies on this traffic. This paper studies 20 popular, Web applications that are representative of 12 application types. We describe a method to isolate and capture browser-generated traffic and plot time series with an `RRDTool` database. We show that modern Web applications present very diverse traffic patterns, and propose a description and classification of these patterns.**

***Index Terms*—traffic measurements, HTTP applications, internet**

I. Introduction

Networked applications traditionally use a dedicated network protocol to communicate. Among many examples, one can cite HTTP for hypertext documents, POP and SMTP for email, FTP for file transfer, NNTP for news and discussions, XMPP for chat, SSH and RDP for remote control. These protocols are largely tied to the TCP/IP protocol framework and have their own registered transport layer port number. In this model, there is a clear separation between the transport layer (mostly TCP), providing an end-to-end communication service, and the application layer, providing application-specific primitives and functionalities.

These applications and their associated protocols have their own traffic features and volumes, which makes them distinguishable from one other. [1]. They also have various QoS requirements and security concerns, therefore network administrators can choose to prioritize some applications, and block or throttle others.

We observe an evolution these last years, where an increasing number of application data concentrate and are transmitted over the sole HTTP(S) protocol. Further, applications also concentrate around a single user interface –a browser such as Firefox or Chrome. As a result, the HTTP protocol, which was designed as an application-layer protocol to browse in hypertext documents, slowly evolves towards a more generic usage with transport capabilities. [2] Fig. 1 illustrates the concentration of applications on top of the HTTP/TCP/IP protocol stack (the "HTTP hourglass").

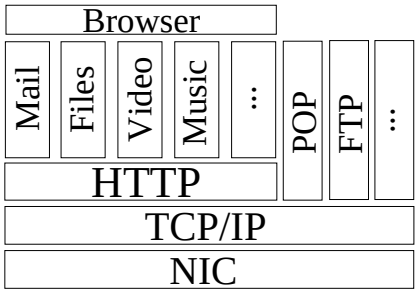

Fig. 1. Applications concentrate on top of HTTP and a unique user interface, the browser.

Several reasons explain this evolution. First, normalizing a network protocol is a long and tedious task made of compromises for consensus (i.e. IETF for RFCs and IANA for port numbers). Second, network administrators apply restrictive policies (e.g., port filtering) that block emerging applications and impede their growth. Third, from a user point of view, it is not convenient to handle a new user interface for each application (e.g., browser, MTA for email, FTP client,...) and it contradicts the current *cloud computing* trends, in which data is stored online to be accessible from any device. HTTP offers solutions to these issues, thanks to its wide flexibility and interoperability.

For network administrators, this evolution has serious consequences. Traffic classification tools are generally port-based, but ports 80 and 443 can now carry much more than just regular Web documents. As a result, the traffic patterns and volumes of HTTP traffic now exhibits a wide diversity, depending on the type of Web application. Understanding the behavior of today's Web applications is a crucial step to apply QoS or security policies on this traffic. For instance, a video application (e.g, YouTube) and a mail reader (e.g., GMail) have completely different QoS requirements, although both applications are built on top of the same HTTP protocol. As a result, this knowledge can help optimizing Web applications. [3], [4]

This paper is a first step towards an accurate classification of Web applications. We explore the diversity of the traffic patterns generated by 20 browser-based applications categorized in 12 classes (video, music, remote control,...). Sec. III describes a method to isolate and capture browser-generated traffic and plot time series with an `RRDTool` database. We

describe and characterize the traffic patterns in Sec. IV. Finally, Sec. V concludes with a preliminary classification of applications and proposes directions for future work.

II. Related work

HTTP has been extensively studied for over a decade. Studies are based on aggregated traffic captured at internet hubs or large organizations (e.g., universities) access links. These studies expose general trends in HTTP usage and workload [5], [2], or focus on some particular Web 2.0 applications such as interactive maps. [6], [7], [8], [9], [10] Our work comes as a complement, by providing a fine-grained study of the various Web applications. In addition to HTTP, the workload of other applications has been characterized. Traditionally, these applications use a dedicated protocol, such as RTSP for media streaming [11] or RDP for remote desktop control. [12] In this work, we show that they may also have a Web-based version that use the sole HTTP protocol.

