NAME

tcp - Internet Transmission Control Protocol

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>

int
socket(*AF_INET*, *SOCK_STREAM*, *0*);

DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of "port addresses". Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets utilizing the TCP protocol are either "active" or "passive". Active sockets initiate connections to passive sockets. By default, TCP sockets are created active; to create a passive socket, the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, termed "wildcard addressing", allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified, the system will assign one. Once a connection has been established, the socket’s address is fixed by the peer entity’s location. The address assigned to the socket is the address associated with the network interface through which packets are being transmitted and received. Normally, this address corresponds to the peer entity’s network.

TCP supports a number of socket options which can be set with setsockopt(2) and tested with getsockopt(2):

TCP_INFO — Information about a socket’s underlying TCP session may be retrieved by passing the read-only option TCP_INFO to getsockopt(2). It accepts a single argument: a pointer to an instance of *struct tcp_info*.

This API is subject to change; consult the source to determine which fields are currently filled out by this option. FreeBSD specific additions include send window size, receive window size, and bandwidth-controlled window space.

TCP_CCALGOOPT Set or query congestion control algorithm specific parameters. See mod_cc(4) for details.

TCP_CONGESTION Select or query the congestion control algorithm that TCP will use for the connection. See mod_cc(4) for details.

TCP_FASTOPEN Enable or disable TCP Fast Open (TFO). To use this option, the kernel must be built with the TCP_RFC7413 option.

This option can be set on the socket either before or after the listen(2) is invoked. Clearing this option on a listen socket after it has been set has no effect on existing TFO connections or TFO connections in progress; it only prevents new TFO connections from being established.

For passively-created sockets, the TCP_FASTOPEN socket option can be queried to determine whether the connection was established using TFO. Note that connections that are established via a TFO SYN, but that fall back to using a non-TFO SYN|ACK will have the TCP_FASTOPEN socket option set.

In addition to the facilities defined in RFC7413, this implementation supports a pre-shared key (PSK) mode of operation in which the TFO server requires the client to be in posession of a shared secret in order for the client to be able to successfully open TFO connections with the server. This is useful, for example, in environments where TFO servers are exposed to both internal and external clients and only wish to allow TFO connections from internal clients.

In the PSK mode of operation, the server generates and sends TFO cookies to requesting clients as usual. However, when validating cookies received in TFO SYNs from clients, the server requires the client-supplied cookie to equal

SipHash24(key=*16-byte-psk*, msg=*cookie-sent-to-client*)

Multiple concurrent valid pre-shared keys are supported so that time-based rolling PSK invalidation policies can be implemented in the system. The default number of concurrent pre-shared keys is 2.

This can be adjusted with the TCP_RFC7413_MAX_PSKS kernel option.

TCP_FUNCTION_BLK Select or query the set of functions that TCP will use for this connection. This allows a user to select an alternate TCP stack. The alternate TCP stack must already be loaded in the kernel. To list the available TCP stacks, see *functions_available* in the *MIB (sysctl) Variables* section further down. To list the default TCP stack, see *functions_default* in the *MIB (sysctl) Variables* section.

TCP_KEEPINIT This setsockopt(2) option accepts a per-socket timeout argument of *u_int* in seconds, for new, non-established TCP connections. For the global default in milliseconds see *keepinit* in the *MIB (sysctl) Variables* section further down.

TCP_KEEPIDLE This setsockopt(2) option accepts an argument of *u_int* for the amount of time, in seconds, that the connection must be idle before keepalive probes (if enabled) are sent for the connection of this socket. If set on a listening socket, the value is inherited by the newly created socket upon accept(2). For the global default in milliseconds see *keepidle* in the *MIB (sysctl) Variables* section further down.

TCP_KEEPINTVL This setsockopt(2) option accepts an argument of *u_int* to set the per-socket interval, in seconds, between keepalive probes sent to a peer. If set on a listening socket, the value is inherited by the newly created socket upon accept(2). For the global default in milliseconds see *keepintvl* in the *MIB (sysctl) Variables* section further down.

