EN PASSANT: PREDICTING HTTP/1.1 TRAFFIC

Balachander Krishnamurthy and Jennifer Rexford
AT&T Labs–Research, Florham Park, NJ
{bala,jrex}@research.att.com

Abstract— **Significant research on the Web is performed using logs collected from proxies and servers. Virtually all of these logs are from sites running the 1.0 version (or earlier) of HTTP. After four years a draft-standard specification of HTTP/1.1 has emerged. However, many proxies and servers still use HTTP/1.0, and most so-called HTTP/1.1 servers do not employ all of the key features available in the protocol. It is not feasible to evaluate the impact of HTTP/1.1 traffic using HTTP/1.0 logs. Thus, there is a need to examine ways of postulating HTTP/1.1 traffic from HTTP/1.0 traffic. This paper presents the En Passant architecture and a tool based on the architecture to convert HTTP/1.0 logs and traffic into plausible HTTP/1.1 traffic based on various parameters representing changes between the two versions of the protocol. The tool takes both the high-level information in HTTP/1.0 server logs and low-level information found in packet traces as input and generates a possible HTTP/1.1 log. We present the set of parameters and the actual feature changes between the protocol that impact the traffic by using a live packet trace as example.**

I. Introduction

With the emergence of the World Wide Web as the dominant Internet application, evaluating the performance of the HTTP protocol and Web client, proxy, and server software has become very important. Since Web traffic patterns are not widely understood, and typically do not follow simple analytic distributions, most performance studies have been based on simulation or actual implementations, drawing on measured traffic or on synthetic workloads derived directly from such measurements. Many of these studies have drawn on collections of Web server logs [1–5]. Virtually all of these studies have focused on server or proxy logs based on HTTP/1.0 traffic.

HTTP/1.1 [6] introduces several new features that may substantially change the characteristics of Web traffic in the coming years. However, there is very little end-to-end HTTP/1.1 traffic in the Internet today, making it difficult to evaluate new Web policies and software systems under representative HTTP/1.1 workloads. Even when servers claim to implement HTTP/1.1, many of the high-traffic sites are not fully compliant with the protocol specification and some of the new features in HTTP/1.1 are disabled by configuration or only partially available to end users [7]. In addition, HTTP/1.1 has had very little penetration at proxies; for example, to date, Apache does not have proxy software that implements HTTP/1.1. As such, it may be several years before the majority of Web transactions use HTTP/1.1 across the entire path between the client and the server. Thus, there is a strong reliance on HTTP/1.0 logs and synthetic load generators to postulate improvements to HTTP/1.1, and to evaluate new proxy and server policies.

We believe that Web performance studies should use more realistic logs that account for the changes to the HTTP protocol. However, proxy and server logs typically do not contain enough information to allow projections of how the same logs would look under HTTP/1.1. We are addressing this problem by developing a tool, *En Passant*, for converting an HTTP/1.0 log into a representative HTTP/1.1 log. *En Passant* is being constructed based on three sources of information: packet-level traces, extended server logs, and the differences between HTTP/1.0 and HTTP/1.1. Towards this end, we are installing a packet monitor at AT&T's Easy World Wide Web (EW3) hosting platform [8], where we are already collecting server logs. The packet traces, combined with extended server logs, will provide detailed information that is not typically available in conventional server logs.

A packet trace, collected at the Web proxy or server site, can provide important information not available in server logs:

- Timing of packets on the wire
- Out-of-order and lost packets
- Interleaving of packets from different response messages
- TCP-level events such as SYN and FIN packets
- TCP and/or HTTP requests that are not processed by the server
- Amount of data transferred on aborted responses

The value of packet traces has been demonstrated in recent studies on the impact of TCP dynamics on the performance of Web proxies and servers [9, 10]. A complete collection of packet traces of both request and response traffic at a Web server would provide a unique opportunity to gauge how a change to HTTP/1.1 would affect the workload. In addition, the extended Web server logs can allow us to measure the components of delay at the server, and record additional information to aid in matching the server log entries with the appropriate parts of the packet trace.

