1 Overview

This document specifies Version 1.3.0 of the “Cookie” Protocol, which runs over TCP. This toy client-server protocol is intended only as a vehicle for learning about protocols; it does not serve any other purpose.

In this protocol the server always sends first. After connecting, the server sends a greeting message that includes a timestamp and a cookie. (Unlike in HTTP, here a “cookie” is a string of at least 20 ASCII characters, each of which is a decimal digit.) Each cookie the server sends the client acts as a “receipt.” The server logs all cookies it issues. The idea is that a client can later present a cookie as evidence that it successfully completed (some part of) a transaction. After the server sends the greeting message, the client sends a request message. What happens next depends on the type of request; there are two types of transactions.

In a Type 0 transaction, after receiving the greeting message the client sends a request message containing a cookie whose value is derived from the cookie the server placed in the greeting message. Thus, the client must prove that it was able to receive and correctly parse the server’s greeting message. In response, the server sends an outcome message. If the client’s request message was correct, the outcome message indicates the result “OK” and contains a second cookie. Otherwise, the outcome message indicates what error occurred.

In a Type 1 transaction, the client sends a different request message, which (in addition to a cookie derived from the one in the greeting message) contains an IP address and port number on which the client is listening for TCP connections. The server then opens a connection to the given address and port; in what follows we call this the “secondary” connection (the other is called the “original” connection). If the server is able to open the secondary connection, the server sends an outcome message on the original connection, indicating “OK” and containing a second cookie. The client then sends a cookie message containing a value derived from that second cookie, on the secondary connection. The server then sends back a third cookie message containing a fresh cookie on the secondary connection, and finally the client returns the same cookie message, unmodified, on the original connection. After receiving this cookie message on the original connection, the server sends back (on the original connection) an outcome message, indicating the overall result of the Type 1 transaction; this message includes a fourth and final cookie if the outcome was “OK”. Thus, successful completion of the Type 1 transaction gives the client four cookies.
2 Message Formats

All messages are encoded as ASCII text strings. (Note: if you are using Java, the encoding you want to use is “ISO-8859-1”. Correctly-formatted messages of this protocol contain only characters that can be encoded in one byte, whose high-order bit is 0.

The end of every message is marked by the two-character delimiter sequence \n\n, that is, two bytes containing the value 0x0A (decimal 10). Note that in the message formats and exchange sequences shown below, the terminating delimiter string is not shown; however, it must be transmitted after each message.

It is not an error for a message to contain single occurrences of \n. For example, a greeting message may be split across lines by using a single \n. However, two consecutive occurrences of \n\n always indicates the end of the message.

Messages consist of one or more fields separated by whitespace. Whitespace consists of a nonzero number of space (decimal value 32) or tab (decimal value 9) characters; white space may also include line feed (\n, decimal value 10) characters, provided no two occur consecutively.

Note well: there is no upper bound on the number of whitespace characters between fields.

Each field consists of some number of non-whitespace characters. Cookies are strings of at least 20 ASCII decimal digits.

Note well: There is no upper bound on the length of a cookie.

The following notational conventions are used in describing the messages:

• String literals are rendered in typewriter text.

• The notation ⟨cookie A⟩ denotes a cookie with value A. So, for example, ⟨cookie A⟩ might denote the string 83928471056749028378110200855. The string denoted is at least 20 characters long and contains only decimal digits.

• Whitespace is denoted by ⟨WS⟩.

• The ASCII line feed character (encoded as byte value decimal 10) is denoted by \n.

• Fields of a message other than literals and cookies are indicated by ⟨fieldname⟩.

2.1 Greeting Message

The greeting message is the first message sent in each session; it is sent by the server. It has the following format:

CS471G–S13⟨WS⟩⟨datestamp⟩⟨WS⟩⟨cookie A⟩
The ⟨datestamp⟩ field contains the server’s idea of the approximate current date and time, something like “Thu Apr 11 18:21:45 EDT 2013”. Note that the datestamp contains white space. The field ⟨cookie A⟩ is the first cookie of the session. Note also that the cookie contains no white space, and is the last field in the message, so it can easily be found by locating the last occurrence of white space in the message.

2.2 Request Message

The request message is the first message sent by the client; it is only sent after the client has received and parsed the greeting message. There are two types of request message: Type 0 and Type 1. The first seven characters of each request message are always the same: 471type; they act as a sanity check, enabling the server to determine that the client is using the correct protocol. Note that there is no space between the string “471type” and the following character, which is either 0 or 1.

The Type 0 request message has the following format:

\[471type0⟨WS⟩⟨cookie f(A)⟩\]

where:

- the character following “471type” is 0 to indicate a Type 0 transaction.
- ⟨cookie f(A)⟩ is a cookie derived from ⟨cookie A⟩ (the cookie in the greeting message) in a manner described in the next paragraph.

This message indicates that the server should proceed with the transaction described in Section 3.1.

The ⟨cookie f(X)⟩ is computed from the ⟨cookie X⟩ by increasing each digit in the string by one (modulo 10). Thus, every ‘1’ becomes ‘2’, every ‘7’ becomes ‘8’, every ‘9’ becomes ‘0’, and so on. For example, if ⟨cookie X⟩ is

019287693492301556731

then ⟨cookie f(X)⟩ would be:

120398704503412667842

The Type 1 request message is similar to the Type 0, except its eighth character is ‘1’, and it also includes an address string:

\[471type1⟨WS⟩⟨cookie g(A)⟩⟨WS⟩⟨dottedquad⟩⟨port⟩\]

where:

- ⟨cookie g(A)⟩ is derived from ⟨cookie A⟩ (the cookie in the greeting message) in a manner described in the next paragraph.
- ⟨dottedquad⟩ is an IP address in the usual form (e.g., 128.163.140.219), and indicates the address of the host where the client is listening for connections.
• \langle port \rangle is a decimal port number between 0 and 65,535, indicating the port where the
client is listening for connections.

