Non-compliant and Proud: A Case Study of HTTP Compliance

Paul Adamczyk
padamczy@uiuc.edu

Munawar Hafiz
mhafiz@uiuc.edu

Ralph E. Johnson
johnson@cs.uiuc.edu

Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL 61801-2302

ABSTRACT

We studied the most popular websites in the US and around the world and discovered that few of them implement the HTTP standard completely. However, the servers are capable of implementing HTTP correctly; it is the configurations that are non-compliant. It is not hard to configure servers correctly, so these websites are non-compliant out of choice, not necessity.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*Standards*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol architecture*

General Terms

Measurement, Experimentation

Keywords

Compliance, Configuration, HTTP

1. INTRODUCTION

HTTP is the cornerstone of the World Wide Web. Understanding HTTP is important, because new Web technologies are built on top of it. Such technologies use HTTP as the basis and extend it in many ways to create more complex and powerful protocols. Without understanding HTTP, one runs the risk of reinventing capabilities that are already present in HTTP or completely misusing the protocol. Alternatively, understanding HTTP just by reading the standard can lead to problems, because not all of its features are widely used. This paper examines the current status of HTTP compliance.

HTTP defines eight methods for interacting with resources on the Web. Most Web users are not aware of them at all. Typical Web programmers know only of two HTTP methods: GET and POST. These methods have visual representations in Web browsers and they are the only two methods supported by HTML forms. GET is invoked when a user elects to go to a URI in the address bar, when a hyperlink is selected, or in response to pressing a button in a form. POST is invoked by pressing a button. Other HTTP methods do not have any representation in the browser.

.

This paper presents the study of the implementation and configuration of HTTP methods on the Web. We make a distinction between the *implementation* of these methods in Web servers and their *configuration* by websites that use these servers. We analyze the compliance results from both perspectives.

Our results are based on the study of the 100 most popular websites in the world, the 100 most popular websites in the USA (both according to Alexa [1]), and the websites of the top 25 computer science departments in the USA (according to US News [28]). While there is some overlap between the first two sets resulting in 176 unique websites, we selected these three groups, because they represent different flavors of popularity. The most popular websites in the world provide social networking, search, and Web portals in the most-used languages in the world. The US websites additionally include many e-commerce sites. The websites of computer science departments represent the most popular research institutions.

This paper makes the following contributions to the understanding of the Web:

- It presents a quantitative analysis of the state of HTTP compliance, 2007 A.D. This study focuses on a selection of HTTP methods, headers, and algorithms defined in the HTTP standards.
- It places the results in historical context by discussing trends that have occurred over time, both in the definitions and the implementations of the standard. Thus it provides the basis for understanding the capabilities of HTTP for the designers of new protocols built on top of it.
- It invites further studies and discussion about the state of the Web by posing questions prompted by the results. More detailed studies, targeting specific aspects of Web protocols can use these findings as a starting point.

This paper begins with an overview of the related work followed by the overview of our experiments. We present experimental results, compare them to previous studies, and discuss them from the perspective of Web servers and proxies. The results show that the Web servers implement all HTTP methods according to the standard, but few websites are configured according to the standard.

Protocol Version	Release Date	Supported Methods	Change from previous
Initial implementation	Before 1991	GET	—
Before HTTP/1.0	1991	GET, PUT, POST, HEAD, DELETE, LINK, UNLINK, *CHECKIN**, *CHECKOUT SEARCH, TEXTSEARCH, SHOWMETHOD, SPACEJUMP*	Added twelve methods
RFC 1945 (HTTP/1.0)	May, 1996	GET, HEAD, POST, PUT, DELETE, *LINK, UNLINK*	Removed six methods
RFC 2068 (HTTP/1.1) (obsoletes RFC 1945)	Nov, 1998	GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE	Removed LINK, UNLINK; Added OPTIONS, TRACE
RFC 2616 (HTTP/1.1) (obsoletes RFC 2068)	June, 1999	GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT	Added CONNECT

Table 1: HTTP methods defined in subsequent incarnations of the protocol

2. THE HTTP STANDARD

Initial implementations of HTTP had only one method, GET [4, 5]. Over time, many methods have been added and removed. Table 1 summarizes the historical development.

After an initial explosion, the number of methods has decreased. The methods have been fairly stable since the standard has been officially published in RFC (Request For Comments) 1945 [6]. The current definition, RFC 2616 [11], consists of eight methods. For the purpose of this paper, we divide these methods into three groups:

(1) Read-only methods (GET and HEAD) that do not modify the state of the server. These are the only methods that all resources are required to implement.

(2) Write methods (POST, PUT, and DELETE) that modify the data on the server.

(3) Supporting methods (OPTIONS, TRACE, and CONNECT) that were added in version 1.1. These are infrastructure methods that provide access to resource options, debugging support, and secure connections.

RFC 2616 does not define the semantics of the CONNECT method. Unlike the other methods, CONNECT is implemented by Web proxies. The client sends this method to the proxy server to initiate the creation of an end-to-end Secure Socket Layer (SSL) tunnel between the client and the end server. RFC 2817 [17] complements the HTTP standard by describing how to use the CONNECT method to create an end-to-end SSL tunnel through a proxy from the requesting client to the server.

HTTP requests have the following structure:

```
Method SP Request-URI SP HTTP-Version CRLF
Headers
CRLF
[ message-body ]
```

`Method` is the HTTP method (*e.g.,* GET). `SP` stands for space. `Request-URI` is the relative part of the website address. It is set to "/" if the address is the home page. `HTTP-Version` is HTTP/1.0 or 1.1. `CRLF` is carriage return and line feed. `Headers` is a list of headers, which may be general, request, or entity headers, *e.g.,* Content-Length, Date. They are separated by `CRLF`. `message-body` (also called *entity*) is optional and contains the payload (*i.e.,* data).

The HTTP response consists of a status line, a set of headers, and an optional message-body:

```
HTTP-Version SP Status-Code SP Reason-Phrase CRLF
Headers
CRLF
[ message-body ]
```

Some elements are identical to the request. `Status-Code` is a numerical value of the result and `Reason-Phrase` is a corresponding textual description. For example, in the popular response "404 Not Found", the status code is 400 and reason phrase is "Not Found". Following common terminology, we refer to status codes in the 400 and 500 ranges as *error codes*. `Headers` include general, response, and entity headers.

3. RELATED WORK

Several experimental studies of HTTP have been conducted in the past. These studies have provided motivation for adding new features to the standard, *e.g.,* caching [8], persistent connections [23], or pipelining [22]. The frequency of such studies decreased after HTTP/1.1 became official. Instead, more focus has been given to studying the characteristics of specific network elements, such as Web servers [3] and Web proxies [9].