The concentration of applications over a single protocol stack has been noticed and predicted for a while. Ethernet and TCP/IP are flagrant examples of this trend. For instance, Deering describes various types of "IP hourglass" [13], predicts a narrowing of Link-level and Network layers, but his work does not focus on higher layer protocols such as HTTP. HTTP tunneling [14] is a way to bypass firewalls' restrictive policies by encapsulating a blocked application in HTTP requests and responses. Tunneling is a quite marginal trick used by computer enthusiasts to use their favorite illegal program. On the contrary, what we observe here is a general trend followed by widely-used and legacy Web applications. Note that the generalization of HTTP usage in all types of applications (non only browser-based) might speed up with the introduction of WebSockets. [15]

Methods for internet flow classification can be either port-based, feature-based or content-based (DPI). [1] However, these methods usually do not provide a specific breakdown of applications running on top of HTTP. Several industry bandwidth shaping and monitoring tools [1] classify some bandwidth-intensive HTTP content (e.g., YouTube videos) as a specific category, apart from a generic HTTP category. However, we believe that all HTTP traffic should be classified into sub-categories such as the ones we introduce in this paper (e.g., video, music,...).

III. Methodology

This section details our methodology to capture, visualize and analyze the traffic patterns generated by a set of heterogeneous and popular Web applications. In the next paragraph, we will start by presenting the applications under study.

Table I summarizes the Web applications studied in this work. We selected 20 applications, that we consider to be representative of the current trends in Web usage. Furthermore, they are all based on the HTTP(S) protocol, i.e. running on port 80 or 443 on the server side. The next section describes a method to isolate, capture and visualize the traffic generated by these applications.

[1]Allot: http://www.allot.com/; PacketShaper: http://www.bluecoat.com/products/packetshaper; SolarWinds: http://www.solarwinds.com/

A. Capturing Web traffic

Capturing Web traffic accurately requires a slightly more sophisticated method than simply launching a packet capture tool, such as `tcpdump`, on the test machine. Indeed, in doing so we might capture traffic generated by other applications running on the machine. Another solution might be to apply a packet filter and capture only traffic on the TCP assigned to HTTP, i.e. 80 and 443. However, other applications, not running in a browser, are also known to generate Web traffic in order to bypass firewalls. Remember that we are interested only in the traffic generated by browser-based applications, so we ignore these applications (e.g., Skype) in this study.

Therefore, we take the approach of capturing the sole HTTP traffic generated by the Web browser that is running the application under study. To label packets and flows with process information, two various of the same technique are available. In the GT and Macroscope [16], [17] approach, a basic packet capture tool sniffs a network interface and saves all packets in a trace file; periodically and independently of the packet capture, a script dumps the host's TCP connection table into a trace file. The TCP connection table is a list of all TCP connections that are currently maintained by the system. For each connection, it lists the remote and local ports, the IP address of the remote host, and also the identifier of the process that initiated the connection. Finally, the two datasets are merged, based on the timestamp and socket informations. Each packet in the trace is assigned to a TCP connection, then to the process identifier that created the connection. This offline approach is not optimal for our needs. It requires to query the connection table regularly, possibly for no use if no new connection was created. Worse, if the query period is too long, the merging might be unable to assign a short-lived TCP connection to its corresponding process identifier, because its entry might have been removed from the connection table before the script queries it. As a result, there is a risk that some connections remain unlabeled.

In another variant, packet capture and application assignment are tightly coupled. To the best of our knowledge, the `NetHogs` [18] network monitoring tool is the only implementation of this variant. For each packet that it captures, it tries to match it with a known connection (and consequently, to its process identifier), looking in a local copy of the TCP connection table. If no match is found, it queries the OS kernel to gather a fresher copy of the table and updates its own copy for the subsequent packet matching. This variant has the advantage of reducing the overhead (it performs only necessary OS queries), while increasing the accuracy (it always uses the freshest connection table). Therefore we take this later approach to implement our tool. We ported the Linux version of NetHogs to Windows, and we use this later version in our experiments.

