PRO-COW: Protocol Compliance on the Web—A Longitudinal Study

Balachander Krishnamurthy
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932
bala@research.att.com

Martin Arlitt
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
arlitt@hpl.hp.com

Abstract

With the recent (draft) standardization of the Hypertext Transfer Protocol – HTTP/1.1 protocol on the Web, it is natural to ask what percentage of popular Web sites speak HTTP/1.1 and how *compliant* are these so-called HTTP/1.1 servers. We attempt to answer these questions through a series of experiments based on the protocol standard. The tests were run on a comprehensive list of popular Web sites to which a good fraction of the Web traffic is directed. Our experiments were conducted on a global extensible testing infrastructure that we built to answer the above questions. The tests were carried out over a period of 16 months and were repeated thrice during the period. Our results show reasons for concern on the state of HTTP/1.1 protocol compliance: some servers do not properly support basic features such as the `HEAD` method, while many popular servers do not support some of the key features (such as persistent connections) that were added in HTTP/1.1. Perhaps most alarming of all, some servers crashed during testing. As a result we believe that small (but significant) changes to the wording of the protocol specification are required.

1 Introduction

With the recent IETF draft standardization of the Hypertext Transfer Protocol—HTTP/1.1 protocol in RFC 2616, it is natural to expect a significant number of clients (browsers), proxies, and servers would be upgraded to adhere to the HTTP/1.1 protocol requirements. In earlier work, we examined the key differences between HTTP/1.0 and HTTP/1.1 [KMK99]. In this work, we report on a longitudinal study that monitored the level of penetration in the Internet of the HTTP/1.1 protocol and the level of protocol *compliance* on the Web (PRO-COW). Many servers (and clients) are configurable and some have already reverted to *disabling* certain features (such as persistent connections) for a variety of reasons.

A wrong way to measure compliance is contacting a Web server and simply checking if the version number field in the response header is HTTP/1.1. This ignores the inherent complexity of the Web (e.g., presence of proxies and gateways), mis-interprets the protocol version number in the response header, and assumes that the server is actually fully compliant based on the protocol version it reports. The question, however, is important and finding a proper answer based on a clear understanding of the Web infrastructure and the HTTP/1.1 protocol would be useful for several reasons. Knowledge of the number of servers actually running fully compliant HTTP/1.1 servers would give us a measure of the protocol adoption rate. If the number is low, it permits us to examine the reasons for the low adoption rate. Repeating the study periodically can be handy to know if future changes are warranted and the speed with which they should be introduced. If compliance with certain features is low, it could have an impact on introduction of similar new features. Web sites are the visible front end for e-commerce companies requiring them to avoid server failures. As we will see later in the paper, failure to adhere to some of the compliance requirements could lead to serious difficulties. An estimate of the traffic that is end-to-end HTTP/1.1 helps us quantify benefits (or disadvantages) of the changes to the protocol that cannot be gleaned by comparing the protocol version numbers.

There are different audiences for this paper. Server implementors may be interested in seeing the level of compliance of their product in practice. Some servers have front ends that distort and alter the semantics of the response leading to potentially incorrect conclusion about the compliance level of that implementation. Protocol designers can see what ideas actually get reflected in usage. Much debated additions to the protocol may not be implemented properly or may be turned off via configuration options in practice. Knowing the realities would help them deal with evolving the protocol. Those interested in learning about compliance issues of protocols in general could see what features are hard to get right and why. Web site administrators deciding on moving to the new version of the protocol can see if a reasonable number of popular sites are running HTTP/1.1. However, if many of the servers running HTTP/1.1 are not compliant or if several of the new features are not enabled, they may want to wait.

Given the enormity of the Web and the complexities of the new version of the protocol (the specification nearly tripled in size between HTTP/1.0 and HTTP/1.1), it is not easy to test the compliance of deployed Web servers. Fortunately however, our task is somewhat simplified for three reasons. Prior work [KMK99] (in which one of the authors of this paper was involved) examined the key differences between HTTP/1.0 and HTTP/1.1, categorizing and identifying the important changes in the protocol. Partitioning of the changes helped isolate the features and enabled the rapid construction of the set of compliance tests. Secondly, although there are tens of millions of sites, only a small fraction of sites attract a significant percentage of traffic. A plausible list of these sites is available from a set of survey sites. Thirdly, access to packet traces from a large ISP helps us verify the accuracy of the survey sites, calibrate the set of popular servers, and determine which (new) HTTP headers are commonly used.

Instead of designing a simple experiment that only addresses the issue of compliance when tested from a client to an origin server we have designed a global experimental infrastructure infrastructure spanning multiple countries and a mix of sites (educational, commercial, and soon residential sites connected via ISPs). The infrastructure is used periodically to study the evolution of protocol compliance. We conducted the compliance tests over a period of 16 months roughly every six months. Along the way several popular sites moved from HTTP/1.0 servers to HTTP/1.1 servers.

We have several significant findings. Over a 16 month period, the percentage of servers claiming to be HTTP/1.1 compliant increased with little improvement in the percentage of servers that were actually compliant. Some popular Web sites failed even the most basic compliance requirements. A significant number of servers have turned off a number of the key HTTP/1.1 improvements. Several servers crashed during our testing. Compliance does not necessarily improve over time. Many of the problems appear to be caused by incorrect server configurations or poorly implemented plugins or filters. A number of implementors have already utilized the results of our study. For example, a prominent browser changed their default behaviour on persistent connections and a server vendor fixed a serious security hole as a result of an earlier version of our study (we notified them discreetly and do not divulge the specific problem here).

The rest of this paper is divided as follows: Section 2 presents motivation for the move to HTTP/1.1 and Section 3 discusses related work in this area. Section 4 presents our compliance testing methodology while Section 5 describes the actual experiment performed to measure the compliance. Section 6 discusses the software used to test and the environment in which the experiment was conducted. Section 7 discusses the results of the experiment. We conclude with a summary of the paper and a discussion of future work.

