Web Timeouts and Their Implications*

Zakaria Al-Qudah[1], Michael Rabinovich[1], and Mark Allman[2]

[1] Case Western Reserve University, Cleveland, Ohio 44106
[2]International Computer Science Institute, Berkeley, CA 94704

Abstract. Timeouts play a fundamental role in network protocols, controlling numerous aspects of host behavior at different layers of the protocol stack. Previous work has documented a class of Denial of Service (DoS) attacks that leverage timeouts to force a host to preserve state with a bare minimum level of interactivity with the attacker. This paper considers the vulnerability of operational Web servers to such attacks by comparing timeouts implemented in servers with the normal Web activity that informs our understanding as to the necessary length of timeouts. We then use these two results—which generally show that the timeouts in wide use are long relative to normal Web transactions—to devise a framework to augment static timeouts with both measurements of the system and particular policy decisions in times of high load.

1 Introduction

One of the historic tenets of networking that has served the Internet well over the past 30 years is that components of the system should be both conservative and liberal at the same time. That is, actions should only be taken as they are strictly necessary—therefore acting *conservatively.* Furthermore, wide tolerance for a range of behavior from other components in the system is also desirable—or acting *liberally.* Another fundamental notion within the Internet is that the only thing we can absolutely count on is the passage of time. This notion naturally led to timeouts as a fundamental fallback mechanism to ensure robust operation. Adhering to the above stated principles tends to make timeouts long such that we can tolerate a range of behavior and the timer only expires when a gross anomaly occurs as opposed to when some task simply happens slower than expected.

Unfortunately, the above narrative becomes muddled in the presence of malicious actors as it creates an opening for the so-called *claim-and-hold* denial of service attacks [13], where an attacker can claim server resources without using them thus preventing the server from utilizing these resources on legitimate activities.

In this paper, we consider the issue of timeouts in the modern Internet within the context of the Web. We conduct an empirical investigation that seeks to understand (*i*) how timeouts are currently set on Web servers and (*ii*) how those settings relate to normal user-driven Web traffic. Our key finding is that

* This work is supported in part by NSF grants CNS-0615190 and CNS-0433702.

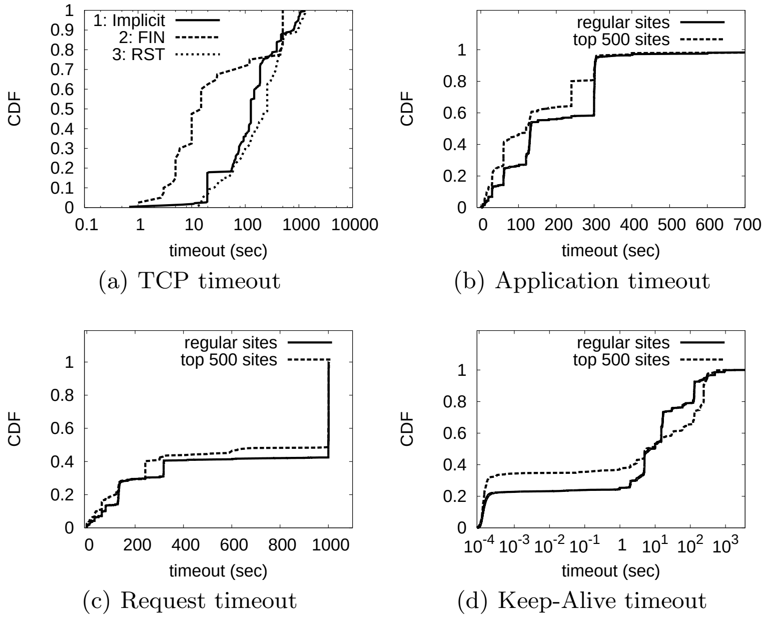
timeout settings are extremely conservative relative to actual traffic patterns and expose Web servers to easy DoS attacks. While this suggests that servers could take a more aggressive posture with respect to timeouts, doing so would run counter to the general tenet mentioned earlier (i.e., would result into dropping legitimate anomalies even at times when enough resources are available to serve them). Instead, we propose an adaptive approach whereby the timeouts are only reduced at times of measured stress. In fact, we observed a small number of Web sites that exhibit a behavior which indicates that they might be already varying their timeouts dynamically. We believe other sites, large or small, would benefit from similar reactions in the face of claim-and-hold attacks. Unfortunately, such timeout adaption is not available out-of-the-box in popular Web servers. As part of this project we have implemented and make available a simplified adaptive mechanism as a modification of the Linux kernel and Apache Web server [1].

