

Bill-Pay Control: An Interactive User Interface to Select IP Service Quality

Aaron Bergstralh Erik Paulson
bergstralh@wisc.edu epaulson@cs.wisc.edu

David Plonka
plonka@cs.wisc.edu
University of Wisconsin - Madison
Computer Sciences Department

December 15, 2006

Abstract

The future Internet may be built to deliver better service to traffic that is accompanied by additional payments. In such a scenario, changes in payments and the resulting performance impact could be realized on very fine timescales, such as within seconds. Thus, users would benefit from (1) a desktop instrument that measures and presents their applications' Internet traffic in a user-friendly fashion and (2) desktop controls that allow the adjustment of desired service level, and corresponding payment, for their application traffic in real time.

In this paper we present *Bill-Pay Control*, an interactive, web-based user interface that we designed and implemented as part of *Bill-Pay Sim*, a simulation system that emulates the performance that such a future Internet might exhibit. *Bill-Pay Control* runs on a user's workstation, continually displays application-level network traffic statistics, and allows the user to select accompanying service levels. This functionality provides the user with the flexibility to improve network and application performance by controlling payments tied to the chosen traffic. We also report our initial experience with the system and make observations about its efficacy as an evaluation environment for proposed Internet billing schemes.

1 Introduction

Traffic in today's network is prioritized and purchased at a very coarse granularity. Most traffic is treated as a flat-rate best effort, and whatever other arrangements exist were negotiated by administrators or ISPs between networks, and subsume all users. In a future network, users may use a micro-payment system for purchasing and prioritizing their own network traffic[4]. A tool to monitor traffic and adjust traffic priority will be necessary for such a network to be possible.

We have built a system called *Bill-Pay Sim* (BPS) in which we provide a user interface, *Bill-Pay Control*, to both monitor traffic and influence its performance. Amongst other things, this

prototype interface could be used in future user surveys for micro-payment systems. The BPS system is comprised of a traffic logger, web based user interface, traffic policer, and a central daemon that coordinates the system.

The BPS traffic logger (bpslog) is a tool that wraps any chosen dynamically-linked application, such as a web browser, and captures all of the network-relevant function calls to get a true view of a user application's network activity. The output of bpslog is then used by the central BPS daemon (bpsd) to relay this information to the user interface.

The BPS user interface allows end-users to view a summary of their recent traffic through any common web browser. This interface allows the end user to view various information about their traffic including current and peak transfer rates, path to the remote host, and current level of service. The user may then use this information to make a decision to increase or decrease the level of service given to a particular traffic aggregate.

In order to allow the user to adjust the service level of their traffic, BPS uses the traffic policer built into Linux to rate limit traffic on the machine. Given that one cannot currently increase the performance of the Internet along arbitrary paths, bpsd initially constrains all incoming traffic to a low level. Lowering the baseline performance of the network allows us to subsequently simulate increasing the performance of the network by loosening that constraint. When a user modifies the service level for some traffic the bpsd modifies the maximum rate for the remote hosts involved.

The final component of BPS is the central bpsd daemon. The bpsd takes the output of the traffic logger and parses and prepares it for the user interface. When a user makes a change in the user interface, the bpsd takes the request and translates it into rules for the traffic policer.

The rest of this paper is organized as follows. Section 2 explains some of our design choices. Section 3 presents the system usage and implementation details. Section 4 discusses our results and observations. Section 5 notes some related work. Our conclusions follow.

2 Design

Our primary design goals for a Bill-Pay Control and simulation system were for it to have: (1) a web-based user interface that is simple and responsive, (2) a network traffic logger for *unmodified* applications that doesn't adversely affect user experience with respect to performance, and (3) traffic control that is implemented solely within the user's workstation so that the overall system is self-contained.

One of the motivating uses for our system is for a "mall-test" environment, where users could be recruited from the general public and asked to interact with the fine-grained control a Bill-Pay system provides. In this environment, we wish to have a system that can be deployed on a single laptop without needing a complicated support infrastructure. An emulated environment that could skew experiences is undesirable; the applications we are using to test network performance should themselves operate at full speed.

In addition, we need an effective way to summarize application traffic so that the user can conveniently express their performance preferences. We would like to display aggregates of IP conversations, both spatially (within the IP address space) and temporally (spanning a sequence of conversations involving the same hosts), so that the user is not overwhelmed by the numerous interactions between their local host and the many remote Internet hosts with which it communicates. Two areas of design concern were how best to associate traffic with abstract handles called

traffic descriptors, and how to best simulate poor Internet performance.

2.1 Traffic Descriptors

It is challenging to displaying Internet traffic in real time such that it can be understood by a typical user. The number of remote hosts involved in an application session can be overwhelming. For instance, when ostensibly visiting even only a few web sites, “hidden” references can result in communication with *hundreds* of hosts. It is desirable to group the traffic involving remote hosts together into the the smallest possible set that still allows the user to meaningfully differentiate between sets of traffic that experience similar network performance. Thus we consider traffic aggregated and named with identifiers that we call “traffic descriptors.” These traffic descriptors are displayed in the user interface and identify the entries that can have their desired service level set by the user.

