PRO-COW: Protocol Compliance on the Web

Balachander Krishnamurthy, Martin Arlitt
Internet Systems and Applications
HP Laboratories Palo Alto
HPL-1999-99
August, 1999

E-mail: bala@research.att.com
arlitt@hpl.hp.com

HTTP/1.1, protocol, World Wide Web, compliancy

With the recent (draft) standardization of the 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 are 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 same infrastructure will be used to answer questions like the percentage of the traffic flow that is end-to-end HTTP/1.1. Our results show reasons for concern on the state of HTTP/1.1 protocol compliancy and the subset of features actually available to end users.

1 Introduction

With the recent IETF draft standardization of HTTP/1.1 protocol in RFC 2616, it is natural to expect that there would be a significant number of clients (browsers), proxies, and servers that are 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 [1]. In this work, we examine the level of current penetration in the Internet of the HTTP/1.1 protocol and the level of *compliance* to the protocol. Many servers (and clients) are configurable and some have already reverted to *turning off* certain features (such as persistent connections) for a variety of reasons.

A wrong way to measure compliance, although unfortunately already widespread, is to contact a set of Web servers and see if the version number field in the response header is HTTP/1.1. This is wrong for several reasons: it ignores the inherent complexity of the way the Web works (presence of proxies and gateways), mis-interprets the protocol version number in the response header, and finally assumes that the server is actually fully compliant based on the protocol version it reports. The difficulties with the interpretation of version numbers has been debated recently [2].

The question, however, is of value and finding a proper answer based on a clear understanding of the world wide 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 could point to problems in the adoption and permit us to examine the reasons for this.
- 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 simply by comparing the protocol versions.
- Repeating the study periodically gives us the rate of adoption of the protocol; this can be handy to know if future changes are warranted and the speed with which they should be introduced. If compliancy is low, especially in certain features, it could have an impact on introduction of similar new features.
- At a meta-level, if there is difficulty in obtaining the answer, then changes in the protocol or infrastructure may be warranted to enable easier measurement of the penetration and compliance.

We view the audience of this paper to be four-fold:

1. Web site administrators who are trying to decide if they should move to the new version of the protocol: if a reasonable number of popular sites are running HTTP/1.1, then they may want to follow the trend. 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 take a wait-and-see attitude. Since Web sites differ in the need for features, examining if features of interest to them are widely available, would give a clear indication of the need for adoption of the new version. As we will see later, there are several well known and popular sites still running HTTP/1.0.
2. Those interested in learning about compliance issues of protocols in general and especially of a protocol that is responsible for 70% of the packets [3] on the Internet: it would be useful to see what features are hard to get right and why. Subtle interaction between features show up in practice leading to either non-adoption or selective *turning-off* of features. Our experimental tests reveal such selective turning-off of features.

3. Server implementors: they may be interested in seeing how popular their version of software is on the Web and the actual level of compliance in practice. Some servers in practice have front ends enabled that distort and alter the semantics of the response leading to potentially wrong conclusion about the compliancy level of that implementation. We were able to identify at least one popular server that had this problem.

4. Protocol designers: they can see what ideas actually get reflected in usage and speculate on reasons for it. Much debated additions to the protocol may not be implemented properly or turned off via configuration options in practice. Knowing the ground realities and reasons would help them deal with evolving the protocol.

Given the enormity of the Web and the complexities of the new version of the protocol (the specification tripled in size between HTTP/1.0 and HTTP/1.1), it is not easy to test the compliancy on the ground. Fortunately however, our task is somewhat simplified due to the following:

- Prior work [1] (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 compliancy tests.
- Even though 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 are available from a set of survey sites.
- Access to packet traces from a large ISP helps us verify the accuracy of the survey sites and calibrate the set of popular servers and use of headers (indicating use of new features).

Instead of designing a simple experiment that only addresses the issue of compliance when tested from a client to a origin server we have designed a larger experimental infrastructure. The global infrastructure spanning multiple countries and a mix of sites (educational, commercial, and soon residential sites connected via ISPs) will be used continuously to see how compliance of the protocol evolves. Additionally, we are already extending the tests to include proxies in the path.

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. An appendix gathers information about a large subset of the HTTP headers.

2 HTTP/1.1 Protocol

Along with the astounding success of HTTP/1.0 protocol (a recent study[3] estimated that nearly 70% of packets in the Internet backbone traffic is HTTP), several problems were discovered in it. 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 [4]) 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 on the features supported by various implementations are available in [5]. 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 compliancy 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 compliancy 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 such work 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 [5] 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 compliancy with HTTP/1.1 to improve their salability. A closer examination reveals that HTTP/1.1 compatibility claim does not always hold up.

4 Methodology

In this section, we present the design decisions of our experiment. We had to decide if the compliancy 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 compliancy 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. We chose not to do this for the following reasons:

1. It is not possible to test the software under various configurations (we counted nearly sixty different configurations of the Netscape server and close to seven hundred different configurations of the Apache server in a recent packet trace from a large ISP).
2. Different answers might result based on where the tests originate.
3. It is more interesting to see what installed servers in the field do than a particular binary release version.
4. It is not possible to obtain the source and binaries of all the servers. Several sites run proprietary servers designed just for their organization.

5. We wanted to test under real world conditions–our requests would be folded in with other "normal" requests. 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[1]. 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. Given the level of vagaries on the Web, nothing short of a direct test of sites would suffice.

It has been well known for a while that few Web sites attract a significant portion of the Web traffic. This has led to compilation of popular Web sites which has significant economic value to those sites (revenue from advertisement). Several rating sites have sprung up that offer statistics on popular Web sites. These include MediaMetrix [6], Netcraft [7], and Hot100 [8]. In addition, some compilation of well known corporations such as Fortune 500 [9] and Global 500 [10] provide lists of globally known corporations some of which presumably attract a significant amount of Web traffic.

