Contents

1 Configuring TCP....1

1.1 Introduction....1

1.1.1 Overview....1

1.1.2 Principles....1

1.1.3 Protocols and Standards....6

1.2 Configuration Task Summary....6

1.3 Configuring TCP SYN Packet Timeout Period....6

1.3.1 Overview....6

1.3.2 Restrictions and Guidelines....7

1.3.3 Procedure....7

1.4 Configuring a TCP Window Size....7

1.4.1 Overview....7

1.4.2 Restrictions and Guidelines....7

1.4.3 Procedure....7

1.5 Configuring the Sending of TCP Reset Packets Upon the Receiving of Port Unreachable Messages....8

1.5.1 Overview....8

1.5.2 Restrictions and Guidelines....8

1.5.3 Procedure....8

1.6 Configuring MSS for a TCP Connection....8

1.6.1 Overview....8

1.6.2 Restrictions and Guidelines....8

1.6.3 Procedure......9

1.7 Configuring the TCP Path MTU Discovery Function......9

1.7.1 Overview......9

1.7.2 Restrictions and Guidelines......9

1.7.3 Procedure......9

1.8 Configuring the MSS Option for TCP SYN Packets......10

1.8.1 Overview......10

1.8.2 Restrictions and Guidelines......10

1.8.3 Procedure......10

1.9 Configuring TCP Connection Exception Detection......10

1.9.1 Overview......10

1.9.2 Procedure......11

1.10 Monitoring......11

1 Configuring TCP

1.1 Introduction

1.1.1 Overview

Transmission Control Protocol (TCP) is a connection-oriented, reliable, and byte stream-based transport layer communication protocol. It is defined in RFC793 of the Internet Engineering Task Force (IETF).

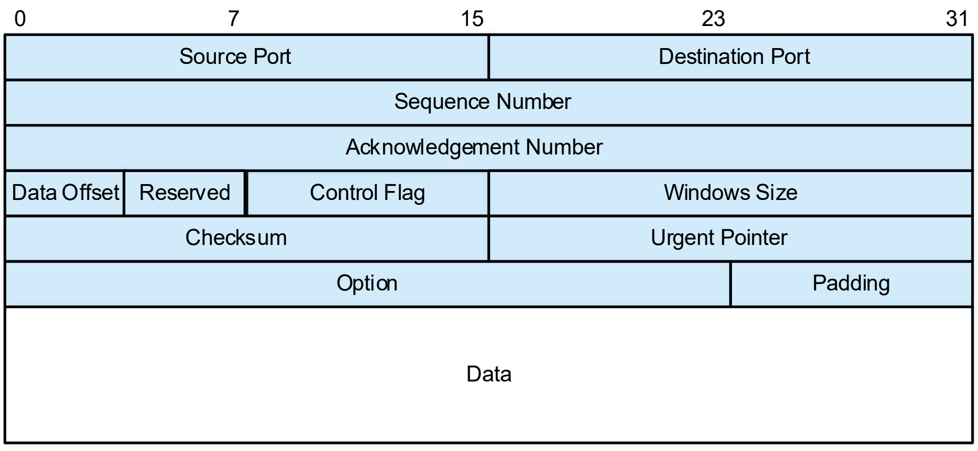
An interconnected network is very different from a single network in that topologies, bandwidth, latency, packet sizes, and other parameters may be entirely different in different parts of the interconnected network. TCP is designed to adapt dynamically to these features of interconnected networks and show robustness in the case of various failures.

- TCP uses a checksum function to check data correctness and validity. The checksum needs to be calculated before transmission and after receiving. Besides, TCP uses the message digest 5 (MD5) authentication to verify data.
- TCP uses sequential transmission, timeout retransmission, and piggyback mechanism to ensure reliability.
- TCP uses the sliding window protocol to control flows. As documented in the protocol, unacknowledged packets in a window need to be retransmitted.
- TCP uses the widely praised TCP congestion control algorithm (also known as the additive increase and multiplicative decrease (AIMD) algorithm) to control congestion.

1.1.2 Principles

1. TCP Header Format

Figure 1-1 TCP Header Format

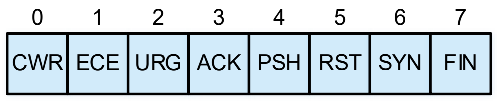


Field descriptions are as follows:

- **Source Port**: Indicates a 16-bit source port number.

