



Connection Settings

This chapter describes how to configure connection settings for connections that go through the ASA, or for management connections, that go to the ASA. Connection settings include:

- Maximum connections (TCP and UDP connections, embryonic connections, per-client connections)
- Connection timeouts
- Dead connection detection
- TCP sequence randomization
- TCP normalization customization
- TCP state bypass
- Global timeouts

This chapter includes the following sections:

- [Information About Connection Settings, page 18-1](#)
- [Licensing Requirements for Connection Settings, page 18-4](#)
- [Guidelines and Limitations, page 18-5](#)
- [Default Settings, page 18-6](#)
- [Configuring Connection Settings, page 18-6](#)
- [Monitoring Connection Settings, page 18-15](#)
- [Configuration Examples for Connection Settings, page 18-15](#)
- [Feature History for Connection Settings, page 18-17](#)

Information About Connection Settings

This section describes why you might want to limit connections and includes the following topics:

- [TCP Intercept and Limiting Embryonic Connections, page 18-2](#)
- [Disabling TCP Intercept for Management Packets for Clientless SSL Compatibility, page 18-2](#)
- [Dead Connection Detection \(DCD\), page 18-2](#)
- [TCP Sequence Randomization, page 18-3](#)
- [TCP Normalization, page 18-3](#)
- [TCP State Bypass, page 18-3](#)

TCP Intercept and Limiting Embryonic Connections

Limiting the number of embryonic connections protects you from a DoS attack. The ASA uses the per-client limits and the embryonic connection limit to trigger TCP Intercept, which protects inside systems from a DoS attack perpetrated by flooding an interface with TCP SYN packets. An embryonic connection is a connection request that has not finished the necessary handshake between source and destination. TCP Intercept uses the SYN cookies algorithm to prevent TCP SYN-flooding attacks. A SYN-flooding attack consists of a series of SYN packets usually originating from spoofed IP addresses. The constant flood of SYN packets keeps the server SYN queue full, which prevents it from servicing connection requests. When the embryonic connection threshold of a connection is crossed, the ASA acts as a proxy for the server and generates a SYN-ACK response to the client SYN request. When the ASA receives an ACK back from the client, it can then authenticate the client and allow the connection to the server.

**Note**

When you use TCP SYN cookie protection to protect servers from SYN attacks, you must set the embryonic connection limit lower than the TCP SYN backlog queue on the server that you want to protect. Otherwise, valid clients can no longer access the server during a SYN attack.

To view TCP Intercept statistics, including the top 10 servers under attack, see [Chapter 23, “Threat Detection.”](#)

Disabling TCP Intercept for Management Packets for Clientless SSL Compatibility

By default, TCP management connections have TCP Intercept always enabled. When TCP Intercept is enabled, it intercepts the 3-way TCP connection establishment handshake packets and thus deprives the ASA from processing the packets for clientless SSL. Clientless SSL requires the ability to process the 3-way handshake packets to provide selective ACK and other TCP options for clientless SSL connections. To disable TCP Intercept for management traffic, you can set the embryonic connection limit; only after the embryonic connection limit is reached is TCP Intercept enabled.

Dead Connection Detection (DCD)

DCD detects a dead connection and allows it to expire, without expiring connections that can still handle traffic. You configure DCD when you want idle, but valid connections to persist.

When you enable DCD, idle timeout behavior changes. With idle timeout, DCD probes are sent to each of the two end-hosts to determine the validity of the connection. If an end-host fails to respond after probes are sent at the configured intervals, the connection is freed, and reset values, if configured, are sent to each of the end-hosts. If both end-hosts respond that the connection is valid, the activity timeout is updated to the current time and the idle timeout is rescheduled accordingly.

Enabling DCD changes the behavior of idle-timeout handling in the TCP normalizer. DCD probing resets the idle timeout on the connections seen in the **show conn** command. To determine when a connection that has exceeded the configured timeout value in the timeout command but is kept alive due to DCD probing, the **show service-policy** command includes counters to show the amount of activity from DCD.

TCP Sequence Randomization

Each TCP connection has two ISNs: one generated by the client and one generated by the server. The ASA randomizes the ISN of the TCP SYN passing in both the inbound and outbound directions.

Randomizing the ISN of the protected host prevents an attacker from predicting the next ISN for a new connection and potentially hijacking the new session.

