HTTP Working Group
INTERNET DRAFT

David M. Kristol
AT&T Bell Laboratories
Lou Montulli
Netscape Communications

<draft-kristol-http-state-mgmt-00.ps>
February 19, 1996

Expires August 19, 1996

Proposed HTTP State Management Mechanism

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as ‘‘work in progress.’’

To learn the current status of any Internet-Draft, please check the ‘‘1id-abstracts.txt’’ listing contained in the Internet- Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

This is authors’ draft 2.4.

1. ABSTRACT

HTTP, the protocol that underpins the World-Wide Web (WWW), is stateless. That is, each request stands on its own; origin servers don’t need to remember what happened with previous requests to service a new one. Statelessness is a mixed blessing, because there are potential WWW applications, like ‘‘shopping baskets’’ and library browsing, for which the history of a user’s actions is useful or essential.

This proposal outlines a way to introduce state into HTTP. New request and response headers, `Cookie` and `Set-Cookie`, carry the state back and forth, thus relieving the origin server from needing to keep an extensive per-user or per-connection database. The changes required to user agents, origin servers, and proxy servers to support state management are modest.

2. TERMINOLOGY

The terms *user agent*, *client*, *server*, *proxy*, and *origin server* have the same meaning as in the HTTP/1.0 specification.

Because it was used in Netscape’s original implementation of state management, we will use the term *cookie* to refer to the state information that passes between an origin server and user agent, and that gets stored by the user agent.

3. STATE AND SESSIONS

This proposal outlines how to introduce state into HTTP, the protocol that underpins the World-Wide Web (WWW). At present, HTTP is stateless: a WWW origin server obtains everything it needs to know about a request from the request itself. After it processes the request, the origin server can ‘‘forget’’ the transaction.

What do we mean by ‘‘state?’’ ‘‘State’’ implies some relation between one request to an origin server and previous ones made by the same user agent to the same origin server. If the sequence of these requests is considered as a whole, they can be thought of as a ‘‘session.’’

Koen Holtman identified these dimensions for the ‘‘solution space’’ of stateful dialogs:

- simplicity of implementation
- simplicity of use
- time of general availability when standardized
- downward compatibility
- reliability
- amount of privacy protection
- maximum complexity of stateful dialogs supported
- amount of cache control possible
- risks when used with non-conforming caches

The paradigm we have in mind obtains the same effect as if a user agent connected to an origin server, carried out many transactions at the user’s direction, then disconnected. Two example applications we have in mind are a ‘‘shopping cart,’’ where the state information comprises what the user has bought, and a magazine browsing system, where the state information comprises the set of journals and articles the user has looked at already. Note some of the key points in the session paradigm:

1. The session has a beginning and an end.
2. The session is relatively short-lived.
3. Either the user agent or the origin server may terminate a session. There is no defined mechanism to inform the origin server that the user agent has terminated a session.
4. State is a property of the session. The user agent can receive, store, and send cookies that reflect this state.

4. PROPOSAL OUTLINE

The proposal we outline here defines a way for an origin server to send state information to the user agent, and for the user agent to return the state information to the origin server. The goal of the proposal is to have a minimal impact on HTTP and user agents. Only origin servers that need to maintain sessions would suffer any significant impact, and that impact can largely be confined to Common Gateway Interface (CGI) programs, unless the server provides more sophisticated state management support. (See **Implementation Considerations**, below.)

4.1 Syntax: General

The two state management headers, `Set-Cookie` and `Cookie`, have common syntactic properties involving attribute-value pairs. The following uses the notation and tokens `ALPHA` (lower and upper case letters), `DIGIT` (decimal digits), and `word` (informally, a `"`-quoted string or sequence of non-special, non-white space characters) from the HTTP/1.1 specification to describe their syntax.

```
<av-pairs>      =       <av-pair> *(";" <av-pair>)
<av-pair>       =       <attr> 0,1*("=" <value>) ; optional <value>
<attr>          =       ALPHA *(ALPHA | DIGIT)
<value>         =       word
```

Attributes are case-insensitive. White space is permitted between tokens. Note that while the above syntax description shows `<value>` as optional, most `<attr>`s require them.

