BBCP

14.09.02.00.0

1 BBCP Syntax 3

1.1 Checksum Considerations (-e and –E) 16

1.2 Tuning Considerations 18

1.2.1 Window Size (-w) 18

1.2.2 Streams (-s) 19

1.2.3 I/O Buffer Size (-B) 20

1.2.4 Output Ordering (-o and –b +) 20

1.2.5 Input Blocking (-b) 21

1.2.6 Compression (-c) 21

1.2.7 Un-buffered I/O (-u) 22

1.2.8 Routing (-z) 22

1.3 Resuming Failed Copies (–a and –k or –K) 23

1.4 Multi-Directory Copying (-d) 24

1.5 Using Pipes (-N) 25

1.5.1 Named Pipe Considerations 25

1.5.2 Program Pipe Considerations 26

1.5.3 Using Program Pipes to Recursively Copy Small Files 26

1.5.4 Restricting Program Pipes 27

1.6 Real-Time Copying (-R) 28

1.6.1 Real-Time Copy Protocol 29

1.6.2 Using an Alternate Lock File 30

1.7 Modifying ssh Startup (-S and –T) 32

1.8 Dealing With Firewalls (-z and -Z) 33

1.9 Configuring New Defaults (-C) 34

1.10 Recursive Copy Compatibility With cp & scp 34

1.11 New Features 35

1.12 Backward Compatibility 37

1.13 Problem Reports & Enhancement Requests 38

1.14 Downloading 38

1.15 Legal Notice 39

Download the pdf version of this document.
Updated 9/2/2014

1 BBCP Syntax

bbcp [*options*] [*srcspec* [. . .]] *snkspec*

srcspec: [[*sid*@]*shost*:]*source* [*srcspec*]

snkspec: [*tid*@]*thost*:]*target*

options:

{**--c**ompress | **-c}** [*lvl*]	{**--config** | **-C**} *fname*
{**--f**orce | **-f**}	{**--in**files | **-I**} *fname*
{**--h**elp | **-h**}	{**--k**eep | **-k**}
{**--license** | **-$**}	{**--l**ogfile | **-l**} *logfn*
{**--mkd**ir | **-A**}	{**--m**ode | **-m**} *mode*
{**--omit** | **-O**}	{**--p**reserve | **-p**}
{**--prog**ress | **-P**} *sec*	{**--pt**ime | **--~**}
{**--r**ecursive | **-r**}	{**--symlink**s | **-@**} *slarg*
{**--v**erbose | **-v**}	{**--version** | **-#**}
{**--w**indowsz | **-w**} [**=**]*wsz*	
[*advanced*]	**--**

---

mode: *dmode***/** | *fmode* | *dmode***/***fmode*

slarg: **follow** | **keep** | **ignore**

shost: *hostname* | **[***ipv6addr***]** | *ipv4addr*

thost: *hostname* | **[***ipv6addr***]** | *ipv4addr*

advanced:

{**--a**ppend | **-a**} [*dir*] {**--b**uffers | **-b**} [**+**]*blkf*

{**--buffsz** | **-B**} *bfsz* {**--debug** | **-D**}

{**--dir**base | **-d**} *path* **-e** | {**-E** | **--check**sum} *csarg*

{**-F** | **--nofs**chk} {**--g**ross **| -g}**

{**--i**dfile | **-i**} *fname* {**--ipv4** | **-4**} [*who*]

-K **-L** *lopts*[**@***lurl*]

{**--n**odns | **-n**} {**--order** | **-o**}

{**--pipe** | **-N**} *nio* {**--port** | **-Z**} *p1*[**:***p2*]

{**--qos** | **-q**} *qos* {**--realtime** | **-R**} [*rtargs*]

{**--s**treams | **-s**} *strms* {**--sync** | **-y**} **d**[**d**]

-S *srccmd* **-T** *trgcmd*

{**--t**imelimit | **-t**} *tlim* {**--u**nbuffered | **-u**} *loc*

-U *wsz* {**--vv**erbose | **-V**}

{**--x**frrate **| -x**} *rate* {**-z** | **--rev**erse}

---

csarg: [%]{**a32** | **c32** | **md5**}[**=**[*csval*|*csfile*]]

lopts: **a** | **b** | **c** | **I** | **o** | **r** | **w** | **x** | [*lopts*]

lurl: **file://***path***/***filename* | **x-netlog://***host*:*port*

x-syslog://localhost

loc: **s** | **t** [*loc*]

nio: **i** | **o** [*nio*]

who: **c** | **s** | **t** [*who*]

Function

Securely and quickly copy data from source to target. Please refer to known problems for a list of current defects and limitations.

Parameters

sid specifies the ssh loginid for the source host. The default is to use your current loginid.

shost specifies the name of the host that holds the source file. By default, the local host is assumed. The *shost* may be expressed as:

hostname	a **DNS** registered hostname
[*ipv6addrs***]**	a complete RFC2732 **IPv6** address enclosed in brackets
Ipv4addr	an ipv4 address in familiar dot notation

source specifies the name of the file to be copied. Any number of source files from any number of hosts may be specified on the command line. If the **-I** option is specified, no source files need to be specified on the command line. When **–N** is specified, *source* must be a named pipe or a program.

tid specifies the ssh loginid for the target host. The default is to use your current loginid.

thost specifies the name of the host to which the file is to be copied. By default, the local host is assumed. The *thost* may be expressed as:

hostname	a DNS registered hostname
[*ipv6addrs***]**	a complete RFC2732 ipv6 address enclosed in brackets
Ipv4addr	an ipv4 address in familiar dot notation

target specifies the name the target location of the file to be copied. If a single source file is specified, *target* may be a filename or a directory. If more than one source file is specified, *target* must be the name of a directory. When **–N** is specified, *target* must be a named pipe or a program.

Options

--compress [*lvl*] | **-c** [*lvl*]

compresses the data prior to sending it across the network using **zlib** written by Jean-loup Gailly and Mark Adler. Specify an integer value from 1 through 9 for *lvl*. A value of 1 gives the best speed while a value of 9 gives the best compression. The default value is 1. If *lvl* is omitted, **-c** may not be the last option on the command line. See the section on "Compression" for a discussion of this option.

--config *fname* | **-C** *fname*

specifies the name of the configuration file. The configuration file is processed when it is encountered on the command line. See the section "Configuring New Defaults" for information about configuration files.

--force | **-f**

forces the copy by erasing the target prior to copying the source file. By default, if the target already exists for the source file, the copy fails. See also the **–K** option modifier.

--infiles *fname* | **-I** *fname*

includes a list of source file specifications from the file identified by *fname*. Each new-line terminated record in *fname* must contain a single source file specification. If **–I** is specified, you need not specify any source files on the command line.

--help | **-h**

prints usage and version information and immediately exits..

--keep | **-k**

keeps any partially created target files and allows full recovery after a copy failure. Refer to "Resuming Failed Copies" for details. The **-o** becomes the default. Normally, partial files are removed after a copy fails.

--license | **-$**

prints the license agreement and immediately exits.

--logfile | **-l** *logfn*

logs standard error to the indicated file, *logfn*, By default, standard error output is written to the terminal.

--mkdir | **-A**

creates the destination directory, as specified by target specification, if it does not exist.

--mode *mode* | **-m** *mode*

sets the final mode for the target file. Specify a 3- or 4-digit mode in octal for *dmode* or *fmode* (see the syntax diagram). The *dmode* specified the access mode to be assigned to directories and the *fmode* to the file itself. The default *dmode* is 0755 while *fmode* is 0644.