B. Capture scenarios

In this paper, we restrict ourselves to a series of capture scenarios (one per application), in which a user is asked to use the application for 15-20 minutes and do whatever he/she would usually do with this application. To make the scenario

	Type	Selected applications	Scenario
1	Radio	Radioways, Live365	Listen to a few radio stations
2	Music	Deezer, Jiwa	Listen to the user's playlist
3	Video	YouTube, Dailymotion, Fox News Live	Search and watch videos
4	File transfer	RapidShare, Microsoft SkyDrive	Upload and download user's files
5	Mail	Gmail	Check, read and send mails
6	News	Google Reader, My Yahoo!	Monitor user's RSS feeds
7	Documents	Google Documents, Microsoft Word Web Apps	Typeset a short text and save it
8	Maps	Google Maps	Point to a few locations; calculate an itinerary
9	Photos	Picasa	View a slide show of user's pics; upload some pics
10	Remote control	LogMeIn, consoleFISH/ajaxterm	Launch and control an application remotely
11	Chat	Meebo	Chat with user's acquaintances
12	Gaming	Runescape	Play an adventure game

TABLE I
SUMMARY OF WEB APPLICATIONS STUDIED IN THIS PAPER.

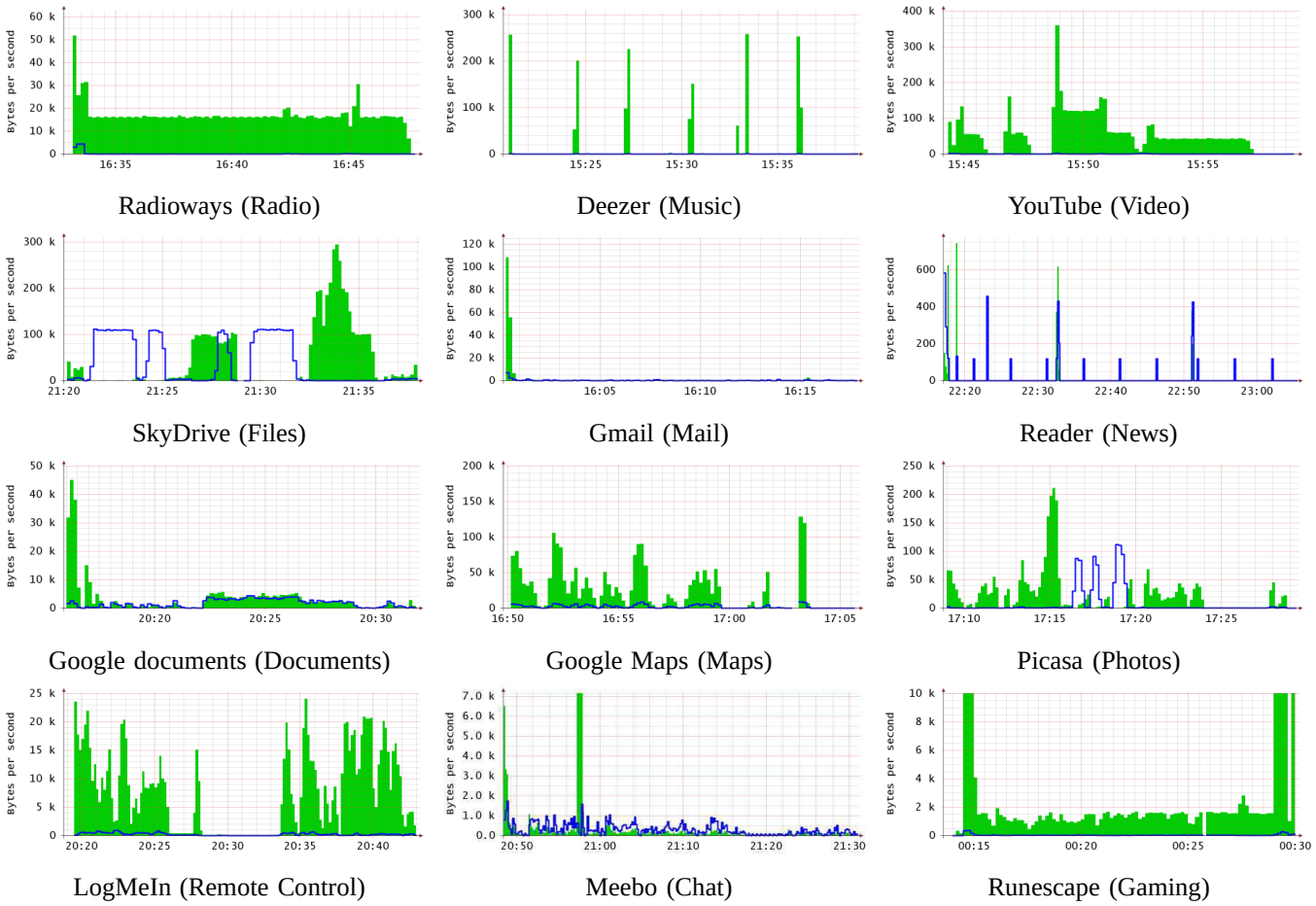

Fig. 2. Traffic times series for selected Web applications.

more realistic, we allow the user to manipulate several Web applications at the same time (e.g., checking email in a browser tab while listening to music in another), but we require the monitored application to be run in a totally separate browser instance. During this period of time, our tool monitors the traffic, filters the traffic generated by the monitored application (using the process identifier of the browser) and maintains two counters: the number of bytes sent and received by the application. An RRDTOOL [19] database stores the value of these counters every 10 seconds and plots the time series at the end of the monitoring period.

We perform all our experiments from a typical home network located in the suburbs of Paris, France. A speed test from the test machine indicates a 750 KB/s download speed, and 110 KB/s upload speed. This benchmark will help us to calibrate the results, in particular regarding the maximum bandwidth required by the tested Web applications.

The last column of Table I describes the actions performed by the users during the capture scenario of each application.

C. Describing traffic patterns