4.2 Origin Server Role

4.2.1 General The origin server initiates a session, if it so desires. (Note that ‘‘session’’ here is a logical connection, not a physical one. Don’t confuse these logical sessions with various ‘‘keepalive’’ proposals for physical sessions.) To initiate a session, the origin server returns an extra response header to the client,

`Set-Cookie.` (The details follow later.)

A user agent returns a `Cookie` request header (see below) to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It may send back to the client a `Set-Cookie` response header with the same or different information, or it may send no `Set-Cookie` header at all. The origin server effectively ends a session by sending back a `Set-Cookie` header that has a null value.

An origin server must be cognizant of the effect of possible caching by other agents of its responses that have a `Set-Cookie` header. Generally a document that has associated state information should not be cached: the cached resource or `Set-Cookie` may be specific to a particular user agent. The origin server must explicitly notify upstream agents not to cache them. To inhibit caching, the origin server should use one of the standard mechanisms that inhibit caching, such as `Cache-control: no-cache` or `Expires:` *<yesterday>*.

An origin server may include multiple `Set-Cookie` headers in a response.
QUESTIONS:

1. Do we need to deal with RFC 822 header folding in that event? That is, multiple `Set-Cookie`s could be folded by some intervening gateway into a single header.
2. How do multiple `Set-Cookie`s play with the `Cache-Control: private=Set-Cookie` stuff? Are all `Set-Cookie`s private? Otherwise, which one?

4.2.2 `Set-Cookie` *Syntax* The syntax for the `Set-Cookie` response header is `Set-Cookie:`, followed by attribute-value pairs. The syntax for attribute-value pairs was shown above. The specific attributes and the semantics of their values follows. The *NAME=VALUE* attribute-value pair must come first. The others, if present, can occur in any order.

NAME=VALUE
: Required. The name of the state information (‘‘cookie’’) is *NAME*, and its value is *VALUE*.

  The *VALUE* is opaque and may be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. ‘‘Opaque’’ implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone that examines the `Set-Cookie` header.

`Expires=`*date*
: Optional. The cookie information expires after *date*, which must be in RFC 1123 format. Because of RFC 1123’s embedded spaces, *date* must be quoted. The timezone **must** be GMT.
  QUESTION: How concerned should we be about clock differences between the origin server and user agent?
  NOTE The Netscape proposal does not require quotes around the expiration date! Because of embedded spaces, I do.

`Domain=`*domain*
: Optional. The `Domain` attribute specifies the host and domain name for which the cookie is valid.

`Path=`*path*
: Optional. The `Path` attribute specifies the subset of URLs to which this cookie applies.

`Secure`
: Optional. The `Secure` attribute (with no value) directs the user agent only to use (unspecified) secure means to contact the origin server whenever it sends back this cookie.

QUESTIONS:

1. Is the order of a-v pairs required to be what’s shown? Must NAME=VALUE come first? Otherwise, what happens if someone chooses a cookie named `domain` or `path`?

4.3 User Agent Role

4.3.1 Interpreting `Set-Cookie` The user agent keeps separate track of state information that arrives via `Set-Cookie` response headers from each origin server (as distinguished by name or IP address and port). The user agent applies these defaults for optional attributes that are missing:

`Expires` Defaults to the end of the session, that is, until the user exits the window or user agent.

`Path` Defaults to the path of the request URL that generated the `Set-Cookie` response.
QUESTION: Don't we really mean the prefix? For example, if the URL were `/a/b/x.html`, the `Path` attribute would default to `/a/b`, I presume.

`Domain` Defaults to the host name of the server that generated the response.
QUESTION: exactly how is this determined? From the URL? (May not be completely specified.)