Although the *En Passant* work focuses on developing a tool for converting HTTP/1.0 server logs into a semi-synthetic HTTP/1.1 logs, the tool could also be used to create synthetic workload generators. Research on Internet workload characterization has typically focused on creating generative models based on packet traces of various applications [11, 12]. A synthetic modeling approach has also been applied to develop workload generators for Web traffic [13, 14]. However, it may be difficult to project how these models should change under the new features in HTTP/1.1.

This position paper highlights the key changes between HTTP/1.0 and HTTP/1.1, and how a tool like *En Passant* could be used (Section 2). We then describe the information available in the combined packet traces/server logs and our measurement approach (Section 3). We discuss the issues surrounding the *En Passant* tool and present some preliminary results based on an existing HTTP packet trace [15] when applied to two areas of changes in HTTP/1.1—caching/coherency and range requests (Section 4). The paper concludes with a summary of our work (Section 5).

II. Changes between HTTP/1.0 and HTTP/1.1

In this section, we survey the main changes between HTTP/1.0 and HTTP/1.1, and identify ways that *En Passant* can track usage (and potential usage) of protocol features.

A. Protocol changes

In an attempt to clarify several parts of the HTTP/1.0 specification and to fix several flaws, several researchers have worked for the last four years to come up with a new version of the HTTP protocol. A comprehensive discussion of the differences between HTTP/1.0 and HTTP/1.1 is presented in [16], upon which this section is based.

Significant effort was expended in improving the semantic transparency of caching; several new headers (such as opaque entity tags to reduce dependency on clock synchronization when comparing timestamps, cache control directives to enable relative expiration times, etc.) were added. Proper deployment of caches and the overall semantic transparency in caching should reduce the number of both `GET If-Modified-Since` requests and consequent 304 (`Not-Modified`) responses with HTTP/1.1.

Reduction in bandwidth usage can be obtained by compressing a resource before transmission and requesting only interesting subsets of a resource (via the `Range` request). The `Expect/Continue` mechanism enables clients to verify that large requests can indeed be handled before actually sending them. Since HTTP is typically implemented on top of TCP, and since most web transactions are short, a significant portion of the time was spent in setting up and tearing down TCP connections. With persistent connections (HTTP connections lasting beyond a single request response sequence) and pipelining (sending additional requests without having to wait for response for previous ones), significant time savings are possible. Thus, embedded images in an HTML document can be downloaded quickly without new TCP connections.

In HTTP/1.0 it was not possible to know if a response was received in its entirety; 1.1 provides ways to detect errors in transmission via support for content length calculation and chunked encoding. The `Host` header was added to reduce the needless proliferation of IP addresses by permitting vanity URLs without requiring a separate hostname to be created with it. Security was strengthened by moving away from HTTP/1.0's model of transmitting passwords in cleartext; it is now encrypted and valid only for a single resource/method. Content negotiation permits a suitable representation for a requested resource if it is available in many forms (language, character sets, etc.).

B. En Passant Tool

The *En Passant* tool will take an HTTP/1.0 log, a set of high level parameters of interest to the server site (such as improving caching or miniminzing bandwidth usage), and an optional packet trace, as input to produce an approximation to the corresponding HTTP/1.1 log. *En Passant* groups the lower level entities whose combination impacts the parameter (e.g., appropriate request and response headers), extracts the necessary information from the server log and packet trace (if present), to produce the potential HTTP/1.1 stream. If the server site is interested in improving caching, *En Passant* would track cache-related request and response headers (such as `ETag` and `Cache-Control` directives). If the server site is interested in reducing bandwidth usage, *En Passant* would track aborted transfers to see if there is a potential for using `Range` requests; or see if large `PUT` requests result in a server error (e.g., 413 `Request Entity Too Large`) indicating the potential for use of the `Expect` request header. A requirement of safe and accurate receipt of data might trigger the use of chunked encoding.

III. Data Collection

We now describe our measurement architecture that combines extended server logging at a Web hosting complex with fine-grain packet monitoring.

A. Server Logs

Server logs range in duration (days to several months), number of records (few hundred to millions), and number of fields in each record. The six common fields found in most logs include:

- IP address or name of the client (remote host)
- Date/time of the request
- First line of the request: HTTP method and URL
- HTTP response status code (200, 304, ...)
- Number of bytes in the response

Logs may also have the remote log and user's name, the referer field—the URL from which the current URL was reached, user agent information, etc. We have developed a robust and efficient process for cleaning and anonymizing server logs, and producing a simplified intermediate format for post-processing [17].

