TCPM WG
Internet Draft
Intended status: Experimental
Expires: January 2019

J. Touch

July 19, 2018

The TCP Service Number Option (SNO)
draft-touch-tcpm-sno-09.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on July 19, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This document specifies a TCP option for service numbers. The current SYN destination port is used both to indicate the desired service and as a connection demultiplexing field. This option separates those two functions, retaining the current destination port solely for demultiplexing and indicating the service separately in a service number option (SNO). By decoupling these two functions, SNO allows a larger number of concurrent connections for a single service, as might be useful between fixed addresses of proxies.

Table of Contents

1. Introduction

TCP connections are defined by a socket pair, where each TCP socket consists of an IP address and a port number. The IP addresses indicate the network endpoints (hosts) of the connection, and the port numbers allow a pair of IP endpoints to have more than one concurrent connection. TCP connections begin when an application on one host sends a SYN segment to a waiting application on the other host, determined by the destination port in that segment.

Port numbers thus serve two distinct purposes. For the entirety of a connection, they help differentiate concurrent connections as part of the socket pair, and are thus used for demultiplexing within a host. For the SYN, the destination port also indicates the waiting application, i.e., the service for that connection, acting as a service identifier.

Service identifiers need to be coordinated between the endpoints of a connection, but need not be coordinated with any other component of the network. To avoid the need for explicit pairwise coordination, most Internet transport protocols currently use globally-assigned destination port numbers as service identifiers; this includes TCP, UDP, SCTP, and DCCP [RFC768] [RFC793] [RFC4960] [RFC4340]. An assigned port number can be requested from the Internet Assigned Numbers Authority (IANA) [IANA].

The use of SYN destination ports as both service identifier and demultiplexing identifier can impact TCP performance. For a given service, a given pair of endpoints can have at most 2^16 concurrent connections, or even connections in the TIME-WAIT state (which is typically expected to last two minutes) [To1999]. This limits services to at most an average of 550 connections per second, which can be a constraint on proxy-to-proxy services.

To reduce this impact, this document specifies the TCP service number option (SNO), which allows services to be specified in an option separate from the current header destination port field. SNO

decouples the use of ports for connection demultiplexing and state management from their use to indicate a desired endpoint service. This decoupling can substantially increase the number of concurrent connections to 2^32; even considering current expected TIME-WAIT delay, that can support up to 35.8M connections per second.

Although it changes TCP SYNs, it does not otherwise affect the processing of other TCP segments or the TCP state machine. SNO must be implemented at both ends of a TCP connection to be effective.

2. Conventions used in this document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Background

TCP supports multiplexing as one of its six core facilities, allowing a single pair of hosts to have multiple concurrent TCP sessions (see Sec. 1.5 of [RFC793]). An endpoint address is associated with a port number, forming a socket; and "A pair of sockets uniquely identifies each connection." Although ports can be bound to services uniquely at each endpoint, RFC 793 notes that it is useful to attach frequently-used services to fixed ports which are publicly known, but that other services may be discovered by dynamic means. This document addresses one impact of that suggestion, and specifies an alternative which alleviates that impact.

The Internet currently relies on the use of fixed, publicly-agreed port numbers for most services, whether for public access (e.g., HTTP, FTP, DNS) or between pre-arranged pairs (e.g., X11, SSL). Some of these services use one public port to negotiate other ports for further exchanges (e.g., FTP, H.323, RPC).

There are several current methods for determining the port for a public service:

- Index the service in IANA's port registry
- Index the service in the host's DNS SRV records
- Ask the host directly using an RPC portmapper/bind-like service
- Ask the host for a hand-off using the TCPMUX port (port #1)

Many of these alternatives, including the use of strings as service identifiers, were described in principle in RFC 814, and have evolved into deployed capabilities [RFC814]. Each of these alternatives is summarized below, and each either significantly limits the number of concurrent connections for a service or incurs additional latency or management overhead compared to SNO.

3.1. IANA port numbers

The Internet Assigned Numbers Authority currently manages globally reserved port numbers [RFC6335]. The desired port number for a service is determined either by an operating system index to a copy of IANA's table (e.g., getportbyname() in Unix, which indexes the /etc/services file), or is fixed in inside the application.