`Secure` If absent, the user agent may send the cookie over an insecure channel.

4.3.2 Cookie Management If a user agent receives a `Set-Cookie` response header with the same `NAME` and `Path` attributes as a pre-existing cookie, the new one supercedes the old. However, if the `Set-Cookie` has an expiration time in the past, the (old and new) cookie is discarded. Otherwise cookies accumulate until they expire (resources permitting), at which time they are discarded.

Because user agents have finite space in which to store cookies, they may also discard older cookies to make space for newer ones, using, for example, a least-recently-used algorithm, along with constraints on the maximum number of cookies that each origin server may set.

Privacy considerations dictate that the user have considerable control over cookie management. The **PRIVACY** section contains more information.

4.3.3 Sending Cookies to the Origin Server When it sends a request to an origin server, the user agent sends a `Cookie` request header to the origin server if it has cookies that are applicable to the request, based on

- the origin server's fully qualified domain name;
- the request URL;
- the current time.

The syntax for the header is `Cookie:` followed by a semi-colon-separated list of the *NAME=VALUE* pairs for the applicable cookies.

These rules apply to choosing applicable cookies from among all the cookies the user agent has.

Domain Selection
: The `Domain` attribute of the cookie must match either

  1. the origin server's fully qualified domain name exactly; or
  2. the tail of the origin server's fully qualified domain name **and** `Domain` must begin with a dot and contain at least three dots. **Note:** I know this is a lie. We need to be real specific.

Path Selection
: The `Path` attribute of the cookie must match a prefix of the request URL.

Expires Selection
: Cookies that have expired should have been discarded and are thus not forwarded to an origin server.

4.3.4 Sending Cookies in Unverifiable Transactions A transaction is *verifiable* if the user has the option to review the Request-URI prior to its use in the transaction. A transaction is *unverifiable* if the user does not have that option. User agents typically use unverifiable transactions when they get inlined entities and when they resolve redirection (3xx) responses. Typically the transaction that the user initiates is verifiable, and that transaction directly or indirectly causes the user agent to make unverifiable transactions.

The following rule eases the implementation of user agent mechanisms that give the user control over sessions (see **PRIVACY** section) by restricting the number of cases where control is needed. The rule prevents malicious service authors from using unverifiable transactions to induce a user agent to start or continue a session with a server in a different domain. The starting or continuation of such sessions could be contrary to the privacy expectations of the user, and could also be a security problem.

When it makes an unverifiable transaction, a user agent must only enable the cookie functionality, that is the sending of `Cookie` request headers and the processing of any `Set-Cookie` response headers, if a cookie with a domain attribute D was sent or received in its origin transaction, such that the host name in the Request-URI of the unverifiable transaction is exactly equal to D, or of the form "H.D", where H is a string possibly containing dots.

User agents may offer configurable options that allow the user agent, or any autonomous programs that the user agent executes, to ignore the above rule, so long as these override options default to ‘‘off.’’

4.4 Caching Proxy Role

One reason for separating state information from both a URL and document content is to facilitate the scaling that caching permits. To support cookies, a caching proxy must obey these rules already in the HTTP specification:

- Honor requests from the cache, if possible, based on cache validity rules.
- Pass along a `Cookie` request header in any request that the proxy must make of another server, and return the response (including any `Set-Cookie` response header) to the client.
- Cache the received response subject to the control of the usual headers, such as `Expires`, `Cache-control: no-cache`, and `Cache-control: private`,

5. EXAMPLES

5.1 Example 1

Most detail of request and response headers has been omitted. Assume the user agent has no stored cookies.

1. User Agent → Server

   ```
   POST /acme/login HTTP/1.0
   [form data]
   ```

   User identifies self.

2. Server → User Agent

   ```
   HTTP/1.0 200 OK
   Set-Cookie: Customer="WILE_E_COYOTE"; Path="/acme"
   ```

   Cookie reflects user’s identity.