- **Destination Port**: Indicates a 16-bit destination port number.
- **Sequence Number**: Indicates a 32-bit sequence number.
- **Acknowledgment Number**: Indicates a 32-bit acknowledgment sequence number.
- **Data Offset**: Indicates a 4-bit data offset. The value is the length of the TCP header (including the option) divided by 4.
- **Control Flag**: Indicates an 8-bit control flag.

Figure 1-2 Control Flag



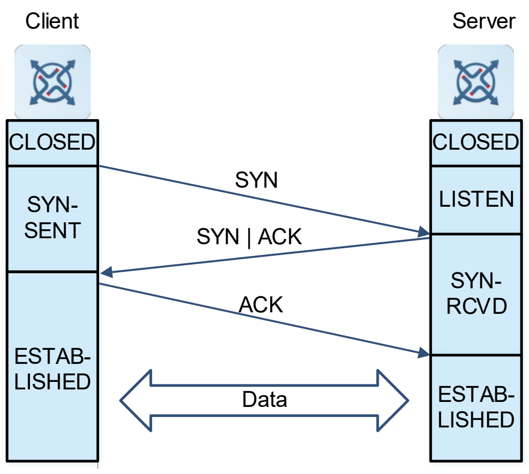
  - Congestion window reduced (**CWR**): The **CWR** flag and the **ECE** flag behind are used for the Explicit Congestion Notification (**ECN**) field in the IP header. When the **ECE** flag is **1**, the peer is notified to make the congestion window smaller.
  - ECN-Echo (**ECE**): The **ECE** flag indicates ECN-Echo. When it is set to **1**, the peer is notified of the congestion from the peer to the local network. When the **ECN** field in the IP header of a received packet is set to **1**, the **ECE** flag in the TCP header is set to **1**.
  - Urgent flag (**URG**): When this bit is set to **1**, the packet contains data that needs urgent processing. Data that needs urgent processing will be further explained in the description of the **Urgent Pointer** field.
  - Acknowledge flag (**ACK**): When this bit is set to **1**, the acknowledgment number is valid. According to the TCP protocol, this bit must be set to **1** except for the SYN packet in the initial establishment of a TCP connection.
  - Push Flag (**PSH**): When this bit is set to **1**, the received data must be immediately transferred to the upper-layer application. When **PSH** is set to **0**, data is buffered instead of being transferred to the upper-layer application immediately.
  - Reset Flag (**RST**): When this bit is set to **1**, the TCP connection must be forcibly closed due to an exception. For example, if a client attempts to establish a TCP connection with a server port that has not provided service externally, the server sends a **RST** packet to the client. In addition, if a device is restarted due to a program crash or power cut, established TCP connections on the device will be closed because all connections will be initialized. In this case, if the peer sends a packet with **RST** set to **1**, the connection is forcibly closed.
  - Synchronize Flag (**SYN**): This bit is used for establishing a connection. If **SYN** is set to **1**, a connection is needed and the initial value of the sequence number is set in the sequence number field.
  - Finish Flag (**FIN**): When this bit is set to **1**, no more data will be sent and disconnection is needed. When the communication ends and disconnection is required, both communication hosts can exchange TCP segments with **FIN** set to **1**. The connection can be closed after each host acknowledges the FIN packet from the peer. However, a host does not have to reply with a FIN packet immediately after receiving a TCP packet with **FIN** set to **1**. Instead, it can wait until all data in the buffer has been successfully sent and is automatically deleted.

- **Window Size**: Indicates 16-bit idle space in the receive buffer. It is used to notify the peer end of a TCP connection of the maximum data length supported by the local end.
- **Checksum**: Indicates a 16-bit TCP checksum.
- **Urgent Pointer**: Indicates a 16-bit field showing the offset of urgent data relative to the value of **Sequence Number**. It is meaningful only when the **URG** flag is set to **1**.
- **Option**: Indicates the option field used to improve the TCP transmission performance. The maximum length is 40 bytes because it is controlled by the data offset (header length). In addition, you are advised to set the **Option** field to an integer multiple of 32 bits.

2. Establishing a TCP Connection

TCP uses the "three-way handshake" mechanism to establish a virtual circuit connection.