TCP initial sequence number randomization can be disabled if required. For example:

- If another in-line firewall is also randomizing the initial sequence numbers, there is no need for both firewalls to be performing this action, even though this action does not affect the traffic.
- If you use eBGP multi-hop through the ASA, and the eBGP peers are using MD5. Randomization breaks the MD5 checksum.
- You use a WAAS device that requires the ASA not to randomize the sequence numbers of connections.

TCP Normalization

The TCP normalization feature identifies abnormal packets that the ASA can act on when they are detected; for example, the ASA can allow, drop, or clear the packets. TCP normalization helps protect the ASA from attacks. TCP normalization is always enabled, but you can customize how some features behave.

The TCP normalizer includes non-configurable actions and configurable actions. Typically, non-configurable actions that drop or clear connections apply to packets that are always bad. Configurable actions (as detailed in [Customizing the TCP Normalizer with a TCP Map, page 18-6](#)) might need to be customized depending on your network needs.

See the following guidelines for TCP normalization:

- The normalizer does not protect from SYN floods. The ASA includes SYN flood protection in other ways.
- The normalizer always sees the SYN packet as the first packet in a flow unless the ASA is in loose mode due to failover.

TCP State Bypass

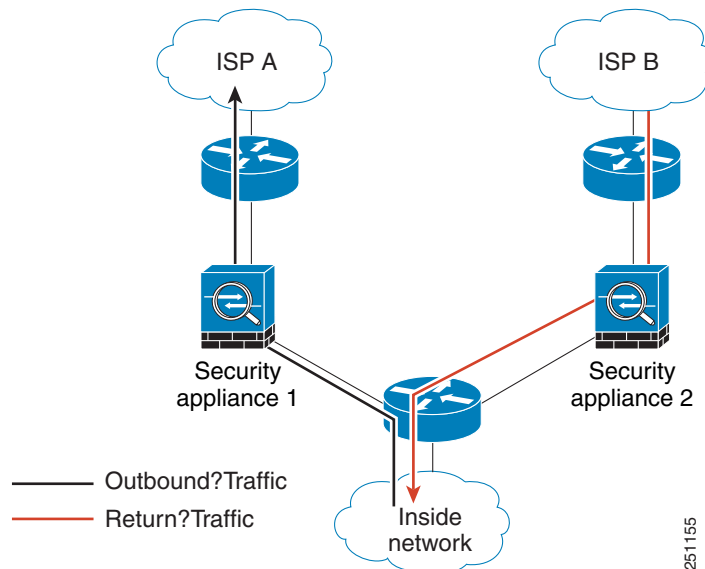
By default, all traffic that goes through the ASA is inspected using the Adaptive Security Algorithm and is either allowed through or dropped based on the security policy. The ASA maximizes the firewall performance by checking the state of each packet (is this a new connection or an established

connection?) and assigning it to either the session management path (a new connection SYN packet), the fast path (an established connection), or the control plane path (advanced inspection). See the general operations configuration guide for more detailed information about the stateful firewall.

TCP packets that match existing connections in the fast path can pass through the ASA without rechecking every aspect of the security policy. This feature maximizes performance. However, the method of establishing the session in the fast path using the SYN packet, and the checks that occur in the fast path (such as TCP sequence number), can stand in the way of asymmetrical routing solutions: both the outbound and inbound flow of a connection must pass through the same ASA.

For example, a new connection goes to ASA 1. The SYN packet goes through the session management path, and an entry for the connection is added to the fast path table. If subsequent packets of this connection go through ASA 1, then the packets will match the entry in the fast path, and are passed through. But if subsequent packets go to ASA 2, where there was not a SYN packet that went through the session management path, then there is no entry in the fast path for the connection, and the packets are dropped. [Figure 18-1](#) shows an asymmetric routing example where the outbound traffic goes through a different ASA than the inbound traffic:

Figure 18-1 Asymmetric Routing



If you have asymmetric routing configured on upstream routers, and traffic alternates between two ASAs, then you can configure TCP state bypass for specific traffic. TCP state bypass alters the way sessions are established in the fast path and disables the fast path checks. This feature treats TCP traffic much as it treats a UDP connection: when a non-SYN packet matching the specified networks enters the ASA, and there is not an fast path entry, then the packet goes through the session management path to establish the connection in the fast path. Once in the fast path, the traffic bypasses the fast path checks.

