XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)

Ian Paterson
mailto:ian.paterson@clientside.co.uk
xmpp:ian@zoofy.com

Dave Smith
mailto:dizzyd@jabber.org
xmpp:dizzyd@jabber.org

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

Jack Moffitt
mailto:jack@chesspark.com
xmpp:jack@chesspark.com

Lance Stout
mailto:lance@andyet.com
xmpp:lance@lance.im

Winfried Tilanus
mailto:winfried@tilanus.com

2014-04-09
Version 1.11

Status	Type	Short Name
Draft	Standards Track	bosh

This specification defines a transport protocol that emulates the semantics of a long-lived, bidirectional TCP connection between two entities (such as a client and a server) by efficiently using multiple synchronous HTTP request/response pairs without requiring the use of frequent polling or chunked responses.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

The Transmission Control Protocol (TCP; RFC 793 [1]) is often used to establish a stream-oriented connection between two entities. Such connections can often be long-lived to enable an interactive "session" between the entities. However, sometimes the nature of the device or network can prevent an application from maintaining a long-lived TCP connection to a server or peer. In this case, it is desirable to use an alternative connection method that emulates the behavior of a long-lived TCP connection using a sequenced series of requests and responses that are exchanged over short-lived connections. The appropriate request-response semantics are widely available via the Hypertext Transfer Protocol (HTTP) as specified in RFC 1945 [2] and RFC 2616 [3].

BOSH, the technology defined in this specification, essentially provides a "drop-in" alternative to a long-lived, bidirectional TCP connection. It is a mature, full-featured technology that has been widely implemented and deployed since 2004. To our knowledge it was the first of many similar technologies, which now include the Comet methodology formalized in the Bayeux Protocol [4] as well as WebSocket RFC 6455 [5] and Reverse HTTP [6].

BOSH is designed to transport any data efficiently and with minimal latency in both directions. For applications that require both "push" and "pull" semantics, BOSH is significantly more bandwidth-efficient and responsive than most other bidirectional HTTP-based transport protocols and the techniques now commonly known as "Ajax". BOSH achieves this efficiency and low latency by using so-called "long polling" with multiple synchronous HTTP request/response pairs. Furthermore, BOSH can address the needs of constrained clients by employing fully-compliant HTTP 1.0 without the need for "cookies" (see RFC 2965 [7]) [8] or even access to HTTP headers.

BOSH was originally developed in the Jabber/XMPP community as a replacement for an even earlier HTTP-based technology called Jabber HTTP Polling (XEP-0025) [9]. Although BOSH assumes that the "payload" of HTTP requests and responses will be XML, the payload formats are not limited to XMPP stanzas (see XMPP Core [10]) and could contain a mixture of elements qualified by namespaces defined by different protocols (e.g., both XMPP and JSON). BOSH connection managers are generally not required to understand anything about the XML content that they transport beyond perhaps ensuring that each XML payload is qualified by the correct namespace.

Note: XMPP Over BOSH (XEP-0206) [11] documents some XMPP-specific extensions of this

[1] RFC 793: Transmission Control Protocol <http://tools.ietf.org/html/rfc0793>.
[2] RFC 1945: Hypertext Transfer Protocol -- HTTP/1.0 <http://tools.ietf.org/html/rfc1945>.
[3] RFC 2616: Hypertext Transport Protocol -- HTTP/1.1 <http://tools.ietf.org/html/rfc2616>.
[4] Bayeux Protocol <http://svn.cometd.org/trunk/bayeux/bayeux.html>.
[5] RFC 6455: The WebSocket Protocol <http://tools.ietf.org/html/rfc6455>.
[6] Reverse HTTP <http://tools.ietf.org/html/draft-lentczner-rhttp>.
[7] RFC 2965: HTTP State Management Mechanism <http://tools.ietf.org/html/rfc2965>.
[8] Requiring cookies is sub-optimal because several significant computing platforms provide only limited access to underlying HTTP requests/responses; worse, some platforms hide or remove cookie-related headers.
[9] XEP-0025: Jabber HTTP Polling <http://xmpp.org/extensions/xep-0025.html>.
[10] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[11] XEP-0206: XMPP Over BOSH <http://xmpp.org/extensions/xep-0206.html>.

protocol that were formerly included in this document.

2 Requirements

The following design requirements reflect the need to offer performance as close as possible to a standard TCP connection.

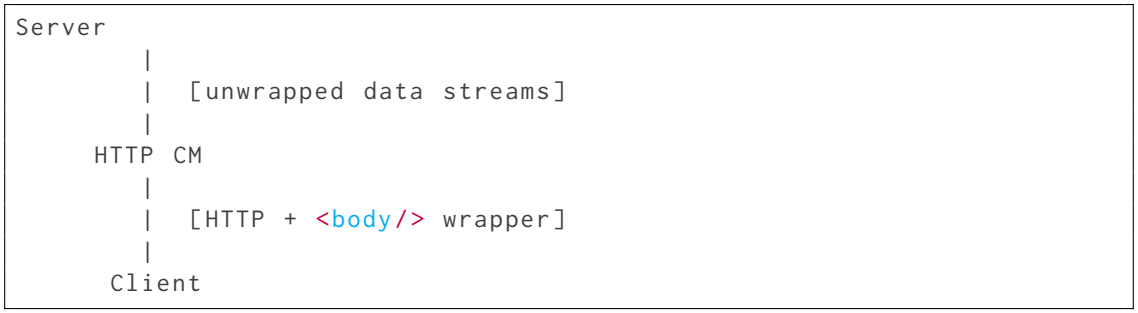
1. Compatible with constrained runtime environments* (e.g., mobile and browser-based clients).
2. Compatible with proxies that buffer partial HTTP responses.
3. Efficient through proxies that limit the duration of HTTP responses.
4. Fully compatible with HTTP/1.0.
5. Compatible with restricted network connections (e.g., firewalls, proxies, and gateways).
6. Fault tolerant (e.g., session recovers after an underlying TCP connection breaks at any stage during an HTTP request).
7. Extensible.
8. Consume significantly less bandwidth than polling-based protocols.
9. Significantly more responsive (lower latency) than polling-based protocols.
10. Support for polling (for clients that are limited to a single HTTP connection at a time).
11. In-order delivery of data.
12. Guard against unauthorized users injecting HTTP requests into a session.
13. Protect against denial of service attacks.
14. Multiplexing of data streams.

*Note: Compatibility with constrained runtime environments implies the following restrictions:

1. Clients are not required to have programmatic access to the headers of each HTTP request and response (e.g., cookies or status codes).
2. The body of each HTTP request and response is parsable XML with a single root element.
3. Clients can specify the Content-Type of the HTTP responses they receive.

3 Architectural Assumptions

This document assumes that most implementations will utilize a specialized connection manager ("CM") to handle HTTP connections rather than the native connection type for the relevant application (e.g., TCP connections in XMPP). Effectively, such a connection manager is a specialized HTTP server that translates between the HTTP requests and responses defined herein and the data streams (or API) implemented by the server with which it communicates, thus enabling a client to connect to a server via HTTP on port 80 or 443 instead of an application-specific port. We can illustrate this graphically as follows:



This specification covers communication only between a client and the connection manager. It does not cover communication between the connection manager and the server, since such communications are implementation-specific (e.g., the server might natively support this HTTP binding, in which case the connection manager will be a logical entity rather than a physical entity; alternatively the connection manager might be an independent translating proxy such that the server might believe it is talking directly to the client over TCP; or the connection manager and the server might use a component protocol or an API defined by the server implementation).

