## NAME

iperf3 - perform network throughput tests

## SYNOPSIS

```
iperf3 -s [ options ]
iperf3 -c server [ options ]
```

## DESCRIPTION

iperf3 is a tool for performing network throughput measurements. It can test TCP, UDP, or SCTP throughput. To perform an iperf3 test the user must establish both a server and a client.

The iperf3 executable contains both client and server functionality. An iperf3 server can be started using either of the -s or --server command-line parameters, for example:

```
iperf3 -s

iperf3 --server
```

Note that many iperf3 parameters have both short (-s) and long (--server) forms. In this section we will generally use the short form of command-line flags, unless only the long form of a flag is available.

By default, the iperf3 server listens on TCP port 5201 for connections from an iperf3 client. A custom port can be specified by using the -p flag, for example:

```
iperf3 -s -p 5002
```

After the server is started, it will listen for connections from iperf3 clients (in other words, the iperf3 program run in client mode). The client mode can be started using the -c command-line option, which also requires a host to which iperf3 should connect. The host can by specified by hostname, IPv4 literal, or IPv6 literal:

```
iperf3 -c iperf3.example.com

iperf3 -c 192.0.2.1

iperf3 -c 2001:db8::1
```

If the iperf3 server is running on a non-default TCP port, that port number needs to be specified on the client as well:

```
iperf3 -c iperf3.example.com -p 5002
```

The initial TCP connection is used to exchange test parameters, control the start and end of the test, and to exchange test results. This is sometimes referred to as the "control connection". The actual test data is sent over a separate TCP connection, as a separate flow of UDP packets, or as an independent SCTP connection, depending on what protocol was specified by the client.

Normally, the test data is sent from the client to the server, and measures the upload speed of the client. Measuring the download speed from the server can be done by specifying the -R flag on the client. This causes data to be sent from the server to the client.

```
iperf3 -c iperf3.example.com -p 5202 -R
```

Results are displayed on both the client and server. There will be at least one line of output per measurement interval (by default a measurement interval lasts for one second, but this can be changed by the -i option). Each line of output includes (at least) the time since the start of the test, amount of data transferred during the interval, and the average bitrate over that interval. Note that the values for each measurement interval are taken from the point of view of the endpoint process emitting that output (in other words, the output on the client shows the measurement interval data for the client.

At the end of the test is a set of statistics that shows (at least as much as possible) a summary of the test as seen by both the sender and the receiver, with lines tagged accordingly. Recall that by default the client is the sender and the server is the receiver, although as indicated above, use of the -R flag will reverse these roles.

The client can be made to retrieve the server-side output for a given test by specifying the --get-server-output flag.

Either the client or the server can produce its output in a JSON structure, useful for integration with other programs, by passing it the -J flag. Because the contents of the JSON structure are only completely known after the test has finished, no JSON output will be emitted until the end of the test.

iperf3 has a (overly) large set of command-line options that can be used to set the parameters of a test. They are given in the "GENERAL OPTIONS" section of the manual page below, as well as summarized in iperf3's help output, which can be viewed by running iperf3 with the -h flag.

## GENERAL OPTIONS

**-p, --port n**
set server port to listen on/connect to to n (default 5201)

**-f, --format**
[kmgtKMGT] format to report: Kbits/Mbits/Gbits/Tbits

**-i, --interval n**
pause n seconds between periodic throughput reports; default is 1, use 0 to disable

**-I, --pidfile file**
write a file with the process ID, most useful when running as a daemon.

**-F, --file name**
Use a file as the source (on the sender) or sink (on the receiver) of data, rather than just generating random data or throwing it away. This feature is used for finding whether or not the storage subsystem is the bottleneck for file transfers. It does not turn iperf3 into a file transfer tool. The length, attributes, and in some cases contents of the received file may not match those of the original file.