Licensing Requirements for Connection Settings

Model	License Requirement
ASAv	Standard or Premium License.
All other models	Base License.

Guidelines and Limitations

Context Mode Guidelines

Supported in single and multiple context mode.

Firewall Mode Guidelines

Supported in routed and transparent mode.

Failover Guidelines

Failover is supported.

TCP State Bypass Unsupported Features

The following features are not supported when you use TCP state bypass:

- Application inspection—Application inspection requires both inbound and outbound traffic to go through the same ASA, so application inspection is not supported with TCP state bypass.
- AAA authenticated sessions—When a user authenticates with one ASA, traffic returning via the other ASA will be denied because the user did not authenticate with that ASA.
- TCP Intercept, maximum embryonic connection limit, TCP sequence number randomization—The ASA does not keep track of the state of the connection, so these features are not applied.
- TCP normalization—The TCP normalizer is disabled.
- SSM and SSC functionality—You cannot use TCP state bypass and any application running on an SSM or SSC, such as IPS or CSC.

TCP State Bypass NAT Guidelines

Because the translation session is established separately for each ASA, be sure to configure static NAT on both ASAs for TCP state bypass traffic; if you use dynamic NAT, the address chosen for the session on ASA 1 will differ from the address chosen for the session on ASA 2.

Maximum Concurrent and Embryonic Connection Guidelines

Depending on the number of CPU cores on your ASA model, the maximum concurrent and embryonic connections may exceed the configured numbers due to the way each core manages connections. In the worst case scenario, the ASA allows up to $n-1$ extra connections and embryonic connections, where n is the number of cores. For example, if your model has 4 cores, if you configure 6 concurrent connections and 4 embryonic connections, you could have an additional 3 of each type. To determine the number of cores for your model, enter the **show cpu core** command.

Default Settings

TCP State Bypass

TCP state bypass is disabled by default.

TCP Normalizer

The default configuration includes the following settings:

```
no check-retransmission
no checksum-verification
exceed-mss allow
queue-limit 0 timeout 4
reserved-bits allow
syn-data allow
synack-data drop
invalid-ack drop
seq-past-window drop
tcp-options range 6 7 clear
tcp-options range 9 255 clear
tcp-options selective-ack allow
tcp-options timestamp allow
tcp-options window-scale allow
ttl-evasion-protection
urgent-flag clear
window-variation allow-connection
```

Configuring Connection Settings

This section includes the following topics:

- [Customizing the TCP Normalizer with a TCP Map, page 18-6](#)
- [Configuring Connection Settings, page 18-11](#)

Task Flow For Configuring Connection Settings

-
- | | |
|---------------|--|
| Step 1 | For TCP normalization customization, create a TCP map according to the Customizing the TCP Normalizer with a TCP Map, page 18-6 . |
| Step 2 | For all connection settings, configure a service policy according to Chapter 1, “Service Policy Using the Modular Policy Framework.” |
| Step 3 | Configure connection settings according to the Configuring Connection Settings, page 18-11 . |
-

Customizing the TCP Normalizer with a TCP Map

To customize the TCP normalizer, first define the settings using a TCP map.

Detailed Steps

- Step 1** To specify the TCP normalization criteria that you want to look for, create a TCP map by entering the following command:

```
hostname(config)# tcp-map tcp-map-name
```

For each TCP map, you can customize one or more settings.

- Step 2** (Optional) Configure the TCP map criteria by entering one or more of the following commands (see [Table 18-1](#)). If you want to customize some settings, then the defaults are used for any commands you do not enter.