We describe the traffic patterns according to three aggregate traffic features:

- Traffic **intensity** is the total volume of traffic exchanged by the application during the measurement period. We define three levels of intensity: *high*, *medium* or *low*.
- Traffic **symmetry** measures the ratio between upstream and downstream traffic. Traffic is *symmetric* if the amount of traffic is similar in both directions, and *asymmetric* otherwise.
- Traffic **shape** describes the general aspect of the traffic pattern. We define four shapes: *dirac* (short burst periods followed by zero traffic), *rectangle* (long burst periods with constant rate), *variable* (variable rate) or *continuous* (rate remains constant during the whole measurement period).

IV. Traffic patterns

Fig. 2 presents a typical time series for each type of applications under study. For space reasons we do not show the plots for all applications, but we rather select one representative application per type, as defined in Table I (music, video, documents,...). For each plot, the X axis represents the time of the day, [2] while the Y axis indicates the traffic rate, in bytes per second. Downstream traffic is plotted with a (green) plain area, upstream traffic with a (blue) solid line. Note that the X and Y ranges vary significantly from one plot to another. Fig. 3 presents general traffic statistics for each application (average and maximum bandwidth, in upstream and downstream).

A first look at the plots show the wide diversity of modern Web traffic patterns. We now describe some interesting characteristics that we observe, and explain the underlying phenomenons that cause these patterns.

Spiky traffic. Music applications (e.g., Deezer and Jiwa) create very short bursts of traffic at regular intervals. These bursts happen when the application fetches the next song to be played in the playlist. The content is downloaded in its entirety at the highest achievable speed (the download lasts a few seconds), and buffered in the browser before it is played. During the play, the application generates no traffic, and the start of the next download is tied to the end of the current song. As song tracks are generally formatted to last 3-5 minutes, this explains the regular traffic bursts that we observe. The traffic is highly asymmetric, with a medium traffic volume (20KB/s on average).