2 HTTP/1.1 Protocol

Along with the astounding increase in network traffic of HTTP packets, several problems were discovered in the HTTP/1.0 protocol. There was no official standard for the HTTP/1.0 protocol though there were various implementations. The latest version of the Hypertext Transfer Protocol HTTP/1.1

was standardized in June 1999 (in RFC 2616 [ea99]) after over four years of discussion in the IETF HTTP Working Group. A primary motivation in coming up with a new version of the protocol was to fix several known weaknesses in HTTP/1.0. However, during the process of developing the standard, several intermediate "HTTP/1.1" implementations of servers and clients began showing up on the Web.

As part of the standardization effort of HTTP/1.1, reports on several interoperable implementations of the components (clients/browsers and servers) had to be submitted to the W3C–World Wide Web consortium. W3C forum reports describing the features supported by various implementations are available in [For]. The information in the forum reports are submitted by the various individuals/organizations who developed the components. The forum report lets developers indicate the subset of features of the protocol they have implemented and if they have tested the features.

The testing that is reported in the forum is done by the developers themselves but occasionally components are made available for others to test. Some of the servers on the forum are very popular on the Web (e.g., Apache, Netscape-Enterprise, and Microsoft-IIS) while others are experimental servers often used in the research community. There is a reasonable amount of variance in the set of features implemented in this collection and compliance testing cannot be done by just testing the subset presented in this forum. Given that there are millions of sites running servers with different configurations, it becomes important to broaden our compliance testing base and methodology from that used in the forum.

3 Related work

Measurement of protocol compliance is not entirely novel but we know of no other independent testing in the HTTP arena. Partially, this is due to the relative recency of the protocol and HTTP/1.1 is the first upgrade since HTTP/1.0. Earlier versions, such as as HTTP/0.9 and HTTP/1.0 were never formally standardized. Formal testing of non-standardized versions would not have been very helpful. However, the deficiencies found in the implementations of HTTP/1.0 and the prematurely (mis-)labeled HTTP/1.1 helped in the clarification of the actual HTTP/1.1 specification. Forum reports [For] from implementors and interoperability testing conducted via the Web consortium aided in finding some problems. In the commercial arena there are more *claims* than actual evidence of compliance: many products claim compatibility or compliance with HTTP/1.1 to improve their marketability.

4 Methodology

In this section, we present the design decisions of our experiment. We had to decide if the compliance tests could be run locally and if not, how to come up with a canonical set of test sites. We then had to assemble a testing infrastructure, identify testing software, and isolate the features to be tested based on the protocol draft standard. We discuss the various tradeoffs and the reasons behind the choice of the approach we took.

One approach to test compliance would have been to choose just a handful of popular servers (e.g., Apache, Netscape, Microsoft-IIS) and run our tests directly on the software in a lab environment. There are many reasons for *not* doing this. First, it is extremely difficult to test all of the software configurations (we counted nearly 60 different configurations of the Netscape server and close to 700 different configurations of the Apache server in a recent packet trace from a large ISP). Secondly, different answers might result based on where the tests originate (although we did not expect the results to vary based on client location, but we wanted to confirm this hypothesis). Thirdly, it is more interesting to see what installed servers in the field do than a particular binary release with a fixed configuration. Fourth, it is not possible to obtain the source and binaries of all the servers. Several popular sites run proprietary servers designed just for their organization. Finally, we wanted to test under real world conditions – our requests would be like any other Web request. As will be seen later, this decision was well warranted, since some server implementors were surprised to see the circumstances under which their server was being used.

Additionally, even the information about the server returned in the HTTP response (in the `Server:` header) didn't always include any configuration information. This leads to anomalies such as the same feature implemented correctly in some sites and not so in others, although both sites apparently ran the "same" server instance. Relying on the server identifier would thus be a mistake leading to wrong conclusions.

It has been well known for a while that a small number of Web sites attract a significant portion of the Web traffic. This has led to the compilation of popular Web sites which has significant economic value to those sites (revenue from advertisement). Rating sites such as MediaMetrix [Med], Netcraft [Net], and Hot100 [Hot] offer statistics on popularity. In addition, some compilations of well known corporations such as Fortune 500 [For99] and Global 500 [Glo98] provide lists of globally known corporations some of which presumably attract a significant amount of Web traffic. Our combined list contained more than 500 unique Web sites.

Each of the survey sites has a different way of gathering their data and none of them have made their methodology transparent enough for independent testing. Some even say that they deliberately withhold this information to ensure surveyed sites do not misuse it to alter their rankings. Hot100 claims to survey 100,000 users (40% of whom are outside the USA). They claim to gather data at "strategic" points on the Internet (not at the browser or server) which they then sift through. Mediametrix claims to use a 50,000 user population. Our longitudinal study showed that throughout the 16 month period in which our study was conducted, over 40% of the top 150 unique sites from the combined MediaMetrix [Med], Netcraft [Net], and Hot100 [Hot] ranking lists were the same.

If we are interested in testing compliance of the HTTP/1.1 protocol, it would make sense to examine the servers running on popular Web sites. We performed an initial test to determine which sites were using HTTP/1.0 and which sites were claiming to be HTTP/1.1 compliant. All of our remaining tests were performed using only the list of sites that claimed to be HTTP/1.1. We contacted each one of these sites from a variety of locations in the world to ensure that the client location didn't introduce any bias. Our clients were either the *httperf* [MJ98] tool or handwritten C code imitating basic aspects of a browser and saving the response headers returned. Part of the reason for testing from a variety of places is to extend the tests in the next round to go through proxies (rather than directly from client to server as in this work). We also considered some of the W3C forum reports [For] submitted to the W3C consortium by various server and client implementors to see if the features that were reported as implemented and examined in the interoperability tests are indeed compliant. We included a subset of tests that most of the implementors had conducted.