Table 18-1 tcp-map Commands

Command	Notes
check-retransmission	Prevents inconsistent TCP retransmissions.
checksum-verification	Verifies the checksum.
exceed-mss {allow drop}	<p>Sets the action for packets whose data length exceeds the TCP maximum segment size.</p> <p>(Default) The allow keyword allows packets whose data length exceeds the TCP maximum segment size.</p> <p>The drop keyword drops packets whose data length exceeds the TCP maximum segment size.</p>
invalid-ack {allow drop}	<p>Sets the action for packets with an invalid ACK. You might see invalid ACKs in the following instances:</p> <ul style="list-style-type: none"> • In the TCP connection SYN-ACK-received status, if the ACK number of a received TCP packet is not exactly same as the sequence number of the next TCP packet sending out, it is an invalid ACK. • Whenever the ACK number of a received TCP packet is greater than the sequence number of the next TCP packet sending out, it is an invalid ACK. <p>The allow keyword allows packets with an invalid ACK.</p> <p>(Default) The drop keyword drops packets with an invalid ACK.</p> <p>Note TCP packets with an invalid ACK are automatically allowed for WAAS connections.</p>

Table 18-1 *tcp-map Commands (continued)*

Command	Notes
queue-limit <i>pkt_num</i> [timeout <i>seconds</i>]	<p>Sets the maximum number of out-of-order packets that can be buffered and put in order for a TCP connection, between 1 and 250 packets. The default is 0, which means this setting is disabled and the default system queue limit is used depending on the type of traffic:</p> <ul style="list-style-type: none"> • Connections for application inspection (the inspect command), IPS (the ips command), and TCP check-retransmission (the TCP map check-retransmission command) have a queue limit of 3 packets. If the ASA receives a TCP packet with a different window size, then the queue limit is dynamically changed to match the advertised setting. • For other TCP connections, out-of-order packets are passed through untouched. <p>If you set the queue-limit command to be 1 or above, then the number of out-of-order packets allowed for all TCP traffic matches this setting. For example, for application inspection, IPS, and TCP check-retransmission traffic, any advertised settings from TCP packets are ignored in favor of the queue-limit setting. For other TCP traffic, out-of-order packets are now buffered and put in order instead of passed through untouched.</p> <p>The timeout <i>seconds</i> argument sets the maximum amount of time that out-of-order packets can remain in the buffer, between 1 and 20 seconds; if they are not put in order and passed on within the timeout period, then they are dropped. The default is 4 seconds. You cannot change the timeout for any traffic if the <i>pkt_num</i> argument is set to 0; you need to set the limit to be 1 or above for the timeout keyword to take effect.</p>
reserved-bits { allow clear drop }	<p>Sets the action for reserved bits in the TCP header.</p> <p>(Default) The allow keyword allows packets with the reserved bits in the TCP header.</p> <p>The clear keyword clears the reserved bits in the TCP header and allows the packet.</p> <p>The drop keyword drops the packet with the reserved bits in the TCP header.</p>
seq-past-window { allow drop }	<p>Sets the action for packets that have past-window sequence numbers, namely the sequence number of a received TCP packet is greater than the right edge of the TCP receiving window.</p> <p>The allow keyword allows packets that have past-window sequence numbers. This action is only allowed if the queue-limit command is set to 0 (disabled).</p> <p>(Default) The drop keyword drops packets that have past-window sequence numbers.</p>

Table 18-1 tcp-map Commands (continued)

Command	Notes
synack-data { allow drop }	Sets the action for TCP SYNACK packets that contain data. The allow keyword allows TCP SYNACK packets that contain data. (Default) The drop keyword drops TCP SYNACK packets that contain data.
syn-data { allow drop }	Sets the action for SYN packets with data. (Default) The allow keyword allows SYN packets with data. The drop keyword drops SYN packets with data.
tcp-options { selective-ack timestamp window-scale } { allow clear } Or tcp-options range <i>lower upper</i> { allow clear drop }	Sets the action for packets with TCP options, including the selective-ack, timestamp, or window-scale TCP options. (Default) The allow keyword allows packets with the specified option. (Default for range) The clear keyword clears the option and allows the packet. The drop keyword drops the packet with the specified option. The selective-ack keyword sets the action for the SACK option. The timestamp keyword sets the action for the timestamp option. Clearing the timestamp option disables PAWS and RTT. The window-scale keyword sets the action for the window scale mechanism option. The range keyword specifies a range of options. The <i>lower</i> argument sets the lower end of the range as 6, 7, or 9 through 255. The <i>upper</i> argument sets the upper end of the range as 6, 7, or 9 through 255.
ttl-evasion-protection	Disables the TTL evasion protection. Do not enter this command if you want to prevent attacks that attempt to evade security policy. For example, an attacker can send a packet that passes policy with a very short TTL. When the TTL goes to zero, a router between the ASA and the endpoint drops the packet. It is at this point that the attacker can send a malicious packet with a long TTL that appears to the ASA to be a retransmission and is passed. To the endpoint host, however, it is the first packet that has been received by the attacker. In this case, an attacker is able to succeed without security preventing the attack.