The port number space 0..65536 is split into three ranges [RFC2780]:

- 0..1023 "well-known", also called "system" ports
- 1024..49151 "registered", also called "user" ports
- 49152..65535 "dynamic", also called "private" ports

The terms "well-known" and "registered" are misnomers; both of those port ranges are managed by IANA, and are equally registered and well-known; they are currently known together as "assigned" [RFC6335]. System ports are intended for services that run in privileged mode, sometimes known as "root", although that distinction is blurred in current operating systems.

IANA-managed ports are allocated globally, for all hosts everywhere on the public Internet, even though the meaning of a port need be known only for a particular host. A given service is typically assigned a single port, which then limits the number of concurrent connections between two hosts to 2^16, i.e., the number expressible in the source port field. This assumes that the source port can be

arbitrary; in may implementations, when a service is bound to a SYN destination port it is prohibited for use in other connections, e.g., as a source port for outgoing SYNs.

3.2. DNS SRV records

DNS SRV resource records provide a way to find the port number for a service based on its string name [RFC2782]. A host asks the DNS to index "_servicename._tcp.hostname" (underscores required) and the response is a record that includes both the port number and host's IP address.

SRV records allow port numbers to be allocated on a per-host basis, and allow multiple ports to be indicated for a given service. A system that wants to support a large number of concurrent HTTP connections could advertise the HTTP service as available on the entire unassigned (dynamic) port range, in addition to port 80. This can increase the number of concurrent connections to 2^30 (2^14 dynamic ports and 2^16 source ports), which would be nearly as good as SNO (2^32).

However, SRV lookups require an additional protocol exchange for each first-time access, which can traverse much of the same path the TCP SYN will, i.e., incurring an additional round-trip time of delay (because DNS servers are often located near the hosts they serve). Further, using SRV records requires that the dynamic ports be allocated in advance, and they cannot be reclaimed once advertised. SRV advertisement may be useful for a single known service, but does not support a larger number of connections for any (or every) service on-demand.

Additional challenges for DNS SRV records are autonomy, robustness, and size of the name space. Many hosts do not have control over their DNS entries; moving port lookup into the DNS could limit the services that a host can deploy for public access. This solution also makes the DNS a required part of the Internet architecture, even for accessing services on hosts with well-known IP addresses (e.g., the DNS itself). This decreases network robustness, because access of services on a host depends on access to the DNS.

3.3. RPC portmapper and RPCBIND

An alternative to indexing the service name at a separate host via the DNS would be to contact the intended host directly and request the lookup there. This is how the RPC portmapper (v2) and RPCBIND (v3 and v4) services work, where the source host contacts the destination on port #111 [RFC1833][RFC5531]. This service was

designed for the same basic reason as the TCP port option of this document: to allow a small subset of a potentially large set of services to be dynamically bound to a small number of ports. The differences between portmapper and RPCBIND are not important here, so they are discussed as a single example.

In both portmapper and RPCBIND the source host contacts the destination host on port 111, and issues a request including the desired destination RPC service name. A response indicates the appropriate port for that RPC service.

Like the DNS SRV solution, portmapper/RPCBIND requires a separate round-trip (one for UDP; more for TCP) to perform the lookup operation. This adds to both the communication overhead and connection establishment latency.

The portmapper service also allows services to be selected on version, i.e., to have different versions of a service on different ports, accessed using the same version name but a different version number. This is handled in some IANA entries, DNS SRV records, and TCPMUX by using a port keyword that embeds the version number in the name, e.g., 'imap' vs. 'imap3'. In most other cases, versioning is indicated and negotiated in-band, inside the protocol (e.g., HTTP).

Unfortunately, portmapper has the same limitation as DNS SRV records; once a port is advertised for a given service, it cannot be reclaimed for use by another service. Further, once a given service is advertised, it is likely that the requesting host will cache the response. As a result, dynamic ports can be used to extend the port space for a given service in advance, but they need to be pinned to that service when it is first requested from that host. Again, this limits the ability to flexibly support large numbers of connections for any (or every) service.

3.4. TCPMUX

TCPMUX is a service on TCP port #1 which allows a host to provide a port name handoff service for itself [RFC1078]. A source host opens a connection to port 1 on a destination host and transmits 'portname<CR><LF>' in the data stream; the destination replies with either '+<CR><LF>' (yes, the service is available) or '-<CR><LF>' (no, the service is not available). If the service is available, the connection is transferred to the desired service while still in the OPEN state.

TCPMUX modifies the semantics of TCP connection establishment; its connections always succeed, and upon receipt of the named service

the application must determine whether to proceed or not. This document seeks a more conventional TCP semantics, where unavailable services result in a rejected connection (e.g., RST in reply and/or ICMP error message).