The meaning of date/time field could be the time at which the server

- started processing the request
- started writing the response data into the send socket
- finished writing the response data into the send socket

These times are shown by a $\times$ symbol in the timeline in Figure 1. With all three time values, we could isolate the various components of server delay. For example, the first two timestamps would allow us to determine the server latency in processing client requests (e.g., due to disk I/O, or the generation of dynamic content), whereas the second two timestamps would capture the time required to write into the send socket. Even these additional timestamps would not indicate when the server finished transmitting the data from the socket buffer, and when the last client acknowledgment arrived, or if the client aborted the response after the server process finished writing the data into the send socket buffer. This information is available from the packet traces.

These three timestamps would also help in matching the server log entries with the packet data. The first timestamp would help in identifying the HTTP request packets, and the last two timestamps would help in identifying the HTTP response packets. The server could also log the client TCP port number for each request which should match with the packet trace. Once an accurate match is performed, the three server timestamps can be used to determine portions of delay that were introduced by server processing. With accurate clock synchronization, it may also be possible to determine the latency in receiving and parsing the request, by considering the time that the server starting processing the request (from the server log) and the time that the server machine received the request (from the packet trace).

At an HTTP/1.1 server, several new fields can be logged, including the various `Cache-Control` headers, the `Expect` header, requests handled on the same persistent connection and the number of such requests; these will aid in better understanding of the potential reduction of latency and bandwidth usage.

B. Packet Traces

Packet traces can complement the server logs by providing detailed information about the timing of packets in the network, as well as header fields that are not logged by the server. The packet monitor also sees packets dropped by the server during transient overload. By parsing the contents of an HTTP response message, a packet monitor could conceivably extract information about hypertext references and embedded images. Currently, we focus on

- fields in the HTTP request and response messages
- information about the actual request and response transfers
- timestamps of the actual request and response transfers

Fields common to the packet trace and the server log can be used to match a server log entry with the corresponding request/response pair from the packet trace. In the request header we record the request method (e.g., `GET, HEAD`), URL, and referer field. From the response header, the HTTP response code (e.g., 200, 304), content type, and content length are extracted. Optional header fields used for caching and authentication, such as the `If-Modified-Since` modifier in a `GET` request and the `Last-Modified` time in an 200 (`OK`) response are recorded.

The packet trace provides additional information:

1. The client port number of the TCP connection which can be used to measure the reuse of TCP

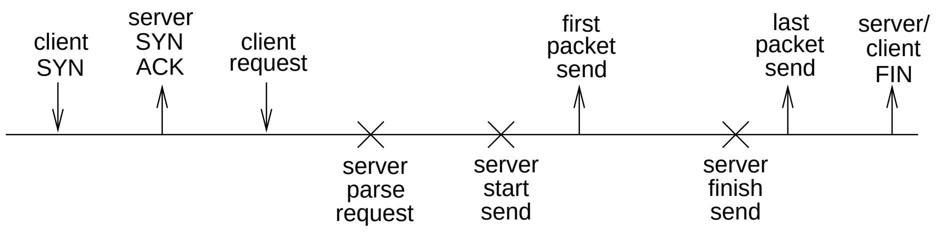

Fig. 1. Example request-response timeline from server log ($\times$) and packet trace ($\uparrow$ and $\downarrow$)

connections, even in HTTP/1.0 (via `Keep-Alive`).

2. TCP SYN (open) requests not handled by the server, and connections that are closed before the server generates a log entry.
3. The way in which the TCP connection was closed (a conventional FIN or a RST) and who closed the connection first (client/server). These can be used to study client aborts and transient server overload.