--omit | **-O**

omits copying files from the source if an identically named file exists at the target. This is particularly useful with **–r** (recursive copy).

--preserve | **-p**

preserves the source file's mode, group name, access time, and modification time. That is, the target file's mode, group, access and modification times are set to match that of the corresponding source file.

--progress *sec* | **-P** *sec*

produces progress messages every *sec* seconds. Specify a *sec* value no less than 1 second.

--ptime | **-~**

only preserves the source file's access time, and modification time. That is, the target file's access and modification times are set to match that of the corresponding source file but the target file's mode and group are unchanged (i.e. determined by the destination defaults).

--recursive | **-r**

performs a recursive copy by copying all files starting at the source directory unless the directory ends with slash, in which case the contents and all descendants are copied. Empty directories and symbolic links are ignored. The **–r** option is mutually exclusive with the **–E** option that specifies a known checksum value.

--symlinks *slarg* | **-@** *slarg*

specifies how symbolic links are to be handled during a recursive copy (**--link**s is a synonym). Valid values for *slarg* are:

follow - follow each symbolic link and copy its target.

keep - recreate the symbolic link at the destination.

ignore - ignore all symbolic links (the default).

--verbose | **-v**

produces additional output during execution.

--version | **-#**

prints the version number and immediately exits.

--windowsz *wsz* | **-w** *wsz*

sets the preferred size of the TCP window. Specify for *wsz* a value no less that 8192 (i.e., 8k). Numbers suffixed by k, m, or g are multiplied by 2^{10}, 2^{20}, or 2^{30}, respectively. Prefixing wsz with an equals sign disables any auto-tuning and attempts to use the specified window size. See the section "Window Size" to choose an appropriate non-default value. The default is to use auto-tuning when available, otherwise a fixed size of 128K.

-- is a null option. Use – when an option with an optional argument would be the last option on the command line.

Advanced Options

--append [*dir*] | **-a** [*dir*]

appends data to the end of the target file if the target is found to be incomplete due to a previously failed copy. The optional *dir* specifies the directory on the target host where checkpoint information is to be written (the default is *home*/.bbcp). The **–a** option is mutually exclusive with the **–E** option. See the section "Resuming Failed Copies" for more details. If *dir* is omitted and **-a** is the last option on the command line it must be followed by the **--** option (double dash).

--buffers *blkf* | **-b** *blkf*

specifies the read blocking buffer factor. That is, *blkf* data blocks are always read from disk and then queued for sending across the network. The maximum is determined by the maximum number of scatter/gather buffers allowed in a readv() system call. See the section "Input Blocking" for cases where it might help. The default is 1 and set to 1 if **–u t** is specified.

--buffers **+***blkf* | **-b +***blkf*

adds *blkf* additional output buffers. This option is meant to be used when ordered output is in effect (see **–o**). See the section "Output Ordering" for cases where it might help. The default is 0.

--buffsz *bfsz* | **-B** *bfsz*

specifies the disk I/O buffer size and becomes the effective I/O transfer unit. Specify for *bfsz* a value no less that 1024 (i.e., 1k). Numbers suffixed by k, m, or g are multiplied by 2^{10}, 2^{20}, or 2^{30}, respectively. The *bfsz* is always made a multiple of the page size, the minimum value possible. The default is *wsz**1.25 or 512K, whichever is smaller. See the section "I/O Buffer Size" for cases where changing the default may help.

--debug | **-D**

turns on debugging.

--dirbase *path* | **-d** *path*

specifies source relative addressing. Each relative *srcspec* is prefixed by path. When the file is copied to the target, then the destination path will be *snkspec*/*srcspec*. That is, the relative path in *srcspec* will be created on the target host relative to *snkspec* and then the file will be copied. See the section "Multi-Target Copying" for more information.

-e calculates a checksum for each block of data sent using the algorithm specified by the **–E** option and is equivalent to "**–E md5**" (see below).

-E [**%**]*cstype*[**=**[*csval*|*csfile*]] | **--check**sum [**%**]*cstype*[**=**[*csval*|*csfile*]]

calculates a checksum either for the whole file and on a block level if –e is specified. It also optionally checks the checksum against a known value (see *csval*) or be reports it (see *csfile* and the **–v** option). **–E** usually implies **–o** and disallows **–a** as well as **–r** if a value is specified. See the section "Checksum Considerations" for details. The arguments are:

% computes the checksum on the source node. By default, the checksum is computed on the target node.

cstype specifies the checksum algorithm to be used. Supported types are:

a32 Adler 32-bit checksum.

c32 Cyclic Redundancy Check (CRC) 32 bit checksum.

md5 Message Digest 128-bit checksum.

csval is the known checksum value specified as a hexadecimal ASCII string with the correct number of digits relative to *cstype*. When *csval* is specified, the calculated checksum is compared to this value and the copy fails if they do not agree. If csval is not specified, the computed checksum is written either to standard error or *csfile*.

csfile is the name of the file where the checksum information is to be appended. The filename must not start with a hexadecimal digit. If *csfile* is not specified, the value is written to standard error. The output format is:

Checksum: *cstype csval host***:***destfile*

cstype checksum type (i.e., **a32**, **c32**, or **md5**).

csval computed checksum value.

host target host name.

destfile name of the file created at *host*.

-F | **--nofs**chk

forces the copy by not checking if there is enough free space on the target host. This option is especially useful for esoteric devices that do not report the correct amount of free space.

--gross | **-g**

recreates the source directory structure even if it is empty (i.e. no files or symbolic links). This is only meaningful for recursive copies.

--idfile *fname* | **-i** *fname*

specifies the name of the **ssh** identity file if one has been specifically created for **bbcp**. The identity filename, prefixed by **–i**, is included in the **ssh** command line when starting the source and target nodes.

--ipv4 [*who*] | **-4** [*who*]

uses the **IPv4 TCP** stack for *command* processing and *source-target* connections. This option is incompatible with **IPv6** addresses. The optional *who* argument restricts **IPv4** mode to the *command* if **c** is specified, the *source* if **s** is specified or the *target* if **t** is specified. The default is **cst** (i.e. everywhere). If *who* is omitted and **–-ipv4** is the last option on the command line it must be followed by the **--** option (double dash).

-K similar to **–k** but removes the ordering restriction unless another option requires it. With **–f** the target file is truncated, not removed, prior to copying. This option is only useful when the destination is a symbolic link to a pre-created file place-holder. The **–K** option is mutually exclusive with **–k**.

-L *lopts*[**@***lurl*]

enables detailed logging of actions via the NetLogger interface. The *lopts* specify what is to be logged while *lurl* determines how it is logged. For *lopts* specify one or more of the following:

a – append to data file
b – buffer information in memory
c – log data compression
i – log network reads
o – log network writes
r – log disk reads
w – log disk writes
x – log data expansion

If *lurl* is not specified, the logging interface uses the value of environmental variable "NETLOGGER_DEST". Specify one of three destinations protocols:

file - data is written to the file identified by *path/filename*
x-netlog - data is sent to *host* listening on *port*
x-syslog - data is sent to the system log on the local host

--nodns | **-n**

does not use **DNS** to resolve IP addresses to host names. This option is especially useful when the source or target machine is neither registered in **DNS** nor has a valid entry in the /etc/hosts file.