As we discuss the design issues involving traffic descriptors we will use the example of accessing “www.cnn.com”, the web server for the Cable News Network.

2.1.1 IP Addresses

IP addresses are confusing for the user. For instance, accessing “http://www.cnn.com” currently yields this set of IP addresses: 64.236.16.20, 64.236.16.52, 64.236.16.84, 64.236.16.116, 64.236.24.12, 64.236.24.20, 64.236.24.28, 64.236.29.120. IP addresses are not persistent, they are foreign, complicated looking and contain too many digits for most to remember. Since IP addresses make poor candidates for traffic descriptors, we chose to only use them as a last resort, i.e. when an application connects to a service using just the IP address, for instance in a URL, and we can find no other reasonable descriptor, such as when it is within the local domain and thus there is no other service provider that carries traffic along the end-to-end path to that remote host.

2.1.2 Domain Names

Domain names are somewhat familiar to users. Users see them as the server names in URLs to which they direct their web browsers, such as “www.cnn.com”. As such they are potentially useful as traffic descriptors.

However, when accessing “http://www.cnn.com” the index document contains very many sub-references to foreign domain names that never appear on the user’s web browser display. In fact, a recent loading of that URL yielded sixteen domain names resolved by the browser, only five of which contained “cnn”.

Still, domain name resolution can be passively intercepted in the application. This enables processing in real time because the system will know the associations between names and IP addresses before connections are made to those IP addresses.

An improvement we considered was to drop variable length prefixes from the domain name. For instance, we could aggregate names matching “*.cnn.com” to just the traffic descriptor “cnn.com”. Unfortunately, automatic abbreviations of this sort are difficult to generalize. For instance, in the United Kingdom, CNN’s web site might be known as “www.cnn.co.uk”. If we simply dropped all but the last two domain name components, we would be left with only “co.uk”. As a traffic

descriptor, this would aggregate all commercial domain names in the UK together, losing desirable fine-grained user control of service levels on a service by service basis.

While we find this domain name abbreviation to be fruitful, convenient automated abbreviation is left for future work.

2.1.3 Reverse Domain Names

Since the socket-based library and system calls generally use IP addresses rather than names, we first intended to capture IP addresses and lookup their “reverse” domain names as the basis for traffic descriptors, then perhaps abbreviate by the method just discussed. These names are entries in the “in-addr.arpa” domain namespace, indexed by the reversed bytes in a given IP address. The notion was that we would observe the application establishing communication with a given IP address and then quickly lookup the associated reverse domain name before reporting that traffic to the user. For instance, the reverse domain name for the IP of “www.cnn.com” is again “www.cnn.com”, which is sensible.

There are a number of problems with this technique. First, a domain name lookup is a blocking operation that can take a prolonged amount of time and can ultimately fail, and often does due to misconfigured pointers or disfunctional reverse name servers. Secondly, the values for reverse domain names are often missing since they are not generally required for proper Internet operation, but exist primarily just for reference. To further complicate matters, these reverse names are sometimes in error. Because reverse domain names are not used to identify services, there is no continuous verification that they are reasonably maintained on the Internet. In any case, even with correct resolution, the resulting names are often unfamiliar to the user since they need not match the “forward” domain names contained in, for instance, web URLs.

Because of the practical complication of performing many concurrent blocking reverse domain name resolutions in near real-time and the potential confusion the resulting names might produce anyway, we decided not to use reverse domain names as traffic descriptors.

2.1.4 IP Address Prefixes

The current BGP table’s IP prefix that best matches for “www.cnn.com” is 64.236.16.0/20. Like IP addresses, IP prefixes, which consist of a base IP address and netmask bit-width, are foreign and confusing to users. For this reason we rejected them as traffic descriptors.

However, these IPv4 prefixes from the global BGP table are likely to have a useful granularity for choosing service levels in that they are often contain a set of co-located servers for services operated by one service provider, and thus match a subnet that might exhibit poor performance such that the user might be willing to pay that provider.

2.1.5 Autonomous System Numbers

Autonomous System Numbers (ASNs) are the identifiers used to route inter-domain traffic for the entire Internet, but they are totally foreign to typical end users. At first this makes them seem a poor choice as traffic descriptors. However, in a system that offers payments to commercial entities, it is these ASNs that likely exactly represent those entities that are responsible for the network performance involving the remote host.

With the aid of the Regional Internet Registry (RIR) databases, such as that of the American Registry for Internet Numbers (ARIN), we can replace the ASNs with short names that are easier for users to understand. For instance, “cnn.com” is operated by an Autonomous System named “ASN-TBS-1” for Turner Broadcast System, the owner of the Cable News Network (CNN).

