XRootD

Third Party Copy Protocol
TPC Version 2.0
Reference

13-April-2020

Andrew Hanushevsky

SLAC
NATIONAL ACCELERATOR LABORATORY

1 Introduction

This document describes **XRootD**'s Third Party Copy (**TPC**) protocol. This application layer protocol allows a third party client to initiate a copy of a file from one data server to another. The data does not flow through the client allowing for superior performance. This protocol is natively supported by the **XRootD xrdcp** data transfer utility. However, since it is implemented at the application layer it can be used by other protocols as well (e.g. **HTTP**). In discussing the protocol the following terminology is used:

- Destination: the node that receives the file.
- LFN: the logical file name at either the source or destination.
- PFN: the physical file name at the destination.
- Source: the node that provides the file.

The **TPC** protocol is based on a rendezvous copy paradigm revolving around a client-supplied rendezvous key. In essence the client must be able to read the source file and create the destination file. If both are true then a rendezvous point is established at the source. This allows the destination to rendezvous with the source using the rendezvous key to transfer the file from the source to the destination. The transfer is a pull request that allows the destination to make sure sufficient resource exist to create the file. At the end of the transfer the client may also request that checksum verification occur to make sure the source and destination files are identical.

While the **CGI** information is normally not encrypted unless the underlying protocol encrypts such information; the protocol is able to detect whether the **CGI** information was possibly stolen from the wire and is being inappropriately used. This requires that any servers participating in a **TPC** transfer be registered in **DNS** and use the same client authentication protocol, if any.

The **TPC** protocol also supports delegation. In this case, the client delegates its credentials to the destination server which uses those credentials to fetch the file on behalf of the user. This is known as **TPCLite**. Delegation is currently supported only for the **GSI** authentication protocol.

2 Client Performing a TPC Transfer

The following steps should be performed to successfully execute a third party copy. Some steps are optional but when executed they assist in diagnostics and predictability.

1. The client may determine whether or not **TPC** is supported by the source and destination servers. This provides early detection of whether a **TPC** transfer is even possible. This step is optional as non-support can be detected later in the protocol steps. See the section on how to determine if **TPC** is supported in **XRootD** protocol.
2. The client should open the file at the source and obtain the file size and indicate that the source will be subject to **TPC** transfer. This step is optional but it may be used by the source in determining whether or not it supports **TPC** and, if so, prepare the file for transfer as needed. The file should be closed after this step completes.
3. The client should open the file at the destination in the write-create mode and supply **CGI** elements to allow the destination to rendezvous with the source. The file should remain open until the transfer completes.
4. The client should issue a **sync()** request against the open destination file. The **sync()** request should not return until the third party copy is initiated or fails to initiate.
5. The client should open the source file in read-only mode and supply specific **CGI** elements to establish a rendezvous point in the source server. The file should remain open until the transfer completes.
6. The client should issue another **sync()** request against the open destination file. The **sync()** request should not return until the third party copy completes or fails.
7. The client should close the source and destination files.

The above steps are detailed in the same sequence in the following sections.

2.1 Determine if TPC is Supported

Various implementations may use different schemes to indicate whether or not **TPC** is supported. In **XRootD** this is done using the "**query config tpc**" request (i.e. **kXR_query** request code with the **kXR_Qconfig** flag set and an argument of "**tpc**"). If the response is a signed integer value it indicates that **TPC** is supported and the value is the protocol version number. If it is not supported, "**tpc**" is returned.

This step is optional but allows you to determine whether the source and destination support **TPC** and avoid initiating a copy that is destined to fail.

2.2 Client Preparatory Source Open

The client may open the source file in read-only mode whose logical file name (i.e. *lfn*) is suffixed with a **TPC CGI** element shown below.

```
lfn?tpc.stage=placement
```

The client may determine the file size at this point, though this can occur in the subsequent step. Certain servers may require advance notification that a **TPC** action will be performed on a file and this step allows for that. The file should be closed before initiating any subsequent steps.