Each of the survey sites has a different way of gathering their data and none of them have made it 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) to get 20 Mb of "data" which is ftp'd daily. They claim to gather data at "strategic" points on the Internet (not at the browser or server) which they then sift through. Mediametrix (which recently merged with Relevant Knowledge, another reporting company) claims to survey 40,000 users (this datum is from August 1998).

If we are interested in testing compliancy of the HTTP/1.1 protocol, it would make sense to examine the servers running on popular Web sites. We compiled a fairly exhaustive list from a variety of sources (including the ones listed above) of (over 500) servers to which a significant portion of Web traffic is directed. We contacted each one of them from a variety of locations in the world (from three continents including a university in North America, a university each in France and Australia, two commercial sites in North America representing both coasts, and a commercial site in South America) to ensure that the client didn't introduce any bias. Our clients were either the *httperf* [11] tool from HP Labs 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 in this work).

We also considered some of the W3C forum reports [5] submitted to the W3C consortium by various server and client implementors to see the features that were actually implemented in the interoperability tests. We tended to go with a subset of tests that most of the implementors had reported.

Next, it is important to measure the origin server's compliancy without having to worry about the influence of proxies, gateways, and tunnels in the path. Proxies, depending on if they are explicit or 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. Even when we send requests directly from clients to origin servers, it is possible for a proxy in front of the server to intercept the request (we noticed several such cases in our test).

In terms of verifying compliancy, we primarily relied on the protocol standard [4]. The protocol specification has three classes of compliance by clients, proxies, and servers for features: MUST, SHOULD, and MAY (for a clear interpretation of these terms used in standards see [12]). The HTTP/1.1 specification states that a server implementation that fails to satisfy one or more MUST requirements is *not*

[1] In a popular server we noticed this and the anonymous contact there believed it to be a wrong design decision to omit this information.

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 compliancy and also speculate on the reasons for any non-compliancy.

While we have not tested every feature of the protocol for compliancy (yet), we have prioritized and tested some of the key features. 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. Continued testing also helps us 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:

1. Important features in the protocol specification (all of which are MUST conditions; i.e., the implementation is not compliant otherwise).
2. Features that we believe are significant additions in HTTP/1.1.
3. Several features that are not mandatory to be present in servers, yet were considered useful in evolving the protocol.

We expect every compliant server to meet the tests of Category One tests and most to meet the Category Two tests. We expect significant variance in compliancy for the Category Three tests.

5.1 Category One tests

In the first category of the experiment we tested the following features:

1. `GET` and `HEAD` methods, with modifiers as warranted
2. Absence of `Host` header

The reasons for choosing the above are simple: 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.

5.1.1 Methods

Over 95% of all requests made to Web servers are `GET`[2] and it is vital that servers implement this properly. The `HEAD` method is often used to debug servers and can be handy to get the metadata of resources. Neither of these methods are new in HTTP/1.1, nor has their behaviour changed significantly. Use of modifiers with `GET` (such as `If-None-Match` and `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.

5.1.2 Host

The `Host` header was added to slow down the depletion of IP addresses, thanks to a rush to 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 did not have the `Host:` header it **must** be rejected. Unfortunately, some servers didn't adhere to this, as we will see in the section on results (Section 7).

5.2 Category Two tests

The second category of tests is 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. We expect to add more features to this category as our testbed evolves. 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 included in this category are:

1. Persistent Connections and Pipelining
2. Range requests

5.2.1 Persistent connections and Pipelining

One of the major innovations in HTTP/1.1 was the introduction of persistent connections. 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 [13]. Persistent connections are the *default* in HTTP/1.1, though servers or clients could close the connection after the first exchange.

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

[2] This figure is from a recent packet trace data from a large ISP.

eliminated. The server however sends the responses in the order of the requests received. Further studies [14] 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 thus tested both persistent connections and pipelining. We tested if servers permitted the basic ability to hold the connection open beyond a single connection and then the separate test of its ability to handle pipelined requests.

5.2.2 Range 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

In the final category are several minor issues. We expect most servers to be compliant and if they are not it is not a disaster.

- OPTIONS and TRACE (additional methods) and POST
- `Expect/100-Continue` mechanism
- `If-None-Match`, `If-Unmodified-Since` cache conditionals
- Miscellaneous tests for size/header violations

5.3.1 Additional method tests

The HTTP/1.1 specification clearly indicates that all general purpose servers MUST implement `GET` and `HEAD` methods, with support for all other methods being optional. However, if a server does implement other methods, the implementation must be conformant with the specification (Section 5.1.1 of [4]). Thus, our tests of conformance is not based on support of other methods but simply one of proper compliance if implemented.

The OPTIONS method is a useful method that 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. To limit the number of proxies or gateways that can forward the request, HTTP provides a `Max-forwards` header. Since our tests were directly from client to origin server we did not test the `Max-forwards` feature.

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`, `If-None-Match`, etc. 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 to be tested for size and other common violations. In this set of tests we examine a variety of method and header combinations for violations of size or incorrect headers. We check for the ability to handle long URIs and incorrect URIs in the request.

6 Testing software and environment

For testing we primarily relied on *httperf* [11]–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. Both the load generator and statistics collector 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 compliancy from a variety of places around the world from diverse organizations. While strictly speaking this should not be necessary, (a server should give the same answer no matter where the requests came from) we did this for two reasons:

Table 1: Global probing sites used for our experiment

Organization/Location	Hardware/OS
AT&T Research, NJ, USA	SGI/Irix 5.3
HP-Labs, California, USA	HP Pentium Pro 200/Linux 2.2.6
Univ. of Kentucky, USA	Sparc/Sun OS 5.6
University of Paris-Sud	Sparc/Sun OS 4.1.3
Univ. of Western Australia	Alpha/Digital OSF/1 V3.2
Santiago, Chile	UltraSPARC/SunOS 5.5.1

- Studying an origin server's compliancy 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.
- 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.