2 Related Work

Qie, et.al. [13] studied, verified, and classified DoS attacks into *busy attacks* and *claim-and-hold attacks*. Web server administrators have reported encountering claim-and-hold attacks [7, 6] and server software vendors seem cognizant of these attacks and typically recommend tuning Web server timeouts [4, 8]. However, as we show in this paper, a large number of Web sites use default timeout values. Barford et. al. observed the negative effect of excessive persistent connections on busy Web servers and recommended an *early close* policy whereby Web clients close persistent connections after downloading a page and all its embedded objects [5]. Rabinovich et. al. suggested adaptive management of persistent connections at Web servers, where a server closes idle connections once it runs out of the connection slots [14]. We argue for a similar but more general approach in Section 4. Park, et.al. also point out the danger of inactive or slow Web clients and propose an independent component to filter and condition external connections for the Web server [12]. In contrast, we suggest an adaptive timeout strategy on the Web server itself.

3 Timeout Measurements

In this section, we assess timeout periods in operational Web servers and compare them with the time needed by Web clients to perform the corresponding activities that these timeout periods control. To this end, we probe two groups of Web servers: (*i*) Alexa's top 500 sites [3] denoted as "high volume" sites and (*ii*) 15,445 sites collected using the Link Harvester tool [15] denoted as "regular" sites. The list of these sites is available from [1]. In the high volume group, 53% of sites reported some version of Apache Web server in the "Server:" response header, 12% Microsoft-IIS, 10% GWS (Google), and the rest reported some other server or nothing at all. Among the regular sites, 68% were Apache, 19% Microsoft-IIS, and the rest other/unknown. As described below, we actively probe these sites for various timeouts. Inevitably for each experiment a small



(a) TCP timeout

(b) Application timeout

(c) Request timeout

(d) Keep-Alive timeout

Fig. 1. Distribution of Web server timeouts.

number of sites are unavailable and so the precise number of sites used varies and is reported for each experiment.

To assess the time consumed by Web clients to perform various normal activities, we analyze a week long packet trace of Web traffic collected at the border of International Computer Science Institute (ICSI) captured between August 11–18 2009. The trace contains nearly 1.6M HTTP connections involving nearly 14K servers and 25K clients. We note that Web clients in our trace are generally well-connected. While it would be desirable to verify our results directly in a qualitatively different environment, we do not expect dial-up clients to affect our findings (as discussed later).

TCP Timeout: The *TCP timeout* represents the length of time a TCP implementation will attempt to retransmit data before giving up on an unresponsive host. We assess this timeout by opening a TCP connection to a given server, sending an HTTP request and disappearing—i.e., sending no further data, ACK, FIN or RST packets. Some sites respond with an HTTP redirection and a FIN (either with the data or closely thereafter). We exclude these sites from further analysis because the timeout we experience in this case is the FIN_WAIT state timeout, not the retransmission timeout. This reduces the number of sites—437 high volume and 13,142 regular sites—involved in this experiment compared to other experiments.

We monitor the server's retransmissions and find three distinct ways for connections to end: (*i*) implicitly with the retransmissions eventually ceasing, (*ii*) explicitly with the server sending a FIN or (*iii*) explicitly with a server sending a RST. We measure the TCP timeout as the interval between the arrival of the first data transmission and the arrival of either the last retransmission or a packet with a FIN or RST bit set (note, this FIN case is distinct from the redirection case discussed above). Figure 1(a) shows the distribution of timeouts measured for each termination method for the high volume sites (the regular sites are omitted due to space constraints but show the same general behavior). In case (*i*)—encompassing 61% of the servers in both sets—the observed timeout is over 100 seconds for two-thirds of the servers. Note that in this case there is no wire event indicating the server has dropped a connection, and we expect that the server waits for some time for an ACK after the last retransmission. Therefore, the measurements represent a lower bound. In case (*ii*)—encompassing 9% of the servers in each set—we believe the FIN transmission is generally triggered by the overall application giving up on the connection rather than TCP terminating the connection itself. Therefore, at best these measurements also represent a lower bound on the length of the TCP timeout, which is illustrated by the order of magnitude difference between cases (*i*) and (*ii*) in Figure 1(a). In case (*iii*)—encompassing 30% of the servers in each set—we observe that servers that send a RST show the longest timeouts by a small margin over servers that silently terminate. We believe this is likely the best representation of the TCP timeout as it encompasses both the entire retransmission process and the additional waiting time that goes unseen in case (*i*).