2.3 Client Destination Open

The client should open the destination file in write-create mode whose logical file name (i.e. *tlfn*) is suffixed with **TPC CGI** elements shown below. Highlighted **CGI** elements are only relevant when delegation is being used.

```
tlfn?tpc.src=hostname&tpc.key=token&tpc.stage=copy[opts]

opts: [&tpc.cks={adler32 | crc32 | md5}]

      [&tpc.dlg=head] [&tpc.dlgon=dval] [&tpc.scgi=scgi]

      [&tpc.lfn=slfn] [&tpc.spr=sprot] [&tpc.str=snum]

      [&tpc.tpr=tprot] [&oss.asize=size]
```

Required Parameters

tlfn Is the logical file name at the destination server that the copied file is to have.

hostname
Is the fully qualified **DNS** name of the source server. This is the server that provides the file.

token Is a client generated hexadecimal ASCII string that is to be used as the rendezvous point by the destination server. The strings should be as unique as possible and may include information unique to the client. The string should be only as long as necessary to achieve relative uniqueness to the client, but no longer than 256 characters.

Optional Parameters

opts Are optional **CGI** elements that must be used in certain circumstances, as described below.

adler32 | crc32 | md5
If specified, the destination server is to verify that the copied file has the same checksum as the source file using the specified checksum type. Otherwise, it is the client's responsibility to perform this check if so desired.

dval Should be '**1**' if the intention is to use delegated TPC. Otherwise it need not be specified but, if specified, it should be '**0**'.

head Is the host name that should be contacted as the source when delegation is being used. Typically, this should be source cluster's head node (i.e. the source host specified on the command line) rather than the final source location. This is only meaningful when delegation is used and the source is a clustered server. The element need not be specified if **tpc.src** already identifies the proper head node.

scgi Is the **CGI** information from the source **URL**. This element needs to be specified only if a) delegation is being used and b) meaningful **CGI** is present on the source **URL** (see the notes on the definition of meaningful). Since a **CGI** string may not be the value of a **CGI** element, all ampersands in *scgi* should be converted to tab characters. The destination server is responsible for converting the tabs to ampersands before initiating the copy.

size If specified, the destination server should use this to determine if sufficient space is available and reserve the appropriate space.

slfn Is the logical file name at the source server of the copied file to be copied. This element must be specified if the *tlfn* is not the same as the *slfn*. If not specified, the *slfn* is assumed to be the *tlfn*. It is always safe to specify this element.

snum If specified, the destination server should use the specified number of **TCP** streams to effect the copy. The actual number of streams may differ from the number specified due to destination server constraints.

sprot Is the source protocol to use. This is an optional token meant to be used for cross-protocol copies.

tprot Is the target protocol to use. This is an optional token meant to be used for cross-protocol copies.

Notes

1) The **tpc.dlgon=1** token is provided to allow constricting a destination copy **URL** that works as with delegated and non-deleted **TPC**, allowing the server to use delegation when so configured. See the section on determining if delegation is supported.
2) Meaningful **CGI** in the **tpc.scgi** element is the presence of any **CGI** elements excluding variables that start with "**tpc.**", "**xrd.**" and "**xrdcl.**". While the client may remove these elements to see if any meaningful elements remain; it is not required to do so. The server should always remove these elements from the *sgci* value before forwarding the **CGI** to the source server.
3) Generally, encoding the full time-of-day value with the process ID and parent ID as a hexadecimal string will achieve relative uniqueness for the TPC rendezvous key, as shown in the code snippet below.

```
char TPCKey[25];
struct timeval  currentTime;
struct timezone tz;
gettimeofday( &currentTime, &tz );
int k1 = currentTime.tv_usec;
int k2 = getpid() | (getppid() << 16);
int k3 = currentTime.tv_sec;
snprintf( TPCKey, 25, "%08x%08x%08x", k1, k2, k3 );
```

2.4 Client TPC Initiation

The is initiated by issuing a **sync()** operation against the destination file. The **sync()** should not return until the copy operation has been initiated at the destination. Because a copy may require scheduling due to limited resources there may be a substantial delay before the copy actually starts. Note that at this point a rendezvous point has not been established at the source. However, any implementation should queue the destination open request for a reasonable amount of time to allow the client to establish a rendezvous point by opening the source file after the copy has been initiated. Initiating a copy request does not necessarily mean the destination has yet opened the file. It only means that the destination copy task has started or is about to start running. It is unpredictable at this point whether the client's source open will arrive before or after the destination's open for the same file.