--order | **-o**

serializes the output stream to force ordered (i.e., sequential) output. The default is to write blocks in the order that they arrive from the network, which is usually somewhat random order. See the section "Output Ordering" for more information. The **–o** becomes the default when **–a**, **–k**, **–E +st**, **–E +t**, **–N o**, or **–u t** is specified.

--pipe *nio* | **-N** *nio*

allows copying using pipes. When *nio* contains an '**i**', the data *source* must be a named pipe or a program. When *nio* contains an '**o**', the data *target* must be a named pipe or a program. Both *source* and *target* may be named pipes or programs. The **–N** option is mutually exclusive **–a**, **–d**, **–K**, **–r**, **–R**, **–u,** and **-@**. Also, '**–N o**' is mutually exclusive with **–f**, **–m**, and **–p**; and **–o** becomes the default. Only a single *source* may be specified with ' **–N i**'. See the section "Using Pipes" for more information and restrictions .

--port *p1*[**:***p2*] | **-Z** *p1*[**:***p2*]

specifies the port or port range to be used in accepting data connections. Normally, the first available port is used. This option restricts the choice to the specified value or range. Specify a value between 1 and 65535, inclusive for *p1* and, optionally, *p2*. If *p2* is specified, it must be greater than or equal to *p1*. See the section "Dealing with Firewalls" for more information.

--qos *qos* | **-q** *qos*

specifies the quality of service to be used. This is router-implementation dependent and may ignored. Specify a value between 0 and 255, inclusive.

--realtime [*rtargs*] | **-R** [*rtargs*]

enables real-time copy mode. See the section "Real-Time Copying" for a full description of this mode and *rtargs*. If *rtargs* is omitted and the option is the last option on the command line it must be followed by the **--** option (double dash).

--streams *strms* | **-s** *strms*

sets the number of parallel network streams to be used for the transfer. The default is 4. See the section "Streams" for other possible values.

--sync d[**d**] | **-y d**[**d**]

synchronizes memory-based file pages with disk pages before closing the output file. A single **d** synchronizes data pages, while **dd** synchronizes data pages as well as inode pages (i.e. meta-data). Review the notes on various admonitions about using this.

-S *srccmd*

is the command to be used to start **bbcp** on the source host. The default is "ssh –x –a –oFallBackToRsh=no %4 %I –l %U %H bbcp". See the usage notes for more information.

-T *srccmd*

is the command to be used to start **bbcp** on the target host. The default is "ssh –x –a –oFallBackToRsh=no %4 %I –l %U %H bbcp". See the usage notes for more information.

--timelimit *tlim* | **-t** *tlim*

is the maximum amount of time that the copy may take before it is aborted. The time limit applies to each source host regardless of the number of files that host supplies. Specify a number greater than zero and optionally suffixed by **s** (the default), **m**, or **h** for seconds, minutes, and hours, respectively. By default, no time limit applies.

--unbuffered *loc* | **-u** *loc*

requests un-buffered (i.e., direct) I/O at the locations specified by *loc*. This option may actually reduce performance. See the section "Un-buffered I/O" for more information. The **–u** option forces **–b 1** and the I/O buffer size is set to a multiple of 8K. The **–u t** option forces **–o**. Specify one or more of the following letters for *loc*:

s - source-side **t -** target-side

-U *wsz*

sets the size of all I/O buffers, including the TCP/IP socket buffer. This option is identical to the **–w** option except that host limits are not checked and buffers are set to the specified size whether or not the operating system considers them valid. Improper use of this option may cause the copy to fail. This option is only useful when dealing with experimental TCP/IP stacks. Most users should use the **–w** option.

--vverbose | **-V**

produces even more output than **–v** allows, including detailed transfer speed statistics.

--xfrrate *rate* | **-x** *rate*

sets the maximum transfer rate. Specify for *rate* a value no less that 1024 (i.e., 1k). Numbers suffixed by k, m, or g are multiplied by 2^{10}, 2^{20}, or 2^{30}, respectively. Data is clocked out from the source at the specified rate per second.

-z | **--rev**erse

uses reverse connection protocol. See the section "Dealing with Firewalls" for more information.

Success

The program exists with a status code of 0.

Failure

The program exits with a non-zero status code.

Notes

1) A list of known problems is detailed on the following web page: http://www.slac.stanford.edu/~abh/bbcp/bbcp_bugs.html
2) Files are copied in the order specified. To minimize start-up and shutdown time, adjacent files are grouped by source host and treated as a copy set (i.e., a related group of files). Avoid inter-mixing different source locations. That is, always specify all the required files from one source location before specifying files from another location.
3) The destination file system must have sufficient space to comfortably hold all of the source files. If sufficient space does not exist at the start of a copy set, the copy is terminated. Use the **–F** option to avoid this requirement.

4) While the copy is in progress, the target file has 0200 as its mode (i.e., owner write-only). The mode is changed only after the copy succeeds.
5) The **sync** option makes sure that all pages have been written to disk before the copy is considered complete. A single **d** argument only makes sure that data pages are synchronized and is supported by all POSIX compliant file systems. A double **d** (i.e. **dd**) argument synchronizes data as well as directory pages (i.e. inodes) and is not uniformly supported by all file systems. If '**sync dd**' is specified and the underlying file system does not support inode synchronization, an error may result. You should check for support before you use this option in production wrokflows.
6) Using the **sync** option may significantly impact the overall speed of the copy as **bbcp** has to wait until all data is actually written to disk.
7) By default, **bbcp** uses **ssh** for authentication on every host that was specified in the source and target specifications. The rules attending to normal **ssh** use always apply to **bbcp**. When in doubt, simply **ssh** to the host in question to validate your ability to copy files to or from that host.
8) Because **bbcp** invokes **ssh** and itself without an absolute path, you must make sure that **bbcp** and **ssh** can be found in one of the directories listed in your PATH environmental variable. Otherwise, you must specify where **bbcp** and **ssh** can be found (see the next note).
9) The **–S** and **–T** options allows you to specify different commands to start **bbcp** on the source and sink nodes. Refer to the section "Modifying ssh Startup" for details on how to change the default location of **bbcp** and **ssh**.
10) **bbcp** allows the source to be /dev/zero and the destination to be /dev/null. This is very useful in measuring actual network bandwidth.
11) Refer to http://netlogger.lbl.gov/home for complete information on NetLogger.
12) You can easily GRID enable **bbcp** from the security standpoint by specifying GSI-OpenSSH as the authentication and launch vehicle using the **-S** and **-T** options.
13) The **–ipv4** option is meant to be used in cases where either the *source* or *target* node has either been improperly configured for **IPv6** or has a non-working **IPv6** implementation. By default, both parties are forced to operate in **IPv4** mode. Consequently, parties must have an **IPv4** address. In mixed mode environments use the *who* argument.
14) Specifying **–ipv4 c** substitutes **-4** for **%4** in the **ssh** command line. If you over-ride the default, you must include **%4** n the command line to have the **-4** option added if it is needed.

1.1 Checksum Considerations (-e and –E)

The **–e** and **–E** options allow you to verify the integrity of the copy but they incur CPU costs. Certain option combinations require that two checksums be calculated at the source node, further increasing the CPU cost but allowing for more flexibility. When you specify a known value for the checksum, **bbcp** can verify that no bit errors are introduced since the file was created. The **–e** and **–E** options interact. Below is a table that describes checksum processing based on various options an arguments and indicates whether or not ordered output (**–o**) is enforced.