TCPMUX further requires all new connections to be received on a single port; this again limits the number of connections between two machines to 2^16, which provides no benefit compared to existing assigned ports as currently used in SYN segments.

3.5. Summary of alternatives and comparison to SNO

Each of the alternatives presented has a significant limitation. These alternatives are summarized as follows:

- IANA ports: limits a given service to 2^16 concurrent connections between two IP addresses; fewer if system/user/dynamic boundaries are preserved

- DNS SRV records: requires an extra round-trip exchange for lookup, not typically under host control, allows up to 2^30 concurrent connections but requires that the additional space of 2^14 be allocated to services on a given host in advance.

- Portmapper: requires an extra round-trip exchange for lookup, allows up to 2^30 concurrent connections but requires that the additional space of 2^14 be allocated to services on a given host in advance

- TCPMUX: destroys semantics of TCP connection establishment, limits connections per endpoint pair to 2^16 over all services

SNO allows the destination host to associate services with processes on a per-connection basis, while avoiding unnecessary additional round-trips or connections and also while reducing message overhead. This enables every service to support up to 2^32 concurrent connections, by decoupling the demultiplexing and service identifier role of SYN destination ports.

The basic operation of SNO is as follows:

- The source host issues a SYN, picking both source and destination port numbers arbitrarily that are not currently in use (active or pending connection).

- The SYN includes SNO, which indicates the IANA assigned port number of the desired service.

- The destination host, upon receiving the SYN with SNO, determines whether the service indicated in the option is running. If so, a SYN-ACK is issued with a zero-length SNO, indicating success of the lookup and handoff. The service is bound to that connection at the destination.

- If the service is not available, the appropriate RST and/or ICMP error messages are returned.

The benefits to TCP SNO are that:

- For a given service, the number of connections between two given IP addresses is no longer limited to 2^16; it is expanded to 2^32.

- SNO support is provided at the same host as the intended service, so the fate of both is shared (i.e., it is more robust than decoupled service such as DNS SRV).

- SNO is embedded in the TCP SYN segment, avoiding extra round trips and messages.

- NAT traversal is preserved.

- TCP connection semantics are maintained, i.e., services not available never connect.

4. TCP Service Number Option

The TCP service number option (SNO) extends the TCP header to include a 16-bit port field indicating desired service, as shown in Figure 1.

>> New implementations of TCP MAY implement SNO.

>> SNO SHOULD NOT appear in any TCP segment except SYN and SYN-ACK. SNO MUST be silently ignored if in any segments except SYN and SYN-ACK.

SNO includes the mandatory KIND and LENGTH fields [RFC793], as well as the desired service port number. The current specification uses the TCP Experimental Option format, with an ExID of 0x5323 in network-standard byte order (ASCII for "S#") [RFC6994].

The KIND is a single octet (byte) which indicates this is an experimental option; SNO is supported on both experimental options (253 and 254); there is no difference as to which experimental

option is used. The LENGTH is a single octet (byte) interpreted as an unsigned number that indicates the length of this option in octets (bytes), including the KIND and LENGTH fields, as well as the octets of the Service-Number.

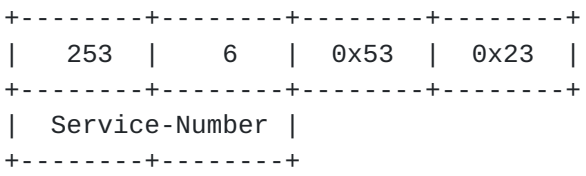

Figure 1 TCP SNO SYN option format

Upon receipt of a TCP SYN segment including SNO ('TCP SYN/SNO'), the Service-Number is matched against a list of available services. Available services are those that listen on the indicated port number. E.g., a web server that listens for incoming connections on port 80 will respond to connections with SYN segments with SNO=80.

The way in which SNO and TCP destination port numbers interacts is described in Section 4.1. When an incoming TCP SYN/SNO is considered valid, the connection is completed by returning a SYN-ACK with a null SNO.

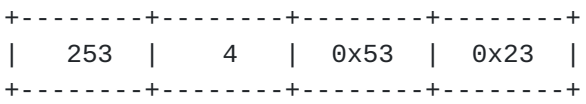

Figure 2 TCP Null SNO format, as used in SYN-ACK

>> A TCP SYN/SNO answered with a TCP SYN with a non-null SNO (LENGTH > 2) or lacking the SNO option MUST cause the initiator to abort the connection via issuing a RST and by reporting an error to the application as if the port were not available.