2.5 Client Source Open

After the previous **sync()** completes, the client should open the source file in read-only mode whose logical file name (i.e. s*lfn*) is suffixed with **TPC CGI** elements shown below. This step is unnecessary when delegation is being used.

```
slfn?tpc.dst=hostname&tpc.key=token&tpc.stage=copy[opts]

opts: &tpc.ttl=sec
```

Parameters

slfn Is the logical file name at the source server to be copied.

hostname
Is the fully qualified **DNS** name of the destination server. This is the server that copies the file.

token Is a client generated hexadecimal ASCII string that was used during the required destination open in the previous step.

opts Are optional **CGI** elements that must be used in certain circumstances, as described below.

sec If specified, is the maximum amount of time the rendezvous point may remain valid. The destination server must rendezvous with the specified key within this time interval. If not specified, system defaults apply. A server may impose a limit for the maximum value and if exceeded use the imposed maximum.

Notes

1) Reasonable short *ttl* values should be used as it is likely that the destination server has already attempted or is about to attempt a source file open.

2.6 Client Wait for Completion

The client should wait for completion by issuing a **sync()** request against the open destination file. The **sync()** should not return until after the copy completes or fails. The **sync()** should also delay its return until the checksum has been validated if the client requested checksum validation.

2.7 Client Closes Files

The copy at this point is finished and the client should close the source and destination files.

3 Destination Server Copying Source File

Once the copy has been initiated by the client at the destination server, the destination server should read-only open the source file (i.e. *slfn*) suffixed with **TPC CGI** elements shown below.

```
slfn?tpc.key=token&tpc.org=user@hostname&tpc.stage=copy
```

Parameters

slfn Is the logical file name at the source server that will be copied file.

token Is a client generated rendezvous token supplied to the destination server.

user@hostname

Is the client's identification:

user is the client's login identifier, followed by a dot, followed by the client's process ID at *hostname* (e.g. *loginid.pid*)

hostname is the fully qualified **DNS** name of the client's host.

Notes

1) If the destination server attempts a **TPC** open and there is no rendezvous point the open should be delayed for a short amount of time to allow the client to establish the rendezvous point via a source open. If the client does not do so with this time window, the open should fail.
2) A third party copy should not be initiated until after the client issues the first **sync()** request against the open destination file. If the copy cannot be started, the **sync()** should be delayed until the copy operation is launched.

3.1 Rendezvous Completion

A source server should only complete a rendezvous requested by a destination server upon source file open when all of the following are true:

- A rendezvous point has been established by the client. This may occur before or after the destination server opens the source file.
- The destination's hostname matches the **TPC.dst** value provided by the client.
- The destination-provided **TPC.key** value matches **TPC.key** value provided by the client.
- The destination-provided **TPC.org** value matches identification of the client that established the rendezvous point.
- The *slfn* in the rendezvous point matches the *slfn* being opened by the destination server.
- The client established rendezvous point has not exceeded it time to live value (*ttl*).

If all of the above are true, the destination server may open the source file. Otherwise, the open should fail.

3.2 Rendezvous Cancellation

A rendezvous requested by a destination but not yet established by a client or one established by a client but not yet requested by a destination is considered pending. A pending rendezvous should be cancelled if any of the following events occur:

- The time to live value (*ttl*) is exceeded. While client may request a specific *ttl* this value should be able to be constrained by the source server. Destination servers should not be allowed override a source server's imposed *ttl*.
- A client attempts to establish a new rendezvous point when matching one is already pending. The new rendezvous request should not be honored.
- A destination server attempts to establish a new rendezvous point when a matching rendezvous point is already pending for the destination. The new rendezvous request should not be honored.