Specification	–o	Checksum Processing
–E *cstype* **–E %***cstype*	N	Calculates block level checksums at the source and target to detect block transmission errors.
–e –E *cstype* **–e –E %***cstype*	N	Equivalent to the above.
–E *cstype*=[*csfile*]	Y	Calculates and *prints* a file checksum at the *target*.
–E %*cstype*=[*csfile*]	N	Calculates and *prints* a file checksum at the *source*.
–e –E *cstype*=[*csfile*]	Y	Calculates and *prints* a file checksum at the *source* and *target* nodes and verifies that they are the same.
–e –E %*cstype*=[*csfile*]	N	Calculates and *prints* a file checksum at the *source*. Additionally, block level checksums are calculated at the source and target to detect transmission errors.
–E *cstype*=*csval*	Y	Calculates the file checksum at the *target* and *compares* it to *csval*.
–E %*cstype*=*csval*	N	Calculates the file checksum at the *source* and *compares* it to *csval*.
–e –E *cstype*=*csval*	Y	Calculates the file checksum at the *source* and *target* nodes, verifies that they are the same, and *compares* it to *csval*.
–e –E %*cstype*=*csval*	N	Calculates the file checksum at the *source* and *compares* it to *csval*. Additionally, block level checksums are calculated at the source and target to detect transmission errors.

File checksums computed at the target always require ordered output while independent block checksums and source-only checksums do not.

Ordered output is difficult to achieve on links with highly variable latency; and may cause copy operations to stall. Even so, it is possible to validate or compute a file checksum without ordering the output by prefixing the *cstype* by a percent sign (%) and specifying **–e**. This combination requests transitive **s**ource checksum mode. Here, the file checksum is computed and optionally validated at the source; then independent checksums are computed for each block and sent to the target where they are validated. A file checksum can be reported because if the source checksum is correct and each block sent to the target suffered no errors, then the same checksum should also exist at the target. Because two checksums must be computed at the source node, the source may become compute-heavy. While not a true end-to-end checksum, it may provide sufficient integrity protection while avoiding output ordering.

If you do specify output ordering (**–o**) or if output ordering is forced by some other option, along with source-only checksumming; **bbcp** automatically reverts to source-target checksums to avoid computing two checksums on the source node. Therefore, "**–o –e –E** %*cstype*=" is equivalent to specifying "**–e –E** *cstype*=".

If full end-to-end checksums are not needed, you can save substantial amount of CPU time by not specifying the percent sign and **–e**. This combination requests direct **t**arget checksum mode. Here, the file checksum is only computed at the target node. If a *csval* is specified, it is only verified at the target. If incorrect, it is impossible to know where the error was introduced. Merely printing a target checksum, of course, is no guarantee that it is correct. Target verification mode still requires ordered output but no additional source CPU resources.

1.2 Tuning Considerations

bbcp has many options for tuning a file transfer to achieve maximum possible performance. Which to use and the appropriate values are completely determined by the type of devices and file systems being used at the source and target node as well as the network that joins them. This section describes what the tuning options actually do and how you might use them.

1.2.1 Window Size (-w)

The first and most important option is **–w**. This determines the TCP window size as well as the default I/O size. The default of 128Kis usually good enough for LAN and larger values are likely to hurt performance.

For wide area networks, you must compute the bandwidth delay product. This is the product of the network bandwidth and round trip time (**RTT**) between the source and target nodes. The basic formula is:

window = *netspeed*/8*RTT

Where *netspeed* is network bandwidth in bits per second and **RTT** is the round-trip time between the source and target; which can be determined using the **ping** or **traceroute** commands. The result is the optimal window size in bytes.

For example, assume you have a 1Gb link and want to send a file from San Francisco (SFO) to Geneva, Switzerland (GVA). The RTT is typically 175ms. Using the formula above, the ideal window would be 20.85MB, much larger than most operating systems allow. So, you will likely choose the largest value allowed that also does not overwhelm the memory resources of the machines being used.

Refer to http://www.speedguide.net/bdp.php for easily calculating the optimal window size.

However, you may not need to calculate a window size at all if the receiving host supports window auto-tuning. This relatively new addition to the TCP/IP stack is fully available in Linux 2.6.17 and above. Auto-tuning is used by default when it is available. While this mode does not require that you manually calculate the window it also assumes that the system administrator has properly configured the auto-tuning parameters and they are appropriate for your particular transfer.

That may not be the case and, if the **–v** option has been specified, **bbcp** will issue a warning if it detects settings that would likely compromise performance. Use the **–V** option display the window settings for an actual transfer.

When auto-tuning is inappropriate or just wrong, you can disable its use by prefixing the window size with an equals sign (e.g., **–w =128k**). This does not mean the window will equal the specified value; as **bbcp** always negotiates the actual window between the sending and receiving hosts.

Refer to http://www.psc.edu/networking/projects/tcptune/ for a good tutorial on tuning TCP stacks for maximum data transfer performance.

1.2.2 Streams (-s)

The **–s** options is the second most important tuning parameter. This specifies the number of parallel TCP streams. A naïve explanation would say that streams can make up for not having a large enough window. The idea is that if you can't get enough packets moving in a single window, then create multiple windows and run them simultaneously. This is only partially true. While multiple streams do provide multiple windows, multiple windows also parallelize traffic with independent time-outs, re-transmissions, and greatly improved I/O overlapping. This is a cumulative effect that dramatically increases the overall bandwidth utilization. But, too much of a good thing can also be bad, as you will see below.

The optimal number of streams can be calculated as:

$$\text{Streams} = window/actual$$

Where *window* is the calculated optimal window size and *actual* is the window size that is actually being used. It follows that the smaller the actual window the larger number of streams will be needed. Using the SFO-GVA example with a window size of 512K would imply that about 42 streams would be optimal. In practice, this is not a practical number and would likely reduce bandwidth utilization. The reason is that each additional stream requires more CPU and memory resources and the scaling is not linear. You can see this for yourself by copying a file and successively increasing the number of streams. As the number is increased the percentage improvement will gradually become successively smaller and will eventually become negative.

As a rule of thumb, start with the optimal calculation divided by two. Then take measurements 25% above and below that number and use whichever provides the best performance. In the local area network the default of 4 streams is usually sufficiently good for 1Gb LAN transfers; while 8 to 12 streams is good for WAN transfers.

1.2.3 I/O Buffer Size (-B)

The third most important tuning parameter is the I/O buffer size. This is the amount of data that will be read from disk, sent over the network in one request, and then written to the target device. By default, the buffer size is set to be the same as the window size. This is ideal to maximize parallelism but might not always be the best value. The ideal buffer size is mostly determined by the file systems being used and its underlying devices. To add more complications, file systems can be tuned for various performance targets and they are rarely tuned to provide the best network performance. Hence, defaulting for maximum parallelism is a good default.

If you know the performance characteristics of source and destination file systems then playing with the buffer size may improve overall network bandwidth utilization. Typically, a larger buffer will make up for device seek-time delays at the expense of reduced parallelism. However, this may still be a win. Generally, faster inter-connects (e.g., 10Gb) fair much better with larger I/O buffers than slower inter-connects; especially in the local area network. In the wide area network try using a buffer size twice the window size to see if any improvement can be realized.