Apart musical applications, only News applications exhibit the "dirac" pattern. Google Reader checks the user's RSS feeds every five minutes, which creates brief, low volume traffic peaks.

Bandwidth-intensive traffic. Video applications present relatively large periods of constant traffic rate, followed by silent periods with unpredictable durations. Contrary to musical applications, where the inter-peaks duration can be predicted because it depends on the duration of the current song track, the traffic for video applications (e.g., YouTube) is mostly regulated by the user, who searches and selects each video before he/she starts watching it. A more detailed analysis of YouTube raises two interesting remarks. First, we notice a short peak of traffic during the first seconds of a video, followed by a longer period where bandwidth is clearly throttled. [3] We believe that this initial boost helps the application buffering enough video in order to compensate for congestion that might occasionally reduce the download speed. Second, we notice three levels of throttling (40, 60 and 120 KB/s), that might be tied to the video duration or even to its bit rate. Resolving the question will require further analysis.

Maps and Photos represent two other bandwidth-intensive Web applications. When a user seeks a new location on Google Maps, large traffic spikes (up to 100 KB/s) are generated. During this initial step, the AJAX application fetches the set of tiles surrounding the location of interest, as well as other metadata such as city and street names. Traffic then decreases progressively as the user adjusts the map position to his exact needs. This effect is particularly visible during the first five minutes of the time series. Note also the non-negligible upload traffic (up to 10 KB/s). Indeed, a map is an assemblage of 256 x 256 pixels tiles, and the application fetches each tile one by one through an HTTP request. These numerous HTTP requests are responsible for the increase of the upstream traffic (each HTTP request weighs about 1KB).

Slide shows and random visualizations in Photo applications also create variable traffic spikes (up to 200 KB/s). Unlike Maps, which are composed by a set of small tiles and thus generate a series of small HTTP requests, Photo applications manipulate larger Web objects (a photo weighs several MB), and therefore require less HTTP requests, which explain the lower upstream traffic during picture visualization. However, another difference with Maps application is the possibility to upload photos on the server. We observe a large increase in upstream traffic (up to 100 KB/s) in the time series, which corresponds to the user who manually uploads a set of photos. In this case, bandwidth throttling is due to the limited upstream bandwidth allocated to the user's ADSL line, as explained in Sec. III-B.

Patterns for "File transfer" applications are very similar to Photos applications, in the sense that they generate high traffic loads, both in the downstream and upstream directions. In our example, SkyDrive's application limits the download speed to 100 KB/s, but this throttling is applied per downloaded file. Indeed, towards the end of the scenario, the user started downloading two, then three files concurrently, which had the effect of increasing the bandwidth to 300 KB/s. Just like for Photos, the upload speed is curbed by the user's ADSL line speed.

"Background" traffic. Chat, Documents, Mail and text-based Remote Control (i.e., Web-based SSH, not plotted here) exhibit very similar patterns with a low, symmetric and quite variable traffic. We explain this similarity by the fact that these applications are text-oriented and involve a high

[2] Although the time of the day may have an impact on traffic patterns, due to network congestion, we did not take this parameter into account in this work.

[3] Additional experiments indicate that bandwidth throttling happens on the client side, in YouTube's Flash application.