Next, it is important to measure the origin server's compliance without having to worry about the influence of proxies, gateways, and tunnels in the path. Proxies, depending on if they are transparent or non-transparent may modify the requests and alter some of the headers in either direction. The response from the server would not be seen directly by the testing client; only the response sent by the proxy. Our knowledge of our local networks allowed us to avoid both non-transparent and transparent proxies at the client sites. However, even when we send requests directly from clients to origin servers, it is possible for an intermediary in front of the server to intercept the request (we noticed several such cases in our test).

In terms of verifying compliance, we primarily relied on the protocol standard [ea99]. The protocol specification has three classes of compliance by clients, proxies, and servers for features: MUST, SHOULD, and MAY [Bra97]. The HTTP/1.1 specification states that a server implementation that fails to satisfy one or more MUST requirements is *not* compliant. If it satisfies all MUST and SHOULD requirements it is *unconditionally* compliant and if it meets all MUST but not all SHOULD requirements it is *conditionally* compliant.

It should be noted that our tests are *not* simply a test of the MUST, SHOULD, and MAY requirements of the HTTP/1.1 draft standard. Instead, we have divided the tests into categories based on importance to the overall Web infrastructure. While some servers may have consciously not complied with some requirements or turned off some features (since they are *not* a MUST requirement), we may still highlight that fact. The goal of our experiment was to both classify the servers in terms of compliance and also speculate on the reasons for any non-compliance. While we have not tested every feature of the protocol for compliance, we have prioritized and tested some of the key features. The primary reason we did not develop a complete test suite for HTTP/1.1 compliance is that it would be extremely difficult to automate some of the tests without having specific knowledge of the design of each site under test (an example of this is provided in Section 7.2.3. Our testing model is extensible—other features simply require extensions to the script which can be plugged into our testing and analysis infrastructure. This enables continued testing over the long haul as more server sites move to HTTP/1.1. Our continued testing also helped us to monitor changes to the servers over time. We also kept our tests free of biases such as time of day and location of clients.

5 Compliance experiment

The actual compliance experiment involved extracting important features from the HTTP/1.1 protocol specification as presented in RFC 2616, the HTTP/1.1 draft standard. Several of the features of 1.1 are carried over from HTTP/1.0 since all HTTP/1.1 servers have to accept HTTP/1.0 style requests. We divided our experiment into three categories: important features in the protocol specification (all of which are MUST conditions; i.e., the implementation is not compliant otherwise), features that we believe are significant additions in HTTP/1.1, and features that are not mandatory in servers yet are considered useful in evolving the protocol. We expect every compliant server to meet the tests of the first category, most to meet the second category tests, and expect significant variance in compliance for the third category tests. Our testing infrastructure can be easily extended to do other compliance tests by augmenting the scripts and reusing the largely automated analysis process.

5.1 Category One tests

In the first category of the experiment we tested `GET` and `HEAD` methods with modifiers as warranted, and tested for the absence of the required `Host` header. We expect these tests to succeed in *any* compliant HTTP/1.1 server. Servers that do not implement the above features correctly are presumably invalidly labeling themselves as HTTP/1.1. It should be noted that the version number in an HTTP message is a hop-by-hop header (as opposed to an end-to-end header) and since our tests are directly from client to origin server, we get exactly the version number the origin server claims it implements.

A vast majority of all HTTP requests made to Web servers are `GET`, the basic way to request a resource on the Web. The `HEAD` method requests that only metadata about the resource be returned and is often used to debug servers. Neither of these methods are new in HTTP/1.1, nor has their behaviour changed significantly. Use of modifiers with `GET` (such as `If-Unmodified-Since`), however, are new and thus included in our tests. These tests are to verify that servers respond with (the new response code) `412 Precondition Failed`, when the precondition fails.

The `Host` header was added to slow down the depletion of IP addresses, due to a rush to obtain vanity URLs (such as `www.foo.com`) and HTTP/1.0 requests not passing the hostname of the request URL. Rather than change the request line format (which would cause massive configuration difficulties), a new (`Host`:) header was *mandated* to be present in every HTTP/1.1 request message. If a HTTP/1.1 request message does not have the `Host:` header it **must** be rejected.

5.2 Category Two tests

The second category of tests consists of important features that have been added to HTTP/1.1. A significant amount of discussion ensued on some of these features during the over four-year development of HTTP/1.1. In the case of a server mis-implementing or partially implementing features tested in this category, we would be curious to know why. Unlike Category One tests, where errors simply imply non-compliance, this category includes tests of features that servers are permitted to selectively implement. There is a general expectation that a HTTP/1.1 server would implement these features. The tests that we include in this category are handling of persistent connections, pipelining, and range requests.

Introduction of persistent connections was a major innovation in HTTP/1.1. In HTTP/1.0, connections lasted just for for a single request/response exchange. This had both a deleterious effect on user perceived latency and the server (each request required TCP setup and teardown) as well as the network (in terms of the additional packets). Most HTTP transactions are short and the TCP handshakes consumed a good chunk of the overall time. For pages with a dozen embedded images (a figure that is relatively common), multiple TCP setups and teardowns were needed. Mogul and Padmanabhan suggested the introduction of persistent connections based on an experimental study [PM95]. Persistent connections are the *default* in HTTP/1.1, though servers or clients could close the connection after the first exchange. In fact, downloading all the embedded images in a single persistent connection (labeled as perfect persistence [KW00]) has the best performance.

Pipelining permits clients to send a stream of requests in a pipeline without waiting for any response from the server. The round trip time of waiting for the acknowledgments of the previous request is eliminated. The server however sends the responses in the order of the requests received. Further studies [ea97] revealed that persistent connections without pipelining could in some cases worsen the performance. In some cases multiple parallel non-persistent connections were found to be better but this came at a cost (minimal to the browser, higher to the server in order to deal with multiple simultaneous connections from each client). Persistent connections with pipelining provided the best combination to reduce latency and the overall number of packets.