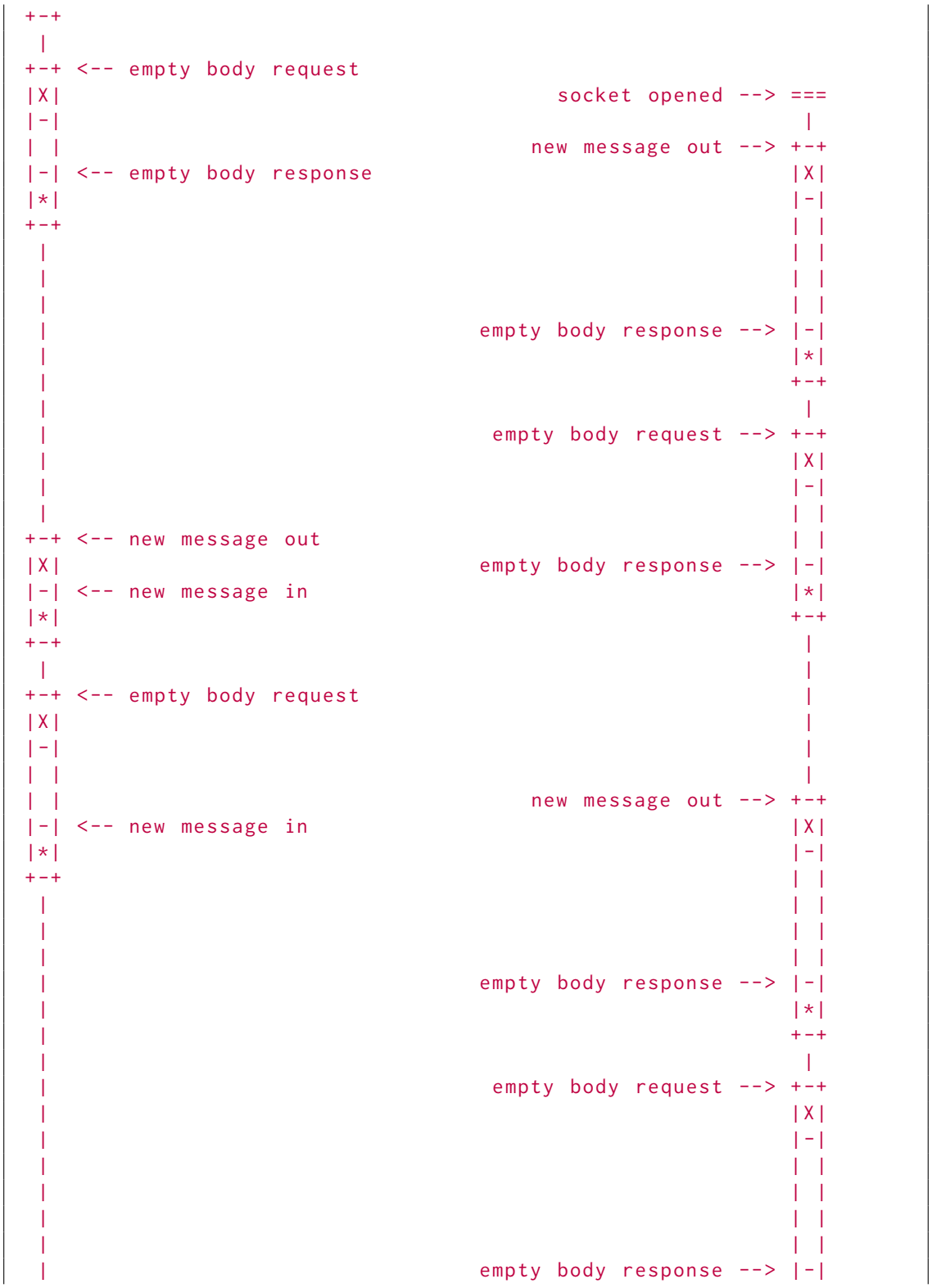
Furthermore, no aspect of this protocol limits its use to communication between a client and a server. For example, it could be used for communication between a server and a peer server if such communication can occur for the relevant application (e.g., in XMPP). However, this document focuses exclusively on use of the transport by clients that cannot maintain arbitrary persistent TCP connections with a server. We assume that servers and components are under no such restrictions and thus would use the native connection transport for the relevant application. (However, on some unreliable networks, BOSH might enable more stable communication between servers.)

4 The BOSH Technique

The technique employed by BOSH, which is sometimes called "HTTP long polling", reduces latency and bandwidth consumption over other HTTP polling techniques. When the client sends a request, the connection manager does not immediately send a response; instead it holds the request open until it has data to actually send to the client (or an agreed-to length

of inactivity has elapsed). The client then immediately sends a new request to the connection manager, continuing the long polling loop.
If the connection manager does not have any data to send to the client after some agreed-to length of time[12], it sends a response with an empty <body/>. This serves a similar purpose to whitespace keep-alives or XMPP Ping (XEP-0199) [13]; it helps keep a socket connection active which prevents some intermediaries (firewalls, proxies, etc) from silently dropping it, and helps to detect breaks in a reasonable amount of time.
Where clients and connection managers support persistent connections (i.e. "Connection: keep-alive" from HTTP/1.0, and which is the default state for HTTP/1.1), these sockets remain open for an extended length of time, awaiting the client's next request. This reduces the overhead of socket establishment, which can be very expensive if HTTP over Secure Sockets Layer (SSL) is used.
If the client has data to send while a request is still open, it establishes a second socket connection to the connection manager to send a new request. The connection manager immediately responds to the previously held request (possibly with no data) and holds open this new request. This results in the connections switching roles; the "old" connection is responded to and left awaiting new requests, while the "new" connection is now used for the long polling loop.
The following diagram illustrates this technique (possibly after XMPP session establishment)

[12] This time is typically on the order of tens of seconds (e.g., 60), and is determined during session creation
[13] XEP-0199: XMPP Ping <http://xmpp.org/extensions/xep-0199.html>.


empty body request
socket opened
new message out
empty body response
empty body response
empty body request
new message out
empty body response
new message in
empty body request
new message out
empty body response
empty body response
empty body request
empty body response

5 HTTP Overview

The requirements of RFC 2616 MUST be met for both requests and responses. Additional HTTP headers not specified herein MAY be included, but receivers SHOULD ignore any such headers. Clients and connection managers MAY omit headers that are not mandated by RFC 2616 and would otherwise be ignored (e.g. if the client has constrained bandwidth), but clients are advised that network and proxy policies could block such requests.
All information is encoded in the body of standard HTTP POST requests and responses. Each HTTP body contains a single <body/> wrapper which encapsulates the XML elements being transferred (see <body/> Wrapper Element).
Clients MUST send all HTTP requests as POST requests in any way permitted by RFC 1945 or RFC 2616. For example, clients can be expected to open more than one persistent connection, or in some cases to open a new HTTP/1.0 connection to send each request. However, clients and connection managers SHOULD NOT use Chunked Transfer Coding, since intermediaries might buffer each partial HTTP request or response and only forward the full request or response once it is available.
Clients MAY include an HTTP Accept-Encoding header in any request. If the connection manager receives a request with an Accept-Encoding header, it MAY include an HTTP Content-Encoding header in the response (indicating one of the encodings specified in the request) and compress the response body accordingly.
The HTTP Content-Type header of all client requests SHOULD be ”text/xml; charset=utf-8”. However, clients MAY specify another value if they are constrained to do so (e.g., ”application/x-www-form-urlencoded” or ”text/plain”). The client and connection manager SHOULD ignore all HTTP Content-Type headers they receive.

6 <body/> Wrapper Element

The body of each HTTP request and response contains a single <body/> wrapper element qualified by the ’http://jabber.org/protocol/httpbind’ namespace. The content of the wrapper is the data being transferred. The <body/> element and its content together MUST conform to the specifications set out in XML 1.0 [14]. They SHOULD also conform to Namespaces in XML [15]. The content MUST NOT contain any of the following (all defined in XML 1.0):

- Partial XML elements

[14] Extensible Markup Language (XML) 1.0 (Fourth Edition) <http://www.w3.org/TR/REC-xml/>.
[15] Namespaces in XML <http://www.w3.org/TR/REC-xml-names/>.

- XML comments
- XML processing instructions
- Internal or external DTD subsets
- Internal or external entity references (with the exception of predefined entities)

The <body/> wrapper MUST NOT contain any XML character data, although its child elements MAY contain character data. The <body/> wrapper MUST contain zero or more complete XML immediate child elements (called "payloads" in this document, e.g., XMPP stanzas as defined in RFC 6120 or elements containing XML character data that represents objects using the JSON data interchange format as defined in RFC 4627 [16]). Each <body/> wrapper MAY contain payloads qualified under a wide variety of different namespaces.
The <body/> element of every client request MUST possess a sequential request ID encapsulated via the 'rid' attribute; for details, refer to the Request IDs section of this document.