2.1.6 Autonomous System Paths

Autonomous System paths are simply the sequences of ASNs that are on the path to a particular destination prefix in the whole Internet’s BGP routing table. For instance, the path from our university to CNN is “WISNET1-AS, LEVEL3, AOL-ATDN, ASN-TBS-1”, meaning that traffic from here to CNN is likely to traverse WiscNet, Level 3 Communication’s network, the AOL Transit Data Network, and ultimately the Turner Broadcasting network. For demonstration purposes, we assume that the AS paths are symmetric such that the reverse of that path would contain identify the Autonomous Systems from CNN to our university.

Ultimately, we chose to use a combination of Autonomous System (AS) paths and domain names as traffic descriptors in our system. The domain names give the user the opportunity to associate the traffic descriptor with the domain names they typically see as server names in URLs. The AS path gives the user some sense of network distance to the remote host and provides the number and identities of the commercial entities along the end-to-end path. Those commercial entities are exactly those that the user may wish to offer payments for improved service.

2.2 Traffic Control

2.2.1 Traffic Control Method

Our design goals and implementation issues left us with a few options to lower baseline network performance. Network emulation tools such as NIST Net[3] and dummynet[6] have more features than necessary for our purpose, so we feared that they would complicate our system and/or require another machine to perform rate-limiting. An embedded traffic shaper inside the application or kernel would be complicated to write. We eventually chose the traffic control mechanisms built into Linux. The Linux Traffic Control system[5] was built-in to most versions of Linux, simplifying deployment. It adds no execution overhead to the application. It also supported ingress traffic control as easily as outgoing control. This combination allowed us to meet our goal of being able to influence the rate at which traffic is delivered to the application in a self-contained system.

2.2.2 Traffic Control Service Levels

The BPS system presents users with a limited set of “service levels.” We allow the user to change the service level of a particular traffic descriptor rather than allowing the user to specify an arbitrary bit or payment rate. The primary motivations for this approach were that the average user of a micro-payment system may find named service levels easier to choose amongst than arbitrary numbers. Another motivation for this approach is that service levels yield traffic rate changes in noticeable steps allowing users to experience a near instantaneous improvement in their traffic. A final motivation is that service levels allow other parts of the system more flexibility in what characteristics a descriptor should exhibit. For example, a “silver” service level might yield twice the throughput of the “bronze” level, whereas the “platinum” level might also be at twice

the bandwidth of “bronze” and exhibit reduced latency. However, at present our simulator only implements bandwidth rates.

2.2.3 Traffic Control Granularity

We next consider effective ways of turning the user’s requests into Traffic Control rules. The Traffic Control utilities can operate on IP/port address pairs, individual IP addresses, and entire network blocks. IP/Port address pairs are too fine-grained to apply Traffic Control rules. Many applications, such as the world web web, result in short-lived sessions involving ephemeral port numbers. Rules for one IP/Port pair would not affect subsequent connections with different IP/Port pairs, and our network activity monitor needs a few seconds to propagate information about new network activity to the Traffic Control rule writer.

Network blocks derived from the BGP table entries are too coarse-grained to apply Traffic Control rules. A rule for an entire BGP prefix would limit all matching traffic in aggregate to a given rate. However, we intend the service level to apply to the traffic involving each individual remote host. For example, consider a traffic descriptor that the user has set to “silver” level, meaning traffic can flow at up to 2Mbps. If there are 20 remote host addresses matching that descriptor, a prefix-based traffic control rule would cause all 20 to share 2Mbps of bandwidth. However, if the Traffic Control rules are written for each individual host, each of the 20 hosts would get their own 2Mbps. This more closely approximates Bill-Pay in that remote hosts that are associated with the same micro-payment amounts still behave individually with respect to their performance. We therefore choose to generate Traffic Control rules for individual remote host IP addresses.

3 Implementation

The Bill-Pay Sim system contains three major components: a user interface, a network activity logging utility, and a daemon. Respectively, these components are “Bill-Pay Control”, the “Bill-Pay Sim Logger” (bpslog), and the “Bill-Pay Sim Daemon” (bpsd). Their relationship is shown in figure 1.

The system’s components are implemented in about 3300 lines lines of javascript, perl, and original C code. The library interposition by bpslog and traffic control techniques employed by bpsd are somewhat Linux-specific and function on systems based on, at least, the Linux 2.4 and 2.6 kernels. The user interface requires a javascript-enabled web browser such as Firefox, but need not necessarily be run on the simulation’s workstation.

3.1 System Operation

Typical operation of the system is described below. The pertinent system elements are annotated with corresponding numbers in figure 1.