**-A, --affinity n/n,m**
Set the CPU affinity, if possible (Linux, FreeBSD, and Windows only). On both the client and server you can set the local affinity by using the n form of this argument (where n is a CPU number). In addition, on the client side you can override the server's affinity for just that one test, using the n,m form of argument. Note that when using this feature, a process will only be bound to a single CPU (as opposed to a set containing potentially multiple CPUs).

**-B, --bind host[%dev]**
bind to the specific interface associated with address host. If an optional interface is specified, it is treated as a shortcut for --bind-dev dev. Note that a percent sign and interface device name are required for IPv6 link-local address literals.

**--bind-dev dev**
bind to the specified network interface. This option uses SO_BINDTODEVICE, and may require root permissions. (Available on Linux and possibly other systems.)

**-V, --verbose**
give more detailed output

**-J, --json**
output in JSON format

**--logfile file**
send output to a log file.

**--forceflush**
force flushing output at every interval. Used to avoid buffering when sending output to pipe.

**--timestamps[=format]**
prepend a timestamp at the start of each output line. By default, timestamps have the format emitted by ctime(1). Optionally, = followed by a format specification can be passed to customize the timestamps, see strftime(3). If this optional format is given, the = must immediately follow the --timestamps option with no whitespace intervening.

**--rcv-timeout #**
set idle timeout for receiving data during active tests. The receiver will halt a test if no data is received from the sender for this number of ms (default to 12000 ms, or 2 minutes).

**--snd-timeout #**
set timeout for unacknowledged TCP data (on both test and control connections) This option can be used to force a faster test timeout in case of a network partition during a test. The required parameter is specified in ms, and defaults to the system settings. This functionality depends on the TCP_USER_TIMEOUT socket option, and will not work on systems that do not support it.

**-d, --debug**
emit debugging output. Primarily (perhaps exclusively) of use to developers.

**-v, --version**
show version information and quit

**-h, --help**
show a help synopsis

## SERVER SPECIFIC OPTIONS

**-s, --server**
run in server mode

**-D, --daemon**
run the server in background as a daemon

**-1, --one-off**
handle one client connection, then exit. If an idle time is set, the server will exit after that amount of time with no connection.

**--idle-timeout n**
restart the server after n seconds in case it gets stuck. In one-off mode, this is the number of seconds the server will wait before exiting.

**--server-bitrate-limit n[KMGT]**
set a limit on the server side, which will cause a test to abort if the client specifies a test of more than n bits per second, or if the average data sent or received by the client (including all data streams) is greater than n bits per second. The default limit is zero, which implies no limit. The interval over which to average the data rate is 5 seconds by default, but can be specified by adding a '/' and a number to the bitrate specifier.

**--rsa-private-key-path file**
path to the RSA private key (not password-protected) used to decrypt authentication credentials from the client (if built with OpenSSL support).

**--authorized-users-path file**
path to the configuration file containing authorized users credentials to run iperf tests (if built with OpenSSL support). The file is a comma separated list of usernames and password hashes; more information on the structure of the file can be found in the EXAMPLES section.

**--time-skew-thresholdsecond seconds**
time skew threshold (in seconds) between the server and client during the authentication process.

## CLIENT SPECIFIC OPTIONS

**-c, --client host[%dev]**
run in client mode, connecting to the specified server. By default, a test consists of sending data from the client to the server, unless the -R flag is specified. If an optional interface is specified, it is treated as a shortcut for --bind-dev dev. Note that a percent sign and interface device name are required for IPv6 link-local address literals.

**--sctp** use SCTP rather than TCP (FreeBSD and Linux)

**-u, --udp**
use UDP rather than TCP

**--connect-timeout n**
set timeout for establishing the initial control connection to the server, in milliseconds. The default behavior is the operating system's timeout for TCP connection establishment. Providing a shorter value may speed up detection of a down iperf3 server.