Figure 1-1 Establishing a TCP Connection



(1) The client initiates a connection and ends the CLOSED phase. In this case, the server enters the LISTEN phase.

(2) The client sends a TCP SYN packet to the server, and then enters the SYN-SENT phase.

(3) The server receives the SYN packet, ends the LISTEN phase, and responds with an SYN ACK packet. Then, the server enters the SYN-RCVD phase.

(4) The client receives the SYN ACK packet from the server, confirming that the data transmission from the client to the server is normal. The client ends the SYN-SENT phase, and responds with an ACK packet. Then, the client enters the ESTABLISHED phase.

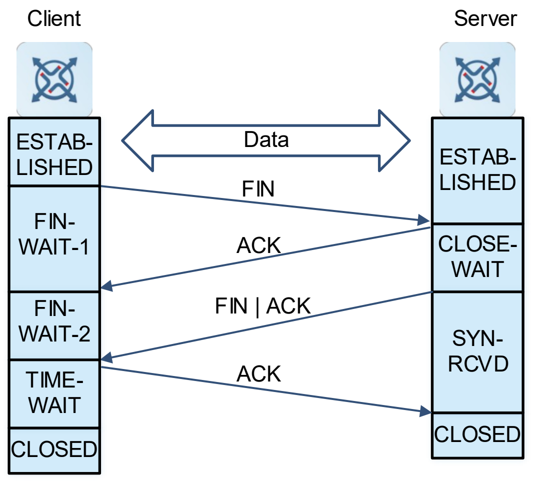
(5) The server receives the ACK packet from the client, confirming that the data transmission from the server to the client is normal. The server ends the SYN-SENT phase, and enters the ESTABLISHED phase.

(6) So far, the three-way handshake is completed, the TCP client and server establish a connection successfully and are ready for data transmission.

3. Releasing a TCP Connection

TCP uses the "four-way wavehand" mechanism to release a virtual circuit connection.

Figure 1-1 Releasing a TCP Connection



(1) To release a connection, the client sends a TCP FIN packet to the server. The client enters the FIN-WAIT-1 phase and stops sending data to the server. At this moment, the client can still receive data from the server.

(2) The server receives the FIN packet, ends the ESTABLISHED phase, enters the CLOSE-WAIT phase, and responds with an ACK packet.

(3) The client receives the ACK packet from the server, confirming that the server has received the connection release request from the client. Then, the client ends the FIN-WAIT-1 phase and enters the FIN-WAIT-2 phase.

(4) After sending the ACK packet, the server goes through the Closed-Wait phase, and sends a FIN ACK packet to the client. At this moment, the server has stopped sending data to the client, but can still receive data from the client.

(5) The client receives the TCP packet from the server, ends the FIN-WAIT-2 phase, enters the TIME-WAIT phase, and sends the ACK packet to the server.

(6) After waiting for a certain period of time, the client ends the TIME-WAIT phase and enters the CLOSED phase. The "four-way wavehand" is completed.

4. SYN Packet Retransmission Upon Timeout

If the server does not reply with an SYN ACK packet after the client sends an SYN packet, the client keeps retransmitting the SYN packet for certain times or until the timeout period expires.

If the server replies with an SYN ACK packet after the client sends an SYN packet but the client does not reply with an ACK packet, the server keeps retransmitting the SYN ACK packet for certain times or until the timeout period expires. This may be SYN flooding.

5. TCP Sliding Window

The TCP sliding window is the receive buffer used to buffer data from the peer. The data will be subsequently read by applications. The TCP window size reflects the size of idle space in the receive buffer. For bulk-data connections, enlarging the window size dramatically promotes TCP transmission performance.

6. MSS

The maximum segment size (MSS) refers to the maximum length of data payload in a TCP segment, excluding the TCP option.

Three-way handshake used for establishing a TCP connection needs MSS negotiation. Both parties of the connection add the MSS option to SYN packets. The option indicates the maximum size of a segment that can be received by the local end, that is, the maximum size of a segment that can be sent by the peer end. Both parties compare the local MSS value with the MSS value received from the peer end and take the smaller MSS value as the MSS of the connection.