1. The user runs the bpslog command to launch an Internet application of their choosing such as the Firefox web browser. The bpslog shared object library intercepts interesting calls and

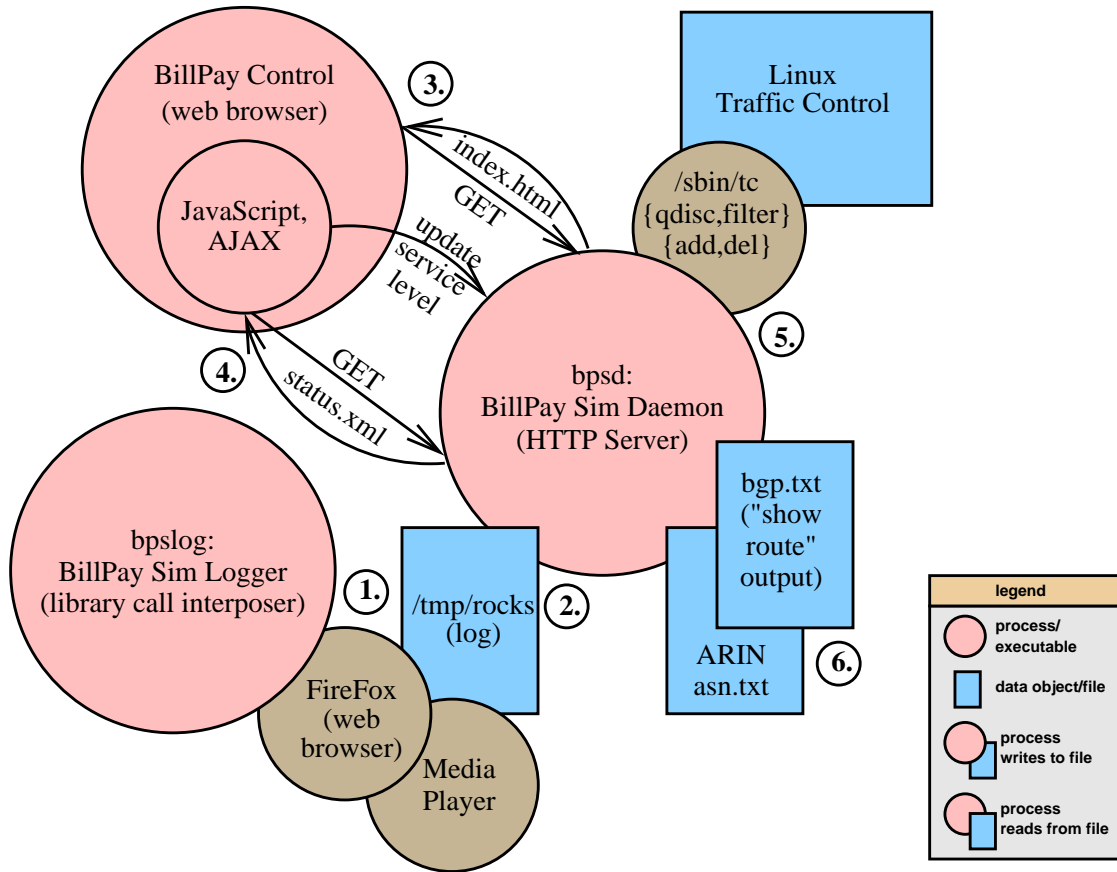


Figure 1: Bill-Pay Simulation System Overview

bpslog’s “-L” option instructs it to log those calls’ arguments and results with high-resolution timestamps to a regular file named “/tmp/rocks”¹. For example:

```
$ bpslog -L firefox &
```

3. The bpsd process runs as root and is launched prior to or shortly after one or more bpslogged processes are launched. For example:

```
# bpsd.pl -v -B
```

The bpsd follows the tail of the “/tmp/rocks” log file, parsing and processing the per-process internet socket related calls reported therein.

3. The user runs Bill-Pay Control in their web browser by fetching an “index.html” document from bpsd. This document contains the HTML and javascript code that forms the interactive user interface. For example:

```
$ firefox http://localhost:12345/index.html &
```

¹This file name is a legacy from the Rocks[7] software, on which bpslog is based.

4. The javascript portion of Bill-Pay Control fetches “status.xml” from bpsd. This virtual XML document continually summarizes all Internet activity performed by bpslogged applications. Bill-Pay Control repeatedly fetches this document at regular intervals, such as every 5 seconds, and updates the user’s display accordingly. The user can then alter the service level for traffic, as identified by traffic descriptor. For instance, they could promote the service level from “default” to “bronze”, which is then conveyed by an HTTP GET request to bpsd.
5. When bpsd receives a request to alter the service level, it uses the Linux “tc” command to alter the ingress traffic control filter rules that it installed at initialization time. The “tc” command modifies Linux’ traffic control table of ordered rules so that the traffic matching the given descriptor has the newly requested service level.
6. If BGP-based descriptors are selected, via bpsd’s “-B” option, bpsd initially loads two text files: the American Registry for Internet Numbers’ “asn.txt” file and the captured output from a “show route” command from the local Autonomous System’s BGP-speaking border router. It processes these files to form an IP prefix search tree of the entire Internet’s BGP table. This level-compressed tree enables bpsd to quickly find the traffic descriptor for any given IP address.

3.2 Bill-Pay Sim Logger

The simulation logger, bpslog, is a library call interposer. To generate a continuous log of chosen applications network activity it replaces calls to functions (including socket, connect, read, write, and close) used to create or manipulate internet sockets or to resolve domain names. Table 1 shows the entire list of functions that bpslog intercepts and logs if the call potentially involves IPv4 sockets.