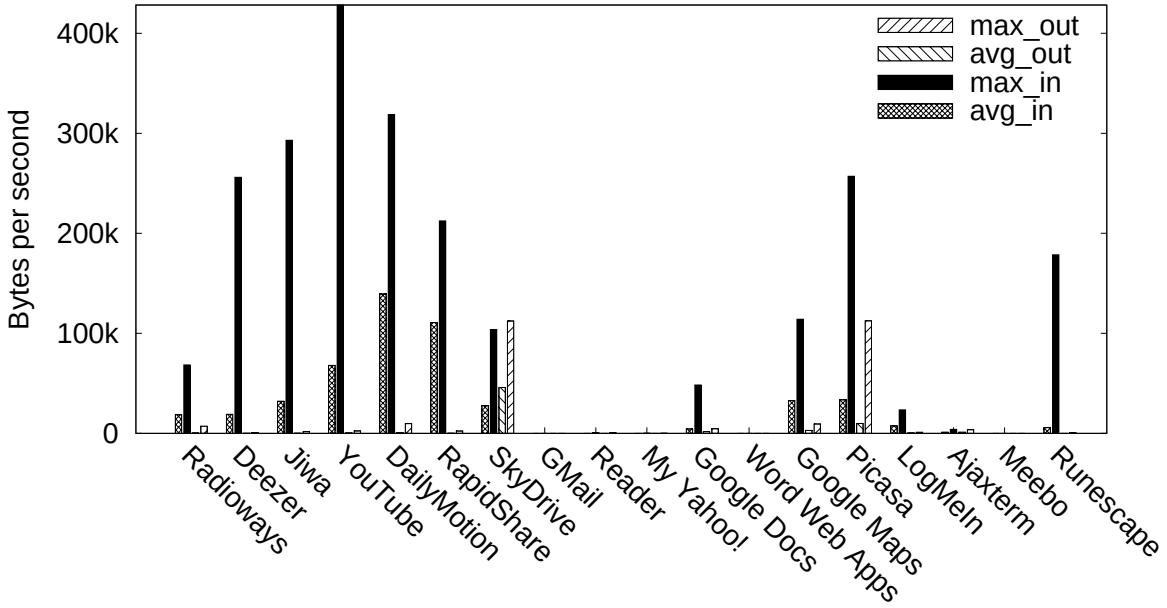

Fig. 3. Web applications statistics (*max*: maximum bandwidth; *avg*: average bandwidth; *out*: upstream traffic; *in*: downstream traffic).

level of interaction (typesetting) with the user. Interestingly, typesetting in Google Documents and Microsoft Word Web Apps generates quite a large amount of traffic (4-5 KB/s in both directions), possibly because each key pressed causes the emission of a request to the server. GMail generates a low background traffic due to small (hundreds of bytes) chunks of data regularly exchanged between the browser and the server. In order to fetch and display emails as soon as they arrive in the user's inbox, the GMail application uses long polls initiated by the client, that enable the server to notify instantaneously the client about new emails.

Constant traffic. Constant traffic is unexpected in Web applications, because of the nature of HTTP itself, that creates and maintains short-lived TCP connections. However, we found three application classes that generate continuous, downstream traffic. Radio streams are capped to about 20 KB/s, and interestingly, the average bandwidth for the Radioways scenario (Radio) is similar to that of the Deezer scenario (Music). Perhaps both applications use the same audio compression ratio and codecs. The only Gaming application that we studied (Runescape) also generates a somewhat constant download traffic. However, gaming applications clearly require more examples and analysis to conclude. Note that a large number of TV channels provide live broadcasts on their Web site, as well as catch-up TV for popular shows and series. However, most of these contents are delivered through RTMP [4], and therefore we ignore them in this study. The only exception we found is Fox News, whose Live service broadcasts a 150 KB/s video stream over HTTP.

V. Conclusion

Fig. 4 proposes a preliminary draft for a classification of Web applications according to the three traffic features introduced in Sec. III-C. The Y axis represents traffic levels, and the X axis indicates the shape of the time series. Finally, unidirectional arrows stand for asymmetric traffic, while the bidirectional ones depict symmetric traffic. Most applications are scattered on the figure, which may indicate that an accurate classification of Web applications is feasible.