Recently, there has been anecdotal evidence (in discussions and mailing lists) that some sites are turning off persistent connections. If it were true, one of the key perceived advantages of HTTP/1.1 (to the network in terms of reduced packets and to users in terms of latency) would turn out not to have been realized. We tested if servers permitted the basic ability to hold the connection open beyond a single connection and then a separate test of its ability to handle pipelined requests.

Another innovation in HTTP/1.1 was the ability to request byte ranges of resources rather than the full contents. There are several reasons for this, including efficiency, such as requiring just the tail of a growing resource, prefetching the headers of resources of certain content-type (such as *gif, jpeg*) to begin outlining images before actually fetching the image, etc. Recovering from aborted connections and transfers is eased by range requests. When parts of the resources are cached, only the missing parts need to be obtained.

5.3 Category Three tests

The third category of tests include minor issues that servers should normally be compliant with. Consequences of non-compliance here are less severe than the first two categories.

5.3.1 Additional method tests

The HTTP/1.1 specification clearly indicates that all general purpose servers MUST implement `GET` and `HEAD` methods. Support for other methods are optional but a server implementing other methods must conform to the specification (Section 5.1.1 of [ea99]). Thus, our tests of conformance is one of proper compliance *if* other methods are implemented. The OPTIONS method indicates the capabilities of the origin server. With a resource specified, any optional features applicable to that resource alone is returned. The TRACE method purely runs a loopback test of the message included in the request and is simply a way to see if the server received exactly what was sent from the client. The server is supposed to return the request it received in the response body.

5.3.2 Expect/Continue mechanism

To prevent clients from needlessly sending large bodies in PUT/POST requests that might not be accepted by a server, HTTP/1.1 introduced a mechanism by which clients could check with the server beforehand. A client would send just the header (without a body but with a content length indicator) including a request header `Expect: 100-Continue`. If the server is willing to accept the request it would reply with a `100 Continue` status response and then the client can send the body; otherwise the server can send a `401 Unauthorized` or a `417 Expectation Failed` response.

5.3.3 Conditional requests

HTTP/1.1 introduced several new conditionals to improve the caching model. Instead of the simple Last-Modified timestamp check that HTTP/1.0 provided in the `GET If-Modified-Since` request, the presence of opaque strings in the form of Entity tags, permits a more general model. If several instances of a resources are maintained at the server and cached at a proxy, the proxy could check if any of its cached instances are current by including conditional headers such as `If-Match`. Additionally, an `If-Unmodified-Since` conditional permits a resource to be sent only if it has *not* changed since the indicated date.

When performing timestamp checks the format of the date string is an issue. HTTP applications have permitted three date formats RFC 822 (updated by RFC 1123), RFC 1036, and ANSI C's `asctime()`. While, HTTP/1.1 clients and servers have to accept all three formats for compatibility with HTTP/1.0, they can *only* generate the RFC 1123 format for representing date values in header fields.

5.3.4 Miscellaneous tests

Several new requests and responses have been added in HTTP/1.1. We examine a variety of method and header combinations for violations of size or incorrect headers. We also check for the server's ability to handle long and incorrect URLs in the request.

6 Testing software and environment

For testing we primarily relied on *httperf* [MJ98] – a performance measurement tool for HTTP. *Httperf* is useful for understanding Web server performance and for analyzing server features and enhancements. The tool has three logical components: the core HTTP engine, the workload generator, and the statistics collector. The latter two components can be configured at runtime via command line options.

We chose to use *httperf* as an analysis tool for several reasons. Since *httperf* already supports the HTTP/1.1 protocol, it saved us from having to provide our own implementation. Although *httperf* does not currently support all of the features that we needed for this study, the tool is available in source code form. This enabled us to modify the tool to issue the desired request headers and collect the appropriate statistics. Our extensions can be rolled back into *httperf* for others to use.

We tested compliance from a variety of places around the world from diverse organizations. Although this should not be necessary since a server should give the same answer no matter where the requests came from, we did this for two reasons. First, studying an origin server's compliance is just the first stage of our experiment. Additional experiments that test other artifacts of the Web require diversity of location to expose biases in the path between client and server (as in [KW00]). Second, if we noticed any dependency on the origin of requests (though the web sites were contacted using hardwired IP addresses) that would be of interest to explore.

When performing tests of such a large scale nature on several large sites, one has to be careful not to let the testing interfere with the normal workings of the site. We did this by identifying ourselves via the `From:` and `User-agent:` headers in every request we sent. We also sent our few test requests serially and just once.

A C program parsed the response headers in the output from each client site (converting them into integer and/or bitmap descriptors) and verified presence/absence of headers, gleaned values, and checked for appropriate headers.

Primary tests were run from the authors' respective organizations (AT&T Research located in New Jersey, USA, and Hewlett-Packard Labs in California, USA). Additional tests were run from the University of Kentucky in Lexington, Kentucky (USA), the University of Paris-Sud, Orsay, (France), the University of Western Australia (Nedlands, Western Australia), and a commercial site in Santiago (Chile). Even though we chose different organizations and locations, the same software was installed and used for all the tests. Each test was run once from each testing site. The same set of servers were contacted from each of the sites. The machines all ran different versions of the UNIX operating system.

Table 1: Breakdown of server software and protocol version

Vendor/Version	Jun 99	Nov 99	Sep 00
Netscape	34.8%	38.8%	38.1%
Microsoft	32.8%	30.9%	33.3%
Apache	28.2%	26.8%	25.3%
Lotus	2.7%	2.6%	1.9%
Zeus	0.4%	0.6%	0.3%
Others	1.1%	0.3%	1.1%
HTTP/1.0	27.0	16.2	7.5
HTTP/1.1	73.0	83.8	92.5

7 Results

7.1 Experiment details

Although the Apache server has a large lead in the server market (over 60% of the market according to Netcraft [Net]), a majority of the popular sites are running Netscape and Microsoft-IIS servers. Table 1 shows vendor-based distribution of servers. The most popular version of the top 3 servers are Netscape-Enterprise/3.x, Microsoft-IIS/4.0, and Apache/1.3.x. Since servers from just three organizations are used by over 95% of the most popular sites running HTTP/1.1, compliance can be improved significantly by ensuring that *all* configurations of these servers are fully compliant.