We contrast the above determined lower bounds with data from our previous work [2]. In that work, we set up two servers: one configured with a normal TCP timeout (default Linux timeout of 15 retransmissions, or ≈13–30 minutes) and one with a quick TCP timeout (3 retransmissions or roughly 600 msec). We then used 59 Keynote [9] clients around the world to download a 2 MB file from each server every 15 minutes for over a week. Reduced retransmissions increased dropped connections due to timeouts by 0.16%, suggesting that continuing to retransmit for long periods of time is often futile.

In summary, while Figure 1(a) shows that—excluding cases where we do not believe TCP terminated the connection—80% of the surveyed servers have TCP timeouts exceeding 57 seconds, and nearly two-thirds of the servers have TCP timeouts exceeding 100 seconds, our preliminary data indicates that most Web interactions would succeed with a sub-second TCP timeout.

Application Timeout: The *application timeout* is the time a server allows between completing the TCP connection establishment and the arrival of the first byte of an HTTP request. To measure the application timeout in operational Web sites, we open a TCP connection to a server without sending an HTTP request using *nc6* [11]. We then measure the time from the completion of the TCP connection establishment until the connection is closed by the server (giving up after 20min). We use 492 high volume sites and 14,985 regular sites in this experiment. We find that just under 36% of sites in both groups do not end the connection after 20min. Potential reasons for this behavior include sites using the TCP_DEFER_ACCEPT Linux TCP option [16] (or like option on other systems). With this option, TCP does not promote a connection from the SYN_RCVD state to ESTABLISHED state—and thus hand it over to the application—until data arrives on the connection. Therefore, the notion of application timeout is not applicable for these sites. (Note however that these sites can still accumulate pending connections in the SYN_RCVD state, which may present a different attack vector.) Another explanation is these sites have an application timeout which is longer than 20min.

Figure 1(b) shows the distribution of measured application timeouts for the remaining ≈64% of sites in the two groups. The figure shows significant modes in both groups around 120s and 300s—the well-known defaults for IIS and Apache respectively. We also observe that high volume sites generally have shorter application timeouts than regular sites. Presumably these sites have determined that shorter timeouts are better for resource management without disrupting users. The figure also has a mode around 240s for the high volume sites which is mostly due to Google's sites (e.g., google.com, google.fr, google.co.uk, gmail.com, etc.). Similarly, we find that the high volume sites responsible for the mode around 30s to be mostly Akamai-accelerated sites. Finally, we find a mode around 60s which we cannot readily explain. Overall, around 54% of high-volume sites and 74% of regular sites have application timeouts of over 100s.

We now turn to our packet trace and measure the time between the last ACK in TCP's three-way handshake and the first packet with the client's HTTP request. We find that 99% of the requests were sent within one second of completing the TCP connection establishment. However, the longest time a client in our trace took to start sending the request after completing the TCP connection establishment is 586 seconds.

Request Timeout: The *request timeout* is the time a Web server allots to a request to completely arrive at the server after the first byte of the request has arrived. To measure the request timeout we drip a 1000 byte request over the network at a rate of one byte/sec and note when (or if) the server terminates the connection. Transmitting the request at a byte/sec factors out a possible effect of another timeout commonly applied to *poll()/select()* calls—which is usually greater than one second. This experiment involves 492 high-volume and 15,033 regular sites.

Figure 1(c) shows the distribution of the measured request timeouts. The plot indicates that 58% of the regular sites and 51% of the high volume sites keep the connection open for the entire 1,000 seconds it took our client to send its request, suggesting that the server does not impose a request timeout. Among the sites that do set a smaller request timeout, high volume sites have generally shorter timeouts than regular sites. Overall, 93% of the high volume sites and 96% of the regular sites have a request timeout period of over 30 seconds.