7 Initiating a BOSH Session

7.1 Session Creation Request

The first request from the client to the connection manager requests a new session.
The <body/> element of the first request SHOULD possess the following attributes (they SHOULD NOT be included in any other requests except as specified under Adding Streams To A Session):

- **'to'** -- This attribute specifies the target domain of the first stream.
- **'xml:lang'** -- This attribute (as defined in Section 2.12 of XML 1.0 [17]) specifies the default language of any human-readable XML character data sent or received during the session.
- **'ver'** -- This attribute specifies the highest version of the BOSH protocol that the client supports. The numbering scheme is "<major>.<minor>" (where the minor number MAY be incremented higher than a single digit, so it MUST be treated as a separate integer). Note: The 'ver' attribute should not be confused with the version of any protocol being transported.

[16] RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON) <http://tools.ietf.org/html/rfc4627>.
[17] Extensible Markup Language (XML) 1.0 (Fourth Edition) <http://www.w3.org/TR/REC-xml/>.

- **'wait'** -- This attribute specifies the longest time (in seconds) that the connection manager is allowed to wait before responding to any request during the session. This enables the client to limit the delay before it discovers any network failure, and to prevent its HTTP/TCP connection from expiring due to inactivity.

- **'hold'** -- This attribute specifies the maximum number of requests the connection manager is allowed to keep waiting at any one time during the session. If the client is able to reuse connections, this value SHOULD be set to "1".

Note: Clients that only support Polling Sessions MAY prevent the connection manager from waiting by setting 'wait' or 'hold' to "0". However, polling is NOT RECOMMENDED since the associated increase in bandwidth consumption and the decrease in responsiveness are both typically one or two orders of magnitude!
A connection manager MAY be configured to enable sessions with more than one server in different domains. When requesting a session with such a "proxy" connection manager, a client SHOULD include a **'route'** attribute that specifies the protocol, hostname, and port of the server with which it wants to communicate, formatted as "proto:host:port" (e.g., "xmpp:example.com:9999"). [18] A connection manager that is configured to work only with a single server (or only with a defined list of domains and the associated list of hostnames and ports that are serving those domains) MAY ignore the 'route' attribute. (Note that the 'to' attribute specifies the domain being served, not the hostname of the machine that is serving the domain.)
The <body/> element of the first request MAY also possess a **'from'** attribute, which specifies the originator of the first stream and which enables the connection manager to forward the originating entity's identity to the application server (e.g., the JabberID of an entity that is connecting to an XMPP server; see XEP-0206).
A client MAY include an **'ack'** attribute (set to "1") to indicate that it will be using acknowledgements throughout the session and that the absence of an 'ack' attribute in any request is meaningful (see Acknowledgements).
Some clients are constrained to only accept HTTP responses with specific Content-Types (e.g., "text/html"). The <body/> element of the first request MAY possess a **'content'** attribute. This specifies the value of the HTTP Content-Type header that MUST appear in all the connection manager's responses during the session. If the client request does not possess a 'content' attribute, then the HTTP Content-Type header of responses MUST be "text/xml; charset=utf-8".

Listing 1: Requesting a BOSH session

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 225
```

[18] Although the syntax of the 'route' attribute bears a superficial resemblance to a URI or IRI, it is not a URI/IRI and MUST NOT be processed in accordance with the rules specified in RFC 3986, RFC 3987, or (for XMPP) RFC 5122.

```
<body content='text/xml; charset=utf-8'
      from='user@example.com'
      hold='1'
      rid='1573741820'
      to='example.com'
      route='xmpp:example.com:9999'
      ver='1.6'
      wait='60'
      ack='1'
      xml:lang='en'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: All requests after the first one MUST include a valid 'sid' attribute (provided by the connection manager in the Session Creation Response). The initialization request is unique in that the <body/> element MUST NOT possess a 'sid' attribute.

7.2 Session Creation Response

After receiving a new session request, the connection manager MUST generate an opaque, unpredictable session identifier (or SID). The SID MUST be unique within the context of the connection manager application. The <body/> element of the connection manager's response to the client's session creation request MUST possess the following attributes (they SHOULD NOT be included in any other responses):

- **'sid'** -- This attribute specifies the SID
- **'wait'** -- This is the longest time (in seconds) that the connection manager will wait before responding to any request during the session. The time MUST be less than or equal to the value specified in the session request.
- **'requests'** -- This attribute enables the connection manager to limit the number of simultaneous requests the client makes (see Overactivity and Polling Sessions). This value must be larger than the 'hold' attribute value specified in the session request. The RECOMMENDED value is one more than the value of the 'hold' attribute specified in the session request.

The <body/> element SHOULD also include the following attributes (they SHOULD NOT be included in any other responses):

- **'ver'** -- This attribute specifies the highest version of the BOSH protocol that the connection manager supports, or the version specified by the client in its request, whichever is lower.

- **'polling'** -- This attribute specifies the shortest allowable polling interval (in seconds). This enables the client to not send empty request elements more often than desired (see Polling Sessions and Overactivity).
- **'inactivity'** -- This attribute specifies the longest allowable inactivity period (in seconds). This enables the client to ensure that the periods with no requests pending are never too long (see Polling Sessions and Inactivity).
- **'hold'** -- This attribute informs the client about the maximum number of requests the connection manager will keep waiting at any one time during the session. This value MUST NOT be greater than the value specified by the client in the session request.
- **'to'** -- This attribute communicates the identity of the backend server to which the client is attempting to connect.

The connection manager MAY include an **'accept'** attribute in the session creation response element, to specify a comma-separated list of the content encodings it can decompress. After receiving a session creation response with an 'accept' attribute, clients MAY include an HTTP Content-Encoding header in subsequent requests (indicating one of the encodings specified in the 'accept' attribute) and compress the bodies of the requests accordingly.
A connection manager MAY include an **'ack'** attribute (set to the value of the 'rid' attribute of the session creation request) to indicate that it will be using acknowledgements throughout the session and that the absence of an 'ack' attribute in any response is meaningful (see Acknowledgements).
If the connection manager supports session pausing (see Inactivity) then it SHOULD advertise that to the client by including a **'maxpause'** attribute in the session creation response element. The value of the attribute indicates the maximum length of a temporary session pause (in seconds) that a client can request.
For both requests and responses, the <body/> element and its content SHOULD be UTF-8 encoded. If the HTTP Content-Type header of a request/response specifies a character encoding other than UTF-8, then the connection manager MAY convert between UTF-8 and the other character encoding. However, even in this case, it is OPTIONAL for the connection manager to convert between encodings. The connection manager MAY inform the client which encodings it can convert by setting the optional **'charsets'** attribute in the session creation response element to a space-separated list of encodings. [19]
As soon as the connection manager has established a connection to the server and discovered its identity, it MAY forward the identity to the client by including a **'from'** attribute in a response, either in its session creation response, or (if it has not received the identity from

[19] Each character set name (or character encoding name -- we use the terms interchangeably) SHOULD be of type NMTOKEN, where the names are separated by the white space character #x20, resulting in a tokenized attribute type of NMTOKENS (see Section 3.3.1 of XML 1.0 [20]). Strictly speaking, the Character Sets registry maintained by the Internet Assigned Numbers Authority (see <http://www.iana.org/assignments/character-sets>) allows a character set name to contain any printable US-ASCII character, which might include characters not allowed by the NMTOKEN construction of XML 1.0; however, the only existing character set name which includes such a character is "NF_Z_62-010_(1973)".

the server by that time) in any subsequent response to the client.

Listing 2: Session creation response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 243

<body wait='60'
      inactivity='30'
      polling='5'
      requests='2'
      hold='1'
      ack='1573741820'
      accept='deflate,gzip'
      maxpause='120'
      sid='SomeSID'
      charsets='ISO_8859-1 ISO-2022-JP'
      ver='1.6'
      from='example.com'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Listing 3: Subsequent response with 'from' attribute

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 71

<body from='example.com'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

8 Sending and Receiving XML Payloads

After the client has successfully completed all required preconditions, it can send and receive XML payloads via the HTTP binding.