Thus bpslog causes unmodified applications themselves to produce a continuous stream of library system call events that our system can use to track application network operations in real time.

3.3 Bill-Pay Sim Daemon

The simulation daemon (bpsd) is a perl script that has a number of responsibilities. First, it maintains traffic byte counters and rates and aggregates them by traffic descriptor based on reports (BPS::Report objects) delivered by the BPS::InetLog object that are prepared from the logged applications function calls by process ID and socket/file descriptor. When BGP-based descriptors are used, bpsd instead uses BPS::InetLogBGP - a derived class that further translates IP addresses into descriptors based on the AS path for the prefix that best matches the remote IP address. BPS::InetLogBGP uses Net::ParseRouteTable and Net::Patricia to perform fast lookups of IPv4 prefixes and corresponding AS paths in a current routing table for the whole Internet.

Second, bpsd is an HTTP server that processes requests from the user interface. The main requests are: (1) request for the “index.html” file that contains the HTML and javascript code that forms the user interface, (2) request for the virtual “status.xml” file containing the list of current descriptors with their traffic rates, total byte count, service level, and descriptor details, and (3) request to update a descriptor’s service level to a new value.

Table 1: Intercepted Library Calls

Functionality	Library Calls
socket creation	dup dup2 fcntl socket
resolver	inet_addr inet_aton inet_pton gethostbyname gethostbyname2 gethostbyname_r gethostbyname2_r getaddrinfo
socket operations	accept bind connect listen getsockname getpeername shutdown
socket input	read, _read readv recv recvfrom
socket output	write writev send sendto
socket termination	close exit

Lastly, when run with root privilege, bpsd performs traffic control according to requests for service level updates from the user interface.

3.4 Bill-Pay Traffic Control

Bill-Pay Sim (BPS) uses Linux's traffic control (tc) mechanisms to degrade traffic performance by default. BPS uses tc through two perl classes. The BPS::TcOps class provides wrapper functions around the actual Linux tc binary where as the BPS::TrafCtrl class provides the bookkeeping for the traffic control element of BPS. The BPS::TrafCtrl class keeps track of all of the rate modification rules for BPS and calls the necessary subroutines in BPS::TcOps when changes need to be made. When using BPS::TrafCtrl, BPS passes a list of IP addresses and a service level. The service level

is simply one of several traffic rates that is setup by BPS upon startup. Currently BPS sets up five basic service levels: *default*, *bronze*, *silver*, *gold*, and *platinum* at the traffic rates of 128Kbps, 512Kbps, 1Mbps, 2Mbps, and 100Mbps, respectively.

In our current BPS implementation, BPS::TrafCtrl is only performing rate limiting on the ingress path since most user applications use much more downstream than upstream traffic. We are currently rate limiting by IP address however, given the flexibility of the tc mechanisms built into Linux, our approach could be modified to take into account almost any field in the TCP/IP headers. The Linux tc mechanism is also somewhat limited when it comes to modifying ingress traffic since our only option is to drop packets exceeding a given rate whereas in egress mode, tc allows for much more complex queuing based techniques.