Table 1 also shows the increase in the number of popular servers that are claiming to be running HTTP/1.1 over the course of our study. While the table shows a significant increase in servers claiming to be running HTTP/1.1, our results indicate that improvement in compliance has not kept pace.

There are three important caveats to be kept in mind while interpreting our results. First, there are several popular sites that still (as of September 2000) use HTTP/1.0 servers. Among these are AOL, Excite, Yahoo, Amazon, and Altavista. In fact over 7% of the HTTP/1.0 servers we contacted are not Y2K compliant in their `Date` header format. Second, the popularity as measured by MediaMetrix [Med], Netcraft [Net], and Hot100 [Hot] is purely based on number of requests sent to these sites and *not* on the bytes of response. One could just as easily make a case that a top n listing based on bytes shipped from servers is a more important metric. If we are seeking bandwidth reduction through the use of new features in HTTP/1.1, byte-wise high volume servers are likely be to better targets. However, the lack of server logs from these top n sites does not give us a way of measuring this. Finally, there is evidence that pornographic sites are downplayed in surveys of sites. Thus it is possible that such sites didn't end up in our top n sites list. Additionally, such sites, given their propensity to include more images, often have a higher volume (bytewise) of response traffic than other sites. We have been able to verify this based on two separate sets of data: a packet trace from a large ISP where half a dozen porn sites running HTTP/1.1 had many more response bytes compared to the rest of the frequently visited sites, and a proxy log from a large content hosting ISP. We chose not to change our methodology to add such sites to our list.

7.2 Experimental results

In Section 5, we divided our tests into three categories. In this section, we examine the results for each of the categories, as well as for most of the set of tests that we perform. Our compliance tests focus on the extreme cases: determining which servers satisfy all MUST and SHOULD requirements (the unconditionally compliant servers); and determining which servers cannot satisfy one or more MUST requirements (the non-compliant servers).

The initial round of testing in June 1999 confirmed our hypothesis that the location of the request origin does not (in general) have an impact on the compliance. However, there were a few minor variations. We identified two potential causes for the variation in the results by client site. Although we attempted to contact the same server from each client (by utilizing the IP address rather than the host name of the server), some sites appear to use load balancers or other devices to distribute incoming requests among a cluster of (heterogeneous) servers. Furthermore, a number of sites were unavailable during several of our tests; since our results are calculated based on the number of sites that responded to a test (unless otherwise noted), this creates a small amount of variability in the computed percentages. In June 1999 approximately 2% of sites were unavailable; in the last two studies the number of unavailable sites was around 1%. Since the results varied only slightly depending on which client site is utilized, we use only

Table 2: Unconditional Compliance Results for Category One Tests (HPL Data). All figures are in percentage.

Date	GET	HEAD	Host	Pass All	Fail All
Jun 99	83.5	72.9	64.5	60.6	7.1
Nov 99	82.4	69.6	60.0	56.8	7.3
Sep 00	81.9	72.4	64.7	59.1	6.5

Table 3: Breakdown of Category One Test Result Ranges (HPL Data) All figures are in percentage.

	GET	HEAD	Host
Unconditional	81.9–83.5	69.4–72.9	60.0–64.7
Missing Headers	16.1–17.9	8.9–10.8	28.6–30.4
Not Compliant	0.2–0.4	17.7–19.6	6.6–9.6

one (the HPL site) for discussion purposes throughout the remainder of this section. Details on the other sites are available in [KA99].

7.2.1 Category One test results

In this first round of tests we examine three required features for HTTP/1.1: the `GET` and `HEAD` methods, and the `Host` header. We consider a server to be unconditionally compliant for these tests if it returns an appropriate status code (200 for `GET` and `HEAD` tests, and 400 for the `Host` test), and if the response includes the appropriate headers (e.g., `Date` and `Content-Length` or `Transfer-Encoding: chunked`).

The results of this analysis are shown in Table 2. The results reveal that across the three measurement periods the results were quite consistent, with around 82% of the sites under study unconditionally compliant with respect to the `GET` method; about 70% of the sites were unconditionally compliant for the `HEAD` method, while over 60% passed the `Host` test. About 60% of the sites under study were unconditionally compliant across all three tests, while around 7% of the sites failed all three tests for unconditional compliance. Table 2 reveals that the results were quite consistent across the studies conducted over a 16 month period. Note that even with the migration of several servers from HTTP/1.0 to HTTP/1.1 during the testing period and the introduction of new server versions, there has been little improvement in compliance. In order to understand why so many servers failed to pass one or more of these three basic tests we examine each test separately.

Table 3 provides a more detailed breakdown of the Category One results. The ranges represent the minimum and maximum observed results over the entire duration of the study. As we have already shown, most servers are unconditionally compliant with respect to `GET` requests. Between 16 to 18% of the servers did not include either a `Content-Length` header or a `Transfer-Encoding:chunked` header to indicate to the client the length of the message body. Due to this omission these servers are characterized as conditionally compliant. Several servers failed the `GET` compliance test either by returning an incorrect status code or an incorrectly formatted `Date` header.

Around 70% of the tested servers were unconditionally compliant with respect to `HEAD` requests. Approximately 9% of the tested servers did not include the same entity headers that were seen in response to the `GET` request. Thus, these servers are deemed to be conditionally compliant. The more intriguing result is that almost 18% of the tested servers failed the compliance test because they returned a status 500 response rather than the expected 200 response.

Table 3 indicates that close to two-thirds of the servers fulfilled all requirements when responding to a request that did not include a `Host` header. Nearly 30% of the tested servers did not include either a `Date` header or one of `Content-Length` or `Transfer-Encoding: chunked`. These servers are considered to be conditionally compliant. Between 6 and 9% of the servers were not compliant, as they did not require the `Host` header to be present.