Listing 4: Transmitting payloads

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 279

<body rid='1249243562'
      sid='SomeSID'
      xmlns='http://jabber.org/protocol/httpbind'>
```

```
  <message to='contact@example.com'
           xmlns='jabber:client'>
    <body>Good morning!</body>
  </message>
  <message to='friend@example.com'
           xmlns='jabber:client'>
    <body>Hey, what&apos;s up?</body>
  </message>
</body>
```

Upon receipt of a request, the connection manager SHOULD forward the content of the <body/> element to the server as soon as possible. In any case it MUST forward the content from different requests in the order specified by their 'rid' attributes.
The connection manager MUST also return an HTTP 200 OK response with a <body/> element to the client. Note: This does not indicate that the payloads have been successfully delivered to the application server.
It is RECOMMENDED that the connection manager not return an HTTP result until a payload has arrived from the application server for delivery to the client. However, the connection manager SHOULD NOT wait longer than the time specified by the client in the 'wait' attribute of its Session Creation Request, and it SHOULD NOT keep more HTTP requests waiting at a time than the number specified in the 'hold' attribute of the session creation request. In any case it MUST respond to requests in the order specified by their 'rid' attributes.
If there are no payloads waiting or ready to be delivered within the waiting period, then the connection manager SHOULD include an empty <body/> element in the HTTP result:

Listing 5: Empty response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 52

<body xmlns='http://jabber.org/protocol/httpbind'/>
```

If the connection manager has received one or more payloads from the application server for delivery to the client, then it SHOULD return the payloads in the body of its response as soon as possible after receiving them from the server. The example below includes payloads qualified by different namespaces:

Listing 6: Response with queued stanza

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 917

<body xmlns='http://jabber.org/protocol/httpbind'
      xmlns:json='http://json.org/'>
  <message from='contact@example.com'
```

```
          to='user@example.com'
          xmlns='jabber:client'>
    <body>Good morning to you!</body>
  </message>
  <message from='friend@example.com'
          to='user@example.com'
          xmlns='jabber:client'>
    <body>Not much, how about with you?</body>
  </message>
  <json:json>
  [
    {
      "precision": "zip",
      "Latitude":  37.7668,
      "Longitude": -122.3959,
      "Address":   "",
      "City":      "SAN FRANCISCO",
      "State":     "CA",
      "Zip":       "94107",
      "Country":   "US"
    },
    {
      "precision": "zip",
      "Latitude":  37.371991,
      "Longitude": -122.026020,
      "Address":   "",
      "City":      "SUNNYVALE",
      "State":     "CA",
      "Zip":       "94085",
      "Country":   "US"
    }
  ]
  </json:json>
</body>
```

The client MAY poll the connection manager for incoming payloads by sending an empty <body/> element.

Listing 7: Requesting XML Payloads

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 83

<body rid='1249243563'
      sid='SomeSID'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

The connection manager MUST wait and respond in the same way as it does after receiving payloads from the client.

9 Acknowledgements

9.1 Request Acknowledgements

When responding to a request that it has been holding, if the connection manager finds it has already received another request with a higher 'rid' attribute (typically while it was holding the first request), then it MAY acknowledge the reception to the client. The connection manager MAY set the 'ack' attribute of any response to the value of the highest 'rid' attribute it has received in the case where it has also received all requests with lower 'rid' values.

Listing 8: Response with request acknowledgement

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 69

<body ack='1249243564'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

If the connection manager will be including 'ack' attributes on responses during a session, then it MUST include an 'ack' attribute in its session creation response, and set the 'ack' attribute of responses throughout the session. The only exception is that, after its session creation response, the connection manager SHOULD NOT include an 'ack' attribute in any response if the value would be the 'rid' of the request being responded to.
If the connection manager is permitted to hold more than one request at a time, then the reception of a lower-than-expected 'ack' value from the connection manager (or the unexpected abscence of an 'ack' attribute) can give the client an early warning that a network failure might have occurred (e.g., if the client believes the connection manager should have received another request by the time it responded).

9.2 Response Acknowledgements

The client MAY similarly inform the connection manager about the responses it has received by setting the 'ack' attribute of any request to the value of the highest 'rid' of a request for which it has already received a response in the case where it has also received all responses associated with lower 'rid' values. If the client will be including 'ack' attributes on requests during a session, then it MUST include an 'ack' attribute (set to '1') in its session creation request, and set the 'ack' attribute of requests throughout the session. The only exception is that, after its session creation request, the client SHOULD NOT include an 'ack' attribute in

any request if it has received responses to all its previous requests.

Listing 9: Request with response acknowledgement

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 100

<body rid='1249243566'
      sid='SomeSID'
      ack='1249243564'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

After receiving a request with an 'ack' value less than the 'rid' of the last request that it has already responded to, the connection manager MAY inform the client of the situation by sending its next response immediately instead of waiting until it has payloads to send to the client (e.g., if some time has passed since it responded). In this case it SHOULD include a 'report' attribute set to one greater than the 'ack' attribute it received from the client, and a 'time' attribute set to the number of milliseconds since it sent the response associated with the 'report' attribute.

Listing 10: Response with report

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 83

<body report='1249243565'
      time='852'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Upon reception of a response with 'report' and 'time' attributes, if the client has still not received the response associated with the request identifier specified by the 'report' attribute, then it MAY choose to resend the request associated with the missing response (see Broken Connections).

10 Inactivity

After receiving a response from the connection manager, if none of the client's requests are still being held by the connection manager (and if the session is not a Polling Session), the client SHOULD make a new request as soon as possible. In any case, if no requests are being held, the client MUST make a new request before the maximum inactivity period has expired. The length of this period (in seconds) is specified by the 'inactivity' attribute in the session

creation response.
If the connection manager has responded to all the requests it has received within a session and the time since its last response is longer than the maximum inactivity period, then it SHOULD assume the client has been disconnected and terminate the session without informing the client. If the client subsequently makes another request, then the connection manager SHOULD respond as if the session does not exist.
If the connection manager did not specify a maximum inactivity period in the session creation response, then it SHOULD allow the client to be inactive for as long as it chooses.
If the session is not a polling session then the connection manager SHOULD specify a relatively short inactivity period to ensure that disconnections are discovered as quickly as possible. The RECOMMENDED time would be a little more than the number of seconds for a comfortable network round trip between the connection manager and the client under difficult network conditions (since the client can be expected to make a new request immediately -- see above).
If a client encounters an exceptional temporary situation during which it will be unable to send requests to the connection manager for a period of time greater than the maximum inactivity period (e.g., while a runtime environment changes from one web page to another), and if the connection manager included a 'maxpause' attribute in its Session Creation Response, then the client MAY request a temporary increase to the maximum inactivity period by including a 'pause' attribute in a request. Note: If the connection manager did not specify a 'maxpause' attribute at the start of the session then the client MUST NOT send a 'pause' attribute during the session.

Listing 11: Requesting a Session Pause

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 94

<body rid='1249243564'
      sid='SomeSID'
      pause='60'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Upon reception of a session pause request, if the requested period is not greater than the maximum permitted time, then the connection manager SHOULD respond immediately to all pending requests (including the pause request) and *temporarily* increase the maximum inactivity period to the requested time. Note: The response to the pause request MUST NOT contain any payloads.
Note: If the client simply wants the connection manager to return all the requests it is holding then it MAY set the value of the 'pause' attribute to be the value of the 'inactivity' attribute in the connection manager's session creation response. (If the client believes it is in danger of becoming disconnected indefinitely then it MAY even request a temporary reduction of the maximum inactivity period by specifying a 'pause' value less than the 'inactivity' value, thus

enabling the connection manager to discover any subsequent disconnection more quickly.) The connection manager SHOULD set the maximum inactivity period back to normal upon reception of the next request from the client (assuming the connection manager has not already terminated the session).

11 Overactivity

The client SHOULD NOT make more simultaneous requests than specified by the 'requests' attribute in the connection manager's Session Creation Response. However the client MAY make one additional request if it is to pause or terminate a session.
If during any period the client sends a sequence of new requests (i.e. requests with incremented rid attributes, not repeat requests) longer than the number specified by the 'requests' attribute, and if the connection manager has not yet responded to any of the requests, and if the last request did not include either a 'pause' attribute or a 'type' attribute set to "terminate", then the connection manager SHOULD consider that the client is making too many simultaneous requests, and terminate the HTTP session with a 'policy-violation' terminal binding error to the client. Note: This behavior applies to equally to normal and polling sessions.

Listing 12: Too many simultaneous requests response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 98

<body type='terminate'
      condition='policy-violation'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: If the connection manager did not specify a 'requests' attribute in the session creation response, then it MUST allow the client to send as many simultaneous requests as it chooses.
If during any period the client sends a sequence of new requests equal in length to the number specified by the 'requests' attribute, and if the connection manager has not yet responded to any of the requests, and if the last request was empty and did not include either a 'pause' attribute or a 'type' attribute set to "terminate", and if the last two requests arrived within a period shorter than the number of seconds specified by the 'polling' attribute in the session creation response, then the connection manager SHOULD consider that the client is making requests more frequently than it was permitted and terminate the HTTP session and return a 'policy-violation' terminal binding error to the client. Note: the behavior for Polling Sessions is slightly different.

Listing 13: Too frequent requests response

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset=utf-8
Content-Length: 98

<body type='terminate'
      condition='policy-violation'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: If the connection manager did not specify a 'polling' attribute in the session creation response, then it MUST allow the client to send requests as frequently as it chooses.

12 Polling Sessions

It is not always possible for a constrained client to open more than one HTTP connection with the connection manager at a time. In this case the client SHOULD inform the connection manager by setting the values of the 'wait' and/or 'hold' attributes in its session creation request to "0", and then "poll" the connection manager at regular intervals throughout the session for payloads it might have received from the server. Note: Even if the client does not request a polling session, the connection manager MAY require a client to use polling by setting the 'requests' attribute (which specifies the number of simultaneous requests the client can make) of its Session Creation Response to "1", however this is NOT RECOMMENDED. If a session will use polling, the connection manager SHOULD specify a higher than normal value for the 'inactivity' attribute (see Inactivity) in its session creation response. The increase SHOULD be greater than the value it specifies for the 'polling' attribute.
If the client sends two consecutive empty new requests (i.e. requests with incremented rid attributes, not repeat requests) within a period shorter than the number of seconds specified by the 'polling' attribute (the shortest allowable polling interval) in the session creation response, and if the connection manager's response to the first request contained no payloads, then upon reception of the second request the connection manager SHOULD terminate the HTTP session and return a 'policy-violation' terminal binding error to the client.

Listing 14: Too frequent polling response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 98

<body type='terminate'
      condition='policy-violation'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: If the connection manager did not specify a 'polling' attribute in the session creation response, then it MUST allow the client to poll as frequently as it chooses.

13 Terminating the BOSH Session

At any time, the client MAY gracefully terminate the session by sending a <body/> element with a 'type' attribute set to "terminate". The termination request MAY include one or more payloads that the connection manager MUST forward to the server to ensure graceful logoff. The payload in the termination request SHOULD NOT need any response from the server.

Listing 15: Session termination by client

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 158

<body rid='1249243565'
      sid='SomeSID'
      type='terminate'
      xmlns='http://jabber.org/protocol/httpbind'>
  <presence type='unavailable'
            xmlns='jabber:client'/>
</body>
```