**-b, --bitrate n[KMGT]**
set target bitrate to n bits/sec (default 1 Mbit/sec for UDP, unlimited for TCP/SCTP). If there are multiple streams (-P flag), the throughput limit is applied separately to each stream. You can also add a '/' and a number to the bitrate specifier. This is called "burst mode". It will send the given number of packets without pausing, even if that temporarily exceeds the specified throughput limit. Setting the target bitrate to 0 will disable bitrate limits (particularly useful for UDP tests). This throughput limit is implemented internally inside iperf3, and is available on all platforms. Compare with the --fq-rate flag. This option replaces the --bandwidth flag, which is now deprecated but (at least for now) still accepted.

**--pacing-timer n[KMGT]**
set pacing timer interval in microseconds (default 1000 microseconds, or 1 ms). This controls iperf3's internal pacing timer for the -b/--bitrate option. The timer fires at the interval set by this parameter. Smaller values of the pacing timer parameter smooth out the traffic emitted by iperf3, but potentially at the cost of performance due to more frequent timer processing.

**--fq-rate n[KMGT]**
Set a rate to be used with fair-queueing based socket-level pacing, in bits per second. This pacing (if specified) will be in addition to any pacing due to iperf3's internal throughput pacing (-b/--bitrate flag), and both can be specified for the same test. Only available on platforms supporting the SO_MAX_PACING_RATE socket option (currently only Linux). The default is no fair-queueing based pacing.

**--no-fq-socket-pacing**
This option is deprecated and will be removed. It is equivalent to specifying --fq-rate=0.

**-t, --time n**
time in seconds to transmit for (default 10 secs)

**-n, --bytes n[KMGT]**
number of bytes to transmit (instead of -t)

**-k, --blockcount n[KMGT]**
number of blocks (packets) to transmit (instead of -t or -n)

**-l, --length n[KMGT]**
length of buffer to read or write. For TCP tests, the default value is 128KB. In the case of UDP, iperf3 tries to dynamically determine a reasonable sending size based on the path MTU; if that cannot be determined it uses 1460 bytes as a sending size. For SCTP tests, the default size is 64KB.

**--cport port**
bind data streams to a specific client port (for TCP and UDP only, default is to use an ephemeral port)

**-P, --parallel n**
number of parallel client streams to run. iperf3 will spawn off a separate thread for each test stream. Using multiple streams may result in higher throughput than a single stream.

**-R, --reverse**
reverse the direction of a test, so that the server sends data to the client

**--bidir**
test in both directions (normal and reverse), with both the client and server sending and receiving data simultaneously

**-w, --window n[KMGT]**
set socket buffer size / window size. This value gets sent to the server and used on that side too; on both sides this option sets both the sending and receiving socket buffer sizes. This option can be used to set (indirectly) the maximum TCP window size. Note that on Linux systems, the effective maximum window size is approximately double what is specified by this option (this behavior is not a bug in iperf3 but a "feature" of the Linux kernel, as documented by tcp(7) and socket(7)).

**-M, --set-mss n**
set TCP/SCTP maximum segment size (MTU - 40 bytes)

**-N, --no-delay**
set TCP/SCTP no delay, disabling Nagle's Algorithm

**-4, --version4**
only use IPv4

**-6, --version6**
only use IPv6

**-S, --tos n**
set the IP type of service. The usual prefixes for octal and hex can be used, i.e. 52, 064 and 0x34 all specify the same value.

**--dscp dscp**
set the IP DSCP bits. Both numeric and symbolic values are accepted. Numeric values can be specified in decimal, octal and hex (see --tos above).

**-L, --flowlabel n**
set the IPv6 flow label (currently only supported on Linux)

**-X, --xbind name**
Bind SCTP associations to a specific subset of links using sctp_bindx(3). The --B flag will be ignored if this flag is specified. Normally SCTP will include the protocol addresses of all active links on the local host when setting up an association. Specifying at least one --X name will disable this behaviour. This flag must be specified for each link to be included in the association, and is supported for both iperf servers and clients (the latter are supported by passing the first --X argument to bind(2)). Hostnames are accepted as arguments and are resolved using getaddrinfo(3). If the --4 or --6 flags are specified, names which do not resolve to addresses within the specified protocol family will be ignored.

