Dispatcher IPC Interface

Version 3.0

Abstract

This document describes the “Dispatcher IPC Interface.” The dispatcher is a command line tool which the application launches in a separate process. The dispatcher manages the transports, as well as sending and receiving data over the network. The application sends and receives data by communicating with this process. This interface is most useful if the application is written in a different language than the transports or changes to the networking code in the application are not possible. A dispatcher application can in fact use the Transport API Interface internally, serving as a proxy process that the client application will communicate through. This is discussed in more detail in Section 4.1. A dispatcher implementation that wraps the Go implementation of the Transports API is available as a reference implementation [PT2-DISPATCHER].

Table of Contents

1. Dispatcher IPC Interface Specification

When the transport runs in a separate process from the application, the two components interact through an IPC interface. The IPC interface serves to ensure compatibility between applications and transports written in different languages.

1.1 Pluggable Transport Configuration Parameters

When using the IPC interface, Pluggable Transport proxy instances are configured by their parent process at launch time via a set of well-defined command line flags.

1.1.1. Common Configuration Parameters

When launching either a PT Client or PT Server Pluggable Transport, all of the common configuration parameters specified in section 1.1.1 are optional unless otherwise specified in the documentation for the specific parameter. Additional configuration parameters specific to PT Clients are specified in section 1.1.2 and configuration parameters specific to PT Servers are specified in section 1.1.3.

-ptversion

Specifies the versions of the Pluggable Transport specification the parent process supports, delimited by commas. All PTs MUST accept any well-formed list, as long as a compatible version is present.

Valid versions MUST consist entirely of non-whitespace, non-comma printable ASCII characters.

The version of the Pluggable Transport specification as of this document is "3.0".

Examples

```
shapeshifter-dispatcher -ptversion 1,1a,2.2,this_is_a_valid_version
```

-state

Specifies a path to a directory where the PT is allowed to store state that will be persisted across invocations. This can be either an absolute path or a relative path. If a relative path is used, it is assumed to be relative to the current directory. The directory is not required to exist when the PT is launched, however PT implementations SHOULD be able to create it as required.

If -state flag is not specified, PT proxies MUST use the current working directory of the PT process as the state location.

PTs MUST only store files in the path provided, and MUST NOT create or modify files elsewhere on the system.

Examples

```
shapeshifter-dispatcher -state /var/lib/pt_state/
```

-exit-on-stdin-close

Specifies that the parent process will close the PT proxy's standard input (stdin) stream to indicate that the PT proxy should gracefully exit.

PTs MUST NOT treat a closed stdin as a signal to terminate unless this flag is present and is set to "1".

PTs SHOULD treat stdin being closed as a signal to gracefully terminate if this flag is set to "1".

Example

```
shapeshifter-dispatcher -exit-on-stdin-close
```

-ipcLogLevel

Controls what log messages are sent from the dispatcher to the application using LOG messages.

The log level MUST be one of the following:

- NONE
- ERROR
- WARN
- INFO
- DEBUG

Logging at the same log level or above will be sent. For instance, if the -ipcLogLevel is set to ERROR then only ERROR messages will be sent to the application, whereas if the -ipcLogLevel is set to INFO then ERROR, WARN, and INFO (but not DEBUG) messages will be sent to the application. The NONE log level is a special case which indicates that no LOG messages should be sent to the application.

The default log level is NONE.

Example

```
-ipcLogLevel DEBUG
```

-transport

Specifies the name of the PT to use.

The application MUST set either a single transport with -transport or a list of transports with -transports. The application MUST NOT set both a single transport and a list of transports simultaneously.

Example

```
shapeshifter-dispatcher -transport shadow
```

-optionsFile

Specifies the path to a file containing the transport options. This path can be either an absolute path or a relative path. If a relative path is used, it is relative to the current directory.

The contents of the file MUST be in the same format as the argument to the -options parameter.

The application MUST NOT specify both -options and -optionsFile simultaneously.

-mode

Sets the proxy <mode>.

The mode MUST be one of the following:

- transparent-TCP
- transparent-UDP
- socks5
- STUN

The default if no mode is specified is socks5.

1.1.2. Pluggable PT Client Configuration Parameters

When launching either a PT Client, the common configuration parameters specified in section 1.1.1 as well as the client-specific configuration parameters specified in section 1.1.2 are optional unless otherwise specified in the documentation for the specific parameter.

-client

Specifies that the PT proxy should run in client mode.

If neither -client or -server is specified, the PT proxy MUST launch in client mode.

Example

```
shapeshifter-dispatcher -client
```

-transports

Specifies the PT protocols the client proxy should initialize, as a comma separated list of PT names.

PTs SHOULD ignore PT names that it does not recognize.

The application MUST set either a single transport with -transport or a list of transports with -transports. The application MUST NOT set both a single transport and a list of transports simultaneously.

Example

```
shapeshifter-dispatcher -transports obfs2,obfs4
```

-proxylistenaddr

This flag specifies the <address>:<port> on which the dispatcher client should listen for incoming application client connections. When this flag is used, the dispatcher client will use this address and port instead of making its own choice.

The <address>:<port> combination MUST be an IP address supported by `bind()`, and MUST NOT be a host name.

Applications MUST NOT set more than one <address>:<port> pair.

A combined address and port can be set using -proxylistenaddr. Alternatively, -proxylistenhost and -proxylistenport can be used to separately set the address and port respectively. If a combined address and port is specified then a separate host and port cannot be specified and vice versa.

Example

```
shapeshifter-dispatcher -proxylistenaddr 127.0.0.1:5555
```

-proxylistenhost

This flag specifies the <address> on which the dispatcher client should listen for incoming application client connections. When this flag is used, the dispatcher client will use this address instead of making its own choice.

The <address> combination MUST be an IP address supported by `bind()`, and MUST NOT be a host name.

Applications MUST NOT set more than one <address>:<port> pair.

A combined address and port can be set using -proxylistenaddr. Alternatively, -proxylistenhost and -proxylistenport can be used to separately set the address and port respectively. If a combined address and port is specified then a separate host and port cannot be specified and vice versa.

Example

```
shapeshifter-dispatcher -proxylistenhost 127.0.0.1 -proxylistenport 5555
```

-proxylistenport

This flag specifies the <port> on which the dispatcher client should listen for incoming application client connections. When this flag is used, the dispatcher client will use this port instead of making its own choice.

The <address>:<port> combination MUST be an IP address supported by `bind()`, and MUST NOT be a host name.

Applications MUST NOT set more than one <address>:<port> pair.

A combined address and port can be set using -proxylistenaddr. Alternatively, -proxylistenhost and -proxylistenport can be used to separately set the address and port respectively. If a combined address and port is specified then a separate host and port cannot be specified and vice versa.

Example

```
shapeshifter-dispatcher -proxylistenhost 127.0.0.1 -proxylistenport 5555
```

-proxy

Specifies an upstream proxy that the PT MUST use when making outgoing network connections. It is a URI [RFC3986] of the format:

<proxy_type>://[<user_name>[:<password>][@]<ip>:<port>.

The -proxy command line flag is OPTIONAL and MUST be omitted if there is no need to connect via an upstream proxy.

Examples

```
shapeshifter-dispatcher -proxy http://198.51.100.3:443
```

1.1.3. Pluggable PT Server Configuration Parameters

When launching a PT Server, the common configuration parameters specified in section 1.1.1 as well as the server-specific configuration parameters specified in section 1.1.3 are optional unless otherwise specified in the documentation for the specific parameter.

-server

Specifies that the PT proxy should run in server mode.

If neither -client or -server is specified, the PT proxy MUST launch in client mode.

Example

```
shapeshifter-dispatcher -server
```

-transports

Specifies the PT protocols the server proxy should initialize, as a comma separated list of PT names.

PTs SHOULD ignore PT names that it does not recognize.

Parent processes MUST set either -transport or -transports when launching a server-side PT reverse proxy instance.

Example

```
shapeshifter-dispatcher -transports obfs4,shadow
```

-options

Specifies per-PT protocol configuration directives, as a JSON string value with options that are to be passed to the transport.

If there are no arguments that need to be passed to any of PT transport protocols, -options MAY be omitted.

If a PT Client requires a server address, then this can be communicated by way of the transport options. For consistency, transports SHOULD name this option “serverAddress” and it SHOULD have a format of <address>:<port>. Unless otherwise specified in the documentation of the specific transport being used, the address can be an IPv4 IP address, an IPv6 IP address, or a domain name. Not all transports require a server address and some will require multiple server addresses, so this convention only applies to the case where the transport requires a single server address.

The application MUST NOT specify both -options and -optionsFile simultaneously.

Example

```
shapeshifter-dispatcher -options ”{shadow: {password: \”password\”, cipherName:
\“AES-128-GC<\”}}”
```

-bindaddr

A comma separated list of <key>-<value> pairs, where <key> is a PT name and <value> is the <address>:<port> on which it should listen for incoming client connections.

The keys holding transport names MUST be in the same order as they appear in -transports.

The <address> MAY be a locally scoped address as long as port forwarding is done externally.

The <address>:<port> combination MUST be an IP address supported by `bind()`, and MUST NOT be a host name.

Applications MUST NOT set more than one <address>:<port> pair per PT name.

The bind address MUST be set, either using a combined -bindaddr flag or in separate parts using -transport, -bindhost, and -bindport.

If a combined -bindaddr is used then -transport, -bindhost, and -bindport MUST NOT be used. Similarly, if -transport, -bindhost, or -bindport is used then -bindaddr MUST NOT be used.

Example

```
shapeshifter-dispatcher -bindaddr obfs4-198.51.100.1:1984,shadow-127.0.0.1:4891
```

-bindhost

Specifies the <address> part of the server bind address when used in conjunction with -transport and -bindport.

The <address> MAY be a locally scoped address as long as port forwarding is done externally.

The <address> MUST be an IP address supported by `bind()`, and MUST NOT be a host name.

Applications MUST NOT set more than one <address> using -bindhost.

The bind address MUST be set, either using a combined -bindaddr flag or in separate parts using -transport, -bindhost, and -bindport.

If a combined -bindaddr is used then -transport, -bindhost, and -bindport MUST NOT be used. Similarly, if -transport, -bindhost, or -bindport is used then -bindaddr MUST NOT be used.

If -bindhost is specified, then -transport and -bindport must also be used.

-bindport

Specifies the <port> part of the server bind address when used in conjunction with -transport and -bindhost.

Applications MUST NOT set more than one <port> using -bindport.

The bind port MUST be set, either using a combined -bindaddr flag or in separate parts using -transport, -bindhost, and -bindport.

If a combined -bindaddr is used then -transport, -bindhost, and -bindport MUST NOT be used. Similarly, if -transport, -bindhost, or -bindport is used then -bindaddr MUST NOT be used.

If -bindport is specified, then -transport and -bindhost must also be used.

-target

Specifies the destination that the PT reverse proxy should forward traffic to after transforming it as appropriate, as an <address>:<port>. Unless otherwise specified in the documentation of the specific transport being used, the address can be an IPv4 IP address, an IPv6 IP address, or a domain name.

Connections to the target destination MUST only contain application payload. If the parent process requires the actual source IP address of client connections (or other metadata), it should set -extorport instead.

The target destination MUST be set. A combined address and port can be set using -target. Alternatively, -targethost and -targetport can be used to separately set the address and port respectively. If a combined address and port is specified then a separate host and port cannot be specified and vice versa.

Examples

```
shapeshifter-dispatcher -target 127.0.0.1:9001
shapeshifter-dispatcher -target 93.184.216.34:9001
shapeshifter-dispatcher -target
[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:1122
shapeshifter-dispatcher -target example.com:9922
```

-targethost

Specifies the <address> of the destination that the PT reverse proxy should forward traffic to after transforming it as appropriate. Unless otherwise specified in the documentation of the specific transport being used, the address can be an IPv4 IP address, an IPv6 IP address, or a domain name.

Connections to the target destination MUST only contain application payload. If the parent process requires the actual source IP address of client connections (or other metadata), it should set -extorport instead.

The target destination MUST be set. A combined address and port can be set using -target. Alternatively, -targethost and -targetport can be used to separately set the address and port respectively. If a combined address and port is specified then a separate host and port cannot be specified and vice versa.

If -targethost is specified, then -targetport must also be specified.

Example

```
shapeshifter-dispatcher -targethost 93.184.216.34 -targetport 9001
```

-targetport

Specifies the <port> of the destination that the PT reverse proxy should forward traffic to after transforming it as appropriate.

Connections to the target destination MUST only contain application payload. If the parent process requires the actual source IP address of client connections (or other metadata), it should set -extorport instead.

The target destination MUST be set. A combined address and port can be set using -target. Alternatively, -targethost and -targetport can be used to separately set the address and port respectively. If a combined address and port is specified then a separate host and port cannot be specified and vice versa.

If -targetport is specified, then -targethost must also be specified.

Example

```
shapeshifter-dispatcher -targethost 93.184.216.34 -targetport 9001
```

-extorport

Specifies the destination that the PT reverse proxy should forward traffic to, via the Extended ORPort protocol [EXTORPORT] as an <address>:<port>.

The Extended ORPort protocol allows the PT reverse proxy to communicate per-connection metadata such as the PT name and client IP address/port to the parent process.

Example

```
shapeshifter-dispatcher -extorport 127.0.0.1:4200
```

-authcookie

Specifies an absolute filesystem path to the Extended ORPort authentication cookie, required to communicate with the Extended ORPort specified via -extorport.

If the parent process is not using the ExtORPort protocol for incoming traffic, -authcookie MUST be omitted.

Example

```
shapeshifter-dispatcher -authcookie
/var/lib/extended_orport_auth_cookie
```

1.2. Pluggable Transport To Parent Process Communication

When using the IPC method to manage a PT in a separate process, in addition to command line flags, a custom protocol is also used to communicate between the application parent process and PT sub-process. This protocol is communicated over the stdin/stdout channel between the processes. This is a text-based, line-based protocol using newline-terminated lines. Lines in the protocol conform to the following grammar:

```
<Line> ::= <Keyword> <OptArgs> <NL>
<Keyword> ::= <KeywordChar> | <Keyword> <KeywordChar>
<KeywordChar> ::= <any US-ASCII alphanumeric, dash, and underscore>
<OptArgs> ::= <Args>*
<Args> ::= <SP> <ArgChar> | <Args> <ArgChar>
<ArgChar> ::= <any US-ASCII character but NUL or NL>
<SP> ::= <US-ASCII whitespace symbol (32)>
<NL> ::= <US-ASCII newline (line feed) character (10)>
```

The parent process MUST ignore lines received from PT proxies with unknown keywords.

1.2.1. Common Messages

IPC messages specified in section 1.2.1 are common to both clients and servers.
When a PT proxy first starts up, if the -ptversion flag is set, then the proxy MUST determine which version of the Pluggable Transports Specification is being used, to ensure that it is compatible.

Upon determining the version to use, or lack thereof, the PT proxy responds with one of two messages: VERSION-ERROR or VERSION. If the application does not specify a list of compatible versions, then the PT proxy MUST respond with a VERSION message. The version reported SHOULD be the latest version supported by the PT proxy.

VERSION-ERROR <ErrorMessage>

The "VERSION-ERROR" message is used to signal that there was no compatible Pluggable Transport Specification version present in the -ptversion list.

The <ErrorMessage> SHOULD be set to "no-version" for historical reasons but MAY be set to a useful error message instead.

As this is an error, this message is written to STDERR.

PT proxies MUST terminate with the exit code EX_CONFIG (78) after outputting a "VERSION-ERROR" message.

Examples

```
VERSION-ERROR no-version
```

VERSION <ProtocolVersion>

The "VERSION" message is used to signal the Pluggable Transport Specification version that the PT proxy will use to configure it's transports and communicate with the parent process.

The version specified by this document is "3.0".

PT proxies MUST either report an error and terminate, or output a "VERSION" message before moving on to client/server proxy initialization and configuration.

This message is written to STDOUT.

Examples

```
VERSION 3.0
```

After version negotiation has been completed the PT proxy MUST then validate that all of the required command line flags are provided, and that all of the configuration values supplied are well formed.

At any point, if there is an error encountered related to configuration supplied via the command line flags, it MAY respond with an error message and terminate.

ENV-ERROR <ErrorMessage>

The "ENV-ERROR" message is used to signal the PT proxy's failure to parse the configuration command line flags (3.2).

The <ErrorMessage> SHOULD consist of a useful error message that can be used to diagnose and correct the root cause of the failure.

As this is an error, this message is written to STDERR.

PT proxies MUST terminate with error code EX_USAGE (64) after outputting a "ENV-ERROR" message.

Examples

```
ENV-ERROR No -authcookie when -extorport set
```

LOG <LogLevel> <Message>

The “LOG” message is used to carry logging information from the dispatcher to the application.

The LogLevel parameter MUST be one of the following:

- ERROR
- WARN
- INFO
- DEBUG

The Message parameter MUST be a UTF-8 encoded string.

Examples

```
LOG DEBUG This is an example debug log message.
```

1.2.2. Pluggable PT Client Messages

IPC messages specified in section 1.2.2 are specific to PT clients.
After negotiating the Pluggable Transport Specification version, PT client proxies MUST first validate the -proxy flag if it is set, before initializing any transports.

If an upstream proxy is provided, PT client proxies MUST respond with a message indicating that the proxy is valid, supported, and will be used OR a failure message.

PROXY DONE

The "PROXY DONE" message is used to signal the PT proxy's acceptance of the upstream proxy specified by -proxy.

This message is written to STDOUT.

PROXY-ERROR <ErrorMessage>

The "PROXY-ERROR" message is used to signal that the upstream proxy is malformed/unsupported or otherwise unusable.

As this is an error, this message is written to STDERR.

PT proxies MUST terminate immediately with error code EX_UNAVAILABLE (69) after outputting a "PROXY-ERROR" message.

Example

```
PROXY-ERROR SOCKS 4 upstream proxies unsupported.
```

After the upstream proxy (if any) is configured, PT clients then iterate over the requested transports in "TOR_PT_CLIENT_TRANSPORTS" and initialize the listeners.

For each transport initialized, the PT proxy reports the listener status back to the parent via messages to stdout and error messages to stderr.

CMETHOD <transport> <'socks5',’transparent-TCP’,’transparent-UDP’,’STUN’> <address:port>

The "CMETHOD" message is used to signal that a requested PT transport has been launched, the protocol which the parent should use to make outgoing connections, and the IP address and port that the PT transport's forward proxy is listening on.

This message is written to STDOUT.

Examples

```
CMETHOD Replicant socks5 127.0.0.1:19999
CMETHOD meeklite transparent-TCP [::1]:19999
CMETHOD shadow transparent-UDP [::1]:1234
CMETHOD obfs4 STUN 127.0.0.1:8888
```

CMETHOD-ERROR <transport> <ErrorMessage>

The "CMETHOD-ERROR" message is used to signal that requested PT transport was unable to be launched.

As this is an error, this message is written to STDERR.

Outputting a "CMETHOD-ERROR" does not result in termination of the PT process, as even if one transport fails to be initialized, other transports may initialize correctly.

Examples

```
CMETHOD-ERROR shadow No password specified in configuration.
```

Once all PT transports have been initialized (or have failed), the PT proxy MUST send a final message indicating that it has finished initializing.

CMETHODS DONE

The "CMETHODS DONE" message signals that the PT proxy has finished initializing all of the transports that it is capable of handling.

This message is written to STDOUT.

Upon sending the "CMETHODS DONE" message, the PT proxy initialization is complete.

1.2.2.1. Notes

Unknown transports in the -transport or -transports flag are ignored entirely, and MUST NOT result in a "CMETHOD-ERROR" message. Thus it is entirely possible for a given PT proxy to immediately output "CMETHODS DONE" without outputting any "CMETHOD" or "CMETHOD-ERROR" lines. This does not result in termination of the PT process.

Parent processes MUST handle "CMETHOD" and "CMETHOD-ERROR" messages in any order, regardless of ordering in -transports.

1.2.3. Pluggable PT Server Messages

IPC messages specified in section 1.2.3 are specific to PT servers.
PT server reverse proxies iterate over the requested transports in -transport or -transports and initialize the listeners.

For each transport initialized, the PT proxy reports the listener status back to the parent via messages to stdout and error messages to stderr.

SMETHOD <transport> <address:port> [options]

The "SMETHOD" message is used to signal that a requested PT transport has been launched, the protocol which will be used to handle incoming connections, and the IP address and port that clients should use to reach the reverse-proxy.

This message is written to STDOUT.

If there is a specific <address:port> provided for a given PT transport via the -bindaddr flag, the transport MUST be initialized using that as the server address.

The OPTIONAL 'options' field is used to pass additional per-transport information back to the parent process.

The currently recognized 'options' are:

ARGS:[<Key>=<Value>,]+[<Key>=<Value>]

The "ARGS" option is used to pass additional key/value formatted information that clients will require to use the reverse proxy.

Equal signs and commas MUST be escaped with a backslash.

Examples

```
SMETHOD obfs2 198.51.100.1:19999

SMETHOD obfs4 198.51.100.1:4444
ARGS:cert=60RNHBMRrf+aOSPzSj8bD4ASGyyPl0mkaOUAQsAYljSkFB0G8B8m9fGvGJC
pOxwoXS1baA;iatMode=0

SMETHOD meeklite [2001:0db8:85a3:0000:0000:8a2e:0370:7334]:2323
ARGS:url=https://meek-reflect.appspot.com/;front=www.google.com
```

SMETHOD-ERROR <transport> <ErrorMessage>

The "SMETHOD-ERROR" message is used to signal that requested PT transport reverse proxy was unable to be launched.

As this is an error, this message is written to STDERR.

Outputting a "SMETHOD-ERROR" does not result in termination of the PT process, as even if one transport fails to be initialized, other transports may initialize correctly.

Example

```
SMETHOD-ERROR shadow No password provided in configuration.
```

Once all PT transports have been initialized (or have failed), the PT proxy MUST send a final message indicating that it has finished initializing.

SMETHODS DONE

The "SMETHODS DONE" message signals that the PT proxy has finished initializing all of the transports that it is capable of handling.

This message is written to STDOUT.

Upon sending the "SMETHODS DONE" message, the PT proxy initialization is complete.

1.3. Pluggable Transport Shutdown

The recommended way for Pluggable Transport using applications and Pluggable Transports to handle graceful shutdown is as follows:

Set -exit-on-close when launching the PT proxy, to indicate that stdin will be used for graceful shutdown notification.

When the time comes to terminate the PT proxy:

- Close the PT proxy's stdin.
- Wait for a "reasonable" amount of time for the PT to exit.
- Attempt to use OS specific mechanisms to cause graceful PT shutdown (eg: 'SIGTERM')
- Use OS specific mechanisms to force terminate the PT (eg: 'SIGKILL', 'TerminateProcess()').

PT proxies SHOULD monitor stdin, and exit gracefully when it is closed, if the parent supports that behavior.

PT proxies SHOULD handle OS specific mechanisms to gracefully terminate (eg: Install a signal handler on 'SIGTERM' that causes cleanup and a graceful shutdown if able).

PT proxies SHOULD attempt to detect when the parent has terminated (eg: via detecting that it's parent process ID has changed on U*IX systems), and gracefully terminate.

PT proxies exiting after a graceful shutdown should use exit code EX_OK (0).

1.4. Pluggable PT Client Per-Connection Arguments

Certain PT transport protocols require that the client provides per-connection arguments when making outgoing connections. On the server side, this is handled by the "ARGS" optional argument as part of the "SMETHOD" messages. Options can also be set using -options or -optionsFile, in which case they apply to all connections.

On the client side, arguments are passed via the Dispatcher IPC protocol. The -options or -optionsFile flags can be used to set arguments for all connections and this works in all modes. The socks5 mode also supports passing per-connection arguments using the SOCKS5 protocol authentication mechanism.

If there are connection settings present, the authentication type 0x09 is used. This authentication type was specifically assigned by the IANA for use in the Pluggable Transports Specification, with the name “JSON Parameter Block”. When this authenticate method is used, the authentication mode selection byte is followed by the serialized per-connection parameter data. The serialization process for the parameters is defined as follows:

- They keys and values are inserted into a map
- This map is serialized JSON to a UTF-8 string.
- The UTF-8 string is converted to a sequence of bytes. (This is trivial for a UTF-8 string.)
- The number of bytes is counted.

- The byte count is encoded as a 4-byte sequence in network byte order (big-endian).
- The encoded count is prepended to the byte sequence.

If no per-connection settings are needed, authentication type 0x00 (no authentication required) is used.

The following error codes are defined for the response when connection settings are present:

- X'10' - Connection settings size too large
- X'11' - Timeout reading connection settings
- X'12' - Error parsing connection settings
- X'13' - Connection settings have invalid or missing keys or values

While the byte count is encoded as a 4-byte sequence, which is capable of expressing connection setting sizes up to 4GB, it is not required that the implementation support the maximum possible size. If a size larger than is supported by the implementation is specified, the X'10' error code can be used. Additionally, an implementation-dependent timeout should be included for receiving the connection settings. If this timeout is exceeded, the X'11' error code can be used. Error code X'12' is returned if the connection parameters are not properly encoded JSON. Error code X'13' is used if the connection settings are not correct for the specific transport being used.

Example

```
\x00\x00\x00\x39{"shared-secret": "rahasia", "secrets-file": "/tmp/blob"}
```

1.5 UDP Support

This section of the specification describes how application data contained in UDP packets can be sent between the application client and application server. Transports are free to use any type of network transport in order to get the data from the transport client to the transport server, such as UDP, TCP, or other protocols like SCTP. A number of scenarios of therefore possible for deployment such as UDP application data over a TCP transport, UDP application data over a UDP transport, as well as other possibilities.

1.5.1. Configuring the Transports

In the UDP application use case, transport configuration options are specified using -options or -optionsFile. There is no mechanism for per-connection options like in the TCP application use case because UDP is connectionless.

1.5.3. Implementation of the PT Client

The role of the PT Client configured in one of the UDP modes is to accept UDP packets and relay them by transport-specific means to the PT Server. The first step is for the PT Client to listen for UDP packets on a designated port. The second step is to relay these packets to the PT Server.

1.5.4. Integration with TCP Transports

Specifically for transports which utilize a connection between the PT Client and PT Server (such as a TCP connection), establishing a transport connection requires bridging a mismatch between the semantics of packet-based UDP protocols and connection-based TCP transports. TCP transports are opened and later closed, ending the connection. However, UDP protocols are connectionless. There is no intrinsic way to tell when the first packet will start arrive or when the last packet has arrived. Therefore, the PT Client must establish transport connections using lazy instantiation. The PT Client will maintain a pool of transport connections. Each connection will be associated with a PT Server destination address. When the PT Client receives a UDP packet with a PT Server destination address not represented in the pool, a new transport connection will be created and added to the pool. Otherwise, the existing connection will be used. Additionally, connections will be closed and removed from the pool based on a timeout system. When a connection has not been used for some time, it will be closed. The specific timeout used can be configured. It is also possible that a connection will be closed by the PT Server or due to an error. In this case, the transport will be removed from the pool. The following table shows the state transitions that occur with this implementation.

Event	Current State	New State	Effect
Packet received	No matching Connection in pool	New Connection added to pool with state = Waiting	Packet dropped
Packet received	Matching Connection in pool with state = Waiting		Packet dropped
Packet received	Matching Connection in pool with state = Connected		Packet sent using Connection
Connection successful	Connection in pool with state = Waiting	Connection in pool with state = Connected	
Connection closed	Connection in pool with state = Connected	Remove Connection from pool	
Connection failed	Connection in pool with state = Waiting	Remove Connection from pool	
Write failure sending packet	Connection in pool with state = Connected	Remove Connection from pool	Packet dropped
Timeout since last packet	Matching Connection in pool	Remove Connection from pool	

Table 1. Client-side UDP state transitions

Configuration of the transports is described in section 1.5.2. The remaining integration necessary is to take the receiving UDP packets and convert them to a data stream that can be transmitted over a TCP-based transport connection. The basic mechanism for doing this is described in RFC 5389, “Session Traversal Utilities for NAT (STUN)”. Section 7.2.2, “Sending over TCP or TLS-over-TCP”, describes the necessity for adding additional framing to tell where individual UDP packets start and end within the datastream. The particular implementation of this framing is left unspecified in the RFC.

Two methods of framing can be used. The first is for transparent UDP proxies where the format of the UDP packets is unknown. An example use case for this mode is an OpenVPN proxy. For this mode, a simple two byte length in network byte order can be used to prefix UDP packet payload data. The second mode is specifically for STUN packets. An example use case for this mode is when proxying to a TURN server. As STUN packets already contain a header including

a length for the payload, STUN packets can simply be concatenated without additional external framing. Extraction of the individual packets from the data stream on the server side requires knowledge of which framing was used by the client.