The connection manager SHOULD respond to this request with an HTTP 200 OK containing an empty <body/> element. The connection manager SHOULD acknowledge the session termination on the oldest connection with a HTTP 200 OK containing a <body/> element of the type 'terminate'. On all other open connections, the connection manager SHOULD respond with an HTTP 200 OK containing an empty <body/> element.

Listing 16: Connection manager acknowledges termination

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 69

<body type='terminate'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Upon receiving the response, the client MUST consider the HTTP session to have been terminated.

14 Request IDs

14.1 Generation

The client MUST generate a large, random, positive integer for the initial 'rid' (see Security Considerations) and then increment that value by one for each subsequent request.

The client MUST take care to choose an initial 'rid' that will never be incremented above 9007199254740991 [21] within the session. In practice, a session would have to be extraordinarily long (or involve the exchange of an extraordinary number of packets) to exceed the defined limit.

14.2 In-Order Message Forwarding

When a client makes simultaneous requests, the connection manager might receive them out of order. The connection manager MUST forward the payloads to the server and respond to the client requests in the order specified by the 'rid' attributes. The client MUST process responses received from the connection manager in the order the requests were made.
The connection manager SHOULD expect the 'rid' attribute to be within a window of values greater than the 'rid' of the previous request. The size of the window is equal to the maximum number of simultaneous requests allowed by the connection manager. If it receives a request with a 'rid' greater than the values in the window, then the connection manager MUST terminate the session with an error:

Listing 17: Unexpected rid error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 96

<body type='terminate'
      condition='item-not-found'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

14.3 Broken Connections

Unreliable network communications or client constraints can result in broken connections. The connection manager SHOULD remember the 'rid' and the associated HTTP response body of the client's most recent requests which were not session pause requests (see Inactivity) and which did not result in an HTTP or binding error. The number of responses to non-pause requests kept in the buffer SHOULD be either the same as the maximum number of simultaneous requests allowed by the connection manager or, if Acknowledgements are being used, the number of responses that have not yet been acknowledged.
If the network connection is broken or closed before the client receives a response to a request from the connection manager, then the client MAY resend an exact copy of the original request. Whenever the connection manager receives a request with a 'rid' that it has already received, it SHOULD return an HTTP 200 (OK) response that includes the buffered copy of the original XML response to the client (i.e., a <body/> wrapper possessing appropriate

[21] 9007199254740991 is 2^{53}-1. Some weakly typed languages use IEEE Standard 754 Doubles to represent all numbers. These Doubles cannot represent integers above 2^{53} accurately.

attributes and optionally containing one or more XML payloads).
If the connection manager receives a request for a 'rid' which has already been received but to which it has not yet responded then it SHOULD respond immediately to the existing request with a recoverable binding condition (see Recoverable Binding Conditions) and send any future response to the latest request. There is a possibility that a client might subvert polling frequency limits by deliberately sending requests for the same 'rid' multiple times, and so a connection manager implementation MAY choose to impose a limit to the frequency or number of requests for the same 'rid'. If the client exceeds this limit then the connection manager SHOULD terminate the HTTP session and return a 'policy-violation' terminal binding error to the client (see Terminal Binding Conditions).
If the original response is not available (e.g., it is no longer in the buffer), then the connection manager MUST return an 'item-not-found' terminal binding error:

Listing 18: Response not in buffer error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 96

<body type='terminate'
      condition='item-not-found'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: The error is the same whether the 'rid' is too large or too small. This makes it more difficult for an attacker to discover an acceptable value.

15 Protecting Insecure Sessions

15.1 Applicability

The OPTIONAL key sequencing mechanism described here MAY be used if the client's session with the connection manager is not secure. The session SHOULD be considered secure only if all client requests are made via SSL (or TLS) HTTP connections and the connection manager generates an unpredictable session ID. If the session is secure, it is not necessary to use this key sequencing mechanism.
Even if the session is not secure, the unpredictable session and request IDs specified in the preceding sections of this document already provide a level of protection similar to that provided by a connection bound to a single pair of persistent TCP/IP connections, and thus provide sufficient protection against a 'blind' attacker. However, in some circumstances, the key sequencing mechanism defined below helps to protect against a more determined and knowledgeable attacker.
It is important to recognize that the key sequencing mechanism defined below helps to protect only against an attacker who is able to view the contents of all requests or responses in an insecure session but who is not able to alter the contents of those requests (in this

case, the mechanism prevents the attacker from injecting HTTP requests into the session, e.g., termination requests or responses). However, the key sequencing mechanism does not provide any protection when the attacker is able to alter the contents of insecure requests or responses.

15.2 Introduction

The HTTP requests of each session MAY be spread across a series of different socket connections. This would enable an unauthorized user that obtains the session ID and request ID of a session to then use their own socket connection to inject <body/> request elements into the session and receive the corresponding responses.
The key sequencing mechanism below protects against such attacks by enabling a connection manager to detect <body/> request elements injected by a third party.