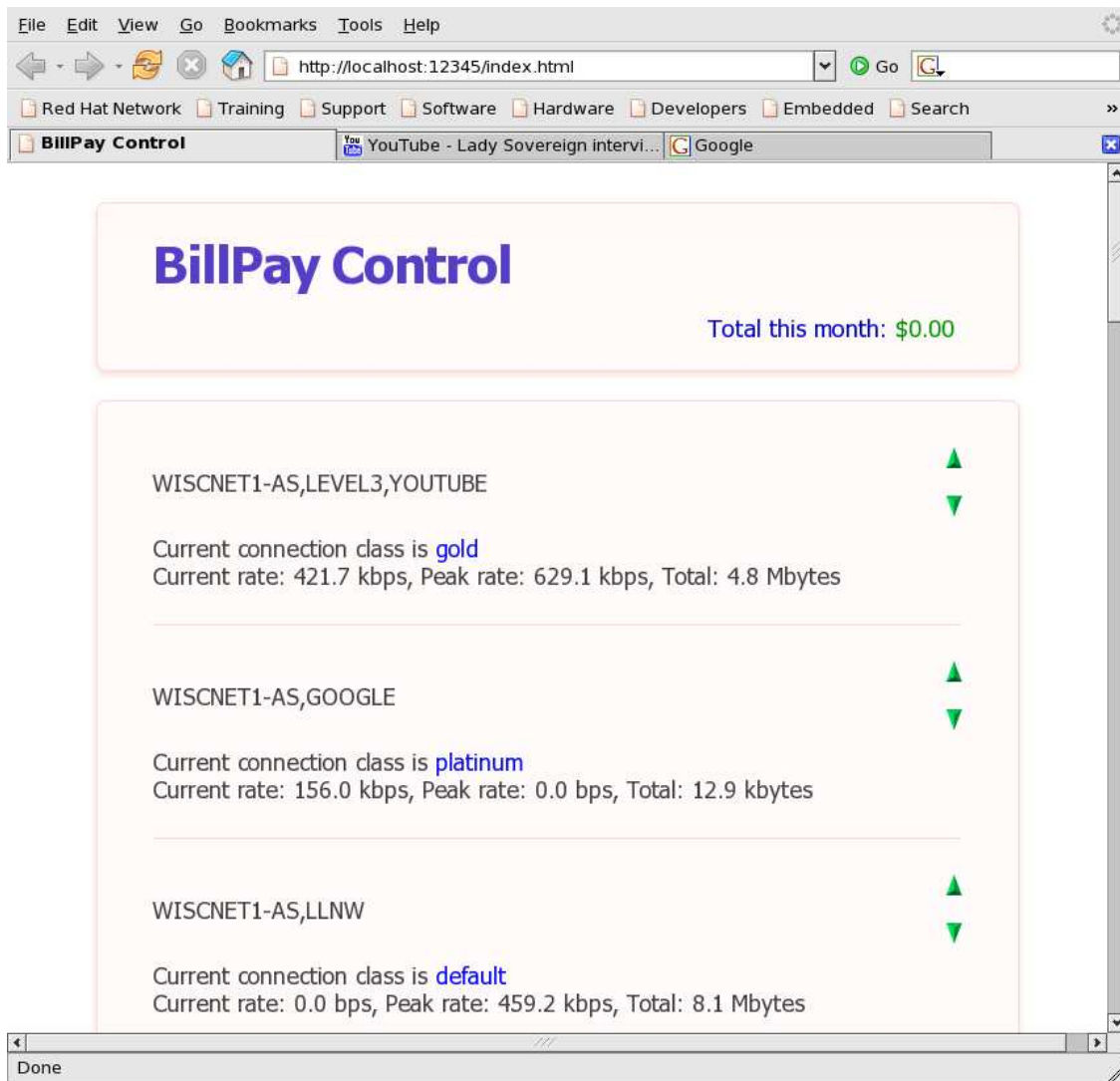


Figure 2: Bill-Pay Control Interface. In the other tabs, we are streaming a video from YouTube and have just completed a Google search.

3.5 Bill-Pay Control

The BillPay Control interface is implemented as a web application. We chose to implement the interface as a web application primarily for ease of development. The functions of the Control Interface are to display current network activity and to allow the user to adjust the performance of individual activities. A web interface performs both of these functions easily, and modern web techniques such as AJAX allow us to develop a web page that behaves interactively like a standalone Java or C++ program. See figure 2 for an example web browsing session as presented by Bill-Pay Control.

The user starts Bill-Pay Control by fetching its “index.html” from bpsd. This web page uses javascript to periodically request the current network activity status via an HTTP request in the background. The bpsd serializes the current status into an XML document. For each traffic descriptor, the XML document contains a record of the current rate, the peak transfer rate, the number of bytes transmitted, the service level, and an array of extra information the display can use to help explain the traffic descriptor to the user. In our current system, the traffic descriptor is usually the AS path used to reach the end hosts, and the extra information fields for the descriptor are the domain names that resolved to hosts matching this path. The meaning of the descriptors is opaque to the BillPay Control display. When the request completes, a javascript callback in the control page is invoked automatically by the web browser and the XML status file is parsed. Any new or updated descriptors are discovered, and the list of descriptors is sorted and converted into appropriate elements for display on the control page. The callback schedules the next status request for a few seconds in the future, and the cycle continues.

When the user clicks on an arrow to request a change to a traffic descriptors service level, the javascript sends a request to the bpsd via HTTP in the background and prevents the browser from leaving the page. The javascript updates the service level on the display for that page to “Updating.” The next periodic refresh of the status information will receive the updated service level, if the bpsd was able to make the change.

The traffic descriptors are displayed sorted by the current rate with higher rates first. If there is a tie for the current rate, the total number of bytes received is the tiebreaker. As the user interacts with sites, the descriptors associated with those sites will change their position in the list as the current transfer rates change. The hope is that the currently active descriptors will be the ones that the user is interested in monitoring and controlling, so they are at the top of the page.

We expect that future operating systems will make heavy use of small widgets to replace small system controls and applications. The Apple Dashboard widgets and Yahoo! Konfabulator widgets are implemented using HTML and javascript, and Microsoft XAML is similar in spirit. The eventual Bill-Pay control system might well be a similar widget.