Finally, and perhaps most importantly, the packet trace has timing details about the steps involved in retrieving Web resources, as shown by the arrows in the timeline in Figure 1. The trace includes timestamps for the TCP open (client SYN and server SYN-ACK) and close (FIN, RST) packets, as well as the timestamp for the first packets of the HTTP request and response headers, as well as the first and last data packets. In a packet trace collected at the server, the timestamps of the client SYN and the server SYN-ACK can be used to estimate the server delay in establishing the TCP connection. In a packet trace collected at the client, the timestamps can be used to estimate the entire TCP-establishment delay, including the network round-trip time. The timestamps of the request and response headers, and the data packets, can be used to study the client throughput and delay.

C. Proposed Measurement Architecture

We propose a hybrid approach that complements extended server logging with passive packet monitoring by a separate machine on the same network segment, as shown in Figure 2. We are realizing this measurement architecture in the AT&T Easy World Wide Web (EW3) web-hosting platform [8], which hosts content for several thousand companies. Server logs from EW3 have been used in several recent studies (e.g.,[18]). To augment the server logging, we are installing a packet monitor on the FDDI rings that connect the platform to the rest of the Internet, and extending the server logging to capture more detailed information.

The packet monitor consists of a 500 MHz Alpha with

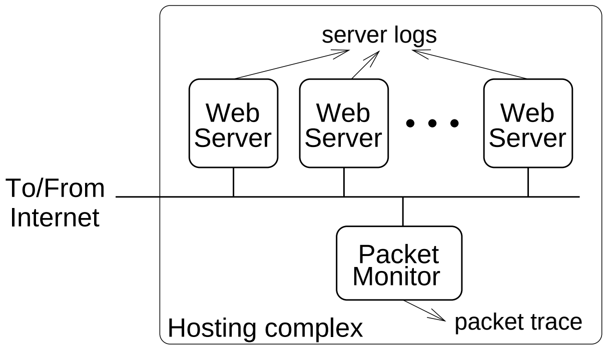

Fig. 2. Measurement architecture

a 10-gigabyte array of striped magnetic disks and a 140-gigabyte magnetic tape robot. We ensure that the monitor is passive by running a modified FDDI driver that can receive but not send packets, and by not assigning an IP address to the FDDI interface. We control the monitor by connecting to it over an AT&T-internal network that does not carry customer traffic. The traces are anonymized by encrypting the IP addresses as soon as packets come off the FDDI link, before writing any packet data to stable storage. To collect Web packet traces, the PacketScope software uses `tcpdump` to capture all port-80 (HTTP) packets and copy them directly to disk. A background process demultiplexes the packets into separate request and response packet flows, reorders any out-of-order packets, and presents summary statistics about each request and response [15,19]. This information includes the timestamps of key TCP and HTTP events, as well as the HTTP headers. A separate offline process matches the requests and responses. Experiences with the packet monitor at a WorldNet modem bank indicates that these instruments can capture more than 150 million packets per day with less than 0.3% packet loss.

IV. Statistics

To demonstrate the potential impact of the changes to HTTP, and the value of a tool like *En Passant*, we have collected statistics on an HTTP packet trace of modem customers in AT&T WorldNet. We focus on two important changes between HTTP/1.0 and HTTP/1.1 — caching/coherency and range requests.

A. HTTP Packet Trace

Our preliminary study of the capabilities of *En Passant* draws on an eleven-day packet trace collected from WorldNet in July 1998. The PacketScope monitored dial-up traffic on a FDDI ring connecting a bank of 450 modems to the WorldNet backbone. The trace includes 12.6 million request/response pairs involving 4.0 million unique URLs. Over 98.42% of the Web transfers involved `GET` requests, with 1.55% `POST` requests and 0.03% `HEAD` requests. Approximately 34.5% of the requests carried cookies. About one-third of the requests claimed to use the 1.1 version of the HTTP protocol, though this does not imply that the proxies and servers involved in these transfers had also used HTTP/1.1. Approximately 70.0% of the responses were 200 (`OK`) and 22.4% were 304 (`Not-Modified`).

B. Caching and Coherency

HTTP/1.0's relatively simple caching model is based on resource expiry times set by the server, where the server include an `Expires` header to indicate the latest time that a cached copy could be used. Before using a cached resource, the client or proxy can issue a conditional `GET` request with an `If-Modified-Since` modifier with the `Last-Modified` time from the last server response message for this resource. The server either responds with a 304 (`Not-Modified`) message if the resource has not changed, else a 200 (`OK`) response and the latest version of resource. The cache coherency model introduces a trade-off between the likelihood of returning a stale resource and the overheads of cache validations. When servers do not send an `Expires` header, the client or proxy is forced to rely on heuristics to decide whether or not to revalidate a cached resource.

