Bidirectional Web Transfer Protocol -- BWTP/1.0
draft-wilkins-hybi-bwtp-00

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 22, 2010.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

The Bidirectional Web Transfer Protocol (BWTP) is an application-level protocol that can be used to upgrade a HTTP/1.1 connection, as defined by [RFC2616] to a bidirectional channel capable of exchanging
It is intended that BWTP can be used as a transport protocol for the HTML 5 Websocket API, as well as for other uses.

Table of Contents

1. Introduction ..................................................... 4
   1.1. Purpose ................................................... 4
2. Terminology ...................................................... 4
3. Overall Operation ................................................ 6
   3.1. Intermediaries .............................................. 7
   3.2. Channel Overview .......................................... 8
   3.3. Message Overview .......................................... 9
   3.4. A Simple Example ......................................... 10
4. Notational Conventions and Generic Grammar ....................... 11
   4.1. Augmented BNF ............................................. 11
   4.2. Basic Rules .............................................. 11
5. Protocol Parameters .............................................. 11
   5.1. BWTP Versions ............................................. 11
   5.2. Uniform Resource Identifiers ............................... 12
   5.3. Character Sets ........................................... 12
   5.4. Missing Charset .......................................... 12
   5.5. Content Codings ........................................... 12
   5.6. Media Types .............................................. 12
6. Frames .......................................................... 12
   6.1. Header Frames ............................................. 13
      6.1.1. OPEN Header Frame ................................... 14
      6.1.2. OPENED Header Frame ................................ 15
      6.1.3. MESSAGE Header Frame ................................. 16
      6.1.4. CLOSE Header Frame .................................. 16
      6.1.5. CLOSED Header Frame ................................ 16
      6.1.6. Message Frames ..................................... 16
7. Messages ........................................................ 17
   7.1. Single Frame Messages .................................... 17
   7.2. Multi Frame Messages ..................................... 17
   7.3. Message Meta Data ....................................... 18
8. Channels ........................................................ 19
   8.1. Opening Channels ........................................ 20
   8.2. Closing Channels ....................................... 20
9. Connection ...................................................... 21
   9.1. Opening Connections ..................................... 22
      9.1.1. Upgrade Request ..................................... 22
      9.1.2. Upgrade Response .................................... 22
      9.1.3. Direct Connection .................................. 23
   9.2. Flow Control ............................................ 23
   9.3. Closing Connections ..................................... 23
      9.3.1. Spontaneous Connection close ......................... 23
      9.3.2. Orderly Connection close ............................. 23
9.3.3. Idle Connection close .............................. 24
9.4. Proxy Connections ...................................... 24
10. Message Meta Data ........................................ 24
11. Access Authentication ..................................... 24
12. Security Considerations .................................. 24
13. IANA Considerations ..................................... 25
14. Acknowledgments .......................................... 25
15. IANA Considerations ..................................... 25
16. Security Considerations .................................. 25
17. Appendices .................................................. 25
    17.1. Appendix B. Mapping to the WebSocket API. ........... 25
    17.2. Appendix B. Carrying HTTP request and reponses ...... 26
18. Normative References ..................................... 27
Authors’ Addresses ............................................. 27
1. Introduction

1.1. Purpose

This IETF draft is intended for further discussion in the hybi@ietf.org mailing list for "Bidirectional communication for hypertext".

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990, but it is intrinsically a request/response protocol and cannot well service the growing use-case for data to be pushed from a server to a client.

The Bidirectional Web Transfer Protocol (BWTP) is an application-level protocol that can be used to upgrade a HTTP/1.1 connection to be able to exchange bidirectional MIME-like messages, containing meta information about the data transferred.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant".

This specification uses a number of terms to refer to the roles played by participants in, and objects of, the HTTP or BWTP communication.

connection: A transport layer virtual circuit established between two programs for the purpose of communication.
frame: The basic unit of BWTP communication, consisting of a structured sequence of octets matching the syntax defined in section 4 and transmitted via the connection.

message: The basic unit of application communication, consisting of one or more BWTP frames.

channel: A stream of BWTP frames for which a definite start, ordering and completion are defined.

request: A HTTP request message, as defined in section 5 of RFC 2616 [RFC2616].

response: A HTTP response message, as defined in section 6 of RFC 2616 [RFC2616].

resource: A network data object or service that can be identified by a URI, as defined in section 3.2. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

client: A program that establishes connections for the purpose of sending HTTP requests or upgrading to BWTP.

user agent: The client which initiates a connection. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.

server: An application program that accepts connections. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program’s capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each message.

origin server: The server on which a given resource resides or is to be created.

end point: Either an origin server or user agent.

proxy: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy MUST implement both the client and server requirements of this specification. A "transparent proxy" is a proxy that does not modify the request or response beyond what is required for proxy
authentication and identification. A "non-transparent proxy" is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering. Except where either transparent or non-transparent behavior is explicitly stated, the HTTP proxy requirements apply to both types of proxies.

gateway: A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.

tunnel: An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

upstream: Upstream describes the flow of a message from an upstream source to a downstream destination.

upstream/downstream: Downstream describes the flow of a message from an upstream source to a downstream destination.

inbound: Inbound applies to frames and messages that are sent towards the origin server.

outbound: Outbound applies to frames and messages that are sent towards the user agent.

3. Overall Operation

The BWTP protocol is designed to be used with the HTTP upgrade mechanism as defined by section 14.42 of [RFC2616]. A client makes an HTTP upgrade request to the server in order to establish a BWTP connection. After a successful upgrade, the BWTP connection implements two independent and asynchronous streams of BWTP length delimited frames, one inbound and one outbound.

BWTP communication usually takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude BWTP from being implemented on top of any other protocol on the Internet, or on other networks. BWTP only presumes a HTTP connection can be established over a reliable transport; any protocol that provides such guarantees can be used.
The following diagram shows how the User Agent (UA) initiates the BWTP connection by sending an upgrade request (an HTTP request) to the Origin Server (OS), over a connection (v):

UA -------------------v------------------- OS
    upgrade request -------------------->
    <----------------- 101 upgrade response
    <------------ BWTP Frames ------------>

Following the upgrade request-response, both server and client can immediately start to send each other frames. The inbound and outbound frames streams are independent and asynchronous (frames do not have responses).

3.1. Intermediaries

One or more intermediaries may be present in a chain between a User Agent (UA) and an Origin Server (OS). There are three common forms of intermediary: proxy, gateway, and tunnel. A proxy is a forwarding agent, receiving a message for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI. A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server’s protocol. A tunnel acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

The figure below shows three intermediaries (A, B, and C) between the user agent and origin server. All intermediaries must accept the initial upgrade request and pass it onwards towards the origin server. If any intermediary does not understand the upgrade request, the BWTP connection will not be established.

UA ----v---- A ----v---- B ----v---- C ----v---- OS
    upgrade request ----------------------------->
    <----------------------- 101 upgrade response
    <---------------- BWTP Frames -------------->
Any party to the communication which is not acting as a tunnel may inject frames into either the inbound or outbound stream, so long as the resulting stream of frames is legal with respect to this specification.

The following diagram illustrates if B has injected an inbound frame:

UA ----v---- A ----v---- B ----v---- C ----v---- OS
upgrade request ---------------------------->
<----------------------- 101 upgrade response
<------------- BWTP Frames --------------->
    ----inbound frame --->
<------------- BWTP Frames ------------->

3.2. Channel Overview

A BWTP connection can carry multiple channels and each frame contains a channel ID and frames for different channels may be interleaved. A channel must be opened with a header frame, which sets origin resource and the default meta-data (eg content type and character encoding) for the channel. Once opened, frames for the channel may be sent in both directions. A user agent, origin server or intermediary may close a channel, even if it did not open it.

The following diagram shows a possible interaction of frames for a single channel, where the end point that opened the channel eventually closes the channel:

UA|OS -------------------v------------------- OS|UA
OPEN channel 1 ----------------------->
<----------------------OPENED channel 1
<------ frames channel 1 ----------->
CLOSED channel 1 --------------------->
<--------------------- frames channel 1
<----------------------CLOSED channel 1

The following diagram shows a possible interaction of frames for two channels, where one end point opens two channels and then each endpoint closes one of the channels.
3.3. Message Overview

A BWTP message is sent in 1 or more message frames on a channel and may be preceded by a header frame containing meta-data. The minimal message is a single message frame containing binary data. Any meta-data (e.g., content-type) sent when the channel was opened will apply to the message.

The following diagram shows a single message frame being sent:

UA|OS -----------------v----------------- OS|UA
message frame channel 1 ------------>

Meta-data for a message may be sent in a header frame prior to the message frames. The following diagram shows a single frame message with meta data being sent:

UA|OS -----------------v----------------- OS|UA
header frame channel 1 ------------>
message frame channel 1 ------------>

A message may be transmitted as multiple message frames. Typically this is done so that a large message sent on a single channel will not unduly delay the transmission of small messages on other channels. The partial message frames for a message on a channel must be sent in order and they may not be interleaved with any other frames for the same channel, but may be interleaved with frames for other channels. The following diagram shows the transmission of a multi-frame message with meta data interleaved with frames for another channel.
3.4. A Simple Example

The following diagram shows the a BWTP connection being established, a single channel being opened and several messages being exchanged:

```
User-Agent | Origin-Server
-----------------------------------------------------------
GET /path HTTP/1.1
Host: host.com:80
Upgrade: BWTP/1.0
Connection: Upgrade

HTTP/1.1 101 BWTP/1.0

BWH 0 38 OPEN /chat/room
Content-Type: text/json;charset=utf-8
Accept-Language: en

BWH 0 32 OPENED
Content-Origin: www.mychat.com
Content-Language: en

BWM 0 46
{user="Bill" text="Bill has joined the room!"}

BWM 0 43
{user="Ted" text="Hello Bill, how are you"}

BWM 0 37
{user="Bill" text="I’m fine thanks"}

BWM 0 47
{user="System" text="the room is closed"}
BWH 0 0 CLOSED

BWH 0 0 CLOSED
```
4. Notational Conventions and Generic Grammar

4.1. Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus–Naur Form (BNF) as defined by section 2.1 of [RFC2616].

Implementers will need to be familiar with the notation in order to understand this specification.

4.2. Basic Rules

The Basic Rules as defined by section 2.2 of [RFC2616] are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986.

5. Protocol Parameters

5.1. BWTP Versions

BWTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further BWTP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values. The <minor> number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The <major> number is incremented when the format of a message within the protocol is changed. See RFC2145 for a fuller explanation.

All messages on a BWTP connection will be of the same version that is negotiated with the upgrade request and response that establishes the BWTP connection:

```
BWTP-Version   = "BWTP" /* 1*DIGIT "." 1*DIGIT
```

Note that the major and minor numbers MUST be treated as separate integers and that each MAY be incremented higher than a single digit. Thus, BWTP/2.4 is a lower version than BWTP/2.13, which in turn is lower than BWTP/12.3. Leading zeros MUST be ignored by recipients and MUST NOT be sent.
The BWTP version of a connection is the highest BWTP version for which all intermediaries and the agent and origin server can handle.

5.2. Uniform Resource Identifiers

BWTP uses URIs as defined by 3.2 of RFC2616 are used to identify BWTP resources.

5.3. Character Sets

BWTP uses the same definition of the term "character set" as that described in section 3.4 of RFC2616.

5.4. Missing Charset

If not otherwise specified, a character set of UTF-8 may be assumed.

5.5. Content Codings

The content codings defined by section 3.5 of RFC2616 and registered with the Internet Assigned Numbers Authority (IANA) MAY be applied to a BWTP message.

5.6. Media Types

HTTP uses Internet Media Types in the Content-Type header field in order to provide open and extensible data typing and type negotiation. The media-type is defined in section 3.7 of RFC2616 as:

```plaintext
media-type = type "/" subtype *( ";" parameter )
type = token
subtype = token
```

6. Frames

A BWTP frame is the basic unit of data exchange over a BWTP connection and is defined by:
Frames are either header frames containing meta data, or messages frames containing the content of the message.

6.1. Header Frames

A header frame is denoted with a "BWH" Frame-Type and carries meta-data. Header frames are defined as:
Header-Frame = Header-Line CRLF
               Header-Body

Header-Line = "BWH" SP
              Channel-ID SP
              Frame-Size SP
              Header-Fields
              URI

Header-Fields = "OPEN" URI
                | "OPENED"
                | "MESSAGE"
                | "CLOSE"
                | "CLOSED"
                | *(SP Frame-Field)

Header-Body = *( Message-Header CRLF )

Message-Header = Field-Name ":" [ Field-Value ]

Field-Name = Content-Header
             | token

Field-Value = *( Field-Content | LWS )

Field-Content = <the OCTETs making up the field-value
               and consisting of either *TEXT or
               combinations of token, separators,
               and quoted-string>

The header fields on the header line define a channel action and/or
meta data related to the channel. The header body of a header frame
contains message headers that represent meta data about subsequent
messages sent on the channel.

The channel ID of a frame is either an integer identifying a specific
channel or "*" which denotes all known channels.

6.1.1. OPEN Header Frame

An OPEN header frame has "OPEN" as the first header field and a URI
as the second. An OPEN header frame MUST be sent as the first frame
on a channel. A BWTP implementation MUST ignore any frames for a
channel that has not been opened. A BWTP implementation MUST ignore
any OPEN header frames sent on a channel that has already been
opened.

The URI in the open header frame identifies a resource such that
Inbound messages are sent to the resource and Outbound messages are sent from the resource. Relative URI’s are relative to the URI used in the upgrade request that established the connection. There is no requirement for the resources identified to be accessible via normal HTTP requests.

The message header fields of an OPEN header frame are the base meta-data for all messages sent on the channel, unless they are replaced by a similarly named message header field in a MESSAGE header frame. Typically the message header fields in a OPEN header frames are used to carry meta-data that is invariant for all messages in the channel.

The channel ID is an integer that is allocated by the opener of a Channel. To avoid ID collisions, User-Agents MUST always create even numbered channel IDs and Origin Servers MUST always create odd numbered channel IDs.

Below is an example OPEN header frame that opens channel 1 for the URI /greeting/service and defines the content type and language of the channel messages to be english in plain text encoded as iso-8859-1:

```
BWH 1 57 OPEN /greeting/service
Content-Type: text/plain;charset=iso-8859-1
Content-Language: en
```

6.1.2. OPENED Header Frame

An OPENED header frame has "OPENED" as the first header field and is sent in response to an OPEN hader frame. When a user agent or origin server receives an OPEN header frame, to accept the channel it MUST send an OPENED header frame with any additional or negotiated meta-data for the channel.

A BWTP implementation MUST ignore OPENED header frames that are not the first message sent for the channel by the end point that received the OPEN header frame.

Below is an example OPENED header frame that set’s an origin for the channel and a locale language:

```
BWH 0 32 OPENED
Content-Origin: www.mychat.com
Content-Language: en
```
6.1.3. MESSAGE Header Frame

A MESSAGE header frame has "MESSAGE" as the first header field. A MESSAGE header frame carries message headers which define meta data that applies to the next message sent on the channel. Message headers in a MESSAGE header frame are replace headers with the same name sent in the OPEN header frame. Typically MESSAGE header frames are used to carry message headers that vary from message to message (eg "Content-MD5" or "Transport-Encoding").

Below is an example MESSAGE header frame that sends a MD5 checksum for the following message.

BWH 1 38 MESSAGE
Content-MD5: 0bbbf28091f83b97246e8a9037a037d5

6.1.4. CLOSE Header Frame

A CLOSE header frame MAY be sent by an intermediary to initiate an orderly shutdown of a channel.

Below is a CLOSE header frame that initiates an orderly shutdown on all known channels:

BWH * 0 CLOSE

6.1.5. CLOSED Header Frame

A CLOSE header frame is sent by a user agent or origin server to either initiate or complete an orderly shutdown of a channel.

Below is a CLOSED header frame that completes an orderly shutdown on channel 3:

BWH 3 0 CLOSED

6.1.6. Message Frames

A message frame is denoted with a "BWM" Frame-Type and contains the binary data of either a complete or partial BWTP message. Message frames are defined as:
Message-Frame  = Message-Line CRLF
                Frame-Body [CRLF]

Message-Line    = "BWM" SP
                Channel-ID SP
                Frame-Size [ '/' Message-Size ]

Frame-Body      = *OCTECT

The message body is any finite sequence of octets. The interpretation of these bytes can be guided by the meta-data sent in the OPEN header frame and/or in a preceding MESSAGE header frame. A message frame with a message size in the message line is part of a multi-frame message.

7. Messages

BWTP messages are composed of an optional MESSAGE header frame followed by 1 or more message frames. The sequence of frames for a message is defined by


    Message-Header-Frame = <Header-Frame conforming to 4.1.1>

Frames other channels MAY be interleaved between the frames of a message.

7.1. Single Frame Messages

A message may be sent in a single message frame. A single frame message does not have message size in the frame start line and the actual message size is equal to the frame size.

Below is an example BWTP message frame that carries a single BTWP message on channel 1:

    BWM 1 11
    HelloWorld!

7.2. Multi Frame Messages

A message may be sent in multiple frames when a message size is provided in the start line. Typically this is done so that a large message sent on a single channel will not unduly delay the transmission of small messages on other channels. A BTWP implementation SHOULD segment a message into multiple frames if
transmission of the message as a significant frame would unduly delay the transmission of messages on other channels.

The message frames for a message MUST be sent in order and they MAY NOT be interleaved with any other frames for the same channel, but MAY be interleaved with frames for other channels. When the message size is known, then the message is complete when sufficient bytes have been sent. Below is an example 11 octet message that is carried in 2 message frames on channel 1:

```
BWM 1 5/11
Hello

BWM 1 6/11
World!
```

If the message size is not known to the sender, then "*" may be given as the message size and the end of the message is indicated by sending a zero length message frame, which SHOULD contain the now known message length. Below is an example 11 octet message that is sent with unknown message size:

```
BWM 1 5/*
Hello

BWM 1 6/*
World!

BWM 1 0/11
```

7.3. Message Meta Data

The meta data for a message consists of the message headers sent in the OPEN header frame combined with an optional MESSAGE header frame sent before each message. If a header field name exists in both the OPEN header frame and a MESSAGE header frame, then the value(s) from the MESSAGE header frame replace those in the OPEN message frame.

The following sequence of frames illustrates how message headers are combined:
In the sequence above: all messages have a Content-Type of "text/plain; charset=iso-8859-1"; the first, second and forth messages are in English; the third message is Italian; the second message has an MD5 checksum.

8. Channels

A channel represents a stream of related BWTP frames. A BWTP implementation MUST process messages with the same channel in the order they are received. There is no ordering guarantee given for messages in different channels. Header frames provide meta data that apply to subsequent frames on the same channel and BWTP messages may be partitioned into multiple frames on the same channel. Channels are created dynamically when an OPEN header frame is sent.

The sequence of possible frames sent by the opener of the channel is defined by:

```
BWTP-Channel-Opener = Open-Header-Frame
  *( BWTP-Message | Header-Frame )
[Closed-Header-Frame]
```

```
Open-Header-Frame = <Header-Frame conforming to 4.1.1>
```
8.1. Opening Channels

Either the user agent or the origin server MAY open a channel by sending an OPEN header frame on an open connection. A connection should only be used to open a channel if it has been newly established or if it has other open channels.

When a user agent or origin server receives an OPENED header frame, it can accept the new channel by sending an OPENED header frame with additional or negotiated meta data for the channel. If the new channel is not accepted, a CLOSED header frame MUST be sent.

The following example shows an user-agent opening a connection and the origin server setting additional and negotiated meta data:

```
User-Agent | Origin-Server
---------------------------------
BWH 0 38 OPEN /chat/room
Content-Type: text/json;charset=utf-8
Accept-Language: en,it,fr

BWH 0 32 OPENED
Content-Origin: www.myhost.com
Content-Language: it
```

A end point that opens a channel SHOULD wait until an OPENED frame is received before sending messages.