The MSS option value in SYN packets to be sent is calculated as follows:

- IPv4 TCP: MSS = IP MTU of the outbound interface corresponding to the peer IP address – IP header size (20 bytes) – TCP header size (20 bytes).
- IPv6 TCP: MSS = Path MTU corresponding to the peer IPv6 address – IPv6 header size (40 bytes) – TCP header size (20 bytes).

Note

The MSS that actually takes effect is the MSS calculated based on the maximum transmission unit (MTU) or the configured MSS, whichever is smaller.

If a connection supports certain options, the option length after 4-byte alignment should be deducted from the MSS value. For example, 20 bytes need to be deducted if the MD5 option is used because the length of the MD5 option is 18 bytes and the length after alignment is 20 bytes.

7. Path MTU Discovery

The TCP path MTU discovery function stipulated in RFC1191 is used to discover the smallest MTU in a TCP path to prevent fragmentation and reassembly and enhance the network bandwidth utilization. The process of IPv4 TCP path MTU discovery is described as follows:

(1) The TCP source sets the Don't Fragment (DF) bit in the outer IP header in a TCP packet to be sent.

(2) If the outbound interface MTU value of a device in the TCP path is smaller than the IP packet length, the packet will be discarded and an ICMP error packet carrying this outbound interface MTU will be sent to the TCP source.

(3) By parsing the ICMP error packet, the TCP source knows the smallest MTU in the TCP path (that is, path MTU).

(4) The size of subsequent data segments sent by the TCP source will not surpass the MSS, which is calculated as follows: TCP MSS = Path MTU – IP header size – TCP header size.

If the MSS has reached the minimum 32 bytes specified by the system and an ICMP error packet asking the system to reduce the MSS is received again, the system will allow the fragmentation of the packets sent from this TCP connection.

The aging mechanism of the path MTU is as follows:

(1) After receiving an ICMP error packet, the TCP source not only reduces the path MTU value, but also starts an aging timer for the path MTU value.

(5) The system gradually increases the MSS value of TCP according to the MTU table specified in RFC 1191 when the timer expires.

(6) If no ICMP error packet is received within 2 minutes after one increment of the MSS, the system continues the increment until the MSS increases to the MSS value advertised by the peer during the three-way handshake.

8. TCP Keepalive Function

You may enable the keepalive function to check whether the peer end of a TCP connection works normally. If the TCP peer end does not send a packet to the local end within a period of time (namely, idle period), the local end starts sending keepalive packets to the peer end for several consecutive times. If no response packet is received, the local end considers the peer end abnormal and closes the TCP connection.

1.1.3 Protocols and Standards

- RFC 793: Transmission Control Protocol
- RFC 1122: Requirements for Internet Hosts -- Communication Layers
- RFC 1191: Path MTU Discovery
- RFC 1213: Management Information Base for Network Management of TCP/IP-based internets: MIB-II
- RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option
- RFC 4022: Management Information Base for the Transmission Control Protocol (TCP)

1.2 Configuration Task Summary

All the configuration tasks below are optional. Perform the configuration tasks as required.

- Optimizing TCP performance
  - Configuring TCP SYN Packet Timeout Period
  - Configuring a TCP Window Size
  - Configuring the Sending of TCP Reset Packets Upon the Receiving of Port Unreachable Messages
  - Configuring MSS for a TCP Connection
  - Configuring the TCP Path MTU Discovery Function
  - Configuring the MSS Option for TCP SYN Packets
- Configuring TCP Connection Exception Detection

1.3 Configuring TCP SYN Packet Timeout Period

1.3.1 Overview

In case of SYN flooding, shortening SYN timeout period can reduce resource consumption. However, it does not work on continuous SYN flooding.

When a device actively requests to establish a connection with an external device, shortening SYN timeout period can reduces users' waiting time, for example, waiting time in the telnet connection.

You may prolong SYN timeout period properly on an unstable network.

1.3.2 Restrictions and Guidelines

Perform the configuration on both ends of a TCP connection.

1.3.3 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Configure SYN timeout.

ip tcp synwait-time *time*

The default timeout period of SYN packets used for connection establishment is 20 seconds.

1.4 Configuring a TCP Window Size

1.4.1 Overview

The TCP receive buffer is used to buffer data from the peer. The data will be subsequently read by applications. The TCP window size reflects the size of idle space in the receive buffer. For bulk-data connections, enlarging the window size dramatically promotes TCP transmission performance.