TCP_KEEPCNT This setsockopt(2) option accepts an argument of *u_int* and allows a per-socket tuning of the number of probes sent, with no response, before the connection will be dropped. If set on a listening socket, the value is inherited by the newly created socket upon accept(2). For the global default see the *keepcnt* in the *MIB (sysctl) Variables* section further down.

TCP_NODELAY Under most circumstances, TCP sends data when it is presented; when outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. The boolean option TCP_NODELAY defeats this algorithm.

TCP_MAXSEG
: By default, a sender- and receiver-TCP will negotiate among themselves to determine the maximum segment size to be used for each connection. The TCP_MAXSEG option allows the user to determine the result of this negotiation, and to reduce it if desired.

TCP_MAXUNACKTIME
: This setsockopt(2) option accepts an argument of *u_int* to set the per-socket interval, in seconds, in which the connection must make progress. Progress is defined by at least 1 byte being acknowledged within the set time period. If a connection fails to make progress, then the TCP stack will terminate the connection with a reset. Note that the default value for this is zero which indicates no progress checks should be made.

TCP_NOOPT
: TCP usually sends a number of options in each packet, corresponding to various TCP extensions which are provided in this implementation. The boolean option TCP_NOOPT is provided to disable TCP option use on a per-connection basis.

TCP_NOPUSH
: By convention, the sender-TCP will set the "push" bit, and begin transmission immediately (if permitted) at the end of every user call to write(2) or writev(2). When this option is set to a non-zero value, TCP will delay sending any data at all until either the socket is closed, or the internal send buffer is filled.

TCP_MD5SIG
: This option enables the use of MD5 digests (also known as TCP-MD5) on writes to the specified socket. Outgoing traffic is digested; digests on incoming traffic are verified. When this option is enabled on a socket, all inbound and outgoing TCP segments must be signed with MD5 digests.

  One common use for this in a FreeBSD router deployment is to enable based routers to interwork with Cisco equipment at peering points. Support for this feature conforms to RFC 2385.

  In order for this option to function correctly, it is necessary for the administrator to add a tcp-md5 key entry to the system’s security associations database (SADB) using the setkey(8) utility. This entry can only be specified on a per-host basis at this time.

  If an SADB entry cannot be found for the destination, the system does not send any outgoing segments and drops any inbound segments. However,

during connection negotiation, a non-signed segment will be accepted if an SADB entry does not exist between hosts. When a non-signed segment is accepted, the established connection is not protected with MD5 digests.

TCP_STATS Manage collection of connection level statistics using the stats(3) framework.

Each dropped segment is taken into account in the TCP protocol statistics.

TCP_TXTLS_ENABLE Enable in-kernel Transport Layer Security (TLS) for data written to this socket. See ktls(4) for more details.

TCP_TXTLS_MODE The integer argument can be used to get or set the current TLS transmit mode of a socket. See ktls(4) for more details.

TCP_RXTLS_ENABLE

Enable in-kernel TLS for data read from this socket. See ktls(4) for more details.

TCP_REUSPORT_LB_NUMA

Changes NUMA affinity filtering for an established TCP listen socket. This option takes a single integer argument which specifies the NUMA domain to filter on for this listen socket. The argument can also have the follwing special values:

TCP_REUSPORT_LB_NUMA_NODOM

Remove NUMA filtering for this listen socket.

TCP_REUSPORT_LB_NUMA_CURDOM

Filter traffic associated with the domain where the calling thread is currently executing. This is typically used after a process or thread inherits a listen socket from its parent, and sets its CPU affinity to a particular core.

TCP_REMOTE_UDP_ENCAPS_PORT

Set and get the remote UDP encapsulation port. It can only be set on a closed TCP socket.

The option level for the setsockopt(2) call is the protocol number for TCP, available from getprotobyname(3), or IPPROTO_TCP. All options are declared in <*netinet/tcp.h*>.