Table 18-1 tcp-map Commands (continued)

Command	Notes
urgent-flag { allow clear }	<p>Sets the action for packets with the URG flag. The URG flag is used to indicate that the packet contains information that is of higher priority than other data within the stream. The TCP RFC is vague about the exact interpretation of the URG flag, therefore end systems handle urgent offsets in different ways, which may make the end system vulnerable to attacks.</p> <p>The allow keyword allows packets with the URG flag.</p> <p>(Default) The clear keyword clears the URG flag and allows the packet.</p>
window-variation { allow drop }	<p>Sets the action for a connection that has changed its window size unexpectedly. The window size mechanism allows TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted too much data. From the TCP specification, “shrinking the window” is strongly discouraged. When this condition is detected, the connection can be dropped.</p> <p>(Default) The allow keyword allows connections with a window variation.</p> <p>The drop keyword drops connections with a window variation.</p>


Configuring Connection Settings


To set connection settings, perform the following steps.

Detailed Steps

	Command	Purpose
Step 1	class-map <i>name</i>	Creates a class map to identify the traffic for which you want to disable stateful firewall inspection.
	Example: hostname(config)# class-map bypass_traffic	
Step 2	match <i>parameter</i>	Specifies the traffic in the class map. See Identifying Traffic (Layer 3/4 Class Maps) , page 1-12 for more information.
	Example: hostname(config-cmap)# match access-list bypass	

	Command	Purpose
Step 3	<code>policy-map name</code> Example: hostname(config)# policy-map tcp_bypass_policy	Adds or edits a policy map that sets the actions to take with the class map traffic.
Step 4	<code>class name</code> Example: hostname(config-pmap)# class bypass_traffic	Identifies the class map created in Step 1
Step 5	Do one or more of the following:	

Command	Purpose
<pre>set connection {[conn-max n] [embryonic-conn-max n] [per-client-embryonic-max n] [per-client-max n] [random-sequence-number {enable disable}]}</pre> <p>Example:</p> <pre>hostname(config-pmap-c)# set connection conn-max 256 random-sequence-number disable</pre>	<p>Sets maximum connection limits or whether TCP sequence randomization is enabled.</p> <p>The conn-max <i>n</i> argument sets the maximum number of simultaneous TCP and/or UDP connections that are allowed, between 0 and 2000000. The default is 0, which allows unlimited connections.</p> <p>If two servers are configured to allow simultaneous TCP and/or UDP connections, the connection limit is applied to each configured server separately.</p> <p>When configured under a class, this argument restricts the maximum number of simultaneous connections that are allowed for the entire class. In this case, one attack host can consume all the connections and leave none of the rest of the hosts matched in the ACL under the class.</p> <p>The embryonic-conn-max <i>n</i> argument sets the maximum number of simultaneous embryonic connections allowed, between 0 and 2000000. The default is 0, which allows unlimited connections.</p> <p>The per-client-embryonic-max <i>n</i> argument sets the maximum number of simultaneous embryonic connections allowed per client, between 0 and 2000000. The default is 0, which allows unlimited connections.</p> <p>The per-client-max <i>n</i> argument sets the maximum number of simultaneous connections allowed per client, between 0 and 2000000. The default is 0, which allows unlimited connections. When configured under a class, this argument restricts the maximum number of simultaneous connections that are allowed for each host that is matched through an ACL under the class.</p> <p>The random-sequence-number {enable disable} keyword enables or disables TCP sequence number randomization. See TCP Sequence Randomization, page 18-3 section for more information.</p> <p>You can enter this command all on one line (in any order), or you can enter each attribute as a separate command. The ASA combines the command into one line in the running configuration.</p> <p> Note For management traffic, you can only set the conn-max and embryonic-conn-max keywords.</p>