HTTP/1.1 provides the `Cache-Control` general header as an explicit directive mechanism that permits clients or servers to override default actions. For example, `max-stale=3600` can be used to extend the time before validation is needed. Likewise, the `no-cache` response directive permits specific header fields from being reused. HTTP/1.1 also permits response headers to include a unique entity tag for each version of a resource, which can be used as a opaque cache validator for reliable validation.

The packet trace data provides an initial estimate of the potential savings from better cache coherency mechanisms. For example, 24.5% of the request messages included an `If-Modified-Since` modifier, and 91.3% of these requests resulted in a 304 (`Not-Modified`) response. As a result, 304 (`Not-Modified`) messages account for 22.4% of all response messages in our trace. Better cache coherency schemes could have avoided the server overhead, network bandwidth, and user-perceived latency introduced by the unnecessary validation traffic. However, the server response messages did not provide sufficient information. Just 56.4% of the responses included a Last-Modified time, and just 6.1% of responses included an Expires header. Future deployments of HTTP/1.1 server software should improve the information available to clients and proxies, allowing them to avoid unnecessary validation traffic. This will reduce the number of server logs entries with 304 (`Not-Modified`) responses. This, in turn, should reduce the number of requests, and affect the spacing of requests for individual clients.

C. Range Requests

HTTP/1.0 does not provide an effective way to request a server to send a portion of a resource. HTTP/1.1 range requests allow a client or proxy to request specific subsets of the bytes in a resource, instead of the full content. This is useful when only a small portion of the resource is of interest, or to retrieve the remainder of a resource after a partial transfer (e.g., an aborted response). Proxy servers could properly cache ranges to generate range responses later from caches. Origin servers and proxies can reduce the amount of bytes transmitted and thus reduce latency. Range requests will alter the size distribution in the traffic mix and lower user-perceived latency. For example, if a client aborts a response, the browser could cache the partial contents of the resource. If the client attempts to download the resource again, the browser can send a request for the remainder of the resource, along with an `If-Modified-Since` modifier to make sure that the resource has not changed in the meantime.

The packet traces suggest that aborts are an important performance consideration. A previous study considered the impact of client aborts on the effectiveness of proxy caching [19]. Aborts are somewhat difficult to detect, so we follow a conservative approach. In some cases, an abort is initiated by an RST (reset) on the client's TCP connection to the server. However, some platforms use a regular FIN (close) packet, making it difficult to distinguish between a close and an abort.

The content-length field enables us to distinguish most cases, since an aborted connection often has fewer bytes transferred than the content-length suggests. However, not all response headers have a content-length field (e.g., dynamically-generated responses).

In the current client trace, our conservative estimate suggests that 5.85% of responses are aborted. On average, the server sends 25.5% of the aborted resource before terminating transmission; the client may not receive all of these bytes, depending on how and when the receive socket is closed. These bytes contribute 8.3% of the total Web response traffic to the modem clients. Under HTTP/1.0, the aborted bytes would have to be transferred a second time if the client requests the resource again. We plan to study how often the client issues a subsequent request for the same resource. Also, we plan to repeat this analysis for the packet trace collected at the Web server platform. This should provide statistics for a more diverse set of clients with different bandwidth resources. The *En Passant* tool can use these results to estimate the bandwidth savings in satisfying repeat requests after an aborted response.

V. Conclusion

We have presented an architecture for gathering detailed data at Web server sites to transform a HTTP/1.0 traffic into a plausible traffic in the new version of the protocol. We discuss the changes that need to be done at the Web server, the network-specific and server-specific data elements that need to be gathered, and the set of parameters needed for the transformation. Based on an actual packet trace from a large ISP network, we examined how the traffic could be converted. We believe that a tool like En Passant will go a long way in helping various sites to convert their HTTP/1.0 logs to see how they might be affected by the changes in the protocol.