Options at the IP transport level may be used with TCP; see ip(4). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

The default congestion control algorithm for TCP is cc_newreno(4). Other congestion control algorithms can be made available using the mod_cc(4) framework.

MIB (sysctl) Variables

The TCP protocol implements a number of variables in the *net.inet.tcp* branch of the sysctl(3) MIB, which can also be read or modified with sysctl(8).

ack_war_timewindow, *ack_war_cnt*
: The challenge ACK throttling algorithm defined in RFC 5961 limits the number of challenge ACKs sent per TCP connection to *ack_war_cnt* during the time interval specified in milliseconds by *ack_war_timewindow*. Setting *ack_war_timewindow* or *ack_war_cnt* to zero disables challenge ACK throttling.

always_keepalive
: Assume that SO_KEEPALIVE is set on all TCP connections, the kernel will periodically send a packet to the remote host to verify the connection is still up.

blackhole
: If enabled, disable sending of RST when a connection is attempted to a port where there is no socket accepting connections. See blackhole(4).

blackhole_local
: See blackhole(4).

cc
: A number of variables for congestion control are under the *net.inet.tcp.cc* node. See mod_cc(4).

cc.newreno
: Variables for NewReno congestion control are under the *net.inet.tcp.cc.newreno* node. See cc_newreno(4).

delacktime
: Maximum amount of time, in milliseconds, before a delayed ACK is sent.

delayed_ack
: Delay ACK to try and piggyback it onto a data packet or another ACK.

do_lrd
: Enable Lost Retransmission Detection for SACK-enabled sessions, disabled

by default. Under severe congestion, a retransmission can be lost which then leads to a mandatory Retransmission Timeout (RTO), followed by slow-start. LRD will try to resend the repeatedly lost packet, preventing the time-consuming RTO and performance reducing slow-start.

do_prr Perform SACK loss recovery using the Proportional Rate Reduction (PRR) algorithm described in RFC6937. This improves the effectiveness of retransmissions particular in environments with ACK thinning or burst loss events, as chances to run out of the ACK clock are reduced, preventing lengthy and performance reducing RTO based loss recovery (default is true).

do_tcpdrain Flush packets in the TCP reassembly queue if the system is low on mbufs.

drop_synfin Drop TCP packets with both SYN and FIN set.

ecn.enable Enable support for TCP Explicit Congestion Notification (ECN). ECN allows a TCP sender to reduce the transmission rate in order to avoid packet drops.

0 Disable ECN.

1 Allow incoming connections to request ECN. Outgoing connections will request ECN.

2 Allow incoming connections to request ECN. Outgoing connections will not request ECN. (default)

3 Negotiate on incoming connection for Accurate ECN, ECN, or no ECN. Outgoing connections will request Accurate ECN and fall back to ECN depending on the capabilities of the server.

4 Negotiate on incoming connection for Accurate ECN, ECN, or no ECN. Outgoing connections will not request ECN.

ecn.maxretries Number of retries (SYN or SYN/ACK retransmits) before disabling ECN on a specific connection. This is needed to help with connection establishment when a broken firewall is in the network path.

fast_finwait2_recycle Recycle TCP FIN_WAIT_2 connections faster when the socket is marked as SBS_CANTRCVMORE (no user process has the socket open, data received on the socket cannot be read). The timeout used here is *finwait2_timeout*.

fastopen.acceptany When non-zero, all client-supplied TFO cookies will be considered to be valid. The default is 0.

fastopen.autokey When this and *net.inet.tcp.fastopen.server_enable* are non-zero, a new key

will be automatically generated after this specified seconds. The default is 120.

fastopen.ccache_bucket_limit
The maximum number of entries in a client cookie cache bucket. The default value can be tuned with the TCP_FASTOPEN_CCACHE_BUCKET_LIMIT_DEFAULT kernel option or by setting *net.inet.tcp.fastopen_ccache_bucket_limit* in the loader(8).