1.2.4 Output Ordering (-o and –b +)

I/O ordering allows you to guarantee that the output stream is sequential in nature. Disk devices work best doing sequential I/O. So, you may wonder why ordering is not the default. The reason is that file systems tend to smooth out unordered output via their memory cache and **bbcp**'s unordered output is largely bounded (i.e., output tends to be more ordered than not). Furthermore, guaranteeing ordered output is memory intensive since an out-of order block must be buffered until its predecessor is received. Depending on how the parallel streams are routed and network latency variability, ordered output may lead to large demands for memory to the point that the copy stalls. The **–b +** option exists solely to inform **bbcp** how much more memory it should use to try to

prevent stalls. There is no formula because the value depends on the network being used and its performance at the time it is being used. If ordered output is desired and copies stall, start with a value twice the number of streams being used to see if that solves the problem.

You may need to worry about order tuning depending on which other options are chosen. The **–a** (append mode), **-c** (compression), **-k** (keep), **-E** (checksum), and **–u t** (un-buffered target) are sequential in nature and enforce an ordered output stream. Some file systems (e.g., **HDFS**) also require an ordered output stream. So, keep in mind these options and their side-effects.

1.2.5 Input Blocking (-b)

The I/O blocking option can provide some improvement in transfer rate by reading several blocks from the input device before queuing the blocks for transmission. Parallel queuing tends to smooth out variability when the TCP streams differ greatly in latency. The cost is a higher initial latency since no stream can start until the requested blocks are read from disk. Eventually, this latency tends to disappear but only if the input device is much faster than the network link. Additionally, there is somewhat less CPU utilization as there are fewer kernel calls to transfer the data. That said, the default is 1 and normally provides the best overall performance.

Consider increasing the blocking factor if the input device is much faster than the network link and window sized transmission units (i.e., -B is not specified) provide the best amount of parallelism.

1.2.6 Compression (-c)

One achieves the highest bandwidth utilization by not using any bandwidth at all and still accomplishing the task. This is what compression tries to provide by reducing the number of bytes sent over the network. Compression is hardly free since it requires a significant amount of CPU resources at the source and target nodes. Indeed, if there is insufficient CPU resources on either end, compression will provide much worse performance. Performance will further degrade if the data does not compress well.

You should considered compression if the data to be sent has a compression ratio of 1.5 or better and the achievable network bandwidth utilization is less than 66% of the stated bandwidth. Be aware of the usable CPU power of the source and target nodes will ultimately determine if compression is a viable option.

1.2.7 Un-buffered I/O (-u)

The **–u** option provides a mechanism to bypass the normal file system memory cache. It can be selectively employed at the source and target nodes. Bypassing the file system cache limits the transfer to the raw speed of the underlying device which, in practice, is usually slower than the network. Consequently, this option should be used as the last resort in improving the transfer rate. There are, however, some cases where un-buffered I/O makes sense either on the source or target. For instance, when transferring a large file (i.e., several gigabytes) using the file system cache may overwhelm the operating system and lead to a general slow-down that is far worse than transferring the data directly from the slow device. Un-buffered I/O also makes sense if the source device is much faster (e.g., 1.5 times) than the network link. You should also consider using the **–B** option to specify an appropriately large I/O transfer size for the device.

Un-buffered I/O places far more constraints on the copy than using a standard file system interface, since **bbcp** must be cognizant of device sectors. When **–u** is specified, all I/O buffers are automatically constrained to be multiples of 8K, the common restriction in many operating systems and device drivers. When **–u t** is specified, ordered output is enforced since many device drivers require it for new files. Even with these constraints, un-buffered I/O may still fail because either the source file is sparse (i.e., has holes) or the operating system does not adequately support "direct I/O".

Consequently, this option is very sensitive to the particular system configuration being used and the nature of the source file. Un-buffered I/O may, in fact, greatly decrease the data transfer rate. It should be treated with care.

1.2.8 Routing (-z)

Ostensibly, the **–z** option is used to reverse connection initiation. It is meant to get around firewall issues (see "Dealing With Firewalls"). However, the option may also cause the path between the source and target to use a different routing. That alone may cause a large variation in transfer speed. If you see an unexpectedly low transfer rate, and firewalls are not an issue, try the same copy with –z to see if you are experiencing routing issues.

1.3 Resuming Failed Copies (–a and –k or –K)

You can resume failed copies in most cases by consistently using the **–a** option. When **–a** is specified, the following occurs:

1) If the target file does not already exist, a new copy is initiated by
    a. creating a checkpoint record to pair the source and target files together,
    b. transmitting all source bytes to the target location, and
    c. upon successful transmission of the source file, erasing the checkpoint file.
2) If the target file exists and is identical in size to the source file and a copy checkpoint record is not found for the file, the copy is assumed to have completed normally and the file is skipped.
3) If the target file is larger than the source file, is smaller in size and a checkpoint record cannot be found, or if the checkpoint record does not pair the source and target files together, **bbcp** terminates with an error unless **–f** has been specified. In this case, the file is removed, or truncated if **–K** has been specified, and the copy is continued as in step 1 above.
4) Otherwise, the copy is resumed by appending all un-transmitted source file bytes to the target file.

The **–k** option maximizes **bbcp**'s ability to resume failed copies. If **–k** is not specified and an error occurs, **bbcp** removes the partially transmitted file. The **–a** option is still useful without **–k**, however, **bbcp** will merely skip over fully copied files. Rarely will **bbcp** be able to resume copying where it left off. The **–k** option forces partially completed files to remain on disk so that a partial copy can be resumed after the fault condition that terminated the copy is corrected.

Proper resumption of partially transmitted files relies on a checkpoint record. By default, this record is written in the command owner's (i.e., the user running **bbcp**) home directory in the ".bbcp" subdirectory. This subdirectory is automatically created if it does not already exist. The file names in this subdirectory have the format

bbcp.*srchost.trgid***.***trgfn*

Where *srchost* is the DNS name of the host that holds the source file, *trgid* is the unique identification of the target location, and *trgfn* is the name of the target file.

The contents of the file uniquely identify the source file at *srchost*. Proper pairing requires that the conditions that created the checkpoint file are still true at the time the copy is resumed. This essentially means that the copy cannot be resumed if any changes have occurred to the source file or if the source or target files have changed location since the copy was terminated.

Users with home directories in **AFS** may wish to change the default location for checkpoint files, especially should they run in batch-mode without an **AFS** token. Refer to the section "Configuring New Defaults" on how to set a new default location.

1.4 Multi-Directory Copying (-d)

You may use **bbcp** to copy source files to multiple directories. The **–d** option enables source relative addressing that, in turn, allows multi-directory copying. The following steps are taken when you specify **–d** *path*:

1) Each relative source file specification (i.e., one that does not start with a slash) is prefixed by path. The source file must be found at the resulting location.
2) The file is transferred to the sink (i.e., target) host along with its associated relative path.
3) The sink host creates the source relative path, if it does not exist, prefixed with the path in the sink specification.
4) The file is then created with a file name identical to the source file name.

For example,

```
bbcp -d /usr/abh/data dir1/data1 dir2/data2 batch:/usr/temp
```

would copy

`/usr/abh/data/dir1/data1` to `/usr/temp/dir1/data1`
`/usr/abh/data/dir2/data2` to `/usr/temp/dir2/data2`

The directories `dir1` and `dir2` are automatically created starting at path `/usr/temp` on host batch should they not exist.

You may mix relative paths with absolute paths. Absolute source paths are not prefixed by the **–d** path and are copied to the directory identified by the sink specification.

1.5 Using Pipes (-N)

The **–N** option enables copying via a combination of named pipes or programs. The arguments to **–N** indicate whether the *source* (**i**) is a named pipe or program or the *target* (**o**) is a named pipe or program. When the *source* is a named pipe or program, only one *source* may be specified. Determination of whether the data source or target is a pipe or program is done at copy-time on each node. If a pipe is detected, it is treated as a regular file with no specified size.