A C program parsed the response headers in the output from each client site converting them into integer and/or bitmap descriptors verify presence/absence, glean 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 University of Kentucky in Lexington, Kentucky (USA), University of Paris-Sud, Orsay, (France), 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 UNIX—Table 1 shows the details.

7 Results

7.1 Experiment details

Table 2 shows the top fifteen sites that we constructed based on the composite from Netcraft, MediaMetrix and Hot100.

Though the Apache server has a large lead in the server market (over 60% of the market in a variety of surveys), a majority of the *request* traffic in our collection of popular sites goes to Netscape and Microsoft-IIS servers. Table 3 shows distribution of request traffic on the basis of vendors. 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 account for over 95% of request traffic of the most popular sites running HTTP/1.1, compliancy can be improved significantly by ensuring that *all* configurations of these servers are fully compliant.

Three important caveats should be kept in mind while interpreting our results:

- There are several well known and popular sites that do not (as of July 1999) run HTTP/1.1. Among these are AOL, Excite, Yahoo, Amazon, and Altavista.

Table 2: Top 15 sites in no particular order

Organization	**URL**
Yahoo	www.yahoo.com
America Online	www.aol.com
Microsoft	www.microsoft.com
Netscape	www.netscape.com
Lycos	www.lycos.com
Hotmail	www.hotmail.com
Excite	www.excite.com
Microsoft Network	www.msn.com
Apple	www.apple.com
Altavista	www.altvista.com
Amazon	www.amazon.com
IBM	www.ibm.com
Sun	java.sun.com
Slashdot	www.slashdot.org
Xoom	www.xoom.com

Table 3: Top server vendors seen in our test

Server vendor	Percentage
Netscape	34.8
Microsoft	32.8
Apache	28.2
Lotus	2.7
Zeus	0.4
Oracle	0.2
Others	0.8

Table 4: Unconditional Compliance Results for Category One Tests

Client Site	GET(%)	HEAD(%)	Host(%)	Pass All(%)	Fail All(%)
AT&T	82.1	72.4	64.6	59.8	7.4
Australia	82.3	72.7	64.4	60.0	7.3
Chile	82.3	70.3	64.4	60.3	7.9
France	82.4	72.4	64.1	59.7	7.4
HPL	83.5	72.9	64.5	60.6	7.1
Kentucky	82.4	72.7	64.2	60.1	7.5

- The popularity 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.
- There is evidence that pornographic sites are eliminated from or downplayed in surveys of sites. Thus it is quite 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. To examine this bias, we looked at two separate sets of data:

  1. A packet trace from a large ISP; we looked for some of the frequently requested sites. Half a dozen sites that are of this ilk (running HTTP/1.1) had many more bytes transferred in response as compared to the rest of the frequently visited sites.
  2. A proxy log from a large content hosting ISP: this also showed that pornographic sites were responsible for a significant fraction of the bytes transferred.

  We did not add such sites to our tests.

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).

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 4. The results reveal that approximately 82% of the sites under study were unconditionally compliant with respect to the `GET` method; about 72% of the sites were unconditionally compliant for the `HEAD` method, while 64% passed the `Host` test. Roughly

Table 5: Breakdown of Category One Test Results (HPL Data)

	GET (%)	HEAD (%)	Host(%)
Unconditional	83.5	72.9	64.5
Missing Expected Headers	16.1	9.4	28.6
Not Compliant	0.4	17.7	6.9

60% of the sites under study were unconditionally compliant across all three tests, while just over 7% of the sites failed all three tests for conditional 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. Since the results varied only slightly depending on which client site is utilized, we use only one (the HPL site) for discussion purposes throughout the remainder of this section. We identified a few potential causes for the variation in the results by client site:

- Clusters of heterogeneous servers at a single 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.
- Site availability: a number of sites appear to have been unavailable during several of our tests; since our results are calculated based on the number of sites that responded to a test, this creates a small amount of variability in the computed percentages.

Table 5 provides a more detailed breakdown of the Category One results. As we have already shown, most servers are unconditionally compliant with respect to `GET` requests. 16.1% 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. Two servers failed the `GET` compliance test: one returned a status 500 response code rather than the expected 200 response code, while the second included an incorrectly formatted `Date` header.

Just under three-quarters 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 5 indicates that close to two-thirds of the servers fulfilled all requirements when responding to a request that did not include a `Host` header. 28.6% 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. 6.9% 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 6. These five servers accounted for 86.5% of all the servers seen in our tests.

Table 6 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

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

Server	%of Svrs	GET(%)	HEAD(%)	Host(%)	Pass All(%)	Fail All(%)
IIS/4.0	32.5	89.7	98.0	98.2	87.3	0.0
Apache/1.3	18.1	100.0	96.7	100.0	96.7	0.0
Netscape/3.5	14.0	70.8	22.2	0.0	0.0	16.9
Netscape/3.6	12.8	50.8	29.2	0.0	0.0	27.7
Apache/1.2	9.1	100.0	100.0	97.8	97.8	0.0