References

[1] M. F. Arlitt and C. L. Williamson, "Internet Web servers: Workload characterization and implications," *IEEE/ACM Trans. on Networking*, vol. 5, pp. 631–644, October 1997. `ftp://ftp.cs.usask.ca/pub/discus/paper.96-3.ps.Z`.

[2] S. Manley and M. Seltzer, "Web facts and fantasy," in *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 125–133, December 1997. `http://www.eecs.harvard.edu/~vino/web/sits.97.html`.

[3] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving end-to-end performance of the web using server volumes and proxy filters," in *Proc. ACM SIGCOMM*, September 1998. `http://www.research.att.com/~bala/papers/sigcomm98.ps.gz`.

[4] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the World Wide Web," in *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 1–12, December 1997. `http://www.research.att.com/~bala/papers/pcv-usits97.ps.gz`.

[5] B. Krishnamurthy and C. E. Wills, "Piggyback server invalidation for proxy cache coherency," in *Proc. World Wide Web Conference*, April 1998. `http://www.research.att.com/~bala/papers/psi-www7.ps.gz`.

[6] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," *Request for Comments 2616*, June 1999. `ftp://ftp.isi.edu/in-notes/rfc2616.txt`.

[7] B. Krishnamurthy and M. Arlitt, "PRO-COW: Protocol compliance on the Web," Tech. Rep. 990803-05-TM, AT&T Labs – Research, August 1999. `http://www.research.att.com/~bala/papers/procow-1.ps.gz` .

[8] AT&T Easy World Wide Web `http://www.ipservices.att.com/wss/hosting`.

[9] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz, "TCP behavior of a busy Internet server: Analysis and improvements," in *Proc. IEEE INFOCOM*, April 1998. `http://http.cs.berkeley.edu/~padmanab/index.html`.

[10] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, "Web proxy caching: The devil is in the details," in *Proc. Workshop on Internet Server Performance*, June 1998. `http://www.cs.wisc.edu/~cao/WISP98.html`.

[11] R. Caceres, P. Danzig, S. Jamin, and D. Mitzel, "Characteristics of wide-area TCP/IP conversations," in *Proc. ACM SIGCOMM*, pp. 101–112, September 1991. `http://www.research.att.com/~ramon/papers/sigcomm91.ps.gz`.

[12] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A parameterizable methodology for internet traffic flow profiling," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1481–1494, October 1995. `http://www.nlanr.net/Flowsresearch/Flowspaper/flows.html`.

[13] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proc. ACM SIGMETRICS*, June 1998. `http://cs-www.bu.edu/faculty/crovella/paper-archive/sigm98-surge.ps`.

[14] B. Mah, "An empirical model of HTTP network traffic," in *Proc. IEEE INFOCOM*, April 1997. `http://www.ca.sandia.gov/~bmah/Papers/Http-Infocom.ps`.

[15] A. Feldmann, "Continuous online extraction of HTTP traces from packet traces," in *Proc. World Wide Web Consortium Web Characterization Workshop*, November 1998. `http://www.research.att.com/~anja/feldmann/papers/w3c98_httptrace.ps.gz`.

[16] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol, "Key differences between HTTP/1.0 and HTTP/1.1," in *Proc. Eighth International World Wide Web Conference*, May 1999. `http://www.research.att.com/~bala/papers/h0vh1.ps.gz`.

[17] B. Krishnamurthy and J. Rexford, "Software issues in characterizing web server logs," in *Proc. World Wide Web Consortium Web Characterization Workshop*, November 1998. `http://www.research.att.com/~bala/papers/ew3c.html`.

[18] E. Cohen, B. Krishnamurthy, and J. Rexford, "Efficient algorithms for predicting requests to web servers," in *Proc. IEEE INFOCOM*, March 1999. `http://www.research.att.com/~bala/papers/inf99-submit.ps.gz`.

[19] A. Feldmann, R. Caceres, F. Douglis, and M. Rabinovich, "Performance of web proxy caching in heterogeneous bandwidth environments," in *Proc. IEEE INFOCOM*, March 1999. `http://www.research.att.com/~anja/feldmann/papers/infocom99_proxim.ps.gz`.