This message indicates that the server should proceed with the transaction described in
Section 3.2.

The \langle cookie g(X) \rangle is computed from \langle cookie X \rangle by replacing each digit with its “comple-
ment” modulo 10. That is, each digit other than 0 is replaced by the difference between
that digit’s value and 10; the digit 0 is left unchanged. (So is 5, because it is its own
complement mod 10). For example, if \langle cookie X \rangle is

019287693492301556731

then \langle cookie g(X) \rangle would be

091823417618709554379

Note that the transformation functions \( f(\ ) \) and \( g(\ ) \) require only local information, and
can be implemented digit-by-digit.

2.3 Cookie Message

Each cookie message consists of a single cookie, which is a string made of the ASCII
in Type 0 transactions, and by both client and server in Type 1 transactions. Like all
cookie protocol messages, it is delimited by \( \backslash n \). The cookie message format is thus:

\langle cookie X \rangle

In other words, the message (excluding the delimiter) contains only decimal digits.

2.4 Outcome Message

The server sends an Outcome message at the conclusion of a transaction. The message
has one of two forms, depending on the success of the transaction. If the transaction was
completed successfully, the message has the form:

\( \text{OK} \langle WS \rangle \langle cookie D \rangle \)

\( D \) is the final cookie of the session. If the transaction is not successful (due to some error
by the client), the message has the general form:

\( \text{Error:} \langle WS \rangle \langle \text{brief description of problem} \rangle \)

The two cases can be distinguished by looking for the initial string \textbf{OK}.
3 Message Exchanges

This section describes the sequences of messages exchanged between the client and server. The first part of each session is always the same:

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>[connect] →</td>
<td>←CS471G-S13 Sun Apr 14 21:37:04 EDT 2013 64···38</td>
</tr>
</tbody>
</table>

Note: Cookies are truncated in what follows because of space limitations. Also, the characters in each message are transmitted in the (left-to-right) order shown.

3.1 Type 0 Exchange

In a Type 0 transaction, after the client receives the cookie message containing the cookie $A$, it sends back another cookie message containing a cookie with value $f(A)$, where $f()$ is a mapping from strings to strings, described above in Section 2.2. After that the server sends an outcome message containing a second cookie:

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>[connect] →</td>
<td>←CS471G-S13 Mon Apr 8 12:00:01 EDT 2013 7···5 471type0 8···6→</td>
</tr>
<tr>
<td>[close]</td>
<td>←OK 5···1</td>
</tr>
</tbody>
</table>

The above diagram assumes no syntax or other errors occurred; in case of such errors, the final message would begin with “Error” or “error”, and no cookie would be included.

Note that there can only be one transaction per connection, and that the final message must be properly delimited. The connection is closed by both sides once the outcome message has been received.

3.2 Type 1 Exchange

In a Type 1 transaction, after the server greeting the client sends a Type 1 request message containing the cookie $g(A)$ (derived from $\langle$cookie $A$ $\rangle$ as described above in Section 2.2), as well as the IP address and port number of a socket where the client (or another program) is listening for connections. If the message is valid, the server attempts to open a connection to the indicated socket. Once that connection (the “second connection”) is complete, the server sends an outcome message on the original client connection to indicate the result; if it connected successfully, the outcome message has the result $\text{OK}$, and a new $\langle$cookie $B$ $\rangle$. 

5
The client then sends back a **cookie message** containing $B$ on the second connection, and if that message is valid, the server sends another **cookie message** containing $C$ on the second connection. The client must then send a cookie message containing $f(C)$ on the original connection. Finally, the server sends an outcome message on the original connection, and then closes both connections.

The following shows the normal sequence of message for a type 1 transaction.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>[connect] →</td>
<td>[listen]</td>
<td>[connect] →</td>
</tr>
<tr>
<td>←CS471G-S13 Thu Apr 11 04:44:55 EDT 2013 3:4 471type1 7:6 98.1.2.3-48765→</td>
<td>←OK 7:1 5:5 →</td>
<td>←OK 2:9 [close]</td>
</tr>
<tr>
<td></td>
<td>←7:1 4:4 →</td>
<td></td>
</tr>
</tbody>
</table>

In some cases, when the address specified by the client is incorrect, it may take several seconds for the server’s connection attempt to time out. The server cannot send an outcome message on the original connection until the fate of the second connection is known. In general, the server sends outcome messages as soon as possible.

### 4 NATs and Other Considerations

When a client is behind a Network Address Translation (NAT) box, it thinks its address is different from what the server sees. This makes the Type 2 transaction as specified above impossible. Therefore the server interprets the address “0.0.0.0” to mean “the same IP address from which the original connection came”. When that special address is given, the server will use the remote IP address from the original connection as the destination address.

Under normal (correct) operation, the server does not close any connection until all messages have been exchanged with the client. However, when the server encounters an error on the second connection it will immediately close it, send an “Error” outcome message on the original connection, and close it also.

Note that the last message on each connection should be delimited by \n\n, and *not* by the end-of-stream indication. A client that closes the connection without properly delimiting the final message will receive an error indication.