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, 16.9% of the Netscape/3.5 servers and 27.7% of the Netscape/3.6 servers failed all three unconditional compliance tests. This observation suggests that perhaps there is an implementation problem with these types of servers. These results also suggest that certain types of Web servers are responsible for much of the behaviour 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 6 suggests that the biggest change between these versions occurs with the `GET` requests. While 70.8% of the Netscape/3.5 servers were unconditionally compliant on this test (the remaining 29.2% were missing a `Content-Length` or `Transfer-Encoding: chunked` header), only 50.8% of the Netscape/3.6 servers passed unconditionally (the remainder were missing a length header). At this time we have no evidence to suggest why this change has occurred. Both the Netscape/3.5 and 3.6 servers did quite poorly on the `HEAD` test, with only 22.2% and 29.2% passing the unconditional compliance tests respectively. The remaining Netscape 3.5 and 3.6 servers were either missing the expected headers (19.4% and 23.1% respectively), or returned an incorrect status code (58.3% and 47.7% respectively). All of the Netscape 3.5 and 3.6 servers failed the `Host` test for unconditional compliance. 77.8% of the 3.5 servers and 86.2% of the 3.6 servers were missing both a `Date` and a length header, while 22.2% of the 3.5 servers and 13.8% of the 3.6 servers were not compliant, as they did not require a `Host` header to be present.

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 6 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 4 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 that 21.5% 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.

Table 7: Breakdown of Category Two Results

Client Site	Persistence(%)	Pipelining(%)	Range(%)	Pass All(%)	Fail All(%)
ATT	70.9	65.9	52.4	40.5	20.8
Australia	71.2	64.3	52.4	40.1	21.3
Chile	71.2	67.2	52.7	41.9	21.6
France	71.5	66.9	52.3	41.4	21.6
HPL	71.9	70.6	52.3	43.9	20.6
Kentucky	71.8	66.3	51.9	41.3	21.5

Table 8: Breakdown of Category Two Results by Server Type (HPL Data)

Server	Persistence(%)	Pipelining(%)	Range(%)	Pass All(%)	Fail All(%)
IIS/4.0	87.9	87.3	52.4	52.4	12.7
Apache/1.3	87.0	87.0	51.1	47.8	9.8
Netscape/3.5	41.1	38.4	67.2	37.5	30.6
Netscape/3.6	41.5	35.4	47.7	35.4	52.3
Apache/1.2	89.1	89.1	52.7	43.5	10.9

7.2.2 Category Two test results

Our next set of tests examined server support of persistent connections, pipelining, and range requests. These features are among the more widely discussed enhancements offered by HTTP/1.1. Note that none of these features are required. 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 explicitly send a `Connection: close` header to indicate non-persistence. The Range feature is only a MAY level requirement; servers uninterested in obeying the request would just send the complete response.

Table 7 reveals that many of the servers supported at least one of the Category Two features. Approximately 70% of the servers supported persistent connections. We expect that many of the remaining sites had persistent connections disabled by the site administrators. Slightly fewer sites allowed the client to pipeline requests. Only about half of the servers supported Range requests (properly). Overall, 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 8 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 these 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 believe that the combination of our Category One and Two tests includes most of the important and useful features of HTTP/1.1. In the remainder of this subsection we examine the number of servers that (unconditionally) supported all six features.

Table 9: Breakdown of Category One and Two Results

Client Site	Pass All Tests(%)	Fail All Tests(%)
ATT	30.3	6.8
Australia	31.1	7.2
Chile	31.4	7.8
France	30.7	7.5
HPL	31.1	7.1
Kentucky	30.6	7.5

Table 9 indicates 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.

Figure 1 provides the cumulative frequency histogram of the Category One and Two results by server type for the HPL data. For example, 28% of the Netscape-Enterprise/3.6 servers (N-E/3.6) failed all six tests (i.e., they supported 0 features). None of the Netscape/3.6 servers supported all six features. All of the Apache/1.3 servers unconditionally supported three or more features (10% supported exactly 3, the remaining 90% 4 or more). 52% of the Microsoft-IIS/4.0 servers (IIS/4.0) supported all six features, as did 48% of the Apache/1.3 servers and 42% of the Apache/1.2 servers.

Figures 2—6 indicate the number of Category One and Two features supported by each of the common servers. These figures differ from Figure 1 in that they indicate the amount of variability in our results as seen from the six client sites. For each data point we have plotted the minimum, median and maximum observed values. In most cases there is little variance in the measured results. For example, Figure 2 shows that 14.1% of Microsoft-IIS/4.0 servers supported three or fewer features (median), with a minimum result of 12.7% and a maximum of 14.6%. The cause of this variability is primarily due to some of the servers not responding to the requests from all clients. The greatest variation in the results occurs for the Netscape-Enterprise/3.5 and 3.6 servers (Figures 4 and 5). We speculate that much of this variation is due to features that do not work consistently for these servers (e.g., persistent connections and pipelining).

7.2.3 Category Three test results

In this section we tested the servers to determine whether they supported ten other suggested/recommended features of HTTP/1.1. The tests that we performed and the results are presented in Table 10.

The results in Table 10 indicate that many of these somewhat lesser known features are available on many Web servers. 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.