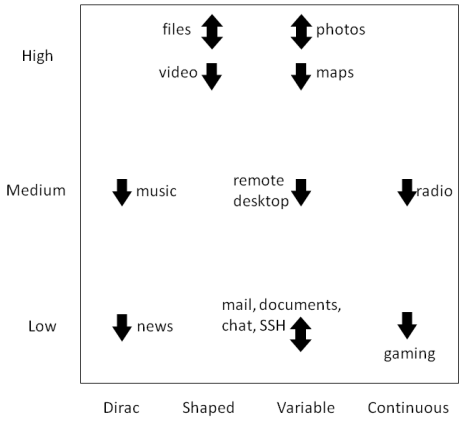

Fig. 4. Preliminary draft for a classification of Web applications.

However, the work presented in this paper is only a first step towards a precise classification of the plethora of HTTP applications available on today's Web. In particular, our next step is to collect data from more users, to evaluate how user diversity impact application usage and the associated traffic patterns. More sources (with various types of internet connections), terminal types (e.g., smartphones, tablets) and measurements at various times of the day, would also perhaps allow us to observe additional traffic patterns.

[4]Real Time Messaging Protocol is a protocol developed by Macromedia/Adobe.

REFERENCES

[1] H. Kim, D. Barman, M. Faloutsos, M. Fomenkov, and K. Lee, "Internet Traffic Classification Demystified: The Myths, Caveats and Best Practices," in *Proc. ACM CoNEXT*, December 2008.

[2] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proc. ACM SIGCOMM Internet Measurement Conference*, November 2009.

[3] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang, "WebProphet: Automating Performance Prediction for Web Services," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, April 2010.

[4] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann, "Improving Content Delivery using Provider-aided Distance Information," in *Proc. ACM SIGCOMM Internet Measurement Conference*, November 2010.

[5] T. Callahan, M. Allman, and V. Paxson, "A Longitudinal View of HTTP Traffic," in *Proc. Passive and Active Measurement Workshop*, April 2010.

[6] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann, "The New Web: Characterizing AJAX Traffic," in *Proceedings of the 9th International Conference on Passive and Active Network Measurement*, April 2008.

[7] S. Lin, Z. Gao, and K. Xu, "Web 2.0 traffic measurement: analysis on online map applications," in *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '09, 2009.

[8] S. Veres and D. Ionescu, "Measurement-based traffic characterization for Web 2.0 applications," in *Instrumentation and Measurement Technology Conference, 2009. I2MTC '09. IEEE*, May 2009.

[9] L. Shuai, G. Xie, and J. Yang, "Characterization of HTTP behavior on access networks in Web 2.0," in *Telecommunications, 2008. ICT 2008. International Conference on*, June 2008.

[10] P. Nagpurkar, W. Horn, U. Gopalakrishnan, N. Dubey, J. Jann, and P. Pattnaik, "Workload characterization of selected JEE-based Web 2.0 applications," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, September 2008.

[11] M. C. Alec, A. Wolman, G. M. Voelker, and H. M. Levy, "Measurement and Analysis of a Streaming-Media Workload," March 2001.

[12] I. Humar, J. Bešter, and S. Tomažič, "Characterizing graphical desktop sharing system's workload in collaborative virtual environments," in *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, ser. CCNC'09, January 2009.

[13] S. Deering, "Watching the Waist of the Protocol Hourglass," IETF 51, 2001, www.iab.org/documents/docs/hourglass-london-ietf.pdf; Last access: January 2011.

[14] L. Brinkhoff, "GNU httptunnel," June 2008, http://www.nocrew.org/software/httptunnel.html; Last access: January 2011.

[15] I. Hickson, "The WebSocket API," February 2011, http://dev.w3.org/html5/websockets/; Last access: March 2011.

[16] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and k. c. claffy, "GT: picking up the truth from the ground for internet traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 39, October 2009.

[17] L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft, "Macroscope: End-Point Approach to Networked Application Dependency Discovery," in *Proc. ACM CoNEXT*, December 2009.

[18] A. Engelen, "NetHogs: What program is using that bandwidth?" 2008, http://nethogs.sourceforge.net/; Last access: January 2011.

[19] T. Oetiker, "RRDTool - Round-Robin Database Tool," 1999, http://www.mrtg.org/rrdtool/; Last access: January 2011.