To assess how long Web clients normally take to transmit their requests, we measure the time between the first and last packets of HTTP requests in our trace. When the entire request fits in one packet, we report the time as zero. We concentrate on HTTP GET requests in this experiment as they are typically small. HTTP POST requests on the other hand could be arbitrarily large and

Group	Impose Limit	IIS with limit	IIS without limit
Top 500	24.1%	32.7%	5.1%
Regular	23.5%	59.0%	7.2%

Fig. 2. Response timeout

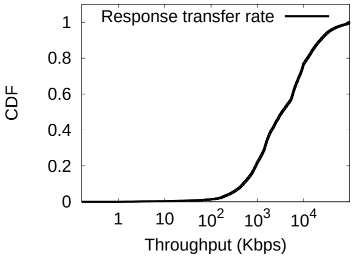


Fig. 3. Response rate.

take longer to send. This suggests that these two request types should be handled with different timeouts. We find that 85% of the requests fit into one packet. Further, 99.9% of the requests are completed within 1 second. Still, the longest time taken by a client in the trace is 592 seconds.

Response Timeout: The *response timeout* is the amount of time the server allocates to delivering an HTTP response. This timeout guards against a client that is alive (i.e., responds with TCP ACKs) but consumes data at a slow rate, by either acknowledging few bytes at a time or advertising a small (or at the extreme, zero) window. Since the client is responding the connection can only be closed by the application and not by TCP.

We are aware of only one major Web server that enforces a response timeout—alternatively presented as a minimum transfer rate—which is Microsoft's IIS. The default minimum transfer rate in IIS is 240 bytes/sec. Even though IIS notationally imposes a minimum *rate-based* limit, internally this is converted to a *time-based* limit. Specifically, IIS divides the response size by the minimum transfer rate with the result used to arm a timer. If the timer fires and the client has not fully consumed the response, IIS will close the connection [8]. This mechanism is efficient in that progress is checked only once. However, an attacker can leverage this mechanism by finding a large object and retrieving it at a low rate—which IIS will only detect after a long time.

To measure the response timeout, we open a connection to a Web server, send a request for the home page, and consume the response at a low rate. Given the IIS's default rate limit of 240 bytes/sec, in our experiments we consume the response at a lower rate of 100 bytes/sec. A site that delivers the entire response at this rate is assumed to not impose a limit, otherwise a limit is in place. This experiment involves 494 high volume sites and 15,034 regular sites. The table in Figure 2 shows our results. We find that less than 25% of sites—regardless of group—impose a limit on the transfer rate. Furthermore, 59% of the regular sites that impose limits identify themselves as IIS, as expected. However, only 33% of the high-volume sites that impose response time limits identify themselves as IIS servers. There could be a myriad reasons that can explain the remaining sites, including IIS servers obscuring their identities, servers behind transparent TCP proxies that keep their own timers, custom built servers, intrusion prevention systems impacting communication, etc. Interestingly, as shown in the last column of the table, there is a small percentage of sites that identify themselves as IIS servers and yet do not impose any response timeout. This could be caused by site administrators disabling the response timeout or transparent TCP proxies that obscure the actual Web server behavior.

We now consider the time needed by normal Web clients to consume responses. This time is determined mainly by the round trip time for small responses and by the available end-to-end bandwidth for large responses. Therefore, while a low limit on the transfer rate such as IIS's 240 byte/sec might be appropriate for small responses (although whether one could tighten this limit at times of stress is an interesting question for future work), we aim at assessing whether such a low limit is appropriate for large responses, especially that attacks against this timeout are particularly dangerous for large responses. To assess that, we consider responses in the ICSI trace with size of at least 50 KB. We approximate the end-to-end transfer rate as the response size divided by the time between the first and last packet of the response. Figure 3 presents the distribution of response transfer rates. The figure shows that nearly 99% of the responses (whether the response was originated from an ISCI server or an external server) were transferred at over 10 Kbps (that is 1,250 bytes/second compared to the default of 240 bytes/second of IIS).

HTTP Keep-Alive: We next turn to persistent HTTP, which attempts to make Web transfers efficient by keeping TCP connections open for more than one HTTP request. The *HTTP keep-alive timeout* is defined as the time the server will keep an idle connection open after successfully satisfying all requests. We start by issuing requests for the home pages of the Web sites using *nc6*. We then measure the time between receiving the last packet of the response and receiving a FIN or RST from the server. This experiment involves 490 high volume and 14,928 regular sites Figure 1(d) shows the distribution of these times. The problem of finding a cut-off point before which we assume servers do not maintain persistent connections is relatively easy in this figure. Indeed, selecting the cut-off point at 100ms or at 1 second produces similar results. Roughly, 65% of the high volume sites and 76% of the regular sites maintain persistent connections. These numbers indicate that the overall support of persistent connections has not changed appreciably since Fall of 2000 [10]. Surprisingly, regular sites seem to have shorter keep-alive timeouts than high volume sites. For instance, nearly 61% of the high volume sites that use persistent connections use a timeout over 30s while it is roughly 32% for the regular sites. We speculate that this is due to the higher incidence of Apache with default configuration of 15s keep-alive timeout among regular sites than it is among high volume sites.

