D5.5: Data streaming software client

Harro Verkouter

September 4 2017

Background

The data set sizes for e.g. high energy physics and radio astronomy have virtually exploded. In itself, sheer data set size is not a particular interesting phenomenon – High Performance Computing- and storage solutions for processing them have scaled accordingly.

The same cannot be said for data transport. When voluminous data sets – from several Gigabytes to a Terabyte – need to be moved, the network protocol becomes the limiting factor.

Network technology and international connectivity rarely are the problem either: 100Gbps network connections are becoming more commonplace, but many tools still rely on the TCP[1] protocol from 1981[2]. It is very robust but designed for small packets over (very) low bandwidth connections. Another problem is that the protocol's performance is quite sensitive to round-trip-time[3]

Many attempts have been made to scale the TCP protocol performance up for long, fat network pipes but with mixed success.

At JIVE experience was gained using the UDP-based Data Transfer (UDT) protocol[4] library. UDP is an unreliable protocol but does not suffer from TCP's shortcomings, thus allowing for higher data transfer speeds. UDT adds the reliability whilst retaining most of UDP's performance.

[1] Transmission Control Protocol
[2] It was designed earlier but the Internet Protocol version 4 (IPv4) version appeared in 1981
[3] The time it takes a packet from sender to receiver and back – from Europe to New Zealand this is approximately 0.3 seconds
[4] http://udt.sourceforge.net/index.html

ASTERICS is a project supported by the European Commission Framework Programme Horizon 2020 Research and Innovation action under grant agreement n. 653477

Deliverable

A data streaming software system ("etransfer") comprising of server- and client programs was developed using the C++11 programming language. The UDT protocol is only available as library and as such application software had to be written around it; extending existing data transfer tools like `scp` or `ftp` was deemed impossible because they implicitly rely on the TCP protocol.

In the etransfer model the server (`etd` – short for e-transer daemon) is started with two separate connections: a command server and a data server. The client software (`etc` – e-transfer client) connects through the command connection and negotiates a file read- or write request. The bulk data transfer then proceeds over the data channel the server was started with; the server sends its data channel address(es) to the client.

The etransfer software supports TCP and UDT over both IPv4 and IPv6 for all of its connections.

The server administrator may start the etransfer server with multiple command- and/or data protocols and/or port numbers. Only the protocol(s) and port number(s) of the command connection(s) should have to be made public. For the data connection the client program loops over all data addresses the server program was started with, in order, and uses the first one which succesfully connects.

In normal client/server data transfer programs like `scp` and `ftp` only transfers between the client and one server are possible: a user can upload a file to a server or download it from there. The etransfer server/client system allows the client program to initiate server to server transfers, just by specifying two remote locations. In such a case the data does ***not*** flow through the user's computer or network link but the remote source connects directly to the remote destination.

The system natively supports remote wildcards; it is possible to transfer multiple files irrespective of wether they are remote or local.

Because the etransfer programs expect to transfer large files over long distances, it supports resuming of interrupted transfers. This is done by comparing file sizes at the source and destination and only the remaining bytes are transferred.

Although called resuming, the system regards each file to be transferred as a "resume" operation. The difference is in how the combination of existence and/or length of the destination file is handled. By default, an existing remote file will not be overwritten and an error message generated. Three modes are supported to change this behaviour:

- overwrite: this restarts the file transfer. The remote file is truncated to zero size after which the whole source file is transferred.
- skip existing: this assumes that if the remote file exists, it is complete. This is not checked. No bytes are transferred and no error is generated. The client continues transferring the next file.
- resume: this is the actual resume operation. If the source file is longer than the destination file, the remaning bytes are transferred. If the source file's size is shorter or equal to the destination no bytes are transferred and no error is generated.

ASTERICS is a project supported by the European Commission Framework Programme Horizon 2020 Research and Innovation action under grant agreement n. 653477

Synopses

Below follows the command line help for both the server- and client for illustration purposes.

The `etd` server

The daemon by default backgrounds and refuses to run with root privilege.