1.4.2 Restrictions and Guidelines

Perform the configuration on both ends of a TCP connection.

1.4.3 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Configures a TCP window size.

ip tcp window-size *size*

The default TCP window size is 65,535 bytes.

If the window size is greater than 65535 bytes, window enlarging will be enabled automatically. The window size advertised to the peer is the configured window size or the idle space in the receive buffer, whichever is smaller.

Note

This command applies to both IPv4 TCP and IPv6 TCP.

1.5 Configuring the Sending of TCP Reset Packets Upon the Receiving of Port Unreachable Messages

1.5.1 Overview

In general, when the device distributes a TCP packet, if the TCP connection, to which the packet belongs cannot be identified, the device sends a reset packet to the peer to terminate the TCP connection. This, however, can also become a target for attackers. A large number of TCP port unreachable messages can impose attacks on the device. You can use this command to prevent the sending of TCP reset packets upon the receiving of port unreachable messages.

1.5.2 Restrictions and Guidelines

Perform the configuration on both ends of a TCP connection.

1.5.3 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Configure the sending of TCP reset packets after port unreachable messages are received.

ip tcp send-reset

TCP reset packets are sent upon the receiving of port unreachable messages by default.

Note

This command applies to both IPv4 TCP and IPv6 TCP.

1.6 Configuring MSS for a TCP Connection

1.6.1 Overview

TCP negotiates about the MSS value when establishing a connection. The MSS refers to the maximum length of data payload in a TCP segment. If the path MTU discovery function at one end of a TCP connection is not available, the device at this end cannot calculate the size of TCP packets based on the MTU value, and the TCP packets may be discarded due to too long size. To prevent this situation, you can manually set the MSS for TCP packets so that the MSS value negotiated by both ends will not exceed this value and the packets can successfully go through the intermediate networks.

1.6.2 Restrictions and Guidelines

- Perform the configuration on both ends of a TCP connection.
- An excessively small MSS reduces transmission performance. You can promote TCP transmission by increasing the MSS. Configure the MSS by referring to the interface MTU. If the former is bigger than the interface MTU, TCP packets need to be fragmented and reassembled, which reduces the transmission performance.
- If an upper MSS limit is configured, the upper MSS limit that actually takes effect is the MSS calculated based on the MTU or configured MSS, whichever is smaller.

1.6.3 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Configure an upper MSS limit.

ip tcp mss *max-segment-size*

The calculated value "IPv4/IPv6 MTU – IPv4/IPv6 header length – TCP header length" is used as the upper MSS limit by default.

Note

This command applies to both IPv4 TCP and IPv6 TCP.

1.7 Configuring the TCP Path MTU Discovery Function

1.7.1 Overview

The path MTU discovery function of TCP is implemented according to RFC1191 to improve the network bandwidth utilization. When TCP is applied to bulk transmit chunk data, this function can improve transmission performance greatly.

As described in RFC1191, after discovering the path MTU, TCP can use a larger MSS to probe a new path MTU at intervals. This interval is specified by using the **age-timer** parameter. When the device discovers a path MTU smaller than the MSS negotiated by both ends of a TCP connection, the device tries to probe a larger path MTU at the configured interval described above. The probe process is stopped when the path MTU reaches the MSS or the user turns off the timer. You may use the **age-timer infinite** parameter to turn off this timer.

1.7.2 Restrictions and Guidelines

Perform the configuration on both ends of a TCP connection.

1.7.3 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Enable the path MTU discovery function.

ip tcp path-mtu-discovery [**age-timer** *time* | **age-timer infinite**]

The path MTU discovery function of TCP is disabled by default.

Note

This command applies to only IPv4 TCP. The path MTU discovery of IPv6 TCP is enabled permanently and cannot be disabled.

1.8 Configuring the MSS Option for TCP SYN Packets

1.8.1 Overview

Both ends of a TCP connection negotiate about the MSS value for subsequent TCP transmission during the handshake. After the three-way handshake is successful, a TCP connection is established between both ends, and the devices at both ends forward the TCP packets. The tunnel header is encapsulated when a TCP packet is forwarded. Therefore, the length of the TCP packet may exceed the MSS value negotiated during connection establishment, resulting in packet fragmentation and reducing transmission efficiency.