15.3 Generating the Key Sequence

Prior to requesting a new session, the client MUST select an unpredictable counter (”n”) and an unpredictable value (”seed”). The client then processes the ”seed” through a cryptographic hash and converts the resulting 160 bits to a hexadecimal string K(1). It does this ”n” times to arrive at the initial key K(n). The hashing algorithm MUST be SHA-1 as defined in RFC 3174 [22].

Listing 19: Creating the key sequence

```
K(1) = hex(SHA-1(seed))
       K(2) = hex(SHA-1(K(1)))
       ...
       K(n) = hex(SHA-1(K(n-1)))
```

Because case is not significant in hexadecimal encoding, key comparisons SHOULD be case insensitive.

15.4 Use of Keys

The client MUST set the ’newkey’ attribute of the first request in the session to the value K(n).

Listing 20: Session Request with Initial Key

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
```

[22] RFC 3174: US Secure Hash Algorithm 1 (SHA1) <http://tools.ietf.org/html/rfc3174>.

```
Content-Length: 203

<body content='text/xml; charset=utf-8'
      hold='1'
      rid='1573741820'
      to='example.com'
      wait='60'
      xml:lang='en'
      newkey='ca393b51b682f61f98e7877d61146407f3d0a770'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

The client MUST set the 'key' attribute of all subsequent requests to the value of the next key in the generated sequence (decrementing from K(n-1) towards K(1) with each request sent).

Listing 21: Request with Key

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 130

<body rid='1573741821'
      sid='SomeSID'
      key='bfb06a6f113cd6fd3838ab9d300fdb4fe3da2f7d'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

The connection manager MAY verify the key by calculating the SHA-1 hash of the key and performing a case insensitive comparison of it to the 'newkey' attribute of the previous request (or the 'key' attribute if the 'newkey' attribute was not set). If the values do not match (or if it receives a request without a 'key' attribute and the 'newkey' or 'key' attribute of the previous request was set), then the connection manager MUST NOT process the element, MUST terminate the session, and MUST return an 'item-not-found' terminal binding error.

Listing 22: Invalid Key Sequence Error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 96

<body type='terminate'
      condition='item-not-found'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

15.5 Switching to Another Key Sequence

A client SHOULD choose a high value for "n" when generating the key sequence. However, if the session lasts long enough that the client arrives at the last key in the sequence K(1) then

the client MUST switch to a new key sequence.
The client MUST:

1. Choose new values for ”seed” and ”n”.
2. Generate a new key sequence using the algorithm defined above.
3. Set the 'key' attribute of the request to the next value in the old sequence (i.e. K(1), the last value).
4. Set the 'newkey' attribute of the request to the value K(n) from the new sequence.

Listing 23: New Key Sequence

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 277

<body rid='1573741822'
      sid='SomeSID'
      key='6f825e81f4532b2c5fa2d12457d8a1f22e8f838e'
      newkey='113f58a37245ec9637266cf2fb6e48bfeaf7964e'
      xmlns='http://jabber.org/protocol/httpbind'>
  <message to='contact@example.com'
           xmlns='jabber:client'>
    <body>I said ”Hi!”</body>
  </message>
</body>
```

16 Multiple Streams

16.1 Introduction

The OPTIONAL feature described in this section enables multiple XML streams to be contained within a single HTTP session. This feature allows for clients to connect using more than one account at the same time. This feature also reduces network traffic for any client that needs to establish parallel streams over HTTP.

16.2 Discovery

If a connection manager supports the multi-streams feature, it MUST include a 'stream' attribute in its Session Creation Response. If a client does not receive the 'stream' attribute

then it MUST assume that the connection manager does not support the feature. [23]
The 'stream' attribute identifies the first stream to be opened for the session. The value of each 'stream' attribute MUST be an opaque and unpredictable name that is unique within the context of the connection manager application.

Listing 24: Session creation response with stream name

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 251

<body wait='60'
      inactivity='30'
      polling='5'
      requests='2'
      hold='1'
      accept='deflate,gzip'
      stream='firstStreamName'
      maxpause='120'
      sid='SomeSID'
      charsets='ISO_8859-1 ISO-2022-JP'
      ver='1.6'
      from='example.com'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

16.3 Adding Streams To A Session

If the connection manager included a 'stream' attribute in its session creation response then the client MAY ask it to open another stream at any time by sending it an empty <body/> element with a 'to' attribute. The request MUST include valid 'sid' and 'rid' [24] attributes, and SHOULD also include an 'xml:lang' attribute. The request MAY include either 'route' or 'from' attributes (see Session Creation Request), but it SHOULD NOT include 'ver', 'content', 'hold' or 'wait' attributes (since a new session is not being created).

Listing 25: Requesting another stream

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 144

<body sid='SomeSID'
```

[23] Therefore a client and a connection manager will be compatible even if one or the other offers no support for multi-stream sessions.
[24] The 'rid' attribute is always incremented normally without reference to any 'stream' attribute.

```
      rid='1573741820'
      to='example.com'
      route='xmpp:example.com:9999'
      xml:lang='en'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

If the connection manager did not indicate its support for multiple streams at the start of the session, then it MUST ignore the extra attributes and treat the request as a normal empty request for payloads (see Sending and Receiving XML Payloads). [25] Otherwise it MUST open a new stream with the specified server (see Session Creation Response), generate a new stream name, and respond to the client with the name. The response MAY also include the 'from' attribute, but it SHOULD NOT include 'sid', 'requests', 'polling', 'hold', 'inactivity', 'maxpause', 'accept', 'charsets', 'ver' or 'wait' attributes.

Listing 26: Add stream response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 97

<body stream='secondStreamName'
      from='example.com'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: If the response did not include a 'from' attribute then they MAY be sent in a subsequent response instead (see Session Creation Response). In that case the 'stream' attribute MUST also be specified.

16.4 Transmitting Payloads

If more than one stream has been opened within a session, then all non-empty <body/> elements sent by the connection manager MUST include a 'stream' attribute that specifies which stream *all* the payloads it contains belong to. The client SHOULD include a 'stream' attribute for the same purpose. The client MAY omit the 'stream' attribute if it wants the connection manager to broadcast the payloads over all open streams. Note: A <body/> element MUST NOT contain different payloads for different streams.
If a stream name does not correspond to one of the session's open streams, then the receiving connection manager SHOULD return an 'item-not-found' terminal binding error, or the receiving client SHOULD terminate the session. However, if the receiving entity has only just closed the stream (and the sender might not have been aware of that when it sent the payloads), then it MAY instead simply silently ignore any payloads the <body/> element contains.
Note: Empty <body/> elements that do not include a 'from' attribute SHOULD NOT include a

[25] This helps to ensure backwards-compatibility with older implementations.

'stream' attribute (since nothing is being transmitted for any stream). If such a <body/> element does include a 'stream' attribute then the receiving entity SHOULD ignore the attribute.

Listing 27: Client sends payload with a stream name

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 207

<body rid='1249243562'
      sid='SomeSID'
      stream='secondStreamName'
      xmlns='http://jabber.org/protocol/httpbind'>
  <message to='contact@example.com'
           xmlns='jabber:client'>
    <body>I said hello.</body>
  </message>
</body>
```

Note: The value of the 'stream' attribute of the response MAY be different than the corresponding request. [26]

Listing 28: Connection manager responds with a different stream name

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 197

<body stream='firstStreamName'
      xmlns='http://jabber.org/protocol/httpbind'>
  <message from='contact@example.com'
           to='user@example.com'
           xmlns='jabber:client'>
    <body>Hi yourself!</body>
  </message>
</body>
```

If no stream name is specified by the connection manager then the client MUST assume the payloads are associated with the first stream (even if the first stream has been closed).
If no stream name is specified by the client then the connection manager MUST broadcast the payloads over all open streams. [27]

[26] Each HTTP response MUST belong to the same session as the request that triggered it, but not necessarily to the same stream.
[27] The broadcast payloads can be of any type.

Listing 29: Client asks for a payload to be broadcast

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 149

<body rid='1249243562'
      sid='SomeSID'
      xmlns='http://jabber.org/protocol/httpbind'>
  <presence xmlns='jabber:client'>
    <show>away</show>
  </presence>
</body>
```

16.5 Closing a Stream

If more than one stream is open within a session, the client MAY close one open stream at any time using the procedure described in the section Terminating the BOSH Session above, taking care to specify the stream name with a 'stream' attribute. If the client closes the last stream the connection manager MUST terminate the session. If the client does not specify a stream name then the connection manager MUST close all open streams (sending any payloads the terminate request contains to all streams), and terminate the session.