Timeout Adaption: To get a preliminary intuition as to whether Web sites currently vary their timeouts over time, we performed periodic probing of the request timeout for the high volume sites. Specifically, we probed each site every 12 minutes for a week. We define a site as having an adaptive timeout if at least m% of the measurements to the server are at least m% different from the mean timeout to the given site (i.e., m is an experimental parameter). This procedure

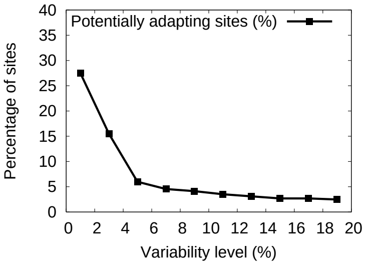


Fig. 4. Variability of request timeouts.

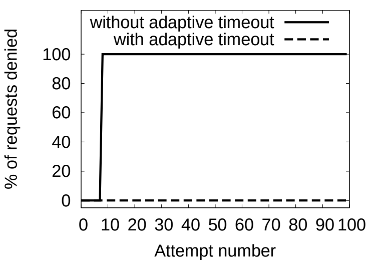


Fig. 5. Performance of adaptive timeouts.

is clearly not conclusive given that we may simply not have observed adaption for a particular site because there was no reason for the site to adapt during our measurements. Further, a timeout change could be caused by reasons other than adaptability such as different requests arriving at different servers with various timeout configurations or a server crash during a connection lifetime. The percentage of sites found to be using an adaptive timeout as a function of m is shown in Figure 4. We find that roughly 3% of the sites tested exhibit behavior suggestive of timer adaption, as shown by the range of m values for which this finding holds.

Summary: our measurements indicate that normal web clients perform their activities quickly as compared to the time allowed by Web servers. [1] Long timeouts leave a server vulnerable to claim-and-hold attacks. These attacks have been reported in practice [7, 6], and we will demonstrate a simple attack utilizing these timeouts in the next section. Short of complex external intrusion detection mechanisms, a naive way to counter these attacks would be to increase the number of allowable concurrent connection slots at the server. But this may cause performance degradation in case the slots are consumed by legitimate connections, since the number of concurrent connections is driven by the server capacity. Furthermore, although our measurements show that current long timeouts are generally unneeded by normal Web clients, slashing them blindly would run counter to the general networking tenet of allowing liberal client behaviors. Therefore, we suggest slashing these timeouts only at the time of stress.[2]

[1] While clients in our trace are generally well-connected, the characteristics of dial-up connections should not affect this finding. Indeed, dial-up connections offer a last-mile bandwidth of 30-40 Kbps—well within the 99^{th} percentile we observe in our trace and also well above the 240 bytes/sec IIS requires. Furthermore, the few hundreds of milliseconds these connections add still leave the time needed by these connections to perform activities much shorter than allowed by Web servers.

[2] One can imagine applications, as possible with AJAX, where HTTP connections could have long idle periods. Our trace accounts for all HTTP interactions including AJAX, and as discussed, did not encounter such connections in large numbers. We note that the content provider controls both ends of the connection in these applications. Therefore, these connections could either be treated differently by the server or the applications can be written to handle possible interruptions gracefully.

While our measurements suggest that a small fraction of sites might already be varying their timeouts, popular Web servers such as Apache and IIS do not offer such mechanisms—which would limit the spread use of these mechanisms.

4 Adaptive Timeouts

We now present our implementation of an adaptive timeout mechanism and demonstrate its usefulness. Our implementation involves changes to the Linux TCP stack and Apache web server (version 2.2.11). The kernel extension allows an application to specify a target response transfer rate and to toggle the kernel between a conservative (current behavior) and aggressive (close any connection below the target transfer rate) modes. The kernel monitors the transfer rate of connections only during periods of non-empty TCP send queue to avoid penalizing a client for the time the server has no data to send. Our modified Apache sets the target transfer rate parameter (500 bytes/second in our experiments) and monitors the connection slots. Once allocated slots reach a certain level (90% of all slots in our experiments), it (a) reduces its application timeout from its current default of 300s to 3s and (b) toggles the kernel into the aggressive mode. While a complete implementation of our framework would consider all timeouts, our current implementation covers application timeout, TCP timeout, and response timeout.