3. User Agent → Server

   ```
   POST /acme/pickitem HTTP/1.0
   Cookie: Customer="WILE_E_COYOTE"
   [form data]
   ```

   User selects an item for ‘‘shopping basket.’’

4. Server → User Agent

   ```
   HTTP/1.0 200 OK
   Set-Cookie: Part_Number="Rocket_Launcher_0001"; Path="/acme"
   ```

   Shopping basket contains an item.

5. User Agent → Server

   ```
   POST /acme/shipping HTTP/1.0
   Cookie: Customer="WILE_E_COYOTE"; Part_Number="Rocket_Launcher_0001"
   ```

   User selects shipping method from form.

6. Server → User Agent

   ```
   HTTP/1.0 200 OK
   Set-Cookie: Shipping="FedEx"; Path="/acme"
   ```

   New cookie reflects shipping method.

7. User Agent → Server

   ```
   POST /acme/process HTTP/1.0
   Cookie: Customer="WILE_E_COYOTE"; Part_Number="Rocket_Launcher_0001";
                Shipping="FedEx"
   [form data]
   ```

   User chooses to process order.

8. Server → User Agent

   ```
   HTTP/1.0 200 OK
   ```

   Transaction is complete.

The user agent makes a series of requests on the origin server, after each of which it receives a new cookie. All the cookies have the same `Path` attribute and (default) domain. Because the request URLs all have `/acme` as a prefix, and that matches the `Path` attribute, each request contains all the cookies received so far.

5.2 Example 2

This example illustrates the effect of the `Path` attribute. All detail of request and response headers has been omitted. Assume the user agent has no stored cookies.

Imagine the user agent has received, in response to earlier requests, the response headers

```
Set-Cookie: Part_Number="Rocket_Launcher_0001"; Path="/acme"
```

and

```
Set-Cookie: Part_Number="Riding_Rocket_0023"; Path="/acme/ammo"
```

A subsequent request by the user agent to the (same) server for URLs of the form `/acme/ammo/...` would include the following request header:

```
Cookie: Part_Number="Riding_Rocket_0023";
          Part_Number="Rocket_Launcher_0001"
```

Note that the NAME=VALUE pair for the cookie with the more specific `Path` attribute comes before the one with the less specific `Path` attribute. Further note that the same cookie name appears more than once.

A subsequent request by the user agent to the (same) server for URLs of the form `/acme/...` (assuming `...` did not have the prefix `ammo`) would include the following request header:

```
Cookie: Part_Number="Rocket_Launcher_0001"
```

Here, the second cookie's `Path` attribute `/acme/ammo` is not a prefix of the request URL, so the cookie does not get forwarded to the server.

6. IMPLEMENTATION CONSIDERATIONS

Here we speculate on likely or desirable details for an origin server that implements state management.

6.1 `Set-Cookie` Content

An origin server's content should probably be divided into disjoint application areas, some of which require the use of state information. The application areas can be distinguished by their request URLs. The `Set-Cookie` header can incorporate information about the application areas by setting the `Path` attribute for each one.

The session information can obviously be clear or encoded text that describes state. However, if it grows too large, it can become unwieldy. Therefore, an implementor might choose for the session information to be a key into a server-side database. Of course, using a database creates some problems that the state management proposal was meant to avoid, namely:

1. keeping real state on the server side;
2. how and when to garbage-collect the database entry, in case the user agent terminates the session by, for example, exiting.

6.2 Stateless Pages

Caching is a good thing for the scalability of WWW. Therefore it's important to reduce the number of documents that have state embedded in them inherently. For example, if a shopping-basket-style application always displayed a user's current basket contents on each page, those pages could not be cached, because each user's basket's contents would be different. On the other hand, if each page contained just a link that allowed the user to ''Look at My Shopping Basket,'' the page could be cached.