Command	Purpose
<pre>set connection timeout {[embryonic hh:mm:ss] {idle hh:mm:ss [reset]] [half-closed hh:mm:ss] [dcd hh:mm:ss [max_retries]]}</pre> <p>Example:</p> <pre>hostname(config-pmap-c)# set connection timeout idle 2:0:0 embryonic 0:40:0 half-closed 0:20:0 dcd</pre>	<p>Sets connection timeouts. For global timeouts, see the timeout command in the command reference.</p> <p>The embryonic <i>hh:mm:ss</i> keyword sets the timeout period until a TCP embryonic (half-open) connection is closed, between 0:0:5 and 1193:00:00. The default is 0:0:30. You can also set this value to 0, which means the connection never times out.</p> <p>The idle <i>hh:mm:ss</i> keyword sets the idle timeout period after which an established connection of any protocol closes, between 0:0:1 and 1193:0:0. The default is 1:0:0. You can also set this value to 0, which means the connection never times out. For TCP traffic, the reset keyword sends a reset to TCP endpoints when the connection times out.</p> <p>The half-closed <i>hh:mm:ss</i> keyword sets the idle timeout period until a half-closed connection is closed, between 0:5:0 (for 9.1(1) and earlier) or 0:0:30 (for 9.1(2) and later) and 1193:0:0. The default is 0:10:0. Half-closed connections are not affected by DCD. Also, the ASA does not send a reset when taking down half-closed connections.</p> <p>The dcd keyword enables DCD. DCD detects a dead connection and allows it to expire, without expiring connections that can still handle traffic. You configure DCD when you want idle, but valid connections to persist. After a TCP connection times out, the ASA sends DCD probes to the end hosts to determine the validity of the connection. If one of the end hosts fails to respond after the maximum retries are exhausted, the ASA frees the connection. If both end hosts respond that the connection is valid, the ASA updates the activity timeout to the current time and reschedules the idle timeout accordingly. The <i>retry-interval</i> sets the time duration in <i>hh:mm:ss</i> format to wait after each unresponsive DCD probe before sending another probe, between 0:0:1 and 24:0:0. The default is 0:0:15. The <i>max-retries</i> sets the number of consecutive failed retries for DCD before declaring the connection as dead. The minimum value is 1 and the maximum value is 255. The default is 5.</p> <p>The default tcp idle timeout is 1 hour.</p> <p>The default udp idle timeout is 2 minutes.</p> <p>The default icmp idle timeout is 2 seconds.</p> <p>The default esp and ha idle timeout is 30 seconds.</p> <p>For all other protocols, the default idle timeout is 2 minutes.</p> <p>To never time out, enter 0:0:0.</p> <p>You can enter this command all on one line (in any order), or you can enter each attribute as a separate command. The command is combined onto one line in the running configuration.</p> <p> Note This command is not available for management traffic.</p>

Command	Purpose
<pre>set connection advanced-options tcp-map-name</pre> <p>Example:</p> <pre>hostname(config-pmap-c)# set connection advanced-options tcp_map1</pre>	Customizes the TCP normalizer. See Customizing the TCP Normalizer with a TCP Map, page 18-6 to create a TCP map.
<pre>set connection advanced-options tcp-state-bypass</pre> <p>Example:</p> <pre>hostname(config-pmap-c)# set connection advanced-options tcp-state-bypass</pre>	Enables TCP state bypass.
<p>Step 6</p> <pre>service-policy policymap_name {global interface interface_name}</pre> <p>Example:</p> <pre>hostname(config)# service-policy tcp_bypass_policy outside</pre>	Activates the policy map on one or more interfaces. global applies the policy map to all interfaces, and interface applies the policy to one interface. Only one global policy is allowed. You can override the global policy on an interface by applying a service policy to that interface. You can only apply one policy map to each interface.

Monitoring Connection Settings

To monitor TCP state bypass, perform one of the following tasks:

Command	Purpose
<code>show conn</code>	If you use the <code>show conn</code> command, the display for connections that use TCP state bypass includes the flag “b.”