fastopen.ccache_buckets The number of client cookie cache buckets. Read-only. The value can be tuned with the TCP_FASTOPEN_CCACHE_BUCKETS_DEFAULT kernel option or by setting *fastopen.ccache_buckets* in the loader(8).

fastopen.ccache_list Print the client cookie cache. Read-only.

fastopen.client_enable When zero, no new active (i.e., client) TFO connections can be created. On the transition from enabled to disabled, the client cookie cache is cleared and disabled. The transition from enabled to disabled does not affect any active TFO connections in progress; it only prevents new ones from being established. The default is 0.

fastopen.keylen The key length in bytes. Read-only.

fastopen.maxkeys The maximum number of keys supported. Read-only,

fastopen.maxpsks The maximum number of pre-shared keys supported. Read-only.

fastopen.numkeys The current number of keys installed. Read-only.

fastopen.numpsks The current number of pre-shared keys installed. Read-only.

fastopen.path_disable_time
When a failure occurs while trying to create a new active (i.e., client) TFO connection, new active connections on the same path, as determined by the tuple {client_ip, server_ip, server_port}, will be forced to be non-TFO for this many seconds. Note that the path disable mechanism relies on state stored in client cookie cache entries, so it is possible for the disable time for a given path to be reduced if the corresponding client cookie cache entry is reused due to resource pressure before the disable period has elapsed. The default is TCP_FASTOPEN_PATH_DISABLE_TIME_DEFAULT.

fastopen.psk_enable When non-zero, pre-shared key (PSK) mode is enabled for all TFO servers. On the transition from enabled to disabled, all installed pre-shared keys are removed. The default is 0.

fastopen.server_enable When zero, no new passive (i.e., server) TFO connections can be created. On the transition from enabled to disabled, all installed keys and pre-shared keys are removed. On the transition from disabled to enabled, if *fastopen.autokey* is non-zero and there are no keys installed, a new key will be generated immediately. The transition from enabled to disabled does not affect any passive TFO connections in progress; it only prevents new ones from being established. The default is 0.

fastopen.setkey Install a new key by writing *net.inet.tcp.fastopen.keylen* bytes to this sysctl.

fastopen.setpsk Install a new pre-shared key by writing *net.inet.tcp.fastopen.keylen* bytes to this sysctl.

finwait2_timeout Timeout to use for fast recycling of TCP FIN_WAIT_2 connections (*fast_finwait2_recycle*). Defaults to 60 seconds.

functions_available List of available TCP function blocks (TCP stacks).

functions_default The default TCP function block (TCP stack).

functions_inherit_listen_socket_stack
Determines whether to inherit listen socket's TCP stack or use the current system default TCP stack, as defined by *functions_default*. Default is true.

hostcache The TCP host cache is used to cache connection details and metrics to improve future performance of connections between the same hosts. At the completion of a TCP connection, a host will cache information for the connection for some defined period of time. There are a number of *hostcache* variables under this node. See *hostcache.enable*.

hostcache.bucketlimit The maximum number of entries for the same hash. Defaults to 30.

hostcache.cachelimit Overall entry limit for hostcache. Defaults to *hashsize* * *bucketlimit*.

hostcache.count The current number of entries in the host cache.

hostcache.enable Enable/disable the host cache:
0 Disable the host cache.
1 Enable the host cache. (default)

hostcache.expire Time in seconds, how long a entry should be kept in the host cache since last accessed. Defaults to 3600 (1 hour).

hostcache.hashsize Size of TCP hostcache hashtable. This number has to be a power of two, or will be rejected. Defaults to 512.

hostcache.histo Provide a Histogram of the hostcache hash utilization.

hostcache.list Provide a complete list of all current entries in the host cache.

hostcache.prune Time in seconds between pruning expired host cache entries. Defaults to 300 (5 minutes).