Listing 30: Client closes one stream

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 184

<body rid='1249243564'
      sid='SomeSID'
      stream='secondStreamName'
      type='terminate'
      xmlns='http://jabber.org/protocol/httpbind'>
  <presence type='unavailable'
            xmlns='jabber:client'/>
</body>
```

16.6 Error Conditions

If more than one stream is open within a session, the connection manager MAY include a 'stream' attribute in a fatal binding error (see Terminal Binding Conditions). If a 'stream' attribute is specified then the stream MUST be closed by both entities but the session SHOULD

NOT be terminated.

Listing 31: Fatal stream error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 132

<body type='terminate'
      condition='remote-connection-failed'
      stream='secondStreamName'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Note: If the connection manager does not include a 'stream' attribute in a fatal binding error then all the session's open streams MUST be closed by both entities and the session MUST be terminated.

17 Error and Status Codes

There are four types of error and status reporting in HTTP responses:

Condition Type	Description
HTTP Conditions (Deprecated)	The connection manager responds to an invalid request from a legacy client with a standard HTTP error. These are used for binding syntax errors, possible attacks, etc. Note that constrained clients are unable to differentiate between HTTP errors.
Terminal Binding Conditions	These error conditions can be read by constrained clients. They are used for connection manager problems, abstracting stream errors, communication problems between the connection manager and the server, and invalid client requests (binding syntax errors, possible attacks, etc.)
Recoverable Binding Conditions	These report communication problems between the connection manager and the client. They do not terminate the session. Clients recover from these errors by resending all the preceding <body/> wrappers that have not received responses.
Transported Protocol Conditions	Errors relating to the XML payloads within <body/> wrappers are, in general, defined in the documentation of the protocol being transported. They do not terminate the session.

Full descriptions are provided below.

17.1 HTTP Conditions

Note: All HTTP codes except 200 have been superseded by Terminal Binding Conditions to allow clients to determine whether the source of errors is the connection manager application or an HTTP intermediary. A legacy client (or connection manager) is a client (or connection manager) that did not include a 'ver' attribute in its session creation request (or response). A legacy client (or connection manager) will interpret (or respond with) HTTP error codes according to the table below. Non-legacy connection managers SHOULD NOT send HTTP error codes unless they are communicating with a legacy client. Upon receiving an HTTP error (400, 403, 404), a legacy client or any client that is communicating with a legacy connection manager MUST consider the HTTP session to be null and void. A non-legacy client that is communicating with a non-legacy connection manager MAY consider that the session is still active.

Code	Name	Superseded by	Purpose
200	OK	-	Response to valid client request.
400	Bad Request	bad-request	Inform client that the format of an HTTP header or binding element is unacceptable (e.g., syntax error).
403	Forbidden	policy-violation	Inform client that it has broken the session rules (polling too frequently, requesting too frequently, too many simultaneous requests).
404	Not Found	item-not-found	Inform client that (1) 'sid' is not valid, (2) 'stream' is not valid, (3) 'rid' is larger than the upper limit of the expected window, (4) connection manager is unable to resend response, (5) 'key' sequence is invalid.

Note: No other HTTP error and status codes were defined in the early versions of BOSH (e.g., Internal Server Error).

17.2 Terminal Binding Conditions

In any response it sends to the client, the connection manager MAY return a fatal error by setting a 'type' attribute of the <body/> element to "terminate". These binding errors imply that the HTTP session is terminated (unless a 'stream' attribute is specified -- see Multiple Stream Error Conditions).
Note: Although many of these conditions are similar to the XMPP stream error conditions specified in RFC 6120, they are not to be confused with XMPP stream errors. In cases where BOSH is being used to transport XMPP, any fatal XMPP stream error conditions experienced between the connection manager and the XMPP server SHOULD only be reported using the "remote-stream-error" condition as described below.

Listing 32: Remote connection failed error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 106

<body type='terminate'
      condition='remote-connection-failed'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

The following values of the 'condition' attribute are defined:

Condition	Purpose
bad-request*	The format of an HTTP header or binding element received from the client is unacceptable (e.g., syntax error).
host-gone	The target domain specified in the 'to' attribute or the target host or port specified in the 'route' attribute is no longer serviced by the connection manager.
host-unknown	The target domain specified in the 'to' attribute or the target host or port specified in the 'route' attribute is unknown to the connection manager.
improper-addressing	The initialization element lacks a 'to' or 'route' attribute (or the attribute has no value) but the connection manager requires one.
internal-server-error	The connection manager has experienced an internal error that prevents it from servicing the request.
item-not-found*	(1) 'sid' is not valid, (2) 'stream' is not valid, (3) 'rid' is larger than the upper limit of the expected window, (4) connection manager is unable to resend response, (5) 'key' sequence is invalid.

Condition	Purpose
other-request	Another request being processed at the same time as this request caused the session to terminate.
policy-violation*	The client has broken the session rules (polling too frequently, requesting too frequently, sending too many simultaneous requests).
remote-connection-failed	The connection manager was unable to connect to, or unable to connect securely to, or has lost its connection to, the server.
remote-stream-error	Encapsulates an error in the protocol being transported.
see-other-uri	The connection manager does not operate at this URI (e.g., the connection manager accepts only SSL or TLS connections at some https: URI rather than the http: URI requested by the client). The client can try POSTing to the URI in the content of the <uri/> child element.
system-shutdown	The connection manager is being shut down. All active HTTP sessions are being terminated. No new sessions can be created.
undefined-condition	The error is not one of those defined herein; the connection manager SHOULD include application-specific information in the content of the <body/> wrapper.

* If the client did not include a 'ver' attribute in its session creation request then the connection manager SHOULD send a *deprecated* HTTP Error Condition instead of this terminal binding condition. If the connection manager did not include a 'ver' attribute in its session creation response then the client SHOULD expect it to send a *deprecated* HTTP Error Condition instead of this terminal binding condition.
The following is an example of a "see-other-uri" condition:

Listing 33: See other URI error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 144

<body condition='see-other-uri'
      type='terminate'
      xmlns='http://jabber.org/protocol/httpbind'>
  <uri>https://secure.jabber.org/xmppcm</uri>
</body>
```

The following is an example including a "remote-stream-error" condition:

Listing 34: Remote error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: 526