To determine whether a specific type of Web server was responsible for the unusual behaviour we observed, we analyzed the data by the type of server. The results for the five most common servers seen in our tests are shown in Table 4. These five server versions accounted for an average of 88.7% of all the servers seen in our tests over the testing period. Each server has three rows associated with it representing data from Jun '99, Nov '99, and Sep '00 respectively.

Table 4 indicates that two of the top five servers (Apache/1.3 and Apache/1.2) were unconditionally compliant on almost every site under study; in the few cases where these servers were not unconditionally compliant, the cause was usually a missing header, perhaps the result of the (mis)configuration of that particular server. Microsoft-IIS/4.0 ranked third in the percentage of servers that passed all three tests for unconditional compliance, trailing the Apache servers by about 10%. Most of the remaining Microsoft-IIS/4.0 servers did not issue the expected response headers. None of the Apache/1.3, Apache/1.2 or Microsoft-IIS/4.0 servers failed all three tests for unconditional compliance.

The results are less positive for the Netscape-Enterprise 3.5 and 3.6 servers. None of these servers passed all three of our tests for unconditional compliance. In fact, over 15% of the Netscape/3.5 servers and over 24% of the Netscape/3.6 servers failed all three unconditional compliance tests. This observation suggests that perhaps there is a problem with the configuration of these servers at some sites. These results also suggest that certain types of Web servers are responsible for much of the absence of compliance we noted earlier in this section. A somewhat discomforting observation is that the Netscape/3.6 server appears to be less compliant than its ancestor, the Netscape/3.5 server; i.e., things do not necessarily improve over time. Reviewing the results in Table 4 suggests that the biggest change between these versions occurs with the `GET` requests. While nearly 70% of the Netscape/3.5 servers were unconditionally compliant on this test (the remaining 30% were missing a `Content-Length` or `Transfer-Encoding: chunked` header), only 56% of the Netscape/3.6 servers passed unconditionally (the remainder were missing a length header). At this time we do not know why this change has occurred. Both the Netscape/3.5 and 3.6 servers did quite poorly on the `HEAD` test, with only around 21% and 27% passing the unconditional compliance tests respectively. The remaining Netscape 3.5 and 3.6 servers were either missing the expected headers or returned an incorrect status code. All of the Netscape 3.5 and 3.6 servers failed the `Host` test for unconditional compliance. About 80% of the 3.5 servers and 85% of the 3.6 servers were missing both a `Date` and a length header, while approximately 20% of the 3.5 servers and 14% of the 3.6 servers were not compliant, as they did not require a `Host` header to be present.

At this time, there are only a few popular server sites that run IIS/5.0 and Netscape 4.x and so there is insufficient data to draw any conclusions. We plan to continue to monitor changes in the distribution of servers used by popular sites, and analyze the compliance of the most common versions.

Even though we found that certain types of Web servers are responsible for a lot of the non-compliant behaviour we observed, the results in Table 4 also indicate that there is a lot of variability in the degree of compliance even among more homogeneous groupings. This suggests that the configuration (including the use of plugins) may have a significant role in determining how compliant a server is and validates the methodology of our testing actual server sites rather than the server software.

The results in Table 2 revealed that 40% of the servers failed one or more of the Category One tests for unconditional compliance to the HTTP/1.1 specification. The most common reason for failure was the lack of appropriate response headers.

Perhaps a more significant observation is more than 20% of the servers tested were not compliant (i.e., they failed a MUST condition) on at least one of the three basic functionality tests. 3% of the servers were not compliant on two of the tests. These servers should definitely not claim to be HTTP/1.1 applications.

7.2.2 Category Two test results

The next set of tests examined some widely discussed enhancements to HTTP/1.1—server support of persistent connections, pipelining, and range requests. Although maintaining a persistent connection is supposed to be the default behaviour of an HTTP/1.1 application, it is a SHOULD requirement and thus a server can be conditionally compliant without maintaining persistent connections. However, servers have to send a `Connection: close` header to indicate non-persistence. The Range feature is a MAY level requirement; servers can always send the complete response.

Table 4: Breakdown of Category One Results by Server Type (HPL Data)

Server	Date	%of Svrs	GET(%)	HEAD(%)	Host(%)	Pass All(%)	Fail All(%)
Apache/1.2	6/99	9.1	100.0	100.0	97.8	97.8	0.0
	11/99	6.3	96.9	100.0	96.9	93.8	0.0
	9/00	2.4	100.0	100.0	100.0	100.0	0.0
Apache/1.3	6/99	18.1	100.0	96.7	100.0	96.7	0.0
	11/99	19.7	99.0	99.0	99.0	98.0	0.0
	9/00	21.8	98.4	98.4	98.4	98.4	0.0
IIS/4.0	6/99	32.5	89.7	98.0	98.2	87.3	0.0
	11/99	30.7	89.7	98.1	98.1	88.4	0.0
	9/00	30.2	90.2	98.9	96.0	89.0	0.0
Netscape/3.5	6/99	14.0	70.8	22.2	0.0	0.0	16.9
	11/99	13.8	71.4	20.3	0.0	0.0	14.5
	9/00	6.2	66.7	22.2	0.0	0.0	13.9
Netscape/3.6	6/99	12.8	50.8	29.2	0.0	0.0	27.7
	11/99	21.7	58.2	23.8	0.0	0.0	23.9
	9/00	27.0	60.2	27.9	0.0	0.0	20.8

Table 5: Breakdown of Category Two Results (HPL Data)

Date	Persistence	Pipelining	Range	Pass All	Fail All All
Jun 99	71.9	70.6	52.3	43.9	20.6
Nov 99	71.5	64.8	54.9	41.0	21.2
Sep 00	74.3	66.8	54.7	42.8	20.8

Table 5 reveals that many of the servers supported at least one of the Category Two features. Over 70% of the servers supported persistent connections. We suspect that many remaining sites had persistent connections disabled by the administrators. Slightly fewer sites allowed the client to pipeline requests. Only about half of the servers supported Range requests properly. Only 40% of the sites supported all three Category Two features, while 20% of the sites supported none of these features.