The TCB for that connection is then associated with the process for the matching service, which then handles all further interactions with the connection.

4.1. Interaction between SNO and the TCP API

TCP currently uses TCP port numbers to demultiplex connections as well as to indicate the desired service at the destination. SNO retains the demultiplexing capability, but overrides service identification.

TCP specifies port numbers for connections in the OPEN command. The current OPEN command is described in RFC 793 Sections 2.7 and 3.8 as:

```
OPEN (local port, foreign socket, active/passive
   [, timeout] [, precedence] [, security/compartment]
   [, options])
   -> local connection name
```

The OPEN call is used to initiate connections, corresponding to Unix connect, and to wait for incoming connection requests, corresponding to Unix listen. The impact of the SNO option on each of these variants is described below.

4.1.1. Active OPEN (Unix connect)

During a TCP active OPEN command, SNO interprets the port number of foreign TCP socket as the SNO Service-Number and selects a random number as the foreign port. The OPEN command can be extended to override that random selection by extending the foreign socket to include both the service identifier and port number as separate fields.

4.1.2. Passive OPEN (Unix listen)

During a TCP passive OPEN command, SNO interprets the local port number as the SNO service identifier. The OPEN command can be extended to allow the listening application to also indicate a specific destination port by extending the local port to include both a service identifier and port number as separate fields.

4.1.3. Impact on the TCP OPEN API

Both active OPEN and passive OPEN may need to extend the current port numbers to include separate service identifiers. It may be useful to consider that only one service identifier is ever used, e.g., an active OPEN may need a separate foreign service identifier, and a passive OPEN may need a separate local service identifier, but separate service identifiers for both foreign and local would never occur. As a result, it may be more convenient to consider the TCP OPEN API as being extended with a single service field as follows:

```
SNOPEN (local port, foreign socket, service, active/passive
   [, timeout] [, precedence] [, security/compartment]
   [, options])
   -> local connection name
```

Legacy uses of the OPEN call can be trivially converted to the new SNOPEN description. A legacy active OPEN uses the port of the foreign socket as the service; a legacy passive OPEN uses the local port as the service.

However, because the most common use is to allow the active foreign port or passive local port "float" (be unspecified, and thus filled by the OS with an arbitrary value), most implementations will not need to modify the TCP OPEN API implementation, or can extend the API using a separate interface (e.g., Unix setsockopt).

4.2. Error conditions

There are two error conditions for a SYN segment with the SNO option to be considered:

- SNO not supported
- Invalid port (i.e., no application listening on that port)

The case where SNO is not supported is already addressed in TCP as an unknown option [RFC793. Implementations are expected to ignore it, which means the SYN-ACK would not include the SNO confirmation response.

>> For an invalid port, the receiving TCP should act as it would if the destination port were a service that is not available, i.e., it SHOULD return an ICMP port unreachable error message [RFC1122]. This message MUST include the received TCP header including the SNO option in its entirety. The destination TCP MUST also send a RST in response. Other interactions are the result of backward compatibility, and are discussed in Section 4.3.

4.3. Backward compatibility

The TCP SNO option is designed to interact correctly only on SNO-supporting implementations.

SNO connection attempts to non-SNO endpoints will be rejected; the SNO SYN will receive a non-SNO SYN-ACK, at which point the SNO endpoint will terminate the connection attempt.

Services on SNO endpoints will support both SNO and non-SNO incoming connections, without the need for recompilation or relinking.

>> Outgoing connections intended to be compatible with both implementations MUST either attempt both SNO and non-SNO connections in parallel or retry a failed SNO attempt with a non-SNO attempt.

5. Issues

The TCP SNO option interacts with some other IP and TCP services, notably security services. Variants of the option may be useful in other transport protocols. Also, there were a number of alternate designs considered which this document captures in summary.

5.1. Interaction with other protocols and features

TCP SNO potentially interacts with any other protocol that interprets or modifies TCP port numbers. This includes IPsec and other firewall systems, TCP/MD5 and other TCP security mechanisms, FTP and other in-band exchange of ports, and network address translators (NATs).

IPsec uses port numbers to perform access control in transport mode [RFC4301]. Security policies can define port-specific access control (PROTECT, BYPASS, DISCARD), as well as port-specific algorithms and keys. