XEP-0037: DSPS - Data Stream Proxy Service

David Sutton
mailto:

”Bac9”
mailto:bac9@bac9.yi.org
xmpp:bac9@jabber.org

2016-10-04
Version 0.8.1

Status	Type	Short Name
Rejected	Standards Track	N/A

A proposal for proxy support in Jabber.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Data Stream Proxy Service (DSPS) is designed to be a common stream protocol for multicast (unicast as special case) over P2S2P (P2P as special case) connections.

2 Startup

This document follows DSPS protocol version 0.5. Any XML data not explicitly defined or mentioned will be ignored without error. On startup, full fledged DSPS starts listening on port 5290 (and 80 if HTTP handshake implemented).

3 General Operation

3.1 Stream Creation/Relay {optional}

(optional) Creating or modifying stream is done like so:

```
<iq
  id='dsps1'
  type='get'
  from='rob@nauseum.org/dspsclient'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a44'>
  <query type='create'
    xmlns='jabber:iq:dsps'
    minthroughput='1.5KB'
    maxpublic='20'>
    <peer port='5290'>
      dsps.myjabber.net/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33
    </peer>
    <comment>public comment</comment>
  </query>
</iq>
```

- **"to"** with resource implies reconnection to previous relay stream with previously supplied credentials and authentication as per section "DSPS relay setup", otherwise implies creation of new stream where creator is granted "master" rights.
- **"minthroughput"** *(optional)* is minimum 16 second average throughput below which peers will be disconnected (but not dropped). Checked after every unit of outgoing transfer against the fourth value returned by "who" query for each peer. Legal only upon initial creation of stream. If omitted or negative, 0 is assumed.

- **"maxpublic"** *(optional)* is maximum number of peers that can join without invitation. If omitted or negative, 0 is assumed. If positive, DSPS generates globally unique id for public peers to acknowledge, reported within "stats" message. Said id remains constant for life of stream.
- **<peer/>** *(optional)* full JID of stream on another DSPS. Relay stream will be treated using "slave" rights. Legal only upon initial creation of stream. Multiple such blocks tried in series until successful connection. First successful (last if all failed) remembered by DSPS, all successive <peer/> ignored.. On connection, DSPS initiates handshake using peer's full JID and contents of block for destination's full JID to the relay destination as per section "Connecting to DSPS via default method". Authentication is done as per section "DSPS relay setup". Upon successful handshake, DSPS sends presence notification to peer as per section "Acknowledge of DSPS connection".
- **"port"** *(optional)* for connecting to destination DSPS. If omitted, default is assumed.
- **<comment/>** *(optional)* all such blocks reported in "stats" message. Multiple such blocks allowed. Block may contain a full XML stack of elements. No order implied for "stats" message.

Possible failure messages:

Code	Message	Description
405	Method Not Allowed	Attempt at reconnect to relay without existing credentials, relay still connected, <peer/> block present in reconnect request, or feature not supported
504	Gateway Timeout	All destination DSPS are unreachable.

3.2 Connection waiting

DSPS creates "id" (empty string is legal), used in "who" replies and notifies client of waiting connection like so:

```
<iq
  id='dsps1'
  type='result'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='rob@nauseum.org/dspsclient'>
  <query type='create' xmlns='jabber:iq:dsps'
    wait='10'
    host='dsps.jabber.org'
```

```
    port='5290'
    minthroughput='1.5KB'
    protocol='0.5'>
    <feature type='http' version='1.1'/>
    <feature type='ssl' version='3.0'/>
  </query>
</iq>
```

- **"from"** full JID of DSPS, internally globally unique. Used in handshake and every subsequent communication with stream. May differ with the one specified in "invite" message.
- **"wait"** amount of time in **milliseconds** that DSPS will wait for client to initiate handshake. If timeout occurs, DSPS will totally forget prepared connection and act accordingly.
- **"host"** *(optional)* for handshake and data stream. If omitted default from the "from" is assumed. Intended for P2P connections to be able to report alternate hostname or IP for connection.
- **"port"** *(optional)* for handshake and data stream. If omitted default is assumed.
- **"minthroughput"** value from "create".
- **"protocol"** version this DSPS supports.
- **<feature/>** *(optional)* supported by this DSPS. Type and version are properties and additional data stored in body. HTTP stream will not follow HTTP protocol. SSL handshake performed encrypted. Both HTTP and SSL connections are only between client and DSPS.

3.3 Establishing prepared connection

Upon receipt of message as per section "Connection waiting", client can either ignore it and connection will timeout, or connect to the DSPS directly via any supported connection method or via relay. There may be a maximum of 1 (one) established connection to DSPS from any Client_full_JID + DSPS_full_JID pair, deviations are handles as per section "Connecting to DSPS via default method". DSPS will not discriminate method via which direct connection is made, even if prior to "disconnect" a different method was used. Any packet from an unauthorized connection is ignored without reporting an error.

3.3.1 DSPS relay setup {optional}

Client may request another DSPS to relay this connection as per section "Stream Creation/Relay", utilizing the "create" body. There is no limit on length of relay chain. Upon initiation of handshake with destination, DSPS reports key like so (message sequence unrelated to current

DSPS handshake):

```
<iq
  id='dsps1'
  type='get'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='rob@nauseum.org/dspsclient'>
  <query type='create' xmlns='jabber:iq:dsps'>acDgH63I27Gb1</query>
</iq>
```

- **"get"** denotes request for auth key.
- **"create"** body contains key returned by destination.

Client must send said key to destination as per section "Connecting to DSPS via default method" and send response to DSPS (which will be transmitted to destination) like so:

```
<iq
  id='dsps1'
  type='result'
  from='rob@nauseum.org/dspsclient'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  <query type='create' xmlns='jabber:iq:dsps'>acDgH63I27Gb1</query>
</iq>
```

- **"result"** denotes reply with auth key.
- **"create"** body contains key returned by destination.

3.3.2 Connecting to DSPS via HTTP method {optional}

Client must connect to DSPS on port 80 and initiate handshake. This may be attempted after "create" result received or "disconnect" occurred, and prior to "wait" timeout expiring, then send HTTP request like so:

```
GET /DSPS/STREAM/ HTTP/1.0<CR>
Host: dsps.server<CR>
<CR>
```

And will receive reply from DSPS before the start of data stream, like so:

```
HTTP/1.0 200 OK<CR>
Content-Type: application/octet-stream<CR>
<CR>
```

Upon completion, Client must resume DSPS handshake as per either section ”Connecting to DSPS via default method” or section ”Connecting to DSPS via SSL method” (if applicable). Subsequent data will not follow HTTP protocol. On error connection closed immediately with optional error messages.
Possible failure messages:

Code	Message	Description
401	Unauthorized	(optional) Returned if any error in HTTP handshake.

3.3.3 Connecting to DSPS via SSL method {optional}

Client must connect to DSPS on specified port and initiate handshake. This may be attempted after ”create” result received or ”disconnect” occurred, and prior to ”wait” timeout expiring, then send following on stream:

```
starttls<CR>
```

Next, regular TLS handshake is initiated. Upon completion, Client must resume DSPS handshake as per section ”Connecting to DSPS via default method”. On error connection closed immediately with optional error messages.
Possible failure messages:

Code	Message	Description
401	Unauthorized	(optional) Returned if any error in SSL handshake.

3.3.4 Connecting to DSPS via default method

Client must connect to DSPS on specified port and initiate handshake. May be attempted after ”create” result received or ”disconnect” occurred, and prior to ”wait” timeout expiring. Standard and SSL handshakes are identical in decrypted state and take the form of:

```
Client_full_JID DSPS_full_JID<CR>
```

- **”Client_full_JID”** client full JID as supplied in either the ”create” or ”acknowledge” message.

- **"DSPS_full_JID"** DSPS full JID as supplied in "from" field of "create" result message from section "Connection waiting"
- **<CR>** regular carriage return, commonly referred to as the newline character.

For example, the appropriate string for the above request would be:

```
rob@nauseum.org/dspsclient dsps.jabber.org/0
    beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33
```

If Client_full_JID and DSPS_full_JID do not have an associated stream, are no longer valid, (e.g. timeout reached or client removed from stream), or connection from said Client_full_JID + DSPS_full_JID pair is in use (i.e. client is still connected to it), connection is closed immediately with possible optional error messages reported. Otherwise DSPS returns uniquely generated key followed by a <CR> like so:

```
uGhhb74d21
```

Client must now send key to DSPS via XML stream like so:

```
<iq
  id='dsps1'
  type='get'
  from='rob@nauseum.org/dspsclient'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'>
  <query type='auth' xmlns='jabber:iq:dsps'>uGhhb74d21</query>
</iq>
```

- **"get"** denotes request for next auth key.
- **"auth"** body contains key returned by DSPS.

DSPS will now check key, if not valid, close connection, report possible optional error message and resume waiting on original key. If valid, generate new key and send to client like so:

```
<iq
  id='dsps1'
  type='result'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='rob@nauseum.org/dspsclient'
  <query type='auth' xmlns='jabber:iq:dsps'>qgqB42Ij784</query>
</iq>
```

- **"result"** denotes return of next auth key.

- **"auth"** body contains key returned by DSPS.

Client must now send received key to DSPS via the stream followed by a <CR>. Once received, DSPS checks key, on mismatch connection is closed immediately with possible optional error messages reported, waiting on key is resumed. Upon successful handshake a message is sent to members of the stream in accordance with the following rules; If the client had type "master" connection, all members of the stream get notified. If the client had type "slave" connection, only other type "master" members get notified. The message takes the form of:

```
<iq
  id='dsps2'
  type='set'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='foo@bar.com/resource'>
  <query type='presence' xmlns='jabber:iq:dsps'>
  <peer status='connect'>JID</peer>
  </query>
</iq>
```

- **"presence"** denotes presence change. Body may contain multiple <peer/> blocks where same JID peers must be placed in chronological order relative to each other from start to end of message.
- **<peer/>** body is full JID of the joined peer unless peer of type "relay", in which case the resource is not reported.
- **"status"** is new status of peer.

Possible failure messages:

Code	Message	Description
401	Unauthorized	(optional) Returned if the DSPS is not aware of said Client_full_JID + DSPS_full_JID pair. Where "from" contains DSPS_full_JID that was used in the handshake and "to" contains Client_full_JID that was used in the handshake.
409	Conflict	(optional) Returned if connection from said full client JID and full DSPS JID is in use (i.e. client is still connected to it). Where "from" contains DSPS_full_JID that was used in the handshake and "to" contains Client_full_JID that was used in the handshake.

3.4 Stream administration

DSPS protocol allows multiple peers to use the same stream. Manipulation of the authorized peer list is done through admin functionality described in next several subsections. DSPS protocol allows for three types of peer connections: ”master”, ”slave”, and ”relay”. ”master” peers get full control of the stream, ”slave” peers get limited control of the stream, and ”relay” are treated similar to ”slave” except in reporting of JIDs where the resource must be omitted. ”master” peers are allowed to invite any other user to the stream and drop any peer registered with the stream, including themselves. ”slave” peers are only allowed to drop themselves from the stream. Any administrative changes coming from a ”slave” peer that are not for the peer’s own connection are ignored. Dropping one’s own connection is the preferred way of permanently disconnecting from the stream.
Any data received from a ”master” gets copied to every other peer on the stream. Any data received from a ”slave” peer gets copied to all ”master” peers on the stream only.
Stream administration request looks like so:

```
<iq
  id='dsps3'
  type='set'
  from='rob@nauseum.org/dspsclient'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'>
  <query type='admin' xmlns='jabber:iq:dsps' expire='20' wait='10'>
    <comment>welcome to the stream</comment>
    <peer type='master'>someone@somewhere.net</peer>
  </query>
</iq>
```

- **”admin”** denotes administrative functions are to follow. Any properties within this block apply to this block alone. Multiple such blocks are allowed.
- **”expire”** time DSPS should wait for the ”acknowledge” message from any invited peer within block. An ”expire” of 0 denotes no time limit. Actual value sent to peer as ”expire” is *minimum* of this value and default value preset for DSPS. If value unparseable or not present, default is used.
- **”wait”** time DSPS should wait for invited peer to connect to DSPS after ”acknowledge” is received and message from section ”Connection waiting”. is sent. A ”wait” of 0 denotes no time limit. Actual value sent to peer as ”wait” is *minimum* of this value and default value preset for DSPS. If the value unparseable or not present, default is used.
- **<comment/>** *(optional)* block sent to each of the peers. Multiple such blocks are allowed. Block may contain a full XML stack of elements. All such blocks are sent to each of the invited peers as is. No guarantee is made on their order in the <invite/> message.

- **<peer/>** JID to execute action upon. If invitation then body will not necessarily be same full JID as one that would respond. Multiple such blocks allowed.
- **"type"** type of action to do. "master" denotes invitation, granting master rights. "slave" denotes invitation, granting slave rights. "drop" denotes request to drop peer from stream. "relay" peers may not be invited but may be dropped using this method.

Possible optional errors include the following:

```
<iq
  id='dsps3'
  type='result'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='user@server.org/resource'>
  <query type='admin' xmlns='jabber:iq:dsps'>
    <peer acknowledge='expire'>abc@company.com/net</peer>
    <peer acknowledge='reject'>friend@someplace.com/home</peer>
    <peer acknowledge='timeout'>abd@company.com/net</peer>
    <peer acknowledge='missing'>who@knows.org/winjab</peer>
  </query>
</iq>
```

- **"admin"** denotes admin response, sent to sender of "admin" request for every peer in block. Peers may be combined from multiple "admin" requests or peers from single "admin" request may be split over multiple "admin" replies.
- **<peer/>** peer in question. For "expire" JID is one from invite request. For "reject" JID is one from which reject received. For "timeout" JID is one from invite request. For "missing" is one from drop request. After this DSPS will totally forget about this peer.
- **"acknowledge"** reason for failure. "expire" denotes "expire" timeout sent, has ended. "reject" denotes peer rejected invite. "timeout" denotes "wait" timeout sent, has ended. "missing" denotes peer marked for drop not found registered on tis stream.

Possible failure messages:

Code	Message	Description
403	Forbidden	(optional) Returned if peer with "slave" rights attempts to use "master" admin privileges.

3.4.1 Invitation to stream {optional}

Upon invite DSPS will attempt to invite each of the peers like so:

```
<iq
  id='dsps4'
  type='get'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='foo@bar.com/resource'>
  <query type='acknowledge'
    xmlns='jabber:iq:dsps'
    status='master'
    expire='20'>
    <peer>rob@nauseum.org/dspsclient</peer>
    <comment>some long comment block or structure</comment>
  </query>
</iq>
```

- **"from"** is unique JID/resource pair generated for this JID, not necessarily same as JID/resource pair specified in section "Connection waiting". It is used for identification of the "acknowledge" message.
- **"acknowledge"** denotes request for invitation acknowledge.
- **"status"** type of connection the client is granted. Same type as tag in invitation request.
- **"expire"** time DSPS will wait for the "acknowledge" message.
- **<peer/>** peer who initiated this invite. Multiple such blocks may exist if multiple distinct peers sent invitation that have not yet been received by the invitee.
- **<comment/>** is <comment/> structure(s) sent in admin request, present if admin request contained it.

3.4.2 Dropping from stream

Upon drop DSPS will immediately closes the connection to the dropped peer. It then will totally forget this peer right after sending it a notification message like so:

```
<iq
  id='dsps5'
  type='set'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='foo@bar.com/resource'>
  <query type='acknowledge' xmlns='jabber:iq:dsps' status='drop'>
    <comment>some long comment block or structure</comment>
  </query>
</iq>
```

- **”from”** is the DSPS JID/resource pair which DSPS has associated with this connection.
- **”acknowledge”** denotes drop notification. Despite the block name, this message does not require a reply.
- **”status”** drop denotes a connection drop.
- **<comment/>** is <comment/> structure(s) sent in admin request, preset if admin request contained it.

For every successfully dropped peer a message is sent to all other stream members, following the rules stated for the ”presence” message, and takes the form of:

```
<iq
  id='dsps6'
  type='set'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='foo@bar.com/resource'>
  <query type='presence' xmlns='jabber:iq:dsps'>
    <peer status='drop'>JID</peer>
  </query>
</iq>
```

- **”from”** is the DSPS JID/resource pair which DSPS has associated with the connection of the recipient of the message.
- **”presence”** denotes presence change. Body may contain multiple <peer/> blocks where same JID peers must be placed in chronological order relative to each other from start to end of message.
- **<peer/>** body is full JID of the dropped peer registered on the stream, unless peer is of type ”relay”, in which case the resource is not reported.
- **”status”** is new status of peer.

3.5 Invitation reply

An invited peer has the option to accept or reject an invitation to a stream.

3.5.1 Accepting an invite

To accept an invitation to a stream, the peer must reply like so:

```
<iq
  id='dsps4'
  type='result'
  from='foo@bar.com/moredsps'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33>
  <query type='acknowledge' xmlns='jabber:iq:dsps' status='connect'/>
</iq>
```

- **"from"** is the JID/resource pair which will be associated with this connection, only it will be allowed to connect to this stream as this user. A peer may be registered to multiple streams from the same full JID, hence all DSPS full JIDs linked to a given peer must be unique.
- **"to"** contains the DSPS JID/resource pair which was the source in the original "acknowledge" message.
- **"acknowledge"** denotes acknowledgment to invitation.
- **"status"** connect denotes an acceptance of invitation.

Upon receipt of this reply the DSPS creates a unique resource for this client JID/resource pair. It then prepares the "create" message as described in section "Connection waiting".

3.5.2 Rejecting an invite

Rejecting an invitation can be done in two ways. A peer can forget about the invitation and let the invitation "expire", or preferably a message can be sent like so:

```
<iq
  id='dsps4'
  type='result'
  from='foo@bar.com/moredsps'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33>
  <query type='acknowledge' xmlns='jabber:iq:dsps' status='drop'/>
</iq>
```

- **"to"** contains the DSPS JID/resource pair which was the source in the original "acknowledge" message.
- **"acknowledge"** denotes acknowledgment to invitation.
- **"status"** drop denotes an rejection of invitation.

Regardless of the way a rejection was achieved a notification message is sent to the inviting peer, as was described in section ”Stream administration”. If unknown ”type” is sent, it will be interpreted as a reject. A maximum of one ”acknowledge” is allowed during the lifetime of an invitation. If multiple such tags are sent, the first tag takes precedence. Any rejection of a public connection will be ignored.

3.6 Disconnection handling

If a peer ever disconnects without first dropping themselves, the following policy applies:
The peer may reconnect within the ”wait” timeout provided in the ”create” reply in section ”Connection waiting”. The peer may choose any supported mode of reconnection supplied in ”create” reply, regardless of mode previously used. The ”wait” timeout is not cumulative over multiple disconnects. After reconnect, peer will not receive any data that exists on the stream while it was disconnected.
Upon such disconnection DSPS notifies all other members of the stream, following the rules stated for the ”presence” message, and takes the form of:

```
<iq
  id='dsps7'
  type='set'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='foo@bar.com/resource'>
  <query type='presence' xmlns='jabber:iq:dsps'>
    <peer status='waiting'>JID</peer>
  </query>
</iq>
```

- **”from”** is the DSPS JID/resource pair which DSPS has associated with the connection of the recipient of the message.
- **”presence”** denotes presence change. Body may contain multiple <peer/> blocks where same JID peers must be placed in chronological order relative to each other from start to end of message.
- **<peer/>** body is full JID of the disconnected peer registered on the stream, unless peer is of type ”relay”, in which case the resource is not reported.
- **”status”** is new status of peer.

Upon reaching ”wait” timeout the procedure is the same if the peer dropped its own connection.

3.7 Ending a connection

Permanent termination of connection can be done in two ways: peer may disconnect from the stream and let the ”wait” timeout expire, or more preferably the peer will drop itself from the stream via an ”admin” message. The ”admin” is still allowed to contain multiple ”peer” blocks.

3.8 Stream use

The use policy for the stream follows the standard rules described in this document. Type and structure of the data must be negotiated by the peers separately (presumably via the normal XML message stream or within <comment/> blocks). The DSPS stream operates at the speed of the slowest connection (or slower if it is so configured in its internal configuration).
Data read from peer in a unit of transfer (decided by DSPS) is sent to other peers in a format like so:

```
0<size><CR><id><CR><data>
```

- **”0”** at beginning for checking start of block.
- **<size>** length in bytes including id and its trailing CR in form of [1-9][0-9]*[0-9A-Z], where last character is base 36 numerical equivalent power of 1024.
- **<id>** id of the sender as per ”who” query.
- **<data>** data sent.
- **<CR>** regular carriage return, commonly referred to as the newline character.

For example, the appropriate string for the above block would be:

```
0340<CR>
010<CR>
this is the data in ASCII form
```

First block received after connection will always be full block. If discrepancy occurs, receiving peer should disconnect and reconnect back to stream.

3.9 Stream information

Two mechanisms exists to gain information about the stream configuration and its members. They are described within next few subsections.

3.9.1 Stream peer listing

To retrieve listing of all registered peers of this stream and their respective connection status any registered peer sends a message like so:

```
<iq
  id='dsps8'
  type='get'
  from='rob@nauseum.org/dspsclient'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'>
  <query type='who' xmlns='jabber:iq:dsps'/>
</iq>
```

- **"to"** has the usual meaning for this client when referring to this DSPS stream.
- **"who"** denotes that this is a listing request. It may not contain a body or attributes, otherwise it will be ignored without error.

The query follows the standard rules: query originating from a "master" peer will return listing of all registered peers and their associated statuses, query originating from a "slave" peer will only return listing of all registered "master" peers and their associated statuses. Returned results do not have any strict order. If multiple "who" queries were requested by a peer that have not yet received a reply, only one reply need be sent.
The query reply is formatted like so:

```
<iq
  id='dsps8'
  type='result'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='rob@nauseum.org/dspsclient'>
  <query type='who' xmlns='jabber:iq:dsps'>
    <peer
      type='master'
      id='0'
      status='connect'
      throughput='1KB////4.8KB//3KB'>
      rob@nauseum.org/dspsclient
    </peer>
  </query>
</iq>
```

- **"from"** is unique DSPS JID/resource pair for the peer receiving the result.
- **<peer/>** body is full JID of registered peer, unless peer is of type "relay", in which case the resource is not reported. A separate block exists for every viewable peer.

- **"type"** the type of connection peer has. May contain value of "master", "slave", or "relay".
- **"id"** id prepended to data coming from this peer.
- **"status"** current status of peer. "connect" denotes peer able to receive data. "wait" denotes peer registered but not connected. "expire" denotes peer was invited but no reply was received yet.
- **"throughput"** shows the average throughput to that peer per second in the units specified after the number (e.g. B, KB, MB, GB, TB, EB) in capital letters. Time is measured only during data transfer. Value contains multiple fields delimited by slash (/). Each field represents time span of power of two (2), relative to its position from start of the string. Each filled-in field contains the average throughput over that timespan. e.g. (1B per sec in last sec)/(1.1B per sec in last 2 sec)/(0.9B per sec in last 4 sec). Only fields representing power of zero (2^0 sec) and power of four (2^4 sec) are required. Last field must be filled-in.

3.9.2 Stream status listing

To retrieve listing of all stream configuration/statistics values or public streams, any registered peer sends a message like so:

```
<iq
  id='dsps9'
  type='get'
  from='rob@nauseum.org/dspsclient'
  to='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'>
  <query type='stats' xmlns='jabber:iq:dsps'/>
</iq>
```

- **"to"** if contains a resource, will return statistics for specified stream. Otherwise will return listing of public streams, i.e. any stream with "maxpublic" greater then 0.
- **"stats"** denotes that this is a configuration/statistics request. It may not contain a body or attributes, otherwise it will be ignored without error.

The query reply is formatted like so:

```
<iq
  id='dsps9'
  type='result'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='rob@nauseum.org/dspsclient'>
  <query type='stats' xmlns='jabber:iq:dsps'
    init='00000000000000'
```

```
    protocol='0.5'
    port='5290'
    minthroughput='1.5KB'
    expiredefault='150'
    waitdefault='100'
    wait='10'
    public='5'
    maxpublic='25'
    mastercount='20'
    slavecount='40'
    relaycount='0'>
    <feature type='http' version='1.1'/>
    <feature type='ssl' version='3.0'/>
    <peer>mydsps@jabber.org/8xd67f56df4f546fdgsfdg65f6g58f</peer>
    <comment>some server comment</comment>
  </query>
</iq>
```

- **"from"** the DSPS JID/resource pair for the peer receiving the result.
- **"init"** UNIX timestamp of the date this stream was initiated.
- **"protocol"** protocol version this DSPS supports.
- **"port"** default port this DSPS listens for connections on.
- **"minthroughput"** as defined in "create".
- **"expiredefault"** default time this DSPS will wait for an "acknowledge" response.
- **"waitdefault"** default time this DSPS will wait for a connection to its default port after an invitation is accepted.
- **"wait"** time this DSPS will wait for this particular client to reconnect if it ever gets disconnected. This is the same value as one sent in the "create" response.
- **"public"** number of "slave" peers registered with stream without invitation.
- **"maxpublic"** maximum number of "slave"peers allowed without invitation.
- **"mastercount"** number of peers with a "master" connection registered on this stream.
- **"slavecount"** number of peers with a "slave" connection registered on this stream.
- **"relaycount"** number of peers with a "relay" connection registered on this stream.
- **<feature/>** *(optional)* denotes a supported feature. All supported features must be listed.
- **<peer/>** *(optional)* list of public stream connections where all reported statistics match. Present only if "to" in original request contained no resource. Multiple allowed where statistics match for all.

- **<comment/>** *(optional)* parameter(s) with a comments from ”create”. This block may contain full XML stack of elements. Multiple such blocks are allowed.

All ”status” attributes are required. Any other undefined blocks with any multiplicity, are legal in this block as long as their tags are not identical to any tag within the protocol. Results returned do not have any strict order. If ”to” in original request contained no resource, multiple ”stats” blocks are allowed, where each contains at least one <peer/> block which has ”maxpublic” greater than 0. To join a public stream a client must send message as per section ”Accepting an invite”.

3.10 Stream shutdown

Stream exists from its ”create”ion time to the time when there are no more ”master” peers registered with the stream.
When last ”master” peer is dropped from the stream, DSPS will make sure that all the data sent by all the ”master” peers was actually copied to all the ”slave” peers still present. For every remaining ”slave” peer DSPS will initiate a drop event. Once stream is void of any peers it will be totally forgotten by the DSPS and all associated data is released.

3.11 Error message format

Error messages look like so:

```
<iq
  id='dsps9'
  type='error'
  from='dsps.jabber.org/0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
  to='rob@nauseum.org/dspsclient'>
  <error code='405'>Method Not Allowed</error>
</iq>
```

- **<error/>** denotes error block where body is text description.
- **”code”** denotes error code.

4 Possible applications

4.1 File transfer

File transfer can be easily accomplished over DSPS. Where one user invites another user to a DSPS stream. File details can be transfered in the invitation comment as such: <meta type='file' name='myfile.txt' size='500K' crc32='12345' sha1='23451' mime='application/octet-stream' timestamp='12345' date='20020412T00:00:00'/>. Where the ”size” would be in bytes.

All properties should reflect their appropriate values for this instance. Once the second peer has accepted, it can simply put a CR on the stream stating that transfer can begin. then the first party simply dumps the contents of file on the stream, closes the stream and ”drop”s itself from the stream. DSPS will make sure the second party gets everything that the first party sent before closing the connection. If multiple recipients of the file are required, the sending client can save a lot of bandwidth and transmit only one copy if the file to the DSPS which in term will transmit the data over to all the other connected clients.

4.2 VoIP

Same idea as the file transfer. However if more then two parties are involved, every party must have a ”master” connection.

4.3 Multicast

A server has a JID which it registers with a stream. Any client wishing to join the multicast sends an XML message to the server, which then invites the client with a ”slave” connection. Thus everything the server sends is received by every client on the stream. If there are multiple back-up servers, they can be invited with a ”master” connection, thus if one of them goes down, the others can take over.

4.4 File Storage

It has long been discussed in many Jabber places that a file storage facility is desired. The communication with such a facility can be easily accommodated with DSPS, as such a facility would merely appear as a user to DSPS which can either be ”invite”ed or ”invite” other users onto personal streams to transfer files as described in 6.1.

5 Why DSPS instead of PASS

PASS has the following design flaws that make it unsuitable for its stated purpose of providing raw data-streams to all classes of users, including those behind firewalls or NAT.

5.1 Ports

PASS requires the use of a large number of individual ports, which on a heavily loaded server can lead to the number of spare ports dropping to zero, causing connections to be refused. This is also problematic if PASS is situated behind a firewall. Firewall administrators are typically loathe to allow incoming connections to a large range of ports.

DSPS only uses one port, and so resolves the first problem, while making the second almost a non-issue.

5.2 Knowledge of IP

PASS requires the client to have some knowledge of IP, which immediately forces the assumption that the XML stream's underlying protocol is in fact, IP. While at the time of writing this is always the case, it may not always be this way.
DSPS uses the Jabber ID to do its routing, and so avoids these problems. And while DSPS does use the concept of a TCP connection and an IP port, this information is never actually used anywhere on the XML stream, making the actual connection to the DSPS implementation-defined.

5.3 IP Addresses

PASS makes the IP address of the remote client available to the local client. While it is rare that this is an actual problem, many users and administrators prefer that IP address information is never actually revealed.
DSPS never transmits IP address information across the XML stream, and so does not have this problem.

5.4 Intuitiveness

PASS requires a client to initiate a connection by opening a (proxied) listening socket, and then soliciting connections. However, TCP works by having the client connect to a remote resource directly. This difference can make the operation of PASS difficult to understand. Also, it is left to the client to distribute the information about this listening socket, which places an additional burden on the client.
DSPS, while it uses listening sockets to do its work, does all the work of setting up the connection after a client initiates it. All the initiating client has to do is request a connection, connect to the DSPS, and wait - everything else is handled automatically.

5.5 Scalability

Due to the master/slave design, DSPS is already able to handle multicasts of streams or such, whilst PASS was only designed for simple p2p stream connections. This will becoming increasingly more important as more emphasis is made on streaming capabilities, for technologies such as audio and video conferencing.
Due to DSPS generality, the protocol can be easily used for either P2P or P2S2P needs. This eliminates the need for a separate protocol for each of the tasks.

6 DSPS with P2P

It is not mandated for DSPS to reside beside a Jabber server. It is entirely possible for any client to implement a stripped down version of such a server. In such a case the only sections that are required are any error reporting, invitation acknowledgment and statistical responses. Any other area of the protocol becomes optional since the recipient peer will not have the ability to use it anyway.
Any client may, but is not required to utilize the striped down functionality. When utilizing such functionality the serving client sends an invitation to the recipient client to join the serving client's DSPS stream. Thus the "create" message would list the serving client as the DSPS and would utilize the "host" attribute to tell the recipient client where to connect to the DSPS.
This ability is advantageous since the recipient client only needs to know one protocol for data transmission over P2P or P2S2P connections, and would not see a difference between the two. The proposed method is for one side to fist try serving a connection to the other. If that fails the other side may attempt to serve the connection. If the second attempt fails the clients may utilize an external DSPS server. The negotiation of who will serve is done outside DSPS protocol. DSPS has no functionality to decide when a P2P connection is possible or desirable, nor does it have enough information to do so reliably.