We also 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 (149 sites versus an average of 7.8 for the other tests), we speculate that most of these

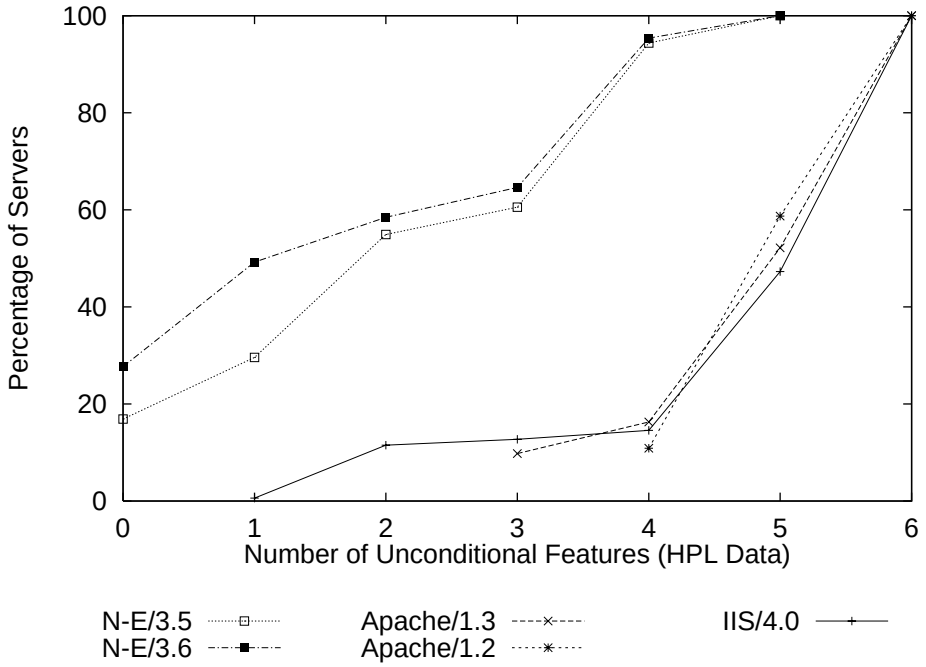

Figure 1: Cumulative frequency histogram of Category One and Two results

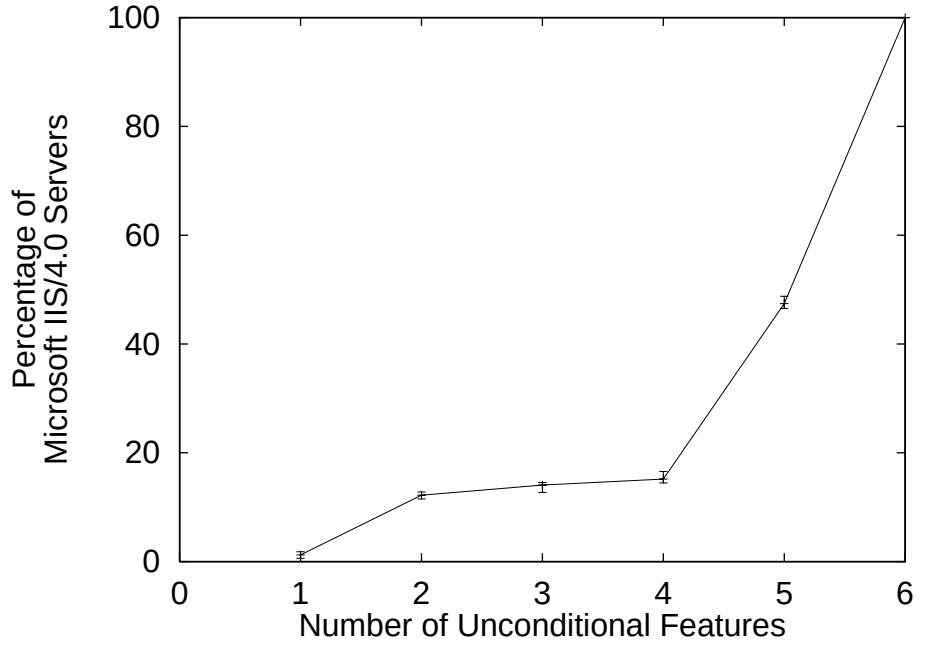

Figure 2: Number of Category One and Two features supported by IIS-4.0

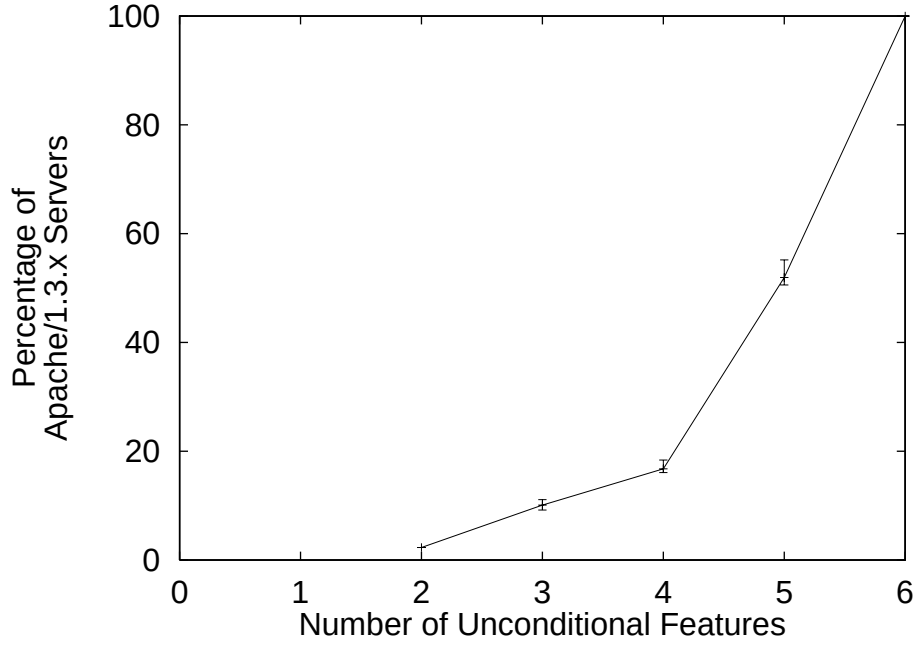

Figure 3: Number of Category One and Two features supported by Apache/1.3

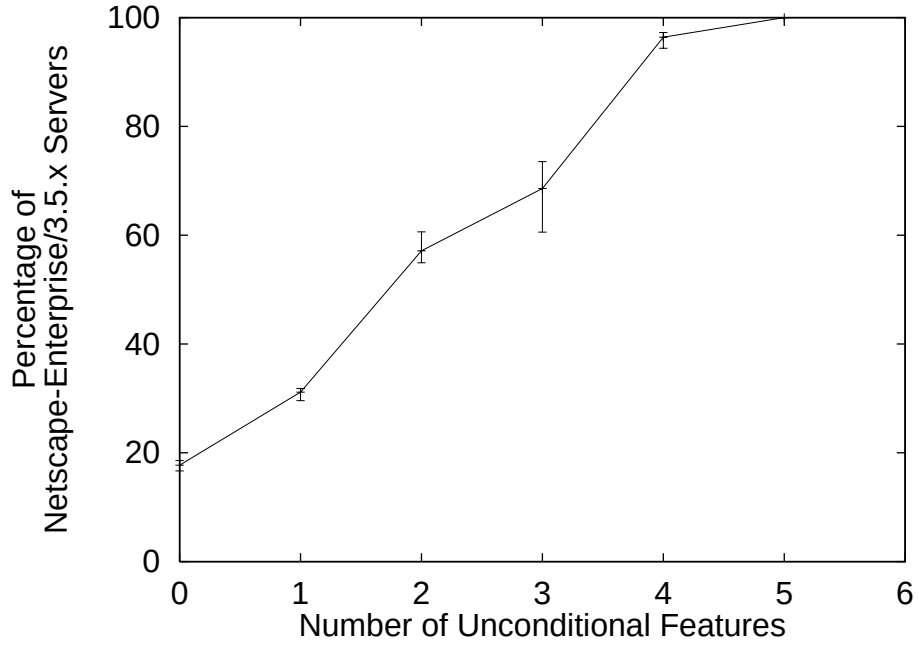

Figure 4: Number of Category One and Two features supported by Netscape-3.5

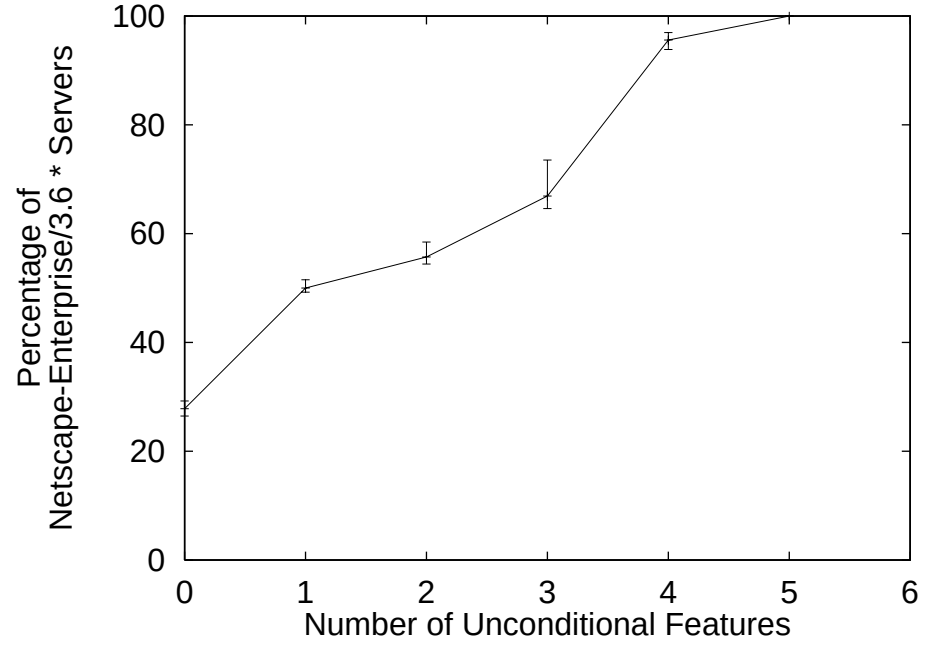

Figure 5: Number of Category One and Two features supported by Netscape-3.6

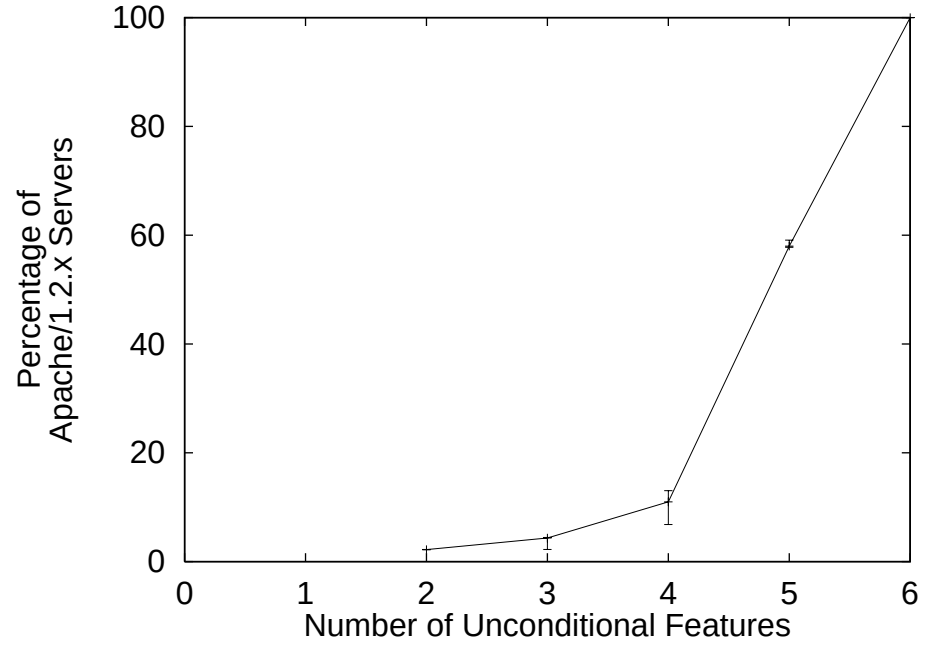

Figure 6: Number of Category One and Two features supported by Apache/1.2

Table 10: Results for Category Three Tests–HPL data

Feature	% Servers Unconditionally Compliant	% Servers Not Compliant
OPTIONS	59.8	0.8
TRACE	97.3	0.2
FOO	54.7	7.1
POST, Expect	63.2	32.0
Incorrect URL	80.5	7.1
Long URL	62.7	2.0
If-None-Match	14.8	0.8
I-U-S (1123)	41.7	57.1
I-U-S (1036)	41.7	57.1
I-U-S (ANSI C)	41.7	57.1

I-U-S == `If-Unmodified-Since` with `Date` in RFC 1123/1026/ANSI-C formats.

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

7.3 Variation within features

Even when servers implement a particular feature (for example, persistent connection and pipelining) they differed in the range of parameters. We looked at one particular feature that the specification does not go into too much detail—how many persistent requests should a compliant server handle. The specification does indicate that a server may close the transport connection at any time. In our experiments, we noticed that different sites accepted different number of requests over a persistent connection, and handled a varying number of pipelined requests before closing the connection. We found no discernible pattern here.

7.4 Impact of location of probing site

The fact that a request originates from a particular location does not in general seem to have an impact on the compliancy. However, there were a few minor variations. We suspect that this could be due to not all sites responding to all requests during the test. The multiple sites served as the redundancy mechanism of the experiment. It should be noted that one of the primary reasons to run the tests from a variety of sites in various countries is for the (ongoing) second phase of our experiment dealing with indirect requests from client to origin servers.