hostcache.purge Expire all entires on next pruning of host cache entries. Any non-zero setting will be reset to zero, once the purge is running.
0 Do not purge all entries when pruning the host cache (default).
1 Purge all entries when doing the next pruning.
2 Purge all entries and also reseed the hash salt.

hostcache.purgenow Immediately purge all entries once set to any value. Setting this to 2 will also reseed the hash salt.

icmp_may_rst Certain ICMP unreachable messages may abort connections in SYN-SENT state.

initcwnd_segments Enable the ability to specify initial congestion window in number of segments. The default value is 10 as suggested by RFC 6928. Changing the value on the fly would not affect connections using congestion window from the hostcache. Caution: This regulates the burst of packets allowed to be sent in the first RTT. The value should be relative to the link capacity. Start with small values for lower-capacity links. Large bursts can cause buffer overruns and packet drops if routers have small buffers or the link is experiencing congestion.

insecure_rst Use criteria defined in RFC793 instead of RFC5961 for accepting RST segments. Default is false.

insecure_syn Use criteria defined in RFC793 instead of RFC5961 for accepting SYN segments. Default is false.

insecure_ack Use criteria defined in RFC793 for validating SEG.ACK. Default is false.

isn_reseed_interval The interval (in seconds) specifying how often the secret data used in RFC 1948 initial sequence number calculations should be reseeded. By default, this variable is set to zero, indicating that no reseeding will occur. Reseeding should not be necessary, and will break TIME_WAIT recycling for a few minutes.

keepcnt Number of keepalive probes sent, with no response, before a connection is dropped. The default is 8 packets.

keepidle Amount of time, in milliseconds, that the connection must be idle before sending keepalive probes (if enabled). The default is 7200000 msec (7.2M msec, 2 hours).

keepinit Timeout, in milliseconds, for new, non-established TCP connections. The default is 75000 msec (75K msec, 75 sec).

keepintvl The interval, in milliseconds, between keepalive probes sent to remote machines, when no response is received on a *keepidle* probe. The default is 75000 msec (75K msec, 75 sec).

log_in_vain Log any connection attempts to ports where there is no socket accepting connections. The value of 1 limits the logging to SYN (connection establishment) packets only. A value of 2 results in any TCP packets to closed ports being logged. Any value not listed above disables the logging (default is 0, i.e., the logging is disabled).

minmss Minimum TCP Maximum Segment Size; used to prevent a denial of service attack from an unreasonably low MSS.

msl The Maximum Segment Lifetime, in milliseconds, for a packet.

mssdflt The default value used for the TCP Maximum Segment Size ("MSS") for IPv4 when no advice to the contrary is received from MSS negotiation.

newcwd Enable the New Congestion Window Validation mechanism as described in

RFC 7661. This gently reduces the congestion window during periods, where TCP is application limited and the network bandwidth is not utilized completely. That prevents self-inflicted packet losses once the application starts to transmit data at a higher speed.

nolocaltimewait Suppress creation of TCP TIME_WAIT states for connections in which both endpoints are local.

path_mtu_discovery Enable Path MTU Discovery.

pcbcount Number of active protocol control blocks (read-only).

perconn_stats_enable Controls the default collection of statistics for all connections using the stats(3) framework. 0 disables, 1 enables, 2 enables random sampling across log id connection groups with all connections in a group receiving the same setting.

perconn_stats_sample_rates

A CSV list of template_spec=percent key-value pairs which controls the per template sampling rates when stats(3) sampling is enabled.

persmax Maximum persistence interval, msec.

persmin Minimum persistence interval, msec.

pmtud_blackhole_detection

Enable automatic path MTU blackhole detection. In case of retransmits of MSS sized segments, the OS will lower the MSS to check if it's an MTU problem. If the current MSS is greater than the configured value to try (*net.inet.tcp.pmtud_blackhole_mss* and *net.inet.tcp.v6pmtud_blackhole_mss*), it will be set to this value, otherwise, the MSS will be set to the default values (*net.inet.tcp.mssdflt* and *net.inet.tcp.v6mssdflt*). Settings:

0 Disable path MTU blackhole detection.
1 Enable path MTU blackhole detection for IPv4 and IPv6.
2 Enable path MTU blackhole detection only for IPv4.
3 Enable path MTU blackhole detection only for IPv6.

pmtud_blackhole_mss MSS to try for IPv4 if PMTU blackhole detection is turned on.

reass.cursegments The current total number of segments present in all reassembly queues.

reass.maxqueuelen The maximum number of segments allowed in each reassembly queue. By default, the system chooses a limit based on each TCP connection's receive buffer size and maximum segment size (MSS). The actual limit applied to a session's reassembly queue will be the lower of the system-calculated automatic limit and the user-specified *reass.maxqueuelen* limit.

reass.maxsegments The maximum limit on the total number of segments across all reassembly queues. The limit can be adjusted as a tunable.

recvbuf_auto Enable automatic receive buffer sizing as a connection progresses.

recvbuf_max Maximum size of automatic receive buffer.

recvspace Initial TCP receive window (buffer size).

retries Maximum number of consecutive timer based retransmits sent after a data segment is lost (default and maximum is 12).

rexmit_drop_options Drop TCP options from third and later retransmitted SYN segments of a connection.

rexmit_initial, rexmit_min, rexmit_slop
Adjust the retransmit timer calculation for TCP. The slop is typically added to the raw calculation to take into account occasional variances that the SRTT (smoothed round-trip time) is unable to accommodate, while the minimum specifies an absolute minimum. While a number of TCP RFCs suggest a 1 second minimum, these RFCs tend to focus on streaming behavior, and fail to deal with the fact that a 1 second minimum has severe detrimental effects over lossy interactive connections, such as a 802.11b wireless link, and over very fast but lossy connections for those cases not covered by the fast retransmit code. For this reason, we use 200ms of slop and a near-0 minimum, which gives us an effective minimum of 200ms (similar to Linux). The initial value is used before an RTT measurement has been performed.

rfc1323 Implement the window scaling and timestamp options of RFC 1323/RFC 7323 (default is 1). Settings:

0 Disable window scaling and timestamp option.
1 Enable window scaling and timestamp option.
2 Enable only window scaling.
3 Enable only timestamp option.

rfc3042 Enable the Limited Transmit algorithm as described in RFC 3042. It helps avoid timeouts on lossy links and also when the congestion window is small, as happens on short transfers.

rfc3390 Enable support for RFC 3390, which allows for a variable-sized starting congestion window on new connections, depending on the maximum segment size. This helps throughput in general, but particularly affects short transfers and high-bandwidth large propagation-delay connections.

rfc6675_pipe Deprecated and superseded by *sack.revised*

sack.enable Enable support for RFC 2018, TCP Selective Acknowledgment option, which allows the receiver to inform the sender about all successfully arrived segments, allowing the sender to retransmit the missing segments only.

sack.globalholes Global number of TCP SACK holes currently allocated.

sack.globalmaxholes Maximum number of SACK holes per system, across all connections. Defaults to 65536.

sack.maxholes Maximum number of SACK holes per connection. Defaults to 128.

sack.revised Enables three updated mechanisms from RFC6675 (default is true). Calculate the bytes in flight using the algorithm described in RFC 6675, and is also an improvement when Proportional Rate Reduction is enabled. Next, Rescue Retransmission helps timely loss recovery, when the trailing segments of a transmission are lost, while no additional data is ready to be sent. In case a partial ACK without a SACK block is received during SACK loss recovery, the trailing segment is immediately resent, rather than waiting for a Retransmission timeout. Finally, SACK loss recovery is also engaged, once two segments plus one byte are SACKed - even if no traditional duplicate ACKs were observed.

sendbuf_auto Enable automatic send buffer sizing.

sendbuf_auto_lowat Modify threshold for auto send buffer growth to account for SO_SNDLOWAT.

sendbuf_inc Incrementor step size of automatic send buffer.

sendbuf_max Maximum size of automatic send buffer.

sendspace Initial TCP send window (buffer size).

syncache Variables under the *net.inet.tcp.syncache* node are documented in syncache(4).

syncookies Determines whether or not SYN cookies should be generated for outbound SYN-ACK packets. SYN cookies are a great help during SYN flood attacks, and are enabled by default. (See syncookies(4).)

syncookies_only See syncookies(4).

tcbhashsize Size of the TCP control-block hash table (read-only). This is tuned using the kernel option TCBHASHSIZE or by setting *net.inet.tcp.tcbhashsize* in the loader(8).

tolerate_missing_ts Tolerate the missing of timestamps (RFC 1323/RFC 7323) for TCP segments belonging to TCP connections for which support of TCP timestamps has been negotiated. As of June 2021, several TCP stacks are known to violate RFC 7323, including modern widely deployed ones. Therefore the default is 1, i.e., the missing of timestamps is tolerated.

ts_offset_per_conn When initializing the TCP timestamps, use a per connection offset instead of a per host pair offset. Default is to use per connection offsets as recommended in RFC 7323.

tso Enable TCP Segmentation Offload.

udp_tunneling_overhead The overhead taken into account when using UDP encapsulation. Since MSS clamping by middleboxes will most likely not work, values larger than 8 (the size of the UDP header) are also supported. Supported values are between 8 and 1024. The default is 8.

udp_tunneling_port The local UDP encapsulation port. A value of 0 indicates that UDP encapsulation is disabled. The default is 0.

v6mssdflt The default value used for the TCP Maximum Segment Size ("MSS") for IPv6 when no advice to the contrary is received from MSS negotiation.

v6pmtud_blackhole_mss MSS to try for IPv6 if PMTU blackhole detection is turned on. See *pmtud_blackhole_detection*.

ERRORS

A socket operation may fail with one of the following errors returned:

[EISCONN] when trying to establish a connection on a socket which already has one;

[ENOBUFS] or [ENOMEM]
when the system runs out of memory for an internal data structure;

[ETIMEDOUT] when a connection was dropped due to excessive retransmissions;

[ECONNRESET] when the remote peer forces the connection to be closed;

[ECONNREFUSED]
when the remote peer actively refuses connection establishment (usually because no process is listening to the port);

[EADDRINUSE] when an attempt is made to create a socket with a port which has already been allocated;

[EADDRNOTAVAIL]
when an attempt is made to create a socket with a network address for which no network interface exists;

[EAFNOSUPPORT] when an attempt is made to bind or connect a socket to a multicast address.

[EINVAL] when trying to change TCP function blocks at an invalid point in the session;

[ENOENT] when trying to use a TCP function block that is not available;

SEE ALSO

getsockopt(2), socket(2), stats(3), sysctl(3), blackhole(4), inet(4), intro(4), ip(4), ktls(4), mod_cc(4), siftr(4), syncache(4), tcp_bbr(4), tcp_rack(4), setkey(8), sysctl(8), tcp_functions(9)

V. Jacobson, B. Braden, and D. Borman, *TCP Extensions for High Performance*, RFC 1323.

D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, *TCP Extensions for High Performance*, RFC 7323.

A. Heffernan, *Protection of BGP Sessions via the TCP MD5 Signature Option*, RFC 2385.

K. Ramakrishnan, S. Floyd, and D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168.

A. Ramaiah, R. Stewart, and M. Dalal, *Improving TCP's Robustness to Blind In-Window Attacks*, RFC 5961.

HISTORY

The TCP protocol appeared in 4.2BSD. The RFC 1323 extensions for window scaling and timestamps were added in 4.4BSD. The TCP_INFO option was introduced in Linux 2.6 and is *subject to change*.