1.5.5. Implementation of the PT Server

The PT Server receives the data over the network using a transport-specific method and reassembles the UDP packets to send on to the UDP-based Application Server. Two modes of operation are available for the PT Server: transparent UDP proxy mode and STUN-aware mode. In the transparent proxy mode, packets retrieved from the data stream are forwarded to a destination address specified as a configuration parameter to the PT Server. The STUN-aware mode is similar, except that it only accepts STUN packets. This is advantageous for transports that use a datastream (such as TCP) because STUN packets carry their own internal length data in the STUN packet headers. In STUN mode, the received packets are forwarded onto a TURN server, the address of which is specified in the PT Server configuration parameters. The goal of the STUN-aware mode is to support the use of existing public TURN servers.

In addition to retrieving packets from the PT Client and relaying them onto a UDP Application Server, the PT Server must also receive UDP packets from the UDP Application Server and relay them back over the transport connection to the PT Client. In this function it follows similar logic to the PT Client. For transports that use connections (such as TCP), the state transitions possible in the PT Server are similar to those in the PT Client, but there are also differences. **If a UDP packet is received and no matching transport connection is available, the packet cannot be delivered and is dropped.** Relatedly, connections in the connection pool are always in a Connected state and never in a Waiting state. Therefore Connection states are removed from the state transition table for the PT Server. The following table shows the state transitions that occur with this implementation.

Event	Current State	New State	Effect
Packet received	No matching Connection in pool		**Packet dropped**
Packet received	Matching Connection in pool		Packet sent using Connection
Connection closed	Connection in pool	Remove Connection from pool	
Write failure sending packet	Connection in pool	Remove Connection from pool	**Packet dropped**
Timeout since last packet	Matching Connection in pool	Remove Connection from pool	

Table 2. Server-side UDP state transitions

1.5.6. Configuring Proxy Modes

There is currently no mechanism for PT Servers to support multiple proxy modes simultaneously. When transport connections are received by the PT Server, the data stream must be interpreted as data from one of the TCP proxy modes (either transparent proxy or SOCKS proxy) or one of the UDP proxy modes (either transparent UDP proxy or STUN-aware proxy to a TURN server). Which mode the PT Server will operate in will be determined by PT Server configuration parameters. It is therefore important to ensure that the PT Client and PT Server are operating in the same mode.

2. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L., "SOCKS Protocol Version 5", RFC 1928, March 1996.

[EXTORPORT] Kadianakis, G., Mathewson, N., "Extended ORPort and TransportControlPort", Tor Proposal 196, March 2012.

[RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

[RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, March 1996.

[PT2-DISPATCHER] Operator Foundation, Shapeshifter Dispatcher.
https://github.com/OperatorFoundation/shapeshifter-dispatcher

Appendix A. Example Client Pluggable Transport Session

Messages the PT Proxy writes to stdin

```
VERSION 2 PROXY DONE
CMETHOD shadow socks5 127.0.0.1:32525
CMETHOD obfs4 socks5 127.0.0.1:37347
CMETHODS DONE
```

Appendix B. Example Server Pluggable Transport Session

Messages the PT Proxy writes to stdin

```
VERSION 2
SMETHOD shadow 198.51.100.1:1984
SMETHOD obfs4 198.51.100.1:43734
ARGS:cert=HszPy3vWfjsESCEOo9ZBkRv6zQ/1mGHzc8arF0y2SpwFr3WhsMu8rK0zyaoyERfbz3ddFw,iat-mode=0
SMETHODS DONE
```

Appendix C. Changelog

PT 3.0

- Implemented proposal 0010 - Reduce Tor-Specific IPC Parameters
- Implemented proposal 0014 - Make Server Address Part of Client Options
- Removed language assuming that transports will use TCP internally to relay UDP application traffic

PT 2.2.1

- Added deprecation warnings

PT 2.2

- Implemented proposal 0006 - User Settable Proxy Listen Address
- Implemented proposal 0008 - Reduce Number of Required IPC Parameters
- Implemented proposal 0009 - Reduce Use of Microformats in IPC Parameters
- Implemented proposal 0012 - Improve Logging in IPC
- Added -client and -server flags
- Deleted section 1.1.4 Command Line Flags as it is has become redundant
- Removed references to obsolete transports

PT 2.1

- Implemented proposal 0002 - Modularization of Specification

PT 2.0

- Modified SOCKS authentication method to use IANA-assigned designator
- Added error response codes for per-connection arguments
- Renamed version flag to ptversion to avoid naming conflict with goptlib
- Standardized use of Dispatcher IPC language throughout
- Added length to per-connection parameter encoding