The results in Table 6 reveal that the server type has less effect on the results than was the case for the Category One tests. Almost all of the Apache/1.2, Apache/1.3 and Microsoft-IIS/4.0 servers supported persistent connections and pipelining. Only about half of these servers supported the Range requests which is the primary reason why so few of these server passed all three Category Two tests. Fewer Netscape-Enterprise/3.5 and 3.6 servers supported persistent connections and pipelining than ranges. We have evidence that suggests there is a problem with the persistent connection implementation in instances of some servers (e.g., the server sends a `Connection: keep-alive` header with the response, then closes the connection without allowing the client to issue any subsequent requests).

We next examine the number of servers that (unconditionally) supported all six features. Table 7 shows that around 30% of the servers under study passed all six tests, while 7% of the servers were either conditionally compliant or not compliant on all six tests. Not shown in the table are the following figures: 15% of the Netscape-Enterprise/3.5 servers (N-E/3.5) failed all six tests (i.e., they supported zero features) and none supported all six features. Over 20% of the Netscape-Enterprise/3.6 servers (N-E/3.6) failed all six tests and none supported all six features. Approximately 50% of the Apache/1.2, Apache/1.3 and Microsoft-IIS/4.0 servers supported all six features.

7.2.3 Category Three test results

In this section we tested the servers to determine whether they supported nine other suggested/recommended but lesser known features of

Table 6: Breakdown of Category Two Results by Server Type (HPL Data)

Server	Date	Persistence(%)	Pipelining(%)	Range(%)	Pass All(%)	Fail All(%)
Apache/1.2	6/99	89.1	89.1	52.7	43.5	10.9
	11/99	90.6	90.6	46.9	40.6	3.1
	9/00	100.0	92.9	50.0	50.0	0.0
Apache/1.3	6/99	87.0	87.0	51.1	47.8	9.8
	11/99	89.0	89.0	54.0	51.0	8.0
	9/00	89.0	88.2	52.8	48.8	7.9
IIS/4.0	6/99	87.9	87.3	52.4	52.4	12.7
	11/99	85.9	85.8	55.8	54.5	12.8
	9/00	86.8	86.7	52.6	49.7	11.4
Netscape/3.5	6/99	41.1	38.4	67.2	37.5	30.6
	11/99	40.0	28.1	68.6	24.6	30.4
	9/00	44.4	31.3	63.9	27.8	36.1
Netscape/3.6	6/99	41.5	35.4	47.7	35.4	52.3
	11/99	50.0	33.6	54.6	32.7	44.6
	9/00	54.2	32.3	57.4	31.6	40.7

HTTP/1.1. Unfortunately many of these features are not straightforward to test. For example, POST requests may not be allowed for certain objects. In such a situation we can test that the server properly rejects the request, but we cannot test whether the server would properly handle a POST request.

Table 8 shows the results of our Category Three tests. The ranges shown in Table 8 indicate the minimum and maximum values seen over the three measurement periods. We observed that a significant number of servers returned non-compliant responses in some of our Category Three tests. For example, some servers return a status 200 response (along with a Web page that indicates an error has occurred) to a request for an incorrect/non-existent URL. This is due to the server configuration rather than the server implementation. The most significant occurrences of non-compliant responses happened with our "POST with `Expect: 100-Continue`" test and our `If-Unmodified-Since` tests. In the `Expect: 100-Continue` test we observed that many sites did not issue a response to the request. Since the number of sites that did not respond was much higher than normal (27–30% of sites versus an average of 1-2% for the other tests), we speculate that most of these sites are simply waiting for further information from the client. Most of the servers we tested do not appear to understand the `If-Unmodified-Since` header, and as a result of ignoring it return a non-compliant response. All of

Table 7: Breakdown of Category One and Two Results (HPL Data)

Date	Pass All Tests(%)	Fail All Tests(%)
Jun 99	31.1	7.1
Nov 99	29.8	7.3
Sep 00	31.4	6.5

Table 8: Category Three Test Results (HPL data)

Feature	% Servers Unconditionally Compliant	% Servers Not Compliant
OPTIONS	26.8–32.3	0.8–2.7
TRACE	94.3–97.3	0.2–1.8
FOO	54.5–60.3	5.3–7.1
POST, Expect	54.6–63.2	31.0–32.0
Incorrect URL	75.9–80.5	5.3–8.2
Long URL	62.7	2.0
I-U-S (1123)	40.0–41.7	57.1–59.3
I-U-S (1036)	40.0–41.7	57.1–59.3
I-U-S (ANSI C)	40.0–41.7	57.1–59.3

I-U-S == `If-Unmodified-Since` with `Date` in RFC 1123/1036/ANSI-C formats.
Long URL test done only in June '99.

the servers that do implement this feature correctly understood all of the three required date formats.

7.3 Intersection of three studies

We examined the sites that were in the top 150 in all three study periods. Of the sites claiming to be running HTTP/1.1, 44% of these sites ran Apache, 37% Netscape and 17% IIS. The results were quite consistent with the overall results presented in Section 7. For example, all Apache servers were unconditionally compliant across all Category One tests, as were 71% of IIS (4.0 and 5.0) servers. None of the Netscape servers passed all three Category One tests for unconditional compliance..

7.4 Reasons for non-compliance

There appear to be several reasons for non-compliance on the part of servers. Some of the reasons are subtle and may not be known even to the server implementors. While most instances of a Microsoft-IIS/4.0 site tested yielded the proper `400 Bad Request` to a HTTP/1.1 request without the `Host:` header, 7 of the 174 sites claiming to run IIS/4.0 returned a `200 OK` response. Closer examination and consultation showed that at least one site probably uses an ISAPI filter [Isa] – a dynamically linked library that intercepts requests—with the intention of modifying it before the core server can parse it. In this situation it is the responsibility of the filter to return the appropriate HTTP headers. The failure of many filters to do this correctly suggests that current server implementations may not be meeting the needs of the people who use those servers. Server architectures may need to be redesigned so that the server retains responsibility for handling the HTTP headers while providing users with the functionality they desire. Some sites have purposely configured their servers to return a success response with HTML text indicating that an error has occurred. Even though the server implementation may be compliant, its configuration causes compliance failure, suggesting that the server does not provide its users with desired functionality.