To avoid packet fragmentation, you may configure the MSS option for TCPv4 SYN packets to be forwarded on the device. After receiving a new TCPv4 SYN packet, the device changes the MSS option value in the TCPv4 SYN packet to the configured value.

1.8.2 Restrictions and Guidelines

You may configure an MSS value with a reference to the interface IP MTU.

This configuration applies to new connections but do not take effect on existing TCP connections.

1.8.3 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Enter the interface configuration mode.

interface *interface-type interface-number*

(4) Configure the MSS option for TCP SYN packets to be forwarded.

(IPv4 network)

ip tcp adjust-mss *max-segment-size*

The MSS option is not configured for TCP SYN packets by default.

(IPv6 network)

ipv6 tcp adjust-mss *max-segment-size*

The MSS option is not configured for TCPv6 SYN packets to be forwarded by default.

1.9 Configuring TCP Connection Exception Detection

1.9.1 Overview

You may enable the TCP keepalive function to check whether the peer works normally.

Suppose that the TCP keepalive function is enabled on a device and default interval, transmission count, and idle period settings are used. If no packet is received from the peer within 15 minutes, the device starts sending keepalive packets every 75 seconds for 6 consecutive times. If the device receives no TCP packet from the peer, it considers the TCP connection invalid and automatically releases the TCP connection.

1.9.2 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Enable the TCP keepalive function.

ip tcp keepalive [**interval** *interval*] [**times** *times*] [**idle-period** *time*]

The TCP keepalive function is disabled by default.

This command is no different to the server and client and applies to all TCP connections.

Note

This command applies to both IPv4 TCP and IPv6 TCP.

1.10 Monitoring

Run the **show** command to check the running status of a configured function to verify the configuration effect.

Run the **debug** command to output debugging information.

Caution

The output debugging information occupies system resources. Therefore, disable the debugging function immediately after use.

Table 1-1 TCP Monitoring

Command	Purpose
show tcp connect [**local-ip** *ip-address*] [**local-port** *port-number*] [**peer-ip** *ip-address*] [**peer-port** *port-number*]	Displays the basic information about an IPv4 TCP connection.
show tcp connect statistics	Displays the statistics on an IPv4 TCP connection.
show tcp pmtu [**local-ip** *ip-address*] [**local-port** *port-number*] [**peer-ip** *ip-address*] [**peer-port** *port-number*]	Displays the path MTU information of IPv4 TCP.
show tcp port [*port-number*]	Displays the usage of IPv4 TCP port.
show tcp parameter	Displays IPv4 TCP parameters.
show tcp statistics	Displays IPv4 TCP statistics.
debug ip tcp packet [**in** | **out**] [**local-ip** *ip-address*] [**peer-ip** *ip-address*] [**global** | **vrf** *vrf-name*] [**local-port** *port-number*] [**peer-port** *port-number*] [	Debugs IPv4 TCP packets.

Command	**Purpose**
deeply]	
debug ip tcp transactions [**local-ip** *ip-address*] [**peer-ip** *ip-address*] [**local-port** *port-number*] [**peer-port** *port-number*]	Debugs IPv4 TCP connections.
show ipv6 tcp connect [**local-ipv6** *ipv6-address*] [**local-port** *port-number*] [**peer-ipv6** *ipv6-address*] [**peer-port** *port-number*]	Displays the basic information about an IPv6 TCP connection.
show ipv6 tcp connect statistics	Displays the statistics on an IPv6 TCP connection.
show ipv6 tcp pmtu [**local-ipv6** *ipv6-address*] [**local-port** *port-number*] [**peer-ipv6** *ipv6-address*] [**peer-port** *port-number*]	Displays the path MTU information of IPv6 TCP.
show ipv6 tcp port [*port-number*]	Displays the usage of IPv6 TCP port.
debug ipv6 tcp packet [**in** | **out**] [**local-ipv6** *ipv6-address*] [**peer-ipv6** *ipv6-address*] [**global** | **vrf** *vrf-name*] [**local-port** *port-number*] [**peer-port** *port-number*] [**deeply**]	Debugs IPv6 TCP packets.
debug ipv6 tcp transactions [**local-ipv6** *ipv6-address*] [**peer-ipv6** *ipv6-address*] [**local-port** *port-number*] [**peer-port** *port-number*]	Debugs IPv6 TCP connections.