If the *source* (or *target*) is not a named pipe, the file name specification is treated as an executable program. For a *source*, the program's standard out is used as the data source. For a *target*, the program's standard in is used as the data target. In short, *bbcp* starts the program(s) and uses its network infrastructure to convey the data from one node to another.

When the *target* is a named pipe or a program, output ordering (**–o**) is enforced. This comes with its own set of tuning options as described earlier in this document. Other considerations specific to named pipes or programs also apply.

1.5.1 Named Pipe Considerations

When a *source* or *target* is a named pipe, the following considerations should be kept in mind.

- As named pipes do not provide any size information, the check for sufficient space on the *target* node is not done.
- When a *source* named pipe indicates end-of-file (eof), it is not possible to determine whether eof was due to a failure in pipe writer. Hence, copies may appear to succeed even though the pipe writer failed.
- Named pipes are never automatically created; they must exist prior to starting the copy.
- To avoid ambiguity between named pipe names and programs, named pipe file names may not contain space characters.
- The **bbcp** process waits until a writer opens a *source* pipe, if used, and a reader opens a *target* pipe, if used.

1.5.2 Program Pipe Considerations

- As program pipes do not provide any size information, the check for sufficient space on the *target* node is not done.
- When a *source* program pipe indicates end-of-file (eof), the condition is deemed to be error-free if the program terminates with a 0 status code. Otherwise, the copy fails.
- When a copy ends using a target program pipe, the program must terminate with a 0 status code in order for the copy to be considered error-free. Otherwise, the copy fails.
- Because bbcp waits for a program pipe to finish executing with a status code of zero before declaring success; the copy does not end until the program pipe ends. This may cause time-out errors when **–t** is specified should the program continue execution without producing (*source*) or needing (*target*) additional data.
- A program pipe specification may contain arguments to the program (i.e. options, etc). Such a specification must be surrounded by quotes or spaces be escaped to be successfully transmitted to **bbcp**. The specification is tokenized and the program is started using **execvp()**. A maximum of 127 tokens are allowed and tokenization ignores quoted sub-arguments. If you need to pass more tokens or use quoted arguments, you must wrap the program with a shell script.
- The **execvp()** system call uses the setting of $PATH to find the program is an absolute path to the program is not specified. Each operating system has its own peculiarities on how $PATH is handled in such cases. Consult the **execvp** man page for more information.

1.5.3 Using Program Pipes to Recursively Copy Small Files

The **–r** option provides a way to recursively copy one or more directories. This tends to be very inefficient when the files are relatively small. You can use **tar** or **gtar** to dramatically speed up recursive copying because all of the source files are combined into a single stream and sent to the destination where the directory structure and files are recreated. For instance,

bbcp -N io 'gtar -c -O -C /tmp mydir ' 'host:gtar -x -C /tmp/foo'

recursively copies the directory **mydir** in **/tmp** to **host** at **/tmp/foo**. Unlike **–r**, symlinks are preserved and it is possible to copy extended attributes as well. Consult the **gtar** man page for available options.

1.5.4 Restricting Program Pipes

Program pipes in essence allow a user to execute arbitrary programs on the *source* or *target* hosts. This is immaterial if the user has full login access to the hosts but does present a security issue when the *source* or *target* host is restricted via the **ssh** key file to a certain set of commands, including **bbcp** (i.e. the user does not have general login access).

Prior to launching a program pipe on the *source* or *target*, **bbcp** checks for the existence of the environmental variable **BBCP_ALLOWPP**. If it has not been defined, the program pipe is launched. Otherwise, **bbcp** performs the following actions:

1. If the value of the variable is '**0**', all program pipes are disallowed and the copy fails.
2. Otherwise, the variable is assumed to consist of a space separated list of allowed programs names. Each item in the list is compared with the name of the program about to be launched. If a match is found, the program is launched.
3. Otherwise, the user specified program pipe is disallowed and the copy fails.

To successfully use this scheme, the following must be true.

1. General login access must be prohibited to the user's account. Typically you would restrict **ssh** usage of the account to the '**bbcp SRC**' or '**bbcp TRG**' commands and perhaps some other select commands.
2. The **BBCP_ALLOWPP** environmental variable should be set in the user's profile (i.e. **.cshrc** or other login script).
3. If a program list is specified, the name should include variants that the user might use (e.g. gtar, /bin/gtar, etc).

1.6 Real-Time Copying (-R)

The **–R** option enables real-time copying mode. In this mode you can start copying a source file while it is still being created by another program. This allows you to implement a streaming copy model as opposed to the more traditional store-and-forward copy model. Since **bbcp** needs to know when the source file is complete, you must use a special file-creation protocol using file locks. Below are the arguments that you can supply to the **–R** option to control how real-time copying is performed. Subsequent sections provide examples.

```
                          -R [rtargs]

   rtargs: {c=csec | b | h | i=sec | v | lfn}[,rtargs]
```

Where:

c=*csec* specifies the minimum interval between checks to see if the input file has grown to sufficient size to continue the copy. File size checks are normally minimized and few occur if the source file is created at least as fast as it can be sent to the target node. The default is 3 seconds.

b blocks copy until there is sufficient data in the source file to utilize all streams. By default, data is sent whenever there are sufficient bytes to send on a single stream. This option may be useful for WAN links or when the file grows relatively slowly in size.

h hides the file at the target node by making it inaccessible (i.e., making it write-only) until the copy completes. By default, the file is accessible on the target node while it is being created there and the s-mode bit is set while the copy is in progress to indicate the file is not yet complete.

i=*isec* specifies the maximum number of seconds the source file may remain idle (i.e., not grow in size) before the copy is aborted. The default does not impose any limit.

v when used in conjunction with *lfn*, verifies the successful completion of the source file by checking that *lfn* is non-zero size after the lock on *lfn* is obtained. The default assumes the source file has been successfully created the moment a shared lock on *lfn* is obtained.

lfn specifies the path to an external file to be used for locking. It must start with dot or slash. This file co-ordinates the real-time copy. The default uses the source file as the target for locking and co-ordination. See the section "Using an Alternate Lock File" for more information.

1.6.1 Real-Time Copy Protocol

The following steps need to be followed to successfully use real-time copy mode:

1. An application opens the future source file in write mode and obtains an exclusive lock on the file using the **fcntl()** system call.
2. The **bbcp** program is launched to copy the source file with the **–R** option. With the **–R** option, **bbcp** opens the source file in read mode and starts the copy while simultaneously trying to obtain a shared lock on the file.
3. When the application is done creating the source file it must close the file. This removes the exclusive lock on the file which allows **bbcp** to obtain the shared lock on the file. At this point, **bbcp** assumes that the source file is complete and the copy continues by sending all outstanding bytes, at the time the shared lock was obtained, to the target node.

Below is a sample snippet of C code that illustrates what the creating application must do.