Our results showed that a significant number of sites disabled HTTP/1.1 features such as persistent connections. There are at least three reasons for this behaviour. First, some sites may be concerned about the performance impact of using such features. One server vendor states that "If your site has hits from many users at any time, then persistent connections may not be good for your server." [IBM]. However, no performance evaluation results were provided to substantiate this statement. Second, problems with TCP implementations on some clients has been reported to cause unexpected behaviour in some browsers when persistent connections are used [IBM]. Some sites may therefore choose to disable persistent connections in order to avoid receiving complaints from visitors to their Web site. Third, some HTTP/1.1 servers actually shipped with persistent connections disabled. If the default configuration was used (or if this feature was not specifically enabled) then the site would not support persistent connections. Early versions of a vendor's server [IBM] had persistent connections enabled by default, but some later versions of the had persistent connections disabled by default. Server implementations can thus become less compliant in newer versions.

8 Conclusions and future work

We examined compliance to the HTTP/1.1 protocol of the most popular Web sites in the world. Although many of the popular sites claim to run HTTP/1.1, some even fail the most basic compliance requirements. Many others run with a number of the significant HTTP/1.1 improvements turned off. A presentation by one author of early results of this work [Kri99] led to a prominent browser making persistent connections the default, a prominent server vendor fixing a denial of service attack problem, and another prominent server handling some of the compliance errors. It is clear to us based on our tests and conversations with server developers that MUST level conditions are more likely to be taken seriously. The more damaging scenarios outlined (such as the one that led to server crashes) should be changed from SHOULD to MUST and we have recommended so to the IETF. Including an appendix in the HTTP specification that highlights all of the MUST and SHOULD level conditions may assist implementors in developing compliant products.

We described the first phase of our experiment examining a list of popular servers with a suite of static tests. Ongoing extensions include dynamic tests examining response headers as they arrive and generating subsequent requests based on the response. It may also be useful to examine entire user sessions rather than just single requests for the home page (or text portion thereof). Currently, we use offline analysis to check for protocol compliance. Incorporating this process into the probing mechanism would simplify the conformance checks. A natural extension is testing compliance of proxies. However, testing proxy compliance is significantly harder than testing servers or clients. An HTTP message can go through several proxies, only some of which may be HTTP/1.1 proxies. Compliant HTTP/1.1 proxies can be detected by the presence of `Via` headers but we would not be able to identify the HTTP/1.0 or non-compliant HTTP/1.1 proxies.

Acknowledgment

We thank Anja Feldmann, Roy Fielding, Jim Gettys, Richard Gray, David Kristol, Scott Lawrence, Jeff Mogul, and David Mosberger for answers to various questions. We thank Michel Beaudouin-Lafon, James Griffioen, Eduardo Krell, and Graeme Yates for giving us access to machines. We thank Mediametrix, Netcraft, Hot100 and other sites that periodically run surveys on popular sites and present server penetration statistics. We thank Jennifer Rexford for comments on an earlier draft.

References

[Bra97] S. Bradner. Key words for use in RFCs to indicate requirement levels. RFC 2119, IETF, March 1997. `ftp://ftp.ietf.org/rfc2119.txt`.

[ea97] H.F.Nielsen et al. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proc. ACM SIGCOMM*, pages 155–166, August 1997. `http://www.inria.fr/rodeo/sigcomm97/program.html`.

[ea99] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, HTTP Working Group, June 1999. `ftp://ftp.ietf.org/rfc2616.txt`.

[For] HTTP/1.1 feature list report summary. `http://www.w3.org/Protocols/HTTP/Forum/Reports/`.

[For99] 1999 Fortune 500 companies, Fortune volume 139 number 8, April 26 1999.

[Glo98] 1998 Global 500 companies, Fortune Magazine 1998.

[Hot] 100 hot.com. `http://100hot.com/`.

[IBM] HTTP Server for AS/400: Persistent Connections. `http://www.as400.ibm.com/products/http/services/persist.htm`.

[Isa] ISAPI callback functions. `http://support.microsoft.com/support/kb/articles/Q150/3/12.asp`.

[KA99] Balachander Krishnamurthy and Martin Arlitt. PRO-COW: Protocol Compliance on the Web, August 1999. `http://www.research.att.com/~bala/papers/procow-1.ps.gz`.

[KMK99] Balachander Krishnamurthy, Jeffrey C. Mogul, and David M. Kristol. Key differences between HTTP/1.0 and HTTP/1.1. In *Proc. Eighth International World Wide Web Conference*, Toronto, May 1999.

[Kri99] Balachander Krishnamurthy. PRO-COW: Protocol comliance on the Web, November 1999. Invited plenary session talk at IETF meeting.

[KW00] Balachander Krishnamurthy and Craig E. Wills. Analyzing factors that influence end-to-end web performance. In *Proc. World Wide Web Conference*, May 2000. `http://www.research.att.com/~bala/papers/e2e.ps.gz`.

[Med] Media Metrix.`http://mediametrix.com/`.

[MJ98] D. Mosberger and T. Jin. httperf—a tool for measuring web server performance. In *Proceedings of WISP '98, Madison, WI*, pages 59–67, June 1998. `http://www.hpl.hp.com/personal/David_Mosberger/httperf`.

[Net] The Netcraft Web Server Survey. `http://netcraft.co.uk/survey`.

[PM95] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving HTTP latency. *Computer Networks and ISDN Systems*, 28(1/2):25–35, December 1995.