To demonstrate how such a simple mechanism can protect sites from claim-and-hold attacks, we set up a Web site with Linux OS and Apache Web server, both using out-of-the box configurations except with Apache configured to allow a higher number of concurrent connections (256 vs. default 150). We then set up a machine that launches an attack targeting the response timeout. In particular, it attempts to keep 300 concurrent connections by requesting a 100 KB file and consuming it at a rate of 200–300 bytes/second on each of these connections. Another machine simulates legitimate traffic by probing the server once every 10 seconds by opening 100 connections to the server with a 5 second timeout period (i.e., a request fails if not satisfied within 5 seconds). This process repeats 100 times. The solid line on Figure 5 shows the results. The attack starts around probe number five. After a short delay (due to Apache's gradual forking of new processes) the attacking host is able to hold all the connection slots and thus completely deny the service to legitimate connections. Further, the attacker accomplishes this at the cost of consuming less than 1Mbps (300 connections with at most 300 bytes/s each) of its own bandwidth—available to a single average residential DSL user let alone a botnet. The dashed line in Figure 5 shows the results of repeating the attack on our modified platform. As seen, our simple mechanism allows the server to cope with the attack load without impinging on legitimate connections by quickly terminating attack connections which leaves open slots for legitimate traffic. Our intent in this experiment is to show that a simple system can perform well. We consider a full study of a range of decision heuristics out of scope for this paper. Further, such decisions can be a policy matter and therefore cannot be entirely evaluated on purely technical grounds.

5 Conclusions

In this paper we study Internet timeouts from two perspectives. We first probe the timeout settings in two sets of operational Web sites (high volume and regular sites). We then study the characteristics of normal Web activity by analyzing passively captured Web traffic. The major finding from these two measurements is that there is a significant mismatch between the time normal Web transactions take and that which Web servers allow for these transactions. While this reflects a conservativeness on the Web server's part it also opens a window of vulnerability to claim-and-hold DoS attacks whereby an attacker claims a large fraction of connection slots from the server and prevents their usage for legitimate clients.

Rather than reducing servers' timeouts to match normal Web activity—a solution that could reduce the tolerance of the server to legitimate activity—we suggest a dynamic mechanism that is based on continuous measurements of both connection progress and resource contention on the server. A decision to reduce the timeouts and drop connections accomplishing little or no useful work is only taken when the server becomes resource constrained. We demonstrate how this simple mechanism can protect Web servers. Our mechanism is implemented in a popular open source Web server and is available for download [1].

References

1. Project Downloads. http://vorlon.case.edu/~zma/timeout_downloads/.
2. Z. Al-Qudah, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. der Merwe. Anycast-aware transport for content delivery networks. In *18th International World Wide Web Conference*, pages 301–301, April 2009.
3. Alexa The Web Information Company. http://www.alexa.com/.
4. Apache HTTP server - Security tips. http://httpd.apache.org/docs/trunk/misc/security_tips.html.
5. P. Barford and M. Crovella. A performance evaluation of hyper text transfer protocols. In *SIGMETRICS*, pages 188–197, 1999.
6. objectmix.com/apache/672969-re-need-help-fighting-dos-attack-apache.html.
7. http://www.webhostingtalk.com/showthread.php?t=645132.
8. Microsoft TechNet Library. http://technet.microsoft.com/en-us/library/cc775498.aspx.
9. Keynote. http://www.keynote.com/.
10. B. Krishnamurthy and M. Arlitt. PRO-COW: Protocol compliance on the web: A longitudinal study. In *USENIX Symp. on Internet Technologies and Sys.*, 2001.
11. nc6 - network swiss army knife. http://linux.die.net/man/1/nc6.
12. K. Park and V. S. Pai. Connection conditioning: architecture-independent support for simple, robust servers. In *USENIX NSDI*, 2006.
13. X. Qie, R. Pang, and L. Peterson. Defensive programming: using an annotation toolkit to build DoS-resistant software. *SIGOPS Oper. Syst. Rev.*, 36(SI), 2002.
14. M. Rabinovich and H. Wang. DHTTP: An efficient and cache-friendly transfer protocol for web traffic. In *INFOCOM*, pages 1597–1606, 2001.
15. SEOBOOK.com. http://tools.seobook.com/link-harvester/.
16. TCP protocol - Linux man page. http://linux.die.net/man/7/tcp.