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
                        ----------------
struct flock flk;
int rc, srcfd;
                        ----------------
flk.l_type  = F_WRLCK; flk.l_whence = SEEK_SET;
flk.l_start = 0;        flk.l_len = 0;
                        ----------------
srcfd = open(srcfn, O_WRONLY|O_CREAT|O_TRUNC,0600);
if (srcfd < 0) {/* Handle open error */}
do {rc = fcntl(srcfd, F_SETLKW, &flk);}
    while(rc < 0 && errno == EINTR);
if (rc < 0) {/* Handle fcntl error */}
                        ----------------
    /* bbcp can be started and the source file written */
```

Sample Application Code Using Source File Locking

1.6.2 Using an Alternate Lock File

The ability to specify a lock file that differs from the source file allows you to enable real-time copy mode using unmodified application as well as positively confirm that the source file was successfully created. An alternate lock name is specified as an argument to the **–R** option. Other than using another file for locking the real-time copy protocol steps are identical to those described in the previous section.

However, with an alternate lock file you can also specify the **v** argument with **–R** to indicate that **bbcp** must get a positive indication that the file was successfully created in order for the copy to complete. The following steps need to be followed to successfully use real-time copy mode with an alternate lock file and the **v** argument:

1. A wrapper script opens the alternate lock file in write mode, truncating to zero length, and obtains an exclusive lock on the file using the **fcntl()** system call.
2. The script launches the application that is to create the source file. The script must wait until the source file appears in the file system or the creating application fails, whichever comes first.
3. When the source file appears in the file system, the script launches **bbcp** to copy the source file with the **–R** option, specifying the alternate lock file name and the v argument. When launched, **bbcp** opens the source and lock files in read mode and starts the copy while simultaneously trying to obtain a shared lock on the alternate lock file.
4. The script then waits for the creating application to complete. Should the creating application end with an error, the script can exit as well. In this case, **bbcp** aborts the copy.
5. Otherwise, the script must write at least one byte into the lock file. It may then exit, implicitly closing the lock file, and let **bbcp** continue.
6. Closing the alternate lock file removes the script's exclusive lock on the file and allows **bbcp** to obtain the shared lock on the file.
7. Then, **bbcp** checks the size of the lock file. If it's zero, the copy is aborted. Otherwise, the copy continues by sending all outstanding bytes, at the time the shared lock was obtained, to the target node.

What follows is a sample snippet of **perl** code that illustrates what the wrapper script must do.

```
#!/bin/perl
Use Fcntl;

$LKfn = '/tmp/myLock';
$LKop = pack('sslllll', F_WRLCK, 0, 0, 0, 0, 0, 0);

                        ----------------

# Open lock file, truncating it to zero, and obtain lock
#
die "Unable to open $LKfn; $!\n"
    if !open(LKFD, "+>$LKfn");
die "Unable to lock $LKfn; $!\n"
    if !fcntl(LKFD, F_SETLKW, $LKop);

                        ----------------

# Launch application and wait for it to open source file
#
$appPID = &Launch_App_Wait_For_File();
exit(1) if !$appPID;# Exit if application failed to launch!

# Now launch bbcp
#
system('bbcp', '-R', "v,$LKfn", srcfn, trgfn);
exit(1) if !$?;# Exit if bbcp failed to launch!

                        ----------------

# Wait for application to end
#
exit(8) if waitpid($appPID,0) <= 0;

# Check for success (allow copy) or failure (abort copy)
#
print LKFD "OK" if $? == 0; # Copy will complete!
```

Sample Script Using Alternate Lock File With* v *Argument

1.7 Modifying ssh Startup (-S and –T)

By default, **bbcp** uses **ssh**, as determined by your PATH environmental variable, for authentication on every host that was specified in the source and target specifications. In order to speed **ssh** start-up, **bbcp** disables **ssh X11** forwarding (**-x**), disables the forwarding of the authentication agent connection (**-a**), and disables the use of **rsh** when **ssh** authentication fails (**-o**).

bbcp executes a copy of itself on the source node as "**bbcp SRC**" and a mirror copy on the target node as "**bbcp SNK**". Because the commands are well known, you may restrict **ssh** usage to exactly these commands when a password-less key-file is used to gain access to a host. The **–I** option provides a mechanism to specify the location of the identity file should it not reside at the default location.

At times, you may need to specify different commands to start **bbcp** on the source node, as well as the sink node. The **–S** and **–T** options allow you to do this. You may also specify the default **–S** and **–T** options using a configuration file. See "Configuring New Defaults" for more information.

Because certain information needs to be substituted in the command line, **bbcp** defines certain character sequences to indicate the location of a substitution. These are:

`%4` - substituted by **-4** if **–ipv4 c** is in effect,
`%I` - substituted by the **–i** *fname* (i.e., **ssh** identify file option) should one exist,
`%H` - the source or target host name, and
`%U` - the source or target user name.

For instance, the command

```
bbcp -S '/bin/ssh %I -l %U %H /bin/bbcp' /tmp/fn abh@host:/tmp
```

would start **bbcp** on the source node using the command

```
/bin/ssh -l abh host /usr/bin/bbcp SRC
```

Since the **ssh** identity file was not specified, the **%I** was deleted. If the identity file were specified as

```
bbcp -S '/bin/ssh %I -l %U %H /bin/bbcp' -i foo /tmp/fn abh@host:/tmp
```

then the command used to start **bbcp** on the source node would be