4 Results & Observations

4.1 Traffic Descriptor Aggregation Effectiveness

Tables 2 shows some of the traffic descriptors resulting from one sample Firefox web browser session. (A full list of descriptors from this sample session is shown in table 3 and 4 in appendix A.) The browsing session lasted about five minutes and consisted of a visit to “http://www.cs.wisc.edu”, a visit to “http://www.cnn.com” and selecting an article, then a visit to “http://www.youtube.com”

Table 2: Traffic Descriptor Aggregation for CNN’s Web Site

AS Path	Domain Name	IP Address	Reverse Domain Name
WISNET1-AS, LEVEL3, AOL-ATDN, ASN-TBS-1	www.cnn.com	64.236.24.20	www5.cnn.com
		64.236.24.28	www7.cnn.com
		64.236.29.120	www.cnn.com
		64.236.16.20	www2.cnn.com
		64.236.16.52	www4.cnn.com
		64.236.16.84	www6.cnn.com
		64.236.16.116	www8.cnn.com
		64.236.24.12	www3.cnn.com
	i.cnn.net	64.236.24.136	i1.cnn.net
		64.236.24.137	i3.cnn.net
		64.236.24.138	i5.cnn.net
		64.236.24.139	i7.cnn.net
		64.236.16.136	i2.cnn.net
		64.236.16.137	i4.cnn.net
		64.236.16.138	i6.cnn.net
		64.236.16.139	i8.cnn.net
	ads.cnn.com	64.236.29.103	64.236.29.103
		64.236.22.63	64.236.22.63
		64.236.22.103	64.236.22.103
		64.236.29.63	64.236.29.63
	cl.cnn.com	64.236.22.12	cl4.cnn.com
		64.236.29.11	cl1.cnn.com
		64.236.29.12	cl2.cnn.com
		64.236.22.11	cl3.cnn.com
	cnn.dyn.cnn.com	64.236.29.20	cnn3.dyn.cnn.com
		64.236.29.21	cnn4.dyn.cnn.com
		64.236.22.20	cnn1.dyn.cnn.com
		64.236.22.21	cnn2.dyn.cnn.com

and selecting and playing one video. Within each AS path, the domain names and IP addresses are listed in the order contacted. IP addresses in the Reverse Domain Name column indicate a lookup failure. The name server is misconfigured or inoperative or that a corresponding in-addr.arpa entry did not exist for that IP address.

For this sample session, there was a total of 106 IP address and/or reverse domain names involved. By our domain name traffic descriptor technique those were summarized to 51 descriptors. By our AS path traffic descriptor technique those were, in turn, summarized into just 13 AS path descriptors. Thus, we propose that these two steps employed by our system are effective at producing a relatively concise set of manipulable traffic descriptors to the user. If the autonomous systems are the commercial entities that wish to receive payments, then this summary is likely the most concise that allows the user complete flexibility with their spending.

4.2 Identifying Critical Paths

Our Bill-Pay Control display uses AS paths as traffic descriptors and sorts those descriptors with recent activity to the top. This is an attempt to help the user to locate the descriptor that *might* be critical to the performance of their application. In many situations this is effective. However, when the user’s application is performing unacceptably but involves Internet traffic having many or varying descriptors, it may be difficult for that user to identify which descriptors’ service levels to increase.

For instance, it is common for web content or advertisements presented on one web page to be gathered from multiple autonomous systems. Advertising and content distribution networks are

typical causes of this phenomenon.

To direct performance improvement efforts, Barford and Crovella[2] developed a method of critical path analysis for bulk TCP transfers. Likewise, a system such as Bill-Pay would benefit from enhancements that would similarly enable the end user to better identify which components along which paths determine or limit their application's performance, so that they can better direct their tuning effort and payments.

5 Related Work

While the basic elements of BPS could be used for almost any micro-payment system, our simulator is closely related to the proposed Bill-Pay[4] system from which our simulator gets its name. Bill-Pay is a system in which an end-user places micro-payments into their outgoing packets and different entities along the traffic path may take a portion of the micro-payments in order to give better service to the traffic. Our BPS system provides one example of how an end-user may add micro-payments to their traffic along with visualizing who may be taking their micro payments along the way.

The INDEX project[1] is also loosely related to Bill-Pay Control. In the INDEX project users were allowed to choose from different rate ISDN lines at various price points along with different methods of paying for services such as a flat rate or per byte. While this project looked at how users were willing to pay for service on a bulk service basis, our system could be used in a similar project to examine how users might be willing to pay for service. The overall results from the INDEX project showed that user activity is sensitive to the method by which they are being charged. If a micro-payment system were to be developed, a similar investigation with a Bill-Pay Control interface would probably be warranted.

6 Conclusion

In conclusion, we presented Bill-Pay Control, an interactive user interface that displays application Internet traffic in near real time and allows the user to select the desired IP service quality. We also presented Bill-Pay Sim, the encompassing simulation system that emulates, to the degree necessary as a testing environment, the performance of a potential future Internet in which users can improve application performance by offering additional payments.

Through the use of our system, our observation is that domain names and Autonomous System paths can be used to fairly effectively and concisely present application traffic. In the absence of additional mapping infrastructure to help identify the candidate payment recipients, our traffic descriptor method is likely as simple as possible to achieve fine-grained control, but no simpler. Still, a Bill-Pay system would be greatly improved by providing better guidance to users so that they may direct their payments down the critical paths that are most likely to improve network and application performance.