<body condition='remote-stream-error'
      type='terminate'
      xmlns='http://jabber.org/protocol/httpbind'
      xmlns:stream='http://etherx.jabber.org/streams'>
  <message from='contact@example.com'
           to='user@example.com'
           xmlns='jabber:client'>
    <body>I said "Hi!"</body>
  </message>
  <stream:error>
    <xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
    <text xmlns='urn:ietf:params:xml:ns:xmpp-streams'
          xml:lang='en'>
      Some special application diagnostic information!
    </text>
    <escape-your-data xmlns='application-ns'/>
  </stream:error>
</body>
```

Naturally, the client MAY report binding errors to the connection manager as well, although this is unlikely.

17.3 Recoverable Binding Conditions

In any response it sends to the client, the connection manager MAY return a recoverable error by setting a 'type' attribute of the <body/> element to "error". These errors do not imply that the HTTP session is terminated.
If it decides to recover from the error, then the client MUST repeat the HTTP request that resulted in the error, as well as all the preceding HTTP requests that have not received responses. The content of these requests MUST be identical to the <body/> elements of the original requests. This enables the connection manager to recover a session after the previous request was lost due to a communication failure.

Listing 35: Recoverable error

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 65

<body type='error'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

17.4 XML Payload Conditions

Application-level error conditions described in the documentation of the protocol being transported are routed to the client through the connection manager. They are transparent to the connection manager, and therefore out of scope for the transport binding defined herein.

18 Implementation Notes

18.1 HTTP Pipelining

HTTP pipelining allows a client to send multiple requests over the same HTTP socket connection without waiting for the corresponding responses. However, RFC 2616 notes that only idempotent methods should be allowed to use HTTP pipelining, which does not include the POST method used extensively by BOSH. Furthermore, there is no guarantee that pipelining will succeed because intermediate proxies might not support it. Therefore, clients and connection managers SHOULD NOT use HTTP Pipelining.

19 Security Considerations

19.1 Connection Between Client and BOSH Service

All communications between a client and a BOSH service SHOULD occur over encrypted HTTP connections. Negotiation of encryption between the client and the connection manager SHOULD occur at the transport layer or the HTTP layer, not the application layer; such negotiation SHOULD follow the HTTP/SSL protocol defined in SSL [28], although MAY follow the HTTP/TLS protocol defined in RFC 2818 [29] or the TLS Within HTTP protocol defined in RFC 2817 [30].
If the HTTP connection used to send the initial session request is encrypted, then all the other HTTP connections used within the session MUST also be encrypted. Furthermore, if authentication certificates are exchanged when establishing the encrypted connection that is used to send the initial session request, then the client and/or connection manager SHOULD ensure that the same authentication certificates are employed for all subsequent connections used by the session. Once such a ”secure session” has been established:

- If the connection manager refuses to establish an encrypted connection or offers a different certificate, then the client SHOULD close the connection and terminate the session without sending any more requests.

[28] SSL V3.0 <http://wp.netscape.com/eng/ssl3/draft302.txt>.
[29] RFC 2818: HTTP Over TLS <http://tools.ietf.org/html/rfc2818>.
[30] RFC 2817: Upgrading to TLS Within HTTP/1.1 <http://tools.ietf.org/html/rfc2817>.

- If the client sends a wrapper element that is part of a ”secure session” over a connection that either is not encrypted or uses a different certificate, then the connection manager SHOULD simply close the connection. The connection manager SHOULD NOT terminate the session since that would facilitate denial of service attacks.

19.2 Connection Between BOSH Service and Application

A BOSH service SHOULD encrypt its connection to the backend application using appropriate technologies such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), and StartTLS if supported by the backend application. Alternatively, the BOSH service can be considered secure (1) if it is running on the same physical machine as the backend application or (2) if it running on the same private network as the backend application and the administrators are sure that unknown individuals or processes do not have access to that private network.
If data privacy is desired, the client SHOULD encrypt its messages using an application-specific end-to-end encryption technology, because there is no way for the client to be sure that the BOSH service encrypts its connection to the application; methods for doing so are outside the scope of this specification.

19.3 Unpredictable SID and RID

The session identifier (SID) and initial request identifier (RID) are security-critical and therefore MUST be both unpredictable and nonrepeating (see RFC 1750 [31] for recommendations regarding randomness of SIDs and initial RIDs for security purposes).

19.4 Use of SHA-1

Recent research has shown that in select cases it is possible to compromise the hashes produced by the SHA-1 hashing algorithm (see RFC 4270 [32]). However, the use to which SHA-1 is put in BOSH will likely minimize the applicability of the attacks described in the literature. Furthermore, current estimates suggest that even with the recently-discovered attack, it would still take one year of computing by a government-sized entity to produce a collision.

20 IANA Considerations

TCP port 5280, conventially used for communication between BOSH clients and BOSH connection mangers, is registered with the Internet Assigned Numbers Authority (IANA) [33] in its

[31] RFC 1750: Randomness Recommendations for Security <http://tools.ietf.org/html/rfc1750>.
[32] RFC 4270: Attacks on Cryptographic Hashes in Internet Protocols <http://tools.ietf.org/html/rfc4270>.
[33] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

port registry at IANA Port Numbers Registry [34], with a keyword of "xmpp-bosh". (Although use of this port is OPTIONAL, it is helpful to define this port in a standardized way so that BOSH clients can contact any given XMPP service via BOSH without the need either for DNS TXT records as described in Discovering Alternative XMPP Connection Methods (XEP-0156) [35] or for more advanced methods such as U-NAPTR.

21 XMPP Registrar Considerations

21.1 Protocol Namespaces

The XMPP Registrar includes 'http://jabber.org/protocol/httpbind' in its registry of protocol namespaces.

22 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    xmlns:stream='http://etherx.jabber.org/streams'
    targetNamespace='http://jabber.org/protocol/httpbind'
    xmlns='http://jabber.org/protocol/httpbind'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0124: http://www.xmpp.org/extensions/xep-0124.html
    </xs:documentation>
  </xs:annotation>

  <xs:import namespace='http://www.w3.org/XML/1998/namespace'
             schemaLocation='http://www.w3.org/2001/03/xml.xsd'/>

  <xs:element name='body'>
    <xs:complexType>
      <xs:choice>
        <xs:element name='uri'
                minOccurs='0'
                maxOccurs='1'
                type='xs:string'/>
```

[34] IANA registry of port numbers <http://www.iana.org/assignments/port-numbers>.
[35] XEP-0156: Discovering Alternative XMPP Connection Methods <http://xmpp.org/extensions/xep-0156.html>.

```
      <xs:any namespace='##other'
              minOccurs='0'
              maxOccurs='unbounded'
              processContents='lax'/>
    </xs:choice>
    <xs:attribute name='accept' type='xs:string' use='optional'/>
    <xs:attribute name='ack' type='xs:positiveInteger' use='optional
       '/>
    <xs:attribute name='authid' type='xs:string' use='optional'/>
    <xs:attribute name='charsets' type='xs:NMTOKENS' use='optional'/
       >
    <xs:attribute name='condition' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='bad-request'/>
          <xs:enumeration value='host-gone'/>
          <xs:enumeration value='host-unknown'/>
          <xs:enumeration value='improper-addressing'/>
          <xs:enumeration value='internal-server-error'/>
          <xs:enumeration value='item-not-found'/>
          <xs:enumeration value='other-request'/>
          <xs:enumeration value='policy-violation'/>
          <xs:enumeration value='remote-connection-failed'/>
          <xs:enumeration value='remote-stream-error'/>
          <xs:enumeration value='see-other-uri'/>
          <xs:enumeration value='system-shutdown'/>
          <xs:enumeration value='undefined-condition'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name='content' type='xs:string' use='optional'/>
    <xs:attribute name='from' type='xs:string' use='optional'/>
    <xs:attribute name='hold' type='xs:unsignedByte' use='optional'/
       >
    <xs:attribute name='inactivity' type='xs:unsignedShort' use='
       optional'/>
    <xs:attribute name='key' type='xs:string' use='optional'/>
    <xs:attribute name='maxpause' type='xs:unsignedShort' use='
       optional'/>
    <xs:attribute name='newkey' type='xs:string' use='optional'/>
    <xs:attribute name='pause' type='xs:unsignedShort' use='optional
       '/>
    <xs:attribute name='polling' type='xs:unsignedShort' use='
       optional'/>
    <xs:attribute name='report' type='xs:positiveInteger' use='
       optional'/>
    <xs:attribute name='requests' type='xs:unsignedByte' use='
       optional'/>
```

```
      <xs:attribute name='rid' type='xs:positiveInteger' use='optional
         '/>
      <xs:attribute name='route' type='xs:string' use='optional'/>
      <xs:attribute name='sid' type='xs:string' use='optional'/>
      <xs:attribute name='stream' type='xs:string' use='optional'/>
      <xs:attribute name='time' type='xs:unsignedShort' use='optional'
         />
      <xs:attribute name='to' type='xs:string' use='optional'/>
      <xs:attribute name='type' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='error'/>
            <xs:enumeration value='terminate'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='ver' type='xs:string' use='optional'/>
      <xs:attribute name='wait' type='xs:unsignedShort' use='optional'
         />
      <xs:attribute ref='xml:lang' use='optional'/>
      <xs:anyAttribute namespace='##other' processContents='lax'/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

23 Acknowledgements

Thanks to Dave Cridland, Mike Cumings, Tomas Karasek, Steffen Larsen, Tobias Markmann, Matt Miller, Chris Seymour, Safa Sofuoğlu, Stefan Strigler, Mike Taylor, Winfriend Tilanus, Matthew Wild, Kevin Winters, and Christopher Zorn for their feedback.