**--nstreams n**
Set number of SCTP streams.

**-Z, --zerocopy**
Use a "zero copy" method of sending data, such as sendfile(2), instead of the usual write(2).

**-O, --omit n**
Perform pre-test for N seconds and omit the pre-test statistics, to skip past the TCP slow-start period.

**-T, --title str**
Prefix every output line with this string.

**--extra-data str**
Specify an extra data string field to be included in JSON output.

**-C, --congestion algo**
Set the congestion control algorithm (Linux and FreeBSD only). An older --linux-congestion synonym for this flag is accepted but is deprecated.

**--get-server-output**
Get the output from the server. The output format is determined by the server (in particular, if the server was invoked with the --json flag, the output will be in JSON format, otherwise it will be in human-readable format). If the client is run with --json, the server output is included in a JSON object; otherwise it is appended at the bottom of the human-readable output.

**--udp-counters-64bit**
Use 64-bit counters in UDP test packets. The use of this option can help prevent counter overflows during long or high-bitrate UDP tests. Both client and server need to be running at least version 3.1 for this option to work. It may become the default behavior at some point in the future.

**--repeating-payload**
Use repeating pattern in payload, instead of random bytes. The same payload is used in iperf2 (ASCII '0..9' repeating). It might help to test and reveal problems in networking gear with hardware compression (including some WiFi access points), where iperf2 and iperf3 perform differently, just based on payload entropy.

**--dont-fragment**
Set the IPv4 Don't Fragment (DF) bit on outgoing packets. Only applicable to tests doing UDP over IPv4.

**--username username**
username to use for authentication to the iperf server (if built with OpenSSL support). The password will be prompted for interactively when the test is run. Note, the password to use can also be specified via the IPERF3_PASSWORD environment variable. If this variable is present, the password prompt will be skipped.

**--rsa-public-key-path file**
path to the RSA public key used to encrypt authentication credentials (if built with OpenSSL support)

## EXAMPLES

### Authentication - RSA Keypair

The authentication feature of iperf3 requires an RSA public keypair. The public key is used to encrypt the authentication token containing the user credentials, while the private key is used to decrypt the authentication token. The private key must be in PEM format and additionally must not have a password set. The public key must be in PEM format and use SubjectPrefixKeyInfo encoding. An example of a set of UNIX/Linux commands using OpenSSL to generate a correctly-formed keypair follows:

```
> openssl genrsa -des3 -out private.pem 2048
> openssl rsa -in private.pem -outform PEM -pubout -out public.pem
> openssl rsa -in private.pem -out private_not_protected.pem
-outform PEM
```

After these commands, the public key will be contained in the file public.pem and the private key will be contained in the file private_not_protected.pem.

### Authentication - Authorized users configuration file

A simple plaintext file must be provided to the iperf3 server in order to specify the authorized user credentials. The file is a simple list of comma-separated pairs of a username and a corresponding password hash. The password hash is a SHA256 hash of the string "{$user}$password". The file can also contain commented lines (starting with the # character). An example of commands to generate the password hash on a UNIX/Linux system is given below:

```
> S_USER=mario S_PASSWD=rossi
> echo -n "{$S_USER}$S_PASSWD" | sha256sum | awk '{ print $1 }'
```

An example of a password file (with an entry corresponding to the above username and password) is given below:

```
> cat credentials.csv
# file format: username,sha256
mario,bf7a49a846d44b454a5d11e7acfaf13d138bbe0b7483aa3e050879700572709b
```

## AUTHORS

A list of the contributors to iperf3 can be found within the documentation located at https://software.es.net/iperf/dev.html#authors.

## SEE ALSO

libiperf(3), https://software.es.net/iperf