The assumptions underlying the HTTP/1.1 standard have been discussed by Kristol [20] and by Mogul [21], but they do not include any experimental evaluation.

We are aware of only one experimental work that is similar to ours. PRO-COW [18] studied the compliance of Web servers with HTTP/1.1 in 1999, soon after the standard was published. This study focused on compliant implementation of the mandatory and optional HTTP features, including methods and headers. It analyzed the responses from 15 most popular websites in 1999. Our study included 10 of these websites. A follow-up PRO-COW study from 2001 [19] included 500 websites, selected based on 3 different services reporting the most popular sites. While the websites of these services still exist today, they no longer provide the lists of popular websites, so we could not use them. We discuss their results to verify our findings in Section 8. We build on

*Methods that were not included in the next protocol version are shown in italics.

their results to provide an alternative, more complete view of HTTP compliance of websites, Web servers, and other Web systems.

4. EXPERIMENTAL SETUP

We implemented an HTTP client to build, send, and receive the HTTP methods using VisualWorks 7.1 (Smalltalk). The experiments consisted of sending OPTIONS, TRACE, GET, conditional GET, and HEAD methods to the home pages of each of the tested websites. The methods that modify the state of the server (POST, PUT, DELETE) were not tested, because it is not possible to set up the same test for all the websites. See section 7.1 for more on this.

To test CONNECT, we used Fiddler [10], an HTTP debugging proxy that logs all the HTTP method traffic between the sender's computer and the Internet. All the requests go through this proxy and it logs the requests and responses. We browsed the Web pages of the tested websites from Alexa and checked the CONNECT requests and responses while communicating with a proxy that tunnels the request to an HTTPS service.

Before discussing the detailed results, it is important to address potential criticisms of our approach. If our requests looked like intruders rather than typical client methods, Web servers may have responded with errors or failed to respond on purpose, *e.g.,* as a protection from a Denial-of-Service attack. We used Web-Sniffer [29], a website that displays headers of various HTTP methods, and Fiddler to verify our findings and collected the same results. We also considered sending the same requests from multiple locations, but decided against it, following the findings of PRO-COW, which concluded that the location of the requestor has little effect on the responses.

Like most popular sites, the sites targeted by our study need many servers to handle all requests. They use load balancing and caching proxies, which means that subsequent requests may go to different machines or may not even reach the server. It would seem that our test results would be difficult to reproduce, but this is not the case. We analyzed all the results manually and collected them multiple times to verify that they are consistent. All websites generated the same response for each request.

One shortcoming of our approach is that we have tested only publicly accessible websites. It is possible that government or corporate intranets are configured to follow the standard more closely. Since intranets have tighter security and reliability requirements, studying their compliance results might provide more insights. Unfortunately, we were unable to obtain access to private intranets.

4.1 The Tested Websites

Our tests included 176 unique websites. We started with the 100 most popular websites in the world, the 100 most popular US websites, and the 25 top computer science departments (we refer to them as World sites, US sites, and CS sites in the remainder of the paper). But, due to duplication, we obtained fewer unique results. The top 100 US sites contains 32 of the same websites as the top world sites. The overlapping sites are only included in the world count. Both world sites and US sites include multiple copies of popular websites, especially Google. The top 100 world sites include 16 versions of Google that correspond to different, country-specific domains. All Google sites are set up exactly the same way, because they return the same headers in the same order in all tests; thus we count them only once. The US sites also include 3 copies of Google site, which are not counted. This means that the results for world sites tests contain 85 results, US sites tests have 66 results, and CS sites tests have 25 results.

Among the most popular sites, Yahoo and eBay also have multiple copies corresponding to different countries. However, each of these sites is configured differently (*i.e.,* each site includes different headers in response to the same HTTP method), so each copy is counted individually.

We selected our test websites based on popularity. This might seem counter-intuitive. What is popular need not be the best. However, on the Web, the popularity translates to more stringent availability and reliability requirements for the sites. Having more traffic means that most popular websites are likely to use more copies of Web servers and face more security threats, which makes them more likely to be concerned with proper use of the HTTP standard. To see how these forces relate to compliance results, we also tested the websites of the highest ranked computer science departments in the US. These websites represent a different type of popularity and have different requirements, because they see lower volume of traffic, fewer market pressures (*i.e.,* no need to make money) and consequently fewer security concerns. However, the admins of these websites are likely to be qualified and be well versed with Web server setup.

4.2 The Classification of the Results

The results are classified according to the definitions from the HTTP standard [11]. A method is *unconditionally compliant* if it implements all "must" and "should" requirements. If it implements all "must" requirements, it is *conditionally compliant*. Otherwise it is *non-compliant*. We also use the term *correctly* to mean "according to the standard".

Results can be classified in more detail based on the response of a method. *Implemented OK* means that the site is unconditionally compliant and returns all the expected data. *Not implemented* means that the site returns a standard error code (405 Method Not Allowed along with the Allow header, or 501 Not Implemented, or 404 Not Found, or 403 Forbidden which "should" include an explanation). This category is also unconditionally compliant. *Error Code in Response* means that the method is not supported/configured, but the response error code was not one of the four listed above. *Runtime Error* means that there was no reply. These two types of responses are non-compliant. Some websites responded to our tests by requesting a redirection to HTTPS. Since we were not testing HTTPS, we could not classify these responses. Consequently, the counts for some methods do not add up to 176.

This paper presents a summary of the results. More detailed results are available at `http://st.cs.uiuc.edu/~padamczy/http_tests.html`.

5. SUPPORTING METHODS

The supporting methods are analyzed first:

- OPTIONS is used to request the list of all the methods that a Web server supports. These methods are listed in the Allow header of the response.
- TRACE is used to monitor an HTTP message as it travels through the Web. The destination server sends the re-

quest message back to the sender and all the HTTP/1.1-compliant intermediate network elements (proxies, gateways) include their names in the trace by adding the Via header.

- CONNECT is used to change the connection with a Web proxy to an SSL tunnel resulting in a secure connection.

Since the OPTIONS method returns the list of methods allowed by a server, we discuss its results before the other methods. Knowing which methods are allowed by a given website makes it easier to determine whether a response with an error code is compliant or not.

5.1 Configuration Compliance Results

OPTIONS method. A standard implementation of OPTIONS (applied to resource "/") "should" return the list of methods supported by the server in the Allow header. Figure 1 summarizes the compliance results of this method.