Configuration Examples for Connection Settings

This section includes the following topics:

- [Configuration Examples for Connection Limits and Timeouts, page 18-15](#)
- [Configuration Examples for TCP State Bypass, page 18-16](#)
- [Configuration Examples for TCP Normalization, page 18-16](#)

Configuration Examples for Connection Limits and Timeouts

The following example sets the connection limits and timeouts for all traffic:

```
hostname(config)# class-map CONNS
hostname(config-cmap)# match any
hostname(config-cmap)# policy-map CONNS
hostname(config-pmap)# class CONNS
```

```
hostname(config-pmap-c)# set connection conn-max 1000 embryonic-conn-max 3000
hostname(config-pmap-c)# set connection timeout idle 2:0:0 embryonic 0:40:0 half-closed
0:20:0 dcd
hostname(config-pmap-c)# service-policy CONNS interface outside
```

You can enter **set connection** commands with multiple parameters or you can enter each parameter as a separate command. The ASA combines the commands into one line in the running configuration. For example, if you entered the following two commands in class configuration mode:

```
hostname(config-pmap-c)# set connection conn-max 600
hostname(config-pmap-c)# set connection embryonic-conn-max 50
```

the output of the **show running-config policy-map** command would display the result of the two commands in a single, combined command:

```
set connection conn-max 600 embryonic-conn-max 50
```

Configuration Examples for TCP State Bypass

The following is a sample configuration for TCP state bypass:

```
hostname(config)# access-list tcp_bypass extended permit tcp 10.1.1.0 255.255.255.224 any

hostname(config)# class-map tcp_bypass
hostname(config-cmap)# description "TCP traffic that bypasses stateful firewall"
hostname(config-cmap)# match access-list tcp_bypass

hostname(config-cmap)# policy-map tcp_bypass_policy
hostname(config-pmap)# class tcp_bypass
hostname(config-pmap-c)# set connection advanced-options tcp-state-bypass

hostname(config-pmap-c)# service-policy tcp_bypass_policy outside

hostname(config-pmap-c)# static (inside,outside) 209.165.200.224 10.1.1.0 netmask
255.255.255.224
```

Configuration Examples for TCP Normalization

For example, to allow urgent flag and urgent offset packets for all traffic sent to the range of TCP ports between the well known FTP data port and the Telnet port, enter the following commands:

```
hostname(config)# tcp-map tmap
hostname(config-tcp-map)# urgent-flag allow
hostname(config-tcp-map)# class-map urg-class
hostname(config-cmap)# match port tcp range ftp-data telnet
hostname(config-cmap)# policy-map pmap
hostname(config-pmap)# class urg-class
hostname(config-pmap-c)# set connection advanced-options tmap
hostname(config-pmap-c)# service-policy pmap global
```


Feature History for Connection Settings

Table 18-2 lists each feature change and the platform release in which it was implemented.

Table 18-2 Feature History for Connection Settings

Feature Name	Platform Releases	Feature Information
TCP state bypass	8.2(1)	This feature was introduced. The following command was introduced: set connection advanced-options tcp-state-bypass .
Connection timeout for all protocols	8.2(2)	The idle timeout was changed to apply to all protocols, not just TCP. The following command was modified: set connection timeout
Timeout for connections using a backup static route	8.2(5)/8.4(2)	When multiple static routes exist to a network with different metrics, the ASA uses the one with the best metric at the time of connection creation. If a better route becomes available, then this timeout lets connections be closed so a connection can be reestablished to use the better route. The default is 0 (the connection never times out). To take advantage of this feature, change the timeout to a new value. We modified the following command: timeout floating-conn .
Configurable timeout for PAT xlate	8.4(3)	When a PAT xlate times out (by default after 30 seconds), and the ASA reuses the port for a new translation, some upstream routers might reject the new connection because the previous connection might still be open on the upstream device. The PAT xlate timeout is now configurable, to a value between 30 seconds and 5 minutes. We introduced the following command: timeout pat-xlate . <i>This feature is not available in 8.5(1) or 8.6(1).</i>

Table 18-2 Feature History for Connection Settings (continued)

Feature Name	Platform Releases	Feature Information
Increased maximum connection limits for service policy rules	9.0(1)	<p>The maximum number of connections for service policy rules was increased from 65535 to 2000000.</p> <p>We modified the following commands: set connection conn-max, set connection embryonic-conn-max, set connection per-client-embryonic-max, set connection per-client-max.</p>
Decreased the half-closed timeout minimum value to 30 seconds	9.1(2)	<p>The half-closed timeout minimum value for both the global timeout and connection timeout was lowered from 5 minutes to 30 seconds to provide better DoS protection.</p> <p>We modified the following commands: set connection timeout half-closed, timeout half-closed.</p>