3.3 Third Party Copy Cancellation

A destination server may queue a client **TPC** request for any reason but typically so as not to exceed its resource limits. When a copy is pending it is considered cancellable under the same conditions as an in-progress copy described below.

Once a destination server successfully rendezvous with a rendezvous point the source file is considered open and the copy is considered in-progress. An in-progress copy cannot be cancelled by the source server except for server failure. The destination server should cancel an in-progress copy if any of the following events occur:

- The client closes the destination file either via **close()** or a disconnect.
- The client requests that the in-progress copy terminate. The mechanism used may vary by implementation. In **XRootD** in-progress termination is requested by an **SFS_FCTL_SPEC1 fctl()** operation against the open destination file with the argument of "**ofs.tpc cancel**".

4 Delegated TPC Copies (a.k.a TPCLite)

The **TPC** protocol also supports delegated copies. The protocol is a subset of the standard **TPC** protocol which is why it is known as **TPCLite**. When a server supports delegated third party copies, the following should occur:

1. The client should determine whether or not delegated **TPC** is supported by the destination server. This provides early detection of whether a delegated **TPC** transfer is even possible. This step is optional as non-support can be detected later in the protocol steps. See the section on how to determine if delegated **TPC** is supported in the **XRootD** protocol.
2. The client may optionally open the file at the source and obtain the file size and indicate that the source will be subject to **TPC** transfer. This step is optional but it may be used to prepare the file for transfer, as needed. The file should be closed after this step completes. This step is detailed under standard TPC.
3. The client should open the file at the destination in the write-create mode and supply **CGI** elements to allow the destination to copy the source file. The **CGI** elements are the same as for standard TPC. Pay special attention to the optional **tpc.dlg** and **tpc.scgi CGI** tokens. The client should issue a **sync()** request against the open destination file. The **sync()** request should not return until the third party copy is initiated or fails to initiate.
4. The client should issue another **sync()** request against the open destination file. The **sync()** request should not return until the third party copy completes or fails.

Notice that under delegated **TPC** the client need not keep the source file open. In fact, there is no requirement that the client ever contact the source server. This significantly reduces **TPC** overhead which is why it is called **TPCLite**.

TPC cancellation follows the same steps as in standard **TPC**.

4.1 Determine if delegated TPC is Supported

Various implementations may use different schemes to indicate whether or not delegated **TPC** is supported. In **XRootD** this is done using the "**query config tpcdlg**" request (i.e. **kXR_query** request code with the **kXR_Qconfig** flag set and an argument of "**tpcdlg**"). If supported, the response is a list of authentication protocols that may be used (e.g. **gsi**). If it is not supported, "**tpcdlg**" is returned.

In order to provide full compatibility between delegated and non-delegated **TPC**, construct the destination **URL** in such a way that the copy would work as long as any kind of **TPC** is supported. This means supplying a proper rendezvous key and specifying "**tpc.dlgon=1**" **CGI** token. After successfully opening the file at the destination, query whether delegated **TPC** is supported at the destination server. If it is, copy with delegation will be used. If not, but **TPC** is supported then perform the subsequent steps to complete non-delegated **TPC**.

4.2 Delegated Destination Server Copying Source File

When delegation is in effect, no **TPC CGI** tokens should be present in the source URL Delegation simply means that the destination server may copy the file as if the client was actually copying the file.

5 Document Change History

29 Mar 2014

- New Document.

21 Oct 2016

- Correct case used in **CGI** elements.

24 Aug 2018

- Document **TPCLite** protocol.
- Document the **tpc.str CGI** token to control number of **TCP** streams.

24 Oct 2018

- Document the **tpc.dlg, tpc.spr,** and **tpc.tpr CGI** tokens.

13 Nov 2019

- Document the **tpc.scgi CGI** token.

17 Jan 2020

- Document the **tpc.dlgon CGI** token.

13 Apr 2020

- Minor editorial changes.