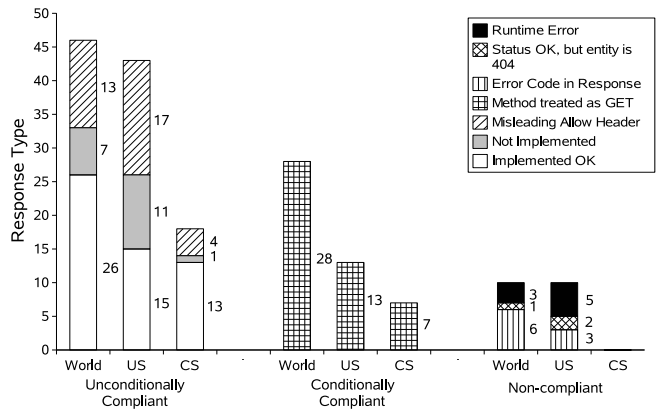

Figure 1: Compliance results for OPTIONS

54 websites implement this method correctly. The unconditionally compliant category includes 34 sites whose response contains an Allow header listing HTTP methods that are *not* allowed by the website. For example, some of these responses indicate that the website allows TRACE or HEAD, but attempts to invoke these methods result in error codes "501 Not Implemented", "405 Method Not Allowed" or "403 Forbidden". The wording in the standard makes this case compliant, but we believe that this is a mistake in the standard. The contents of the Allow header that are inconsistent with other methods make OPTIONS results misleading.

48 websites that do not respond to OPTIONS correctly treat it as if it were a GET, which, surprisingly, could be considered conditionally compliant*. As the result, none of the CS sites are non-compliant.

An interesting example of non-compliance is sending responses with the status code "200 OK" but including an empty entity named `404.html`. 3 websites return this result. We suspect that this response is meant to circumvent the default behavior of some Web browsers that do not display entities when the response contains an error code. PRO-COW [19] study observed similar behavior.

*RFC 2616, section 9.2 states: *200 response SHOULD include any header fields that indicate optional features implemented by the server and applicable to that resource (e.g., Allow)* The Allow header is not required in the response, thus a response without it is still conditionally compliant.

OPTIONS/* method. The HTTP standard states that OPTIONS applies only to the methods that are defined for the specified Request-URI. To receive the list of methods supported by the server as a whole, the Request-URI must be set to "*". But in our tests we found that the OPTIONS method is interpreted in two different ways. OPTIONS sent to the Request-URI corresponding to the home page can return two results: (1) all possible methods defined by the server, or (2) all methods defined only for the requested page. This inconsistency means that this OPTIONS test is not sufficient to measure the compliance of this method.

To perform an accurate analysis of the implementation of OPTIONS, we ran a second set of tests with the Request-URI set to "*". We refer to this version of the method as OPTIONS/* in the remainder of this paper. The results for OPTIONS/*, are shown in Figure 2.

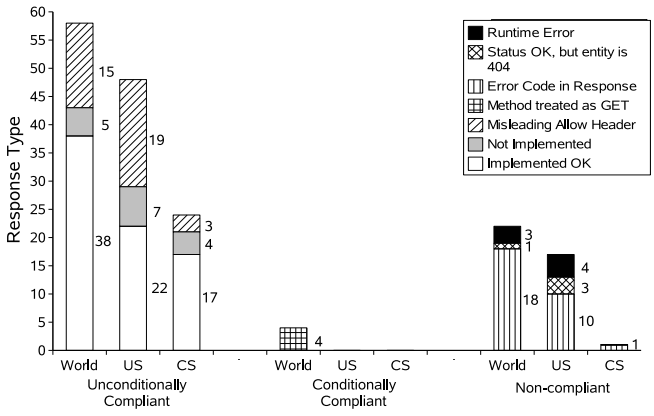

Figure 2: Compliance results for OPTIONS/*

The OPTIONS/* method has more compliant responses than OPTIONS. In total, 130 websites return unconditionally compliant responses. Only 4 responses from the World sites treat OPTIONS/* like GET, but the number of misleading Allow headers increases slightly. Since OPTIONS/* should list all the methods valid for all URIs, its results should be a superset of all the methods listed in the corresponding OPTIONS method. But some websites report fewer methods in the Allow header of OPTIONS/* than in response to OPTIONS. The number of error codes in responses increases as well; 4 websites return an empty entity named `404.html`. 69 websites return the same response for both versions of OPTIONS.

Allowed HTTP methods. The Allow header (included in OPTIONS or OPTIONS/* methods) would seem like a good indicator of which methods are configured on a website. Unfortunately, not all websites provide this information. Table 2 shows the counts of HTTP methods reported by websites. Depending on the server type, the allowed methods are listed in Allow and/or Public header. For websites where OPTIONS and OPTIONS/* return completely different results, the table reflects the contents of the response where the Allow header lists more methods.

The results show a discrepancy between most popular sites and CS department sites. Only about 2/3 of most popular sites (both in US and worldwide) return the Allow header in responses to OPTIONS or OPTIONS/* methods. In contrast, all but 2 CS department sites provide this information. All the responding websites list GET and HEAD as

	GET	HEAD	POST	PUT	DE-LETE	OPT-IONS	TRA-CE	CON-NECT	Other	Web-DAV
World	65	65	42	4	3	65	61	0	3	1
US	42	42	34	18	17	41	39	3	17	7
CS	23	23	17	2	2	23	23	1	2	1
Total	130	130	93	24	22	129	123	4	22	9

Table 2: Allowed HTTP methods count, as reported in OPTIONS

supported methods. OPTIONS, and TRACE, and POST are listed in most responses. Other HTTP methods are listed sporadically, primarily by the US sites. Section 7 expands on this observation and explains the relevance of WebDAV. Some of the sites also return methods (not shown in the table), such as INDEX, RMDIR, which were never a part of the standard.

TRACE method. A standard implementation of the TRACE method "should" return the original message in the response with the Content-Type header set to message/http. It "should" also include a Via header listing the gateways and proxies that processed this method, but this requirement does not affect the compliance results of the server – our findings related to proxies, collected from Via and other non-standard headers, are discussed in Section 9.2. The summary of the results for TRACE is shown in Figure 3.

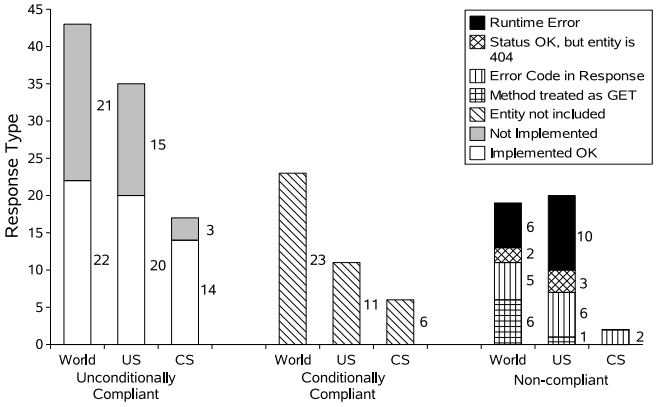

Figure 3: Compliance results for TRACE

Overall, 95 responses to TRACE are unconditionally compliant. About half of the unconditionally compliant World sites choose not to implement this method and indicate so with the expected error codes. The conditionally compliant responses fail to include the original message in the response entity (23, 11, and 6 websites respectively). Non-compliant responses return wrong error codes to report that the method is not supported (13 websites in total), treat TRACE like GET (7 websites), use the `404.html` entity as the error code (5 websites), or fail to respond (16 websites).