Bill-Pay Control gives users the flexibility to experimentally influence Internet performance with feedback in near real time so that they can make informed financial decisions.

7 Acknowledgments

Cristian Estan provided the inspiration for a Bill-Pay simulator and referred us to related work. Barton Miller provided useful hints to understanding library call interposition behaviors. Victor Zandy wrote Rocks[7], on which bpslog is based. Mike Blodgett, Ken Hahn, and Bill Taylor helped us obtain machines for a testbed. Paul Beebe and Stefan Strandberg provided a network connection for our demo. The original HTML design came from Google Page Creator. The mouseover code is from Swazz.org and is made available under the terms of the GNU Public License.

References

- [1] J. Altmann, B. Rupp, and P. Varaiya. Internet demand under different pricing schemes. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 9–14, New York, NY, USA, 1999. ACM Press.
- [2] P. Barford and M. Crovella. Critical path analysis of tcp transactions. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 127–138, New York, NY, USA, 2000. ACM Press.
- [3] M. Carson and D. Santay. Nist net: a linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, 2003.
- [4] C. Estan, A. Akella, and S. Banerjee. Achieving good end-to-end service using bill-pay. Technical Report 1582, University of Wisconsin Computer Sciences Department, November 2006.
- [5] Linux advanced routing and traffic control. <http://lartc.org/>.
- [6] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.
- [7] V. C. Zandy and B. P. Miller. Reliable network connections. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 95–106, New York, NY, USA, 2002. ACM Press.

A Sample Traffic Descriptors

Table 3: Sample Traffic Descriptor Aggregation *continued in Table 4*

AS Path	Domain Name	IP Address	Reverse Domain Name
PRIVATE	pc-11-23-j 10.0.0.2	10.2.23.124 10.0.0.2	10.2.23.124 10.0.0.2
WISNET1-AS, GOOGLE	pagead2.googleadsyndication.com	72.14.223.104 72.14.223.147 72.14.223.99	ar-in-f104.google.com ar-in-f147.google.com ar-in-f99.google.com
WISNET1-AS	spe.atdmt.com img-cdn.mediaplex.com cdn5.tribalfusion.com	205.213.110.44 205.213.110.7 205.213.110.9 205.213.110.8	akamai-44.wisnet.net akamai-7.wisnet.net akamai-9.wisnet.net akamai-8.wisnet.net
WISNET1-AS, LEVEL3, YOUTUBE	www.youtube.com sjc-static15.sjc.youtube.com sjc-static9.sjc.youtube.com sjc-static2.sjc.youtube.com sjc-static8.sjc.youtube.com 208.65.153.150 sjl-static16.sjl.youtube.com 208.65.153.146 208.65.153.14 64.15.124.83 sjc-static11.sjc.youtube.com 64.15.124.96 208.65.153.11 64.15.124.97 64.15.124.90 sjc-static1.sjc.youtube.com sjl-static1.sjl.youtube.com sjc-static13.sjc.youtube.com sjl-static9.sjl.youtube.com sjc-static6.sjc.youtube.com 208.65.153.15 208.65.153.145 208.65.153.12 sjc-static12.sjc.youtube.com sjl-static5.sjl.youtube.com 64.15.124.87 64.15.124.85 sjl-v61.sjl.youtube.com	208.65.153.242 208.65.153.245 208.65.153.251 208.65.153.241 64.15.124.95 64.15.124.89 64.15.124.84 64.15.124.88 208.65.153.150 208.65.153.151 208.65.153.146 208.65.153.14 64.15.124.83 64.15.124.91 64.15.124.96 208.65.153.11 64.15.124.97 64.15.124.81 64.15.124.81 208.65.153.9 64.15.124.93 208.65.153.144 64.15.124.86 208.65.153.15 208.65.153.145 208.65.153.12 64.15.124.92 208.65.153.13 64.15.124.87 64.15.124.85 208.65.153.98	www.youtube.com www.youtube.com 208.65.153.251 www.youtube.com 64.15.124.95 64.15.124.89 64.15.124.84 64.15.124.88 208.65.153.150 208.65.153.151 208.65.153.146 sjl-static6.sjl.youtube.com 64.15.124.83 64.15.124.91 64.15.124.96 sjl-static3.sjl.youtube.com 64.15.124.97 64.15.124.81 64.15.124.81 sjl-static1.sjl.youtube.com 64.15.124.93 208.65.153.144 64.15.124.86 sjl-static7.sjl.youtube.com 208.65.153.145 sjl-static4.sjl.youtube.com 64.15.124.92 sjl-static5.sjl.youtube.com 64.15.124.87 64.15.124.85 sjl-v61.sjl.youtube.com
WISNET1-AS, LEVEL3, VALUECLICK	altfarm.mediaplex.com	216.34.207.71	ad.la.mediaplex.com
WISNET1-AS, SPRINTLINK, DOUBLECLICK	ad.doubleclick.net	209.62.176.52	eqnjmegaadvip1.doubleclick.net
...
13 total	51 total	106 total	106 total