8.2. Closing Channels

Either the user agent or the origin server MAY initiate the close of a channel by sending a CLOSED header frame for the channel. An
intermediary MAY initiate the close of a channel by sending a CLOSE header frame on either the inbound or outbound channel.

If a user-agent or origin server receives a CLOSE header frame or CLOSED header frame for an open channel, then it MUST: stop accepting new messages from the application for the channel; flush all queued messages for the channel; send a CLOSED header frame. Any messages received after the CLOSED header should be handled normally.

If a user-agent or origin server receives a CLOSED header frame for a channel on which it has already sent a CLOSED header, then the channel can be considered closed and the implementation MUST ignore all other frames for that channel other than a new OPEN header frame for the channel.

The interaction below shows a user agent or origin server closing a channel:

```
UA|OS -------------------v------------------- OS|UA
<-------- frames channel 1 ----------->
CLOSED channel 1 --------------------->
<--------------------- frames channel 1
<----------------------CLOSED channel 1
```

The interaction below shows an intermediary I closing a channel:

```
UA|OS ------------v------------- I --------v-------- OS|UA
<-------- frames channel 1 ----------->
CLOSE channel 1 ---->
<------------------ frames channel 1
<------------------ CLOSED channel 1
frames channel 1 ---------------------------->
CLOSED channel 1 ---------------------------->
```

9. Connection

A BWTP connection is sequence of BWTP frames which MAY be separated by multiple CRLF characters:

```
BWTP-Connection  = *( *CRLF BWTP-Frame )
```
9.1. Opening Connections

The BWTP protocol is an HTTP upgrade protocol. A client makes an HTTP upgrade request to the server in order to establish a BWTP connection. After a successful upgrade, the BWTP connection implements two independent and asynchronous streams of BWTP frames, one inbound and one outbound.

9.1.1. Upgrade Request

A BWTP connection is initiated by a client issuing an upgrade request, which is a valid HTTP/1.1 GET request with an Upgrade header field set to the desired BWTP-Version:

```
BWTP-Upgrade-Request = "GET" SP URI SP "HTTP/1.1" CRLF *(message-header CRLF) "Upgrade"":" BWTP-Version *(message-header CRLF) CRLF [ message-body ]
```

Since Upgrade is only for the immediate connection, the Upgrade Request SHOULD also contain a Connection header with "Upgrade" in its values.

Below is an example of an Upgrade Request with basic authentication:

```
GET /path HTTP/1.1
Host: host.com:80
Upgrade: BWTP/1.0
Connection: Upgrade
Authorization: BASIC amd73ksosh3hj210sd3d0dk1
```

If an intermediary in the HTTP connection accept the upgrade, they MUST forward the Upgrade header in their requests towards the origin server.

9.1.2. Upgrade Response

When an origin server receives an Upgrade request and the resource specified by the URI accepts the upgrade, then acceptance is indicated by sending an Upgrade Response:
BWTP-Upgrade-Response = "HTTP/1.1" SP 101 SP "BWTP/1.0" CRLF
*(message-header CRLF)
"Upgrade" "BWTP-Version"
*(message-header CRLF)
CRLF

Below is an example of an Upgrade Response:

HTTP/1.1 101 BWTP/1.0
Content-Origin: the.origin.com

9.1.3. Direct Connection

It would be possible to support direct creation of BWTP connections with the use of a bwtp:// uri schema. A future version of this document may contain such a proposal.

9.2. Flow Control

BWTP servers SHOULD maintain connections and use TCP’s flow control mechanisms to resolve temporary overloads, rather than terminating connections with the expectation that clients will retry. The latter technique can exacerbate network congestion.

A BWTP implementation SHOULD NOT read from a connection more data than it has the capacity to handle. In practice this will mean imposing limits either on: the amount of unprocessed data; the number of unprocessed messages held; the number of channels with unprocessed messages; a combination of these limits.

9.3. Closing Connections

9.3.1. Spontaneous Connection close

Either the client or server or any intermediary MAY at any time close the BWTP connection. Message delivery is not guaranteed if a spontaneous close is performed. Clients, servers and intermediaries SHOULD use orderly connection close if guaranteed delivery is required.

9.3.2. Orderly Connection close

Either the user agent or origin server or any intermediary MAY initiate an orderly close of a BWTP connection by initializing the close of all known channels. Any OPEN header frames received during
orderly close should immediately be closed. Once the has closed all channels on the connection, the connection may be closed.

9.3.3. Idle Connection close

When the last open channel on a connection is close, the connection should be closed.

9.4. Proxy Connections

TBD

10. Message Meta Data

The meta data message headers that can be sent in a header frame are meta data related only to the message body and do not affect the protocol or transport. The headers are extensible and applications are able to define new headers.

Content-Header = "Content-Encoding"
                 | "Content-Language"
                 | "Content-Location"
                 | "Content-MD5"
                 | "Content-Origin"
                 | "Content-Type"
                 | "Access-Control-Allow-Origin"
                 | Token

11. Access Authentication

HTTP provides several OPTIONAL challenge-response authentication mechanisms which can be used by a server to challenge a client request and by a client to provide authentication information. The general framework for access authentication, and the specification of "basic" and "digest" authentication, are specified in RFC2617. This specification adopts the definitions of "challenge" and "credentials" from that specification.

12. Security Considerations

BWTP uses the origin model used by Web browsers to restrict which messages can be seen or sent by third party code running within a user agent. The origin may refer to either the source of a message
from an origin server or the source of the code executing within the user agent.

Unless overridden by a "Access-Control-Allow-Origin" header applied to a channel or to a message, the same origin security policy is applied.

13. IANA Considerations

This document has no actions for IANA.

14. Acknowledgments

Giants with large shoulders.

15. IANA Considerations

This document does not require any actions by the IANA.

16. Security Considerations

To follow.

17. Appendices

17.1. Appendix B. Mapping to the WebSocket API.

HTML5 defines the WebSocket javascript API for two-way communication with a remote host from a web browser. The API is defined as:
interface WebSocket {
  readonly attribute DOMString URL;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute long readyState;

  // networking
  attribute Function onopen;
  attribute Function onmessage;
  attribute Function onclose;
  void send(in DOMString data);
  void close();
};

Each instance of the WebSocket is mapped to a BWTP channel created by
the user agent:

+ when a WebSocket instance is created a new BWTP channel of
  Content-Type "text/plain;charset=utf-8" MUST be opened.

+ when the user agent receives an OPENED header frame, it MUST
call the onopen function of the associated WebSocket instance.

+ when the user agent receives a CLOSED header frame or the con-
  nection is closed, it MUST call the onclose function of the
  associated WebSocket instance.

+ when the user agent receives a message that is of a text mime
  type of a known encoding, it MUST call the onmessage function
  of the associated WebSocket instance.

+ when the send(in DOMString data) function is called, a message
  is sent on the associated channel

+ when the close() function is called, a CLOSE header frame for
  the channel is sent.

17.2. Appendix B. Carrying HTTP request and responses

HTTP request and responses may be carried by using header fields in
header frames to carry methods, status etc. The request/response
ordering of HTTP applies only within a given channel, so requests
issued on different channels may receive responses out of order. HTTP style pipelining is supported within a single channel, however it would be subject to the same limitations as pipelining over a single connection. The use of multiple channels for simultaneous requests is recommended rather than pipelining.

User-Agent | Origin-Server
-------------|------------------

```
BWH 0 24 HTTP GET /index.html
Accept-Encoding: gzip

BWH 0 24 HTTP 200 OK
Content-Type: text/html

BWM 0 27
<html><h1>hello</h1></html>
```

With such a mapping, a client and server could communicate both HTTP and Websocket API traffic over a single connection. It is anticipated that the full mapping of HTTP to BWTP will be the subject of another IETF draft.

18. Normative References


Authors’ Addresses

Greg Wilkins
Webtide

Email: gregw@webtide.com