```
/bin/ssh –i foo –l abh host /usr/bin/bbcp SRC
```

Identical rules apply to the **–T** option which specifies the command to start **bbcp** on the sink (i.e., target) node.

You should also add the "**-a -x -oFallBackTo Rsh=no**" options to your **ssh** command to reduce start-up time; as well as **%4** if you wish to control TCP/IP protocol usage using the **–ipv4** option.

1.8 Dealing With Firewalls (-z and -Z)

bbcp is a peer-to-peer application. Mainly, this means that copies of **bbcp** on the source and sink nodes appear to be both client as well as server applications . This may not be possible at some sites because of firewall restrictions. Specifically, some installation may prohibit incoming TCP/IP connections at arbitrary ports.

Normally, **bbcp** source nodes will connect to their counterpart running on the target node. If the target host prohibits incoming connections, the copy will fail. However, should the source host allow arbitrary connections, you can specify the **–z** option. This option reverses the connection protocol so that the **bbcp** sink node will always try to connect to its counterpart running on the source host.

When the source and target nodes prohibit arbitrary connections, you will need assistance of an administrator at either node. If the **--port** option **(-Z**) is not specified, **bbcp** checks the /etc/services file for the existence of two services: **bbcpfirst** and **bbcplast**. The **bbcpfirst** service identifies the starting port number and **bbcplast** identifies the ending port number that can be used for incoming connections. When **--port** option **(-Z**) is not specified and neither service name can be found, **bbcp** resorts to using an arbitrary port number. If the services are found, **bbcp** restricts its port usage to one of the ports in the indicated range.

Ask the administrator at the source or target nodes to allow a range of well-known port numbers to be used for incoming connections (i.e., allowed to pass through the firewall). This will require that the administrator register these port numbers in the /etc/services file using the names **bbcpfirst** and **bbcplast** (the default names can be changed). Make sure that at least 8 port numbers exist in the range (more if possible). If restricted port access is only allowed in the source site, you must specify the **–z** option when invoking **bbcp**.

1.9 Configuring New Defaults (-C)

When starting, **bbcp** checks the environmental variable **bbcp_CONFIGFN**. When this variable is set, the contents are used as the location of the configuration file. Otherwise, **bbcp** looks to see if the file .bbcp.cf exists in the home directory. If it does, then this file is used as the initial configuration file. A configuration file may also be specified on the command line using the **–C** option. Command line configuration files are processed when they are encountered. Thus, any option specified prior to **–C** may be overridden by the configuration file and the file's values may be overridden by subsequent options. The **–C** option, when specified, should be the first option on the command line.

Each line in the configuration file may contain an option-value pair. The option name is identical to that specified on the command line (e.g., -a,-b, -c, etc.). The value is the value, if any, that would be specified along with the corresponding option. The only difference between options specified on the command line and those specified in the configuration file is that each option must be on a separate line and option values must *not* be quoted.

It is critical to remember that **bbcp** is a peer-to-peer application. Therefore, it can have up to three different execution locations *at the same time*: the host that initiated the **bbcp** command (i.e., agent), the host that holds the source data (i.e., source), and the host that is to receive the source data (i.e., target). In order to simplify the management of this environment, the configuration file is only read on the agent's host (i.e., the host that initiated the copy) and the values are transmitted to the source and target hosts.

1.10 Recursive Copy Compatibility With cp & scp

By default, **bbcp** does not produce the exactly same results as **cp** or **scp** for recursive copies unless you specify the **--gross**, **--mkdir** and **--symlinks follow** options. Even then, differences do occur under certain conditions. For instance, if the source is a regular file and **--mkdir** *and* **--recursive** are in effect, a directory is created and the file is copied into that directory should the directory does not exist. Otherwise, handling is the same.

1.11 New Features

Version **14.09.02.00.0** of documented here has the following new feature:

- The new **–pt**ime option forces **bbcp** to preserve only source file's access and modification time at the destination; leaving the group and mode unchanged.

Version **14.07.01.00.0** of documented here has the following new features:

- The new **–gross** option forces **bbcp** to copy empty directory structures in recursive mode.
- The new **–mkdir** option creates the destination directory for a recursive copy should it not exist.
- The new **–symlinks** option specifies how symbolic links are to be handled during a recursive copy.

Version **14.04.14.00.0** of documented here has the following new features:

- Dual stack **IPv6** mode is fully functional and is the default mode of operation. The new **–ipv4** option forces **bbcp** to use the **IPv4 TCP** stack.
- The new **–license** option prints the license under which **bbcp** is distributed.
- The new **–version** option displays **bbcp**'s version number installed on the current host.
- The **BBCP_ALLOWPP** environmental variable may be used to restrict program pipes to a specified set of programs or disable the feature altogether.

Version **12.08.17.00.0** of **bbcp** documented here has the following new feature:

- The new **–Z** option allows you to specify the port range to be used for incoming connections on the command line instead of relying on /etc/services to contain those values.

Version **12.01.30.00.0** of **bbcp** documented here has the following new feature:

- The new **–N** option allows you to specify that the source or destination (or both) is a named or program pipe.
- The new **–y** option allows you to make sure that data and, optionally, inodes are fully written to disk before the output file is closed.
- This version of **bbcp** follows the documented behavior of **–r** where symlinks are ignored.
- Most options now accept a long form that is more descriptive.

Version **10.08.13.01.0** of **bbcp** documented here has the following new feature:

- The new **–O** option allows one to omit files that already exist on the target node. This is particularly useful with **–r**.
- The new **–R** option enables real-time copy mode.

Version **10.08.04.00.0** of **bbcp** documented here has the following new features:

- The **–a** option is no longer mutually exclusive with **–f** or **–K**. Instead, when **–a** is specified together with **–f** or **–K** , the latter option is applies if the file cannot be appended to. This allows a file copy to restart from scratch if an append copy cannot be executed.
- The **–m** option has been extended to allow the specification of a directory mode to be used when directories are created.

Version **10.07.26.00.0** of **bbcp** introduced the following new features:

- Window auto-tuning is now supported as the default. The **–w** option supports a mechanism to defeat auto-tuning when so desired.
- **–B** option now defines the overall I/O size and is not tied to the **–c** option.
- **–b +** option allows adding more output buffers to prevent stalls when **–o** has been specified or forced.
- **–E** option that allows extensive checksum checking and reporting.
- **–K** option that removes some restrictions of the **–k** option in order to handle pre-created symbolic links.
- **–u** option to specify un-buffered (i.e., direct) I/O at the source or target.
- The default window size has been increased to 128K for improved performance on most current operating systems.
- Recursive name space indexing now happens in the background. This allows large name spaces to be traversed without timing out the copy.

1.12 Backward Compatibility

Version **14.09.02.00.0** of **bbcp** documented here is backward compatible with the following caveat:

- Older version will fail if any of the new option: **–pt**ime is specified.

Version **14.07.01.00.0** of **bbcp** documented here is backward compatible with the following caveat:

- Older version will fail if any of the new options: **–gross**, **--mkdir**, or **--symlinks** is specified.

Version **14.04.11.00.0** of **bbcp** documented here is backward compatible with the following caveat:

- Older version will fail if the new **–ipv4** option is specified. Older versions always use the **IPv4 TCP** stack, making the option generally unnecessary.

Version **12.08.17.00.0** of **bbcp** documented here is backward compatible with the following caveat:

- Older versions will fail if the new **–Z** option is specified.

Version **12.01.30.00.0** of **bbcp** documented here is backward compatible with the following caveat:

- Older versions will fail if the new **–N** or **–y** options are specified.
- This version is incompatible because in that it follows documented behavior and ignores symlinks when doing a recursive copy. Previous versions copy the symlink target as a data file.

Version **10.08.13.01.0** of **bbcp** documented here is backward compatible with the following caveat:

- Older versions will fail if the new **–O** or **–R** options are specified.

Version **10.08.04.00.0** of **bbcp** documented here is backward compatible with the following caveats:

- Older version will fail if **–a** is combined with, **–f** or **–K**.
- Older version will fail if a directory mode is specified with the **–m** option.

Version **10.07.26.00.0** of **bbcp** is backward compatible with the following caveats:

- Older version will fail if **–E**, **–K**, **–u** , or '**–w =**' is specified. These are new options.
- The **–b +** option is converted to **–b** by older versions. While the conversion is compatible, performance characteristics are not comparable.
- The default window size has been increased to 128K. This generally increases performance on today's systems. However, this is incompatible with defective routers and packet firewalls that rewrite the window scaling factor during transmission. When this happens, poor and erratic bandwidth utilization will result. Should you see this, revert to using the old default of 64K.
- The **–W** option is deprecated but still accepted. However, it has now the same meaning as **–w**.
- The format and information provided by the **–v** option (verbose) has changed.

1.13 Problem Reports & Enhancement Requests

Please direct all problem reports, modifications, and requests for enhancements to:

Andrew Hanushevsky abh@stanford.edu

1.14 Downloading

First, please read the legal notice (see below). *Use of this software implies that you have read and agreed to all of the terms and conditions for use.*

You can download one or more of the following **bbcp** binary executables at

http://www.slac.stanford.edu/~abh/bbcp/bin/

Once in the bin directory, you will find platform-specific directories that contain the bbcp binary. The program may or may not work in other versions of the same operating system. Should you run into trouble or wish to extend the range of operating systems available, feel free to clone the git archive:

git clone http://www.slac.stanford.edu/~abh/bbcp/bbcp.git

and send back any required modification as git patches.

1.15 Legal Notice

bbcp is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

bbcp is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with **bbcp** in a file called COPYING.LESSER (LGPL license) and file COPYING (GPL license). If not, see <http://www.gnu.org/licenses>.

Copies of **bbcp** that are not **IPv6** enabled (i.e. versions prior to **14.04.14.00.0**) were distributed under a modified **BSD** license. All **IPv6** enabled version of **bbcp** (i.e. version **14.04.11.00.0** and any subsequent version) are distributed under an **LGL** license.