Usage: etd [--buffer <unsigned long>...] --command <string>... --data <string>... [-f] [-h] [--help] [-m <int>...] [--mss <unsigned int>...] [--run-as <user name>] [--version]

'ftp' like etransfer server daemon, to be used with etransfer client for high speed file/directory transfers.
Addresses are given like (tcp|udt)[6]://[local address][:port]
where:
  [local address] defaults to all interfaces
  [port] defaults to 4004 (command) or 8008 (data)

IPv6 coloned-hex format is supported for [local address] by enclosing the IPv6 address in square brackets: [fe80::1/64%enp4]

[--buffer <unsigned long>...]
  Set send/receive buffer size. Default 33554432

--command <string>...
  Listen on this(these) address(es) for incoming client control connections

--data <string>...
  Listen on this(these) address(es) for incoming client data connections

[-f]
  Run in foreground, i.e. do NOT daemonize
  Default: false
  Constraints:
   xor:mutually exclusive with { [--run-as <user name>] }

[-h]
  Print short usage and exit succesfully

[--help]
  Print full help and exit succesfully

[-m <int>...]
  Message level - higher = more output
  Constraints:
   constraint:maximum value less than or equal 5
   constraint:minimum value greater than or equal -1

[--mss <unsigned int>...]
  Set UDT maximum segment size. Not honoured if data channel is TCP. Default 1500
  Constraints:
   constraint:minimum value greater than or equal 64
   constraint:maximum value less than or equal 65536

[--run-as <user name>]
  Run daemon under this user name
  Default: verkouter
  Constraints:
   constraint:user name must exist on this system
   xor:mutually exclusive with { [-f] }

[--version]
  Print version and exit succesfully

ASTERICS is a project supported by the European Commission Framework Programme Horizon 2020 Research and Innovation action under grant agreement n. 653477

The etc client

Usage: etc [--buffer <unsigned long>...] [-h] [--help] [--list <URL>] [-m <int>] [--mode <file copy mode>] [--mss <unsigned int>...] [--overwrite] [--resume] [--skipexisting] [-v --verbose] [--version] <URL>...

'ftp' like etransfer client program.
This is to be used with etransfer daemon (etd) for high speed file/directory transfers or it can be used to list the contents of a remote directory, if the remote etransfer daemon allows your credentials to do so.

positional arguments:
<URL>...
SRC and DST URL/PATH: remote: ((tcp|udt)[6]://)[user@]host[:port]/path or local: /<path> (i.e. absolute path)
Constraints:
constraint:At most one local PATH can be given
xor:mutually exclusive with { [--list <URL>] }
xor:at least one of these alternatives must be present

[--buffer <unsigned long>...]
Set send/receive buffer size. Default 33554432

[-h]
Print short usage and exit succesfully

[--help]
Print full help and exit succesfully

[--list <URL>]
Request to list the contents of URL
Constraints:
constraint:Can only list remote URLs
xor:mutually exclusive with { <URL>... }
xor:at least one of these alternatives must be present

[-m <int>]
Message level - higher = more output
Constraints:
constraint:maximum value less than or equal 5
constraint:minimum value greater than or equal -1

[--mss <unsigned int>...]
Set UDT maximum segment size. Not honoured if data channel is TCP. Default 1500
Constraints:
constraint:minimum value greater than or equal 64
constraint:maximum value less than or equal 65536

[--overwrite]
Existing target file(s) will be overwritten
Constraints:
xor:mutually exclusive with { [--resume] | [--skipexisting] }

[--resume]
Existing target file(s) will be appended to, if the source file is larger
Constraints:
xor:mutually exclusive with { [--overwrite] | [--skipexisting] }

[--skipexisting]
Existing target file(s) will be skipped
Constraints:
xor:mutually exclusive with { [--overwrite] | [--resume] }

[-v --verbose]
Verbose output for each file transferred
Default: true

[--version]
Print version and exit succesfully

ASTERICS is a project supported by the European Commission Framework Programme Horizon 2020 Research and Innovation action under grant agreement n. 653477