6.3 Implementation Limits

Practical user agent implementations have limits on the number and size of cookies that they can store. In general, user agents' cookie support should have no **fixed** limits. Furthermore, they should provide the following **minimum** capabilities:

- at least 300 cookies
- at least 4096 bytes per cookie (as measured by size of `Set-Cookie` header)
- at least 20 cookies per unique host or domain name

The information in a `Set-Cookie` response header must be retained in its entirety. If for some reason there is inadequate space to store the cookie, it must be discarded, not truncated.

6.3.1 Denial of Service Attacks User agents may choose to set an upper bound on the number of cookies to be stored from a given host or domain name or on the size of the cookie information. Otherwise a malicious server could attempt to flood a user agent with many cookies, or large cookies, on successive responses, which would force out cookies the user agent had received from other servers. However, the minima specified above must still be supported.

7. PRIVACY

An origin server could create a `Set-Cookie` header to track the path of a user through the server. Users may object to this behavior as an intrusive accumulation of information, even if their identity is not evident. (Identity might become evident if a user subsequently fills out a form that contains identifying information.) The state management proposal therefore requires that a user agent give the user control over such a possible intrusion, although the choice of mechanisms is unspecified. However, the control mechanisms provided shall at least allow the user

- to completely disable the sending of cookies.
- to determine whether a stateful session is in progress.

- to control the saving of a cookie on the basis of the cookie's Domain attribute.

Such control could be provided by, for example, mechanisms

- to notify the user when the user agent is about to send a cookie to the origin server, offering the option not to begin a session.
- to display a visual indication that a stateful session is in progress.
- to let the user decide which cookies, if any, should be saved when the user concludes a window or user agent session.
- to let the user examine the contents of a cookie at any time.

A user agent usually begins execution with no remembered state information. It should be possible to configure a user agent never to send `Cookie` headers, in which case it can never sustain state with an origin server. (The user agent would then behave like one that is unaware of how to handle `Set-Cookie` response headers.)

When the user agent terminates execution, it should let the user discard all state information. Alternatively, the user agent may ask the user whether state information should be retained; the default should be ''no.'' Retained state information would then be restored when the user agent begins execution again.

NOTE: User agents should probably be cautious about using files to store cookies long-term. If a user runs more than one instance of the user agent, the cookies could be comingled or otherwise messed up.

8. SECURITY CONSIDERATIONS

The information in the `Set-Cookie` and `Cookie` headers is unprotected. Two consequences are:

1. Any sensitive information that is conveyed in them is exposed to intruders.
2. A malicious intermediary could alter the headers as they travel in either direction, with unpredictable results.

These facts imply that information of a personal and/or financial nature should only be sent over a secure channel. For less sensitive information, or when the content of the header is a database key, an origin server should be vigilant to prevent a bad `Cookie` value from causing it to fail.

9. OTHER, SIMILAR, PROPOSALS

Three other proposals have been made to accomplish similar goals. This proposal is an amalgam of Kristol's State-Info proposal and Netscape's Cookie proposal.

Brian Behlendorf proposed a `Session-ID` header that would be user-agent-initiated and could be used by an origin server to track ''clickstreams.'' It would not carry any origin-server-defined state, however.

10. ACKNOWLEDGEMENTS

This document really represents the collective efforts of the following people, in addition to the authors: Roy Fielding, Marc Hedlund, Koen Holtman, Shel Kaphan, Rohit Khare.

11. AUTHORS' ADDRESSES

David M. Kristol
AT&T Bell Laboratories
600 Mountain Ave. Room 2A-227
Murray Hill, NJ 07974

Phone: (908) 582-2250
FAX: (908) 582-5809
Email: dmk@research.att.com

Lou Montulli
Netscape Communications Corp.
501 E. Middlefield Rd.
Mountain View, CA 94043

Phone: (908) 528-2600
Email: montulli@netscape.com