7.5 Speculations on 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 every other instance of a Microsoft-IIS/4.0 site tested yielded the proper `400 Bad Request` to a HTTP/1.1 request without the `Host:` header, one site (also claiming to run IIS/4.0) returned a `200 OK` response. Closer examination

and consultation showed that this site probably uses an ISAPI filter [15]–a dynamically linked library that intercepts requests—with the intention of modifying it before the core server can parse it. We believe this may be the reason for the error.

A possible reason for turning off features (such as persistent connections/pipelining or range requests) could be performance concerns. There does not appear to be any pattern in the choice of features that are enabled/disabled.

7.6 Stability of results

In the rapidly changing Web, how long will our results remain valid? Obviously, we hope that the non-compliant servers are upgraded to the more compliant ones and more features are enabled. However, one possible way our results might change is when several of the popular servers that are still running HTTP/1.0 begin to upgrade to HTTP/1.1.

8 Conclusions and future work

We have examined the compliance to the HTTP/1.1 protocol in a variety of Web sites that receive a significant portion of the traffic.

We described the first phase of our experiment where we examined a list of popular servers with a suite of static tests. Ongoing extensions include dynamic tests with the extended tool examining response headers as they arrive and generating subsequent requests based on the response (an example of this is the Entity tag mechanism). It may also be useful to examine entire user sessions rather than just single requests for the home page (or text portion thereof) of a site. Currently, we use offline analysis to check for protocol compliance. Incorporating this process into the probing mechanism would simplify the conformance checks. We intend to continue this analysis process over time, to allow us to monitor changes in the Web architecture, and to study the adoption rate of protocol modifications.

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.

We are carrying out additional tests dealing with variation within features. We plan to replay actual trace sequences (if available) or synthetically generated requests varying inter-arrival time. Our analysis process is being automated currently. We would like to do a head to head comparison of two versions of the same server to see what changed between the implementations.

Our experiment shows that even though many of the popular sites claim to run HTTP/1.1, some fail the basic compliancy requirement, while others run with many of the significant improvements turned off. The results of our experiment show that the situation on the Web must first be improved at the origin server level before we can worry about end to end improvement.

9 Acknowledgment

We thank Anja Feldmann, Roy Fielding, Jim Gettys, Richard Gray, David Kristol, Scott Lawrence, and Jeff Mogul for answers and pointers to various questions. We thank David Mosberger for answering questions related to *httperf*.

We thank Michel Beaudouin-Lafon, James Griffioen, Eduardo Krell, and Graeme Yates for readily giving us access to machines in Orsay (France), Kentucky (USA), Santiago (Chile), Nedlands (Western Australia) respectively for running our tests.

We thank Mediametrix, Netcraft, Hot100 and other sites that periodically run surveys on popular sites and present server penetration statistics.

We thank Adam Bradley for help with generating the Appendix and Jennifer Rexford for comments on an earlier draft.

A Appendix

We list the HTTP request, response, and general headers, indicating which are new in HTTP/1.1, if they are hop-by-hop or end-to-end, and where it is legal for them to occur. Additionally, we point to the sections in RFC 2616 where they are discussed.

Table 11: Key for the headers table

N	New in HTTP/1.1
H	Hop-by-hop header
H*	Pseudo-hop-by-hop header
Req	Appears in Request
Rsp	Appears in Response
Both*	Appears in any message
MUST	MUST appear in all messages
SHOULD	SHOULD appear in all messages
opt	Can appear in messages (implicit or explicit MAY)
–	MUST NOT appear
(cache)	Clause applies to caching entities
(proxy)	Clause applies to proxies/gateways
(see X)	This clause is tempered by rules in RFC2616 section X

Note: pseudo-hop-by-hop header means it is not protected by Connection header.

Table 12: HTTP/1.1 Headers

New/Hop	Header Name	Who	Request	Response
N	Accept (14.1)	Req	opt	–
N	Accept-Charset (14.2)	Req	opt	–
N	Accept-Encoding (14.3)	Req	opt	–
N	Accept-Language (14.4)	Req	opt	–
N	Accept-Ranges (14.5)	Rsp	–	opt
N	Age (14.6)	Rsp	–	MUST(cache)
	Allow (5.1.1,9.2,14.7)	Both	PUT:MAY	405:MUST OPTIONS 200:SHOULD
	Authorization (10.4.2,14.8)	Req	opt	–
N	Cache-Control (13,14.9,14.32)	Both	opt	opt
NH	Connection(8,13.5.1,14.10)	Both	opt	opt
	Content-Encoding(14.11)	Both	opt	opt
N	Content-Language(14.12)	Both	opt	opt
	Content-Length(4.4,9.2,10.2.7)	Both	SHOULD (see 4.4)	SHOULD (see 4.4)

continued on next page

Table 12: (continued)

New/Hop	Header Name	Who	Request	Response
N	Content-Location(14.14,14.30)	Both	opt	SHOULD (see 14.14) for content-negotiated responses
N	Content-MD5(14.15)	Both	opt	opt
N	Content-Range(10.2.7,10.4.17)	Both	opt 417:SHOULD	opt
	Content-Type(7.2.1,9.2,14.17)	Both	SHOULD (message with entity-body)	OPTIONS: if body, MUST
	Date(14.18)	Both	opt PUT,POST(14.18) others: SHOULD not	MUST
N	ETag(14.19)	Rsp	–	opt
N	Expect(8.2.3,10.4.18,14.20)	Req	MUST if waiting –	
	Expires(13.2.4,14.18.1,14.21)	Both	opt	opt
	From(14.22)	Req user agents: SHOULD NOT without approval	robots:SHOULD	–
N	Host(5.1.2,5.2,9,14.23)	Req	MUST	–
N	If-Match(14.24)	Req	opt	–
	If-Modified-Since(14.25)	Req	opt	–
N	If-None-Match(14.26)	Req	opt	–
N	If-Range(14.27)	Req	opt	–
N	If-Unmodified-Since(14.28)	Req	opt	–
NH	Keep-Alive(8.1.3,13.5.1,...)	Both	(this header only defined for compatibility)	
	Last-Modified(14.29)	Both	opt	SHOULD (whenever feasible)
	Location(14.30,...)	Rsp	–	201:SHOULD 300:conditional SHOULD (9.5) 301,302,303,307:SHOULD 305:implicit MUST rest:undefined
N	Max-Forwards(14.31)	Req	TRACE,OPTIONS:MAY	–
	Pragma(14.9,14.32)	Both	opt	deprecated, no tokens defined (historic)
NH*	Proxy-Authenticate (10.4.8,14.33, RFC2617:3.6)	Rsp	–	407:MUST
NH*	Proxy-Authorization (10.4.8,14.34, RFC2617:3.6)	Req	opt	–
N	Range(10.2.7,10.4.17,14.35)	Req	opt	–
	Referer(14.36)	Req	opt	–
N	Retry-After(14.37,10.5.4)	Rsp	–	503,3xx:opt
	Server(14.38)	Rsp	–	opt
NH	TE(14.39)	Req	opt	–
NH	Trailer(14.40)	Both	chunked w/trailers:SHOULD MUST NOT include: Transfer-Encoding, Content-Length, Trailer	
NH	Transfer-Encoding(14.41)	Both	opt	opt
NH	Upgrade(10.1.2,14.42)	Both	opt	101:MUST
	User-Agent(14.43)	Req	SHOULD	–
N	Vary(12.1,13.6,14.44)	Rsp	–	negotiated: SHOULD(13.6,14.44) others: opt(14.44)
N	Via(14.38,14.45)	Both	MUST(proxy)	MUST(proxy)
N	Warning (13.1.2,13.5.3,14.46)	Both	opt	when appropriate:MUST (e.g., 13.2.4,14.9.3)
	WWW-Authenticate (10.4.2,14.47,RFC2617)	Rsp	–	401:MUST

References

[1] 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*, (Toronto), May 1999. `http://www.research.att.com/library/trs/TRs/98/98.39/98.39.1.body.ps`.

[2] J. C. Mogul, "What's wrong with HTTP (and why it doesn't matter)," June 1999. Invited talk at the 1999 USENIX Technical Conference.

[3] K. Claffy, G. J. Miller, and K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone," in *Proc. INET '98*, (Geneva), July 1998. `http://www.caida.org/Papers/Inet98/`.

[4] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, HTTP Working Group, June 1999. `ftp://ftp.ietf.org/rfc2616.txt`.

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

[6] "Media Metrix." `http://mediametrix.com/`.

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

[8] "100 hot.com." `http://100hot.com/`.

[9] "1999 Fortune 500 companies," Fortune volume 139 number 8, April 26 1999.

[10] "1998 Global 500 companies," Fortune Magazine 1998.

[11] D. Mosberger and T. Jin, "httperf—a tool for measuring web server performance," in *Proceedings of the First Workshop on Internet Server Performance (WISP '98), Madison, WI*, pp. 59–67, June 1998.

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

[13] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP latency," *Computer Networks and ISDN Systems*, vol. 28, pp. 25–35, Dec. 1995.

[14] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG," in *Proc. ACM SIGCOMM*, pp. 155–166, Aug. 1997. `http://www.inria.fr/rodeo/sigcomm97/program.html`.

[15] "ISAPI callback functions." `http://msdn.microsoft.com/library/sdkdoc/iisref/isre6b1v.htm`.