CONNECT method. The end servers implement Secure HTTP (HTTPS). HTTPS uses SSL to transport the HTTP methods. The entire communication between the SSL connection endpoints is encrypted.

According to RFC 2817 [17], a client can create an SSL connection in two ways. Direct SSL connection can be created if the client communicates directly with the server. Alternatively, a client can create an indirect connection by communicating with a proxy and requesting the proxy to create an SSL tunnel ending at the origin server. This tunnel setup request is initiated by the CONNECT method.

A client that is connected directly to the server can issue an optional or mandatory upgrade request (asking the server to switch to the SSL protocol) with a GET or OPTIONS method correspondingly. However, clients almost never communicate directly with the server, but instead they communicate with a proxy. Direct connections are not supported by any of the websites we tested.

The CONNECT request is sent to the proxy to request a tunnel setup. The proxy communicates with the end server, creates the tunnel, and once the tunnel is set, silently forwards the packets from the client to the end server.

We successfully sent the CONNECT method to a proxy and created an SSL tunnel with 32 out of 85 World sites. We did not see CONNECT in 53 other cases, because these end servers do not implement HTTPS. For the 66 US sites, we found that proxy servers are configured to use CONNECT in 43 cases; the remaining 23 do not implement HTTPS. All the CS sites implement HTTPS. The fact that CONNECT is not supported in many World and US websites does not mean that it is unnecessary and it does not mean that the communication is not going through a proxy. This may be because the origin servers only contain public material (*e.g.,* CNN, Washington Post, IMDB), or they contain publicly editable material (*e.g.,* Wikipedia), or they use some other encryption scheme to achieve confidentiality (*e.g.,* Livejournal clients send their data using POST, but the authentication data is encrypted; Yahoo China sends username in plaintext, but password in encrypted format).

The Allow header of OPTIONS indicates that only 4 websites claim to implement CONNECT (see Table 2). They are Earthlink (Cable provider), Statcounter (Real-time web statistics provider), Digitalpoint (Business solutions provider), and the CS Department of the University of Pennsylvania (Academic website). Surprisingly, Statcounter and Digitalpoint do not appear to implement CONNECT.

5.2 Discussion

Neither OPTIONS nor TRACE is configured sufficiently well to perform the functionality defined in the HTTP standard. In practice, only CONNECT, defined outside of RFC 2616, is configured according to its specification.

Ideally, the clients could use OPTIONS to dynamically discover what methods are supported by a resource and then select the most appropriate method to invoke. However, the fact that only few websites configure this method to return unambiguous, compliant results indicates that dynamic discovery is probably not such a great idea. The OPTIONS method appears to be used mainly to "confuse the enemy" (*i.e.,* hackers) by providing misleading results in the Allow header. It prevent hackers from learning the capabilities of the system. Browsers never send the OPTIONS method to the server. Only hackers use it. Hence it is natural to obscure the information to achieve security through obscurity.

The TRACE method appears to be more useful, because message tracing and debugging is very difficult in distributed systems, such as the Web. TRACE is reminiscent of the popular `traceroute` command. Yet, our results show that few websites handle this method correctly. We suspect that the reason for low compliance of these two methods is security.

Many websites do not use CONNECT, because security is not an important requirement for them. CONNECT is used by 65% of the US sites, but by only 37% of the World sites. This is because the top ranking websites in the world represent different regions; and typically the social networking and Web portal sites are highly ranked in all regions. These sites have low security requirements and do not use CONNECT. In contrast, many e-commerce sites are featured in the top US sites (along with the more popular social networking sites). Security is a key requirement for these websites. So their providers configure the end servers and the proxies correctly to execute the protocol steps. CONNECT is a lightweight mechanism that provides strong security guarantee. This is why the standard is universally adopted for secure communication.

6. READ-ONLY METHODS

The read-only methods retrieve the data from the server:

- GET is used to retrieve the current representation of a resource.
- HEAD is used to retrieve the headers (*i.e.,* metadata) of the corresponding GET method for a given Request-URI, but without the entity.

6.1 Configuration Compliance Results

GET method. A standard implementation of the GET method "must" contain an entity and two headers, Content-Length and Date. Our results indicate that all websites are fully compliant with this definition regardless of the Web server or the protocol version they are using.

Since all Web servers implement the GET method correctly, we also tested one of the flavors of "conditional" GET to see how well the basic caching is supported. A conditional GET is a GET method that includes a header whose name begins with "If", *e.g.,* If-None-Match. The conditional GET returns an entity only if the specified condition is true.

Conditional GET. A standard implementation of conditional GET "should" return an entity only if the resource on the server has been modified since the time specified by the condition. But all the websites generate the content of their homepage dynamically – each subsequent response has a different Date header and different Content-Length.

For this test we needed an entity that does not change as often, so we sent conditional GETs to request the favicon instead of the homepage. Favicon is a small icon logo of a website; Firefox displays it to the left of the URL in the address bar. We tested the websites that do not define a favicon by requesting a small image from their homepage. We excluded from our test 7 websites that store all their graphics in a different domain.

To test the conditional GET, we selected the If-Unmodified-Since header, because PRO-COW [18] also tested this condition[†]. This header was added to the standard improve caching. This method "must" return status code "412 Precondition Failed" if the requested resource has not been modified since the given date. We used a date 5 years ago to make sure that all favicons were modified since. The results are summarized in Figure 4.

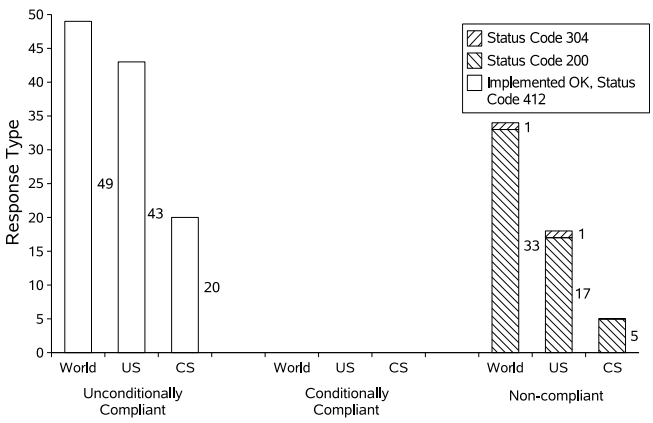

Figure 4: Compliance results for GET with If-Unmodified-Since header

The majority of websites in each category (49, 43, and 20 respectively) respond to this method correctly. There are two types of non-compliant responses. 55 websites (33, 17, and 5 for each category) include an entity that has been modified after the date we requested. Two websites respond with status code "304 Use Local Copy," which the standard defines as the expected response to a conditional GET with a different header, If-Modified-Since. There are no conditionally compliant responses.

HEAD method. A standard implementation of HEAD "must not" contain an entity and it "should" contain the same headers as the corresponding GET method. The results are summarized in Figure 5.

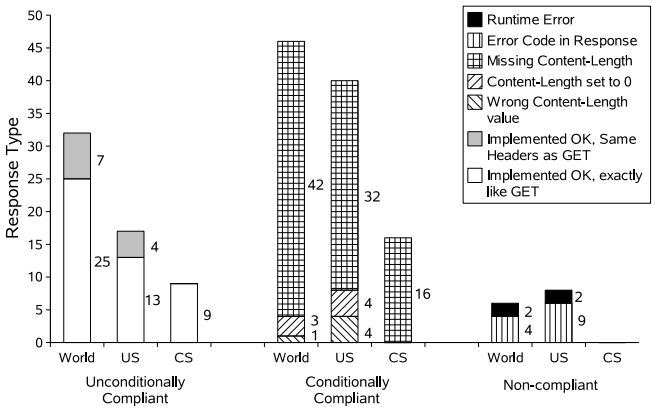

Figure 5: Compliance results for HEAD

All 3 website categories return more conditionally compliant responses (102 in total) than unconditionally compliant ones (58 in total) for this method. Most unconditionally compliant responses include exactly the same headers for GET and HEAD (25, 12, and 9 respectively). Others re-

[†]Initially we were also testing If-Modified-Since, but the results were almost identical for both conditions, so we discontinued that test. All but two US sites had the same compliance results for both tests. We did not run it for the other websites.

turn the same headers, but with slightly different values (7 World sites and 4 US sites). For example, CNN and AOL return a different value of Connection ("close" for HEAD and "keep-alive" for GET), but all the other header values are the same.

All the conditionally compliant responses have problems with one header, Content-Length, which specifies the size of the entity in the corresponding GET method. Recall that all GET responses "must" include Content-Length. Half or more of all the websites in each category (42, 32, and 16 respectively) do not include the Content-Length header[‡]. Some other responses set this header to a completely different value than the value in the corresponding GET method (these 5 websites are counted under *Wrong Content Length* in the figure). Yet other websites set it to 0 (3 World and 4 US responses).

Since HEAD is a required method, every website "must" configure it. Otherwise, it is non-compliant and counted under *Error Code in Response*.

6.2 Discussion

It is not surprising that all the tested websites configure the GET method correctly, because GET is the essence of the Web. Even more advanced versions of GET, such as conditional GET, are configured correctly by the majority of websites. This is because the conditional GET allows Web clients to take advantage of caching.

In contrast, the HEAD method is seldom configured correctly. This result is surprising for three reasons. First, HEAD is a required method. Every website is required to support GET and HEAD. Second, HEAD seems to be easy to implement by reusing the code for GET. In fact, Apache uses the same code to handle both methods. The only place in the code where their handling differs is the check whether to include the entity in the response. But even Apache Web servers sometimes (due to bugs) fail to generate unconditionally compliant responses to HEAD. The most amusing problem with HEAD is Content-Length set to 0, which suggests that the code for GET is reused for HEAD and the size of the entity is calculated dynamically before it is sent. Since HEAD has no entity, the server sets the value to 0.

Third, HEAD performs useful tasks. The metadata retrieved with HEAD is used (1) to determine the size of entity before requesting a large resource, (2) to verify that a resource still exists, and (3) to check when the resource has been last modified. Over time, the first task has become less significant, because improving network speeds have made downloading data easier. The second task will always be important. However the third task can be better accomplished with conditional GET. When the client uses HEAD to check if a resource was modified and the resource was indeed changed, it must then send a GET to retrieve it. With conditional GET, the resource is returned in one message exchange, but only if it was changed.

It would seem that HEAD should be used often, yet the test results show that it is not. Had it been used often, the website admins would have to configure it correctly. We suspect that this method is underutilized, because clients do not know about it.

[‡]We shared all our test results with Apache developers. They informed us that this is a known bug in Apache 2.x. They did not provide any insights about the reasons for other non-compliant results.

7. WRITE METHODS

The HTTP standard defines three methods for modifying resources stored on the server:

- POST is used to update the state of an existing resource.
- PUT is used to create a new resource or to replace the entire contents of an existing one.
- DELETE is used to delete a resource.

7.1 Configuration Compliance Results

Unfortunately, the write methods cannot be tested uniformly. It is not possible to set up the same test for all websites, because each website has its own policy for updating resources. To set up a comprehensive test, we would need to find unique modifiable resources on each website. To modify these resources, we would also need to gain access rights (*e.g.,* register with the website). In the end, all tests would be unique rather than uniform.

PRO-COW collected compliance results for the POST method by checking if Web servers report that POST is not allowed using the standard-defined error codes. Such a test is not convincing, because it tests the configuration of the error reporting, rather than the configuration of the method itself. The error reporting in the methods discussed in the previous two sections shows that most websites do not configure error codes correctly. This is another reason why we did not test POST responses the way PRO-COW did.

Instead, we use our test data to determine the relative popularity of write methods by analyzing the contents of the Allow header. The Allow header, returned by the OPTIONS (and OPTIONS/*) method, lists all methods supported by a website. The results collected from this header indicate that POST is configured by 71.5% of websites (See Table 2). Much fewer websites report that they support PUT or DELETE (18.5% and 16.9% respectively). Moreover, some websites do not include POST in the Allow header even if they support it. For instance, Yahoo does not list any write methods in the Allow header, but it must support at least one write method (most likely POST); otherwise the users could not send messages through Yahoo email.

7.2 Discussion

The POST method is configured by most websites, while PUT and DELETE are not. One reason for this is that the definition of write methods in the standard is too complex.

The difference between PUT and POST is a subtle one. According to the standard, a client wishing to set up the resource in its entirety would send it to the server via PUT. Alternatively, a client wishing to provide the data to the server and allowing the server to determine how to update the resource would use POST. In practice, POST is used almost always, because it places the responsibility for updating the resource on the server. If the server determines that the data submitted via POST should replace the current data, it can do so anyway. A client wishing to update an existing resource according to the semantics of PUT, needs to first obtain the current data stored in the resource (via GET), make the updates, and then send the updated representation to the server via PUT.

[**]The initial PRO-COW paper [18] does not include results for OPTIONS/*. The second paper [19] presents all results as ranges.

	OPTIONS				OPTIONS/*			
	PRO-COW	World	US	CS	PRO-COW	World	US	CS
Unconditional	59.8	54.8	65.2	72.0	26.8-32.3**	69.0	73.8	96.0
Conditional	39.4	33.3	19.7	28.0	65.0-72.4	4.8	0.0	4.0
Non-compliant	0.8	11.9	15.1	0.0	0.8-2.7	26.2	26.2	0.0
	TRACE				**HEAD**			
	PRO-COW	World	US	CS	PRO-COW	World	US	CS
Unconditional	97.3	50.6	53.0	68.0	72.9	38.1	26.2	36.0
Conditional	2.5	27.1	16.7	24.0	9.4	54.8	61.5	64.0
Non-compliant	0.2	22.3	30.3	8.0	17.7	7.1	12.3	0.0
	GET				**Conditional GET**			
	PRO-COW	World	US	CS	PRO-COW	World	US	CS
Unconditional	83.5	100.0	100.0	100.0	41.7	59.0	70.5	80.0
Conditional	16.1	0.0	0.0	0.0	1.2	0.0	0.0	0.0
Non-compliant	0.4	0.0	0.0	0.0	57.1	41.0	29.5	20.0

Table 3: Website compliance results comparison with PRO-COW (All values are in percentages)

Similarly, POST is used to delete resources. This is typically done by sending POST to a Request-URI that includes the word "delete." This is not a good interpretation of the standard, because the URI is supposed to identify a resource. Including "delete" in the URI means that the URI is used to identify the delete method on the server.

The easiest way to avoid this complexity when configuring a website is to use one method for all cases. In practice, websites define all writes (including creation and deletion of resources) with POST. Since POST has been in use for a long time, it is usually configured relatively well. The other write methods, PUT and DELETE, are not configured/allowed in practice.

It might seem that it would be beneficial to remove PUT and DELETE from the HTTP standard altogether, because they are often unused and their tasks can be accomplished with POST. However there exist HTTP extensions that use these methods. WebDAV, a standard for distributed authoring [13] is one of such extensions. It uses existing HTTP methods (including PUT and DELETE) as well as several new ones to define tasks for Web authoring: management of resource versions, access to collections of resources, and access control. WebDAV is relatively popular; 7 of the top 100 US sites support it. As a result, it is important to preserve PUT and DELETE as part of HTTP, because they are relevant in other contexts.

8. CHANGES IN COMPLIANCE

Our results show that the majority of the websites configure only the GET method correctly. All other methods are often configured incorrectly. To put these results in the proper perspective, they should be compared with prior experiments. PRO-COW project was a HTTP/1.1 compliance study that tested 13 different features (algorithms, methods, and headers) defined in HTTP/1.1. Our experiments tested the same methods as PRO-COW. Other PRO-COW tests, such as the presence of mandatory headers in methods or persistence and pipelining features were outside of the scope of our study, because they tested low-level details of the protocol, while we are focusing solely on methods that the protocol users can call. Table 3 compares the results of the 6 tests common to our studies. Each method shows the percentage of unconditionally compliant, conditionally compliant and non-compliant configurations. They are calculated from values described in the previous sections.

There are two PRO-COW papers. The first one describes results for 15 sites in 1999 [18]. The second one tested 500 websites and produced similar results in 2001 [19]. Except for OPTIONS/*, we use the results from the first paper, because they show a single value for each test.

Table 3 shows two major trends:

(1) increased compliance of the GET method,

(2) unchanged or decreased compliance of other methods.

The results of GET, which were very good in PRO-COW, are perfect in our tests. The results of the conditional GET show an improvement in unconditional compliance, but also show a lot of non-compliance. The increased compliance of the conditional GET indicates that the implementation and configuration of features that are useful to clients (*e.g.,* caching) are more likely to improve over time.

On the other hand, TRACE and HEAD show significant decrease in unconditional compliance, while TRACE shows significant increase in non-compliance as well. OPTIONS show some increase of unconditional compliance, but a larger increase in non-compliance. This result indicates that these methods are not used in practice. Since the clients do not use them, the website administrators have no incentive to configure them correctly (as indicated by mismatched headers of GET and HEAD or responses without the original request in TRACE). Some ambiguity in the standard definition (as in the case of OPTIONS) is another cause of bad implementations and configurations. Lastly, the results of OPTIONS, and especially TRACE, illustrate a change of culture prompted by the Web security concerns that occurred after the PRO-COW study.

9. COMPLIANCE OF IMPLEMENTATION

While our test messages were sent to websites, the data collected contains a lot of information about the Web servers that generated the responses and Web proxies through which our test methods were passing. This section discusses how

Web servers and Web intermediaries (proxies, caches, etc) implement the HTTP standard, based on our results.

9.1 Web Servers

To determine the compliance of Web servers, we extracted the names of Web servers from the responses. Typically the Server header identifies the type and version of the Web server. Only 10 websites that we tested do not include this header or send meaningless values (*e.g.,* "server"). Among them are Yahoo and Amazon. Table 4 shows the counts of various Web servers used by the tested websites.

Vendor	Server Type	World	US	CS
Apache	Apache/1.3	15	10	7
	Apache/2.x	14	8	12
	Apache (Unspecified)	24	19	4
	Total Apache	**53**	**37**	**23**
Microsoft IIS	IIS/5.0	5	6	1
	IIS/6.0	9	8	1
	Total IIS	**14**	**14**	**2**
Other	Netscape	1	4	0
	Sun-ONE	1	3	0
	AOL	2	1	0
	lighttpd	3	0	0
	Other	4	4	0
	Total Other	**11**	**12**	**0**
	Not Specified	7	3	0
	Total	**85**	**66**	**25**

Table 4: Web servers used by tested websites

Apache is the most popular Web server in our tests. Most non-US sites use Apache, as do almost all top CS departments. Microsoft's IIS is more popular in the US, but less known Web servers are also more often used in the US sites.

We mapped the correctly configured websites to specific Web server versions to find which servers implement the standard correctly. Most Web servers versions listed in Table 4 have a corresponding, correctly-configured website. Some examples are listed in Table 5.

Server type	Website	Description
Apache/1.3	`www.apple.com`	Apple Inc.
Apache/2.0	`www.cs.ucsd.edu`	University of California, San Diego, CS Department
Apache/2.2	`www.cs.utexas.edu`	University of Texas CS Department
IIS/5.0	`www.realtor.com`	Realtor real estate
SunONE/6.1	`www.nytimes.com`	NY Times newspaper

Table 5: Example websites that pass all tests

In addition, some websites using IIS/6.0 and Netscape/6.0 come very close to passing all our tests. MySpace, a social networking site using IIS/6.0, is unconditionally compliant with all methods except OPTIONS. The Allow header of OPTIONS lists the TRACE method, but when it is called, it returns error code "501 Not Implemented". Similarly, Comcast (a digital cable site using Netscape/6.0) responds to all methods, but it returns wrong status codes - 413 instead of 501 for the unsupported TRACE and 304 instead of 412 for GET If-Unmodified-Since. Had we tested more websites that use these two servers, we would find perfect configurations.

These results show that the type of Web server used does not influence the compliance of a website. Since we were able to find at least one example of a fully-compliant website for almost every server type, these server types *are* fully-compliant. Our test results show that well-implemented servers were configured to be non-compliant on purpose.

The PRO-COW study does not consider the relationship between Web server implementation and configuration. It counts the number of compliant results per Web server type thus implying that website configuration and Web server implementations are closely related. This is not correct. There may be many reasons why websites that use a certain version of a Web server are misconfigured more often. This may be the most popular server, so everyone wants to use it, and it is easier to find non-compliant configurations. Or the server may have many security flaws and people misconfigure it on purpose. Or the server might be very hard to configure.

It is not possible to make any claims about server compliance based on percentages of website results. However, it is possible to determine which Web servers tested by PRO-COW were implemented correctly. They show the percentage of servers that pass (and fail) all tests sorted by server types. Some Apache/1.2, Apache/1.3 and IIS/4.0 servers passed all tests, while none of Netscape/3.5 and 3.6 did. This is comparable to our results. Most Web servers implement the HTTP standard correctly.

Web server security. It might seem that if a website identifies the type and version, of the server it creates a security vulnerability, but this is not the case. Although there are few major server vendors, they offer multiple versions of the software and for each version there are multitudes of possible configuration settings.

On the other hand, revealing too much internal details can lead to problems. For example, Apache servers misconfigured for PHP have been reported to reveal the full path of the `php.exe` script handler in response to an OPTIONS method that inquired about `index.php` [7]. Another possible source of vulnerability we have observed is that the Server header, in addition to the version of the server software, includes information about other software running on the system. For example, here are two headers included in the response to GET from `alibaba.com`:

```
Server: Apache/2.0.55 (Unix) mod_AliCookie(for
        apache2.x)/1.1 aliBeacon/1.0 mod_jk/1.2.15
X-powered-by: Servlet 2.4; JBoss-4.0.2
        (build: CVSTag= JBoss_4_0_2 date=20050502
        2023)/Tomcat-5.5
```

This appears to be the default behavior of some Web servers (including Apache), but most websites wisely exclude the extra information from the Server header.

9.2 Web Intermediaries

Our tests of OPTIONS, OPTIONS/* and TRACE produced responses with spoofed Server headers. This spoofing is typically done by Web intermediaries that override the Server header with their own name. SquidCache and AkamaiGHost were the two most popular Web intermediaries in our tests, each of them overriding the Server header of 9 different websites. SquidCache is a caching proxy, while AkamaiGHost is a content delivery system. Most spoofed headers were in responses with error codes. For example, if a method is not supported, the intermediary, not the server, sends an error response. We know this because the Server header is changed to the name of the intermediary and the HTTP version is set to 1.0. That is not to say that all websites are HTTP/1.1-compliant. Some of today's most popu-

lar websites (Wikipedia, AOL, Mapquest) use HTTP/1.0. In total, 13 websites return all responses as version 1.0. Other non-standard ways of including the information about the intermediary in the response is to add new headers (*e.g.,* S, X-Server, X-cache) or use existing headers (*e.g.,* From, which is supposed to return an e-mail address).

This behavior is non-compliant. The HTTP standard states that gateways and proxies "should" identify themselves in the Via header and "must not" modify the Server header. Moreover, HTTP errata [16] clarifies that the Via header "must" be used in all methods, but this rule is rarely followed. We collected only 13 responses, from 5 different websites, that include Via. Even responses from the websites at the other end of the globe do not include Via although they are likely to pass through several intermediaries. It is possible that Via header is not used, because clients are not interested in this information. For example, Web browsers hide this information from the users. Proxies may be configured not to report their identity in the Via header to overcome the problem of finding open proxies on the Web.

Although some of the HTTP methods were defined only in HTTP/1.1, all our results include the data obtained from both HTTP/1.0 and HTTP/1.1 responses. This is because many of the 1.0 responses are 1.1-compliant. The HTTP version header is not end-to-end, which means that Web intermediaries that handle the response may change its value, but they do not modify the contents of the response. Most intermediaries we observed implement only HTTP/1.0. But HTTP/1.0 is still in use by websites as well. Some of today's most popular websites (Wikipedia, AOL, Mapquest) use it. In total, 13 websites return HTTP/1.0 in all responses.

The above results seem to imply that Web proxies still do not implement the HTTP/1.1 standard correctly. But the results of the CONNECT method show that proxies implement secure connections (defined after HTTP/1.1) quite well. Web proxies implementing CONNECT support SSL 2.0 [15], SSL 3.0 [12], and TLS 1.0 (*a.k.a.* SSL 3.1) [2]. We used Internet Explorer 6.0 on Windows XP for the tests. Internet Explorer sends a TLS 1.0 compatible ClientHello handshake request to initiate the protocol. The server returns an SSL 3.0 compatible response through the proxy and the tunnel is set up according to SSL 3.0. This is possible, because TLS 1.0 can be downgraded to SSL 3.0. The reason behind non-adoption of TLS 1.0 may be backward compatibility, because SSL 3.0 also supports SSL 2.0. We obtained the same results with Firefox 1.5.

Proxy non-compliance with HTTP is not caused by a lag in implementation, but rather it indicates that some HTTP features are not important from the proxies' perspective.

Web proxy security. Efficient HTTP tunneling with CONNECT raises a couple of issues. The tunnels between clients and servers are not necessarily created end-to-end. If the end server is behind a firewall, then the HTTP tunnel is created between the client and the gateway acting as the access point of the firewall. The data is then available in plaintext at the gateway.

Another issue with tunneling is that the proxies have no way to monitor the data passing through the tunnel [25] [27]. Almost any TCP-based protocol can be forwarded through the proxy service.

Security vulnerabilities of TRACE have an indirect effect on proxies. Websites that enable TRACE can become targets of 'cross site tracing' attack that could reveal user information [26]. A script on the client machine can forward a response to a TRACE method with Cookie headers to a malicious server [14]. To prevent this, many servers return the length of the TRACE request instead of the whole request to the client. We suspect that this vulnerability is the reason why proxies do not include the Via header.

10. ANALYSIS OF THE RESULTS

By now, the key result of our experiments should be obvious: HTTP methods do not behave according to the standard. The PRO-COW paper [18] concludes: "The results of our experiment show that the situation on the Web must first be improved at the origin server before we can worry about end-to-end improvements." Yet 6 years after the PRO-COW experiments, our tests show that though the results have not changed much, the Web is doing just fine. There are several possible explanations of this phenomenon.

Perhaps website admins are incompetent. The websites we tested generate the most traffic and probably face the most security threats. For many of these websites (Google is the best example), the Web presence is their entire business. To ensure availability, they must be set up well. Therefore this hypothesis is not true.

Perhaps it is difficult to set up Web servers, because they come with bad defaults. While this hypothesis seems plausible, it is not true either. We installed two versions of Apache to see what their default configurations are. In Apache 1.3 for RedHat, all the methods (even PUT and DELETE) are configured correctly out of the box. The same is true for Apache 2.0 for Windows XP. A website is operational in minutes after the Web server software is downloaded. More effort is required to disable correct configurations.

Perhaps bad website configuration is done on purpose. As noted in the previous paragraph, the Web servers have standard-compliant default settings, yet the compliance of configurations varies. The non-standard setup of less popular methods could be a way to achieve security through obfuscation. Security vulnerabilities mentioned in the previous section support this hypothesis.

Perhaps it is not important to be compliant with the entire HTTP standard. Even the most popular sites seem to be satisfied that the key features are working. If GET or POST method were to suddenly stop working, they would need to be fixed immediately. The other methods are not used often enough to demand proper configuration.

Perhaps most Web traffic is handled by HTTP-agnostic systems, such as content delivery systems. Such Web intermediaries do not operate at the HTTP layer, but use different protocols in their communication with the servers. When they produce HTTP responses on behalf of the servers, such responses are seldom HTTP-compliant.

There are many other explanations for the low compliance results, but the last three reasons - security concerns, the limited use of most HTTP methods, and HTTP-agnostic systems - shed some light on this problem. There is a disconnect between the theory (HTTP standard) and practice (system compliance) in the Web systems. In theory, HTTP is a simple protocol for the Web. It was designed so that it can be extended with more specialized protocols. WebDAV is a good example of this. In practice, a small subset of HTTP is used. Web systems built on top of the simplified HTTP are adding capabilities that already exist in the full-fledged HTTP, but not in the commonly used subset.

11. CONCLUSION AND FUTURE WORK

Our study of different types of popular websites shows that most of them are not compliant with HTTP. While Web servers implement the standard very well, few of the websites are configured correctly. This is a continuing trend, yet it has not affected the growth of the Web.

Our results provide experimental evidence for the debates about the future of the Web. More specific experiments are needed to address them fully, but some debates, *e.g.,* REST vs. SOAP [24], lightweight vs. standards-based security for Web services, can benefit from our results.

The relationship between theory and practice of building Web systems is very complex. Studying only one small aspect of it is not likely to produce comprehensive results, but it is an important step toward a better understanding of the Web. We provide three hypotheses explaining the HTTP non-compliance, but obtaining a clearer picture requires more studies, including surveys of website admins and server implementers. We hope to have raised enough interesting questions for others to join in this conversation.

12. REFERENCES

[1] Alexa. `http://www.alexa.com`.

[2] Christopher Allen and Tim Dierks. The TLS protocol — version 1.0. Internet proposed standard RFC 2246, January 1999.

[3] Martin Arlitt and Carey Williamson. Understanding Web server configuration issues. *Software Practice and Experience*, 34(2):163–186, February 2004.

[4] Tim Berners-Lee. The original HTTP as defined in 1991. `http://www.w3.org/Protocols/HTTP/AsImplemented.html`, 1991. W3C webpage.

[5] Tim Berners-Lee. Is there a paper which describes the WWW protocol. `http://lists.w3.org/Archives/Public/www-talk/1992JanFeb/0000.html`, Jan 9 1992. WWW-talk mailing list.

[6] Tim Berners-Lee, Roy T. Fielding, and Henrik Frystyk Nielsen. Hypertext Transfer Protocol — HTTP/1.0. Internet informational RFC 1945, May 1996.

[7] Bugtraq ID 4057. Apache 2 for Windows OPTIONS request path disclosure vulnerability. `http://www.securityfocus.com/bid/4057/info`, Feb 2002.

[8] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of change and other metrics: A live study of the World Wide Web. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[9] Bradley M. Duska, David Marwood, and Michael J. Freeley. The measured access characteristics of World-Wide-Web client proxy caches. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[10] Fiddler. `http://www.fiddlertool.com/fiddler/`.

[11] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. Internet proposed standard RFC 2616, June 1999.

[12] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol — version 3.0. Internet Draft, Transport Layer Security Working Group, November 1996.

[13] Yaron Y. Goland, E. James Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring — WebDAV. Internet proposed standard RFC 2518, February 1999.

[14] Jeremiah Grossman. Cross-Site Tracing (XST). `www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf`, Jan 2003.

[15] Kipp E. B. Hickman and Taher ElGamal. The SSL protocol. RFC draft, Netscape Communications Corp., June 1995.

[16] HTTP/1.1 specification errata. `http://skrb.org/ietf/http_errata.html`, Oct 2004.

[17] Rohit Khare and Scott D. Lawrence. Upgrading to TLS within HTTP/1.1. Internet proposed standard RFC 2817, May 2000.

[18] Balachander Krishnamurthy and Martin Arlitt. PRO-COW: Protocol compliance on the Web. Technical Report 990803-05-TM, HP Labs, 1999.

[19] Balachander Krishnamurthy and Martin Arlitt. PRO-COW: Protocol compliance on the Web — A longitudinal study. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS-01)*, San Francisco, CA, 2001.

[20] David M. Kristol. HTTP Cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology (TOIT)*, 1(2):151–198, November 2001.

[21] Jeffrey C. Mogul. Clarifying the fundamentals of HTTP. *Software Practice and Experience*, 34(2):103–134, February 2004.

[22] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, Cannes, France, 1997.

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

[24] Paul Prescod. Roots of the REST/SOAP debate. In *EML 2002: Proceedings of the Extreme Markup Languages 2002 conference*, Montreal, Canada, 2002.

[25] US-CERT Vulnerability Note Vu150227. HTTP proxy default configurations allow arbitrary TCP connections. `https://www.kb.cert.org/vuls/id/150227`, Feb 2002.

[26] US-CERT Vulnerability Note VU867593. Multiple vendors' Web servers enable HTTP TRACE method by default. `https://www.kb.cert.org/vuls/id/867593`, Jan 2003.

[27] US-CERT Vulnerability Note VU868219. Multiple vendors' HTTP contentvirus scanners do not check data tunneled via HTTP CONNECT method. `http://www.kb.cert.org/vuls/id/868219`, Feb 2002.

[28] US News and World Report. `http://http://www.usnews.com/usnews/home.htm`.

[29] Web-sniffer v1.0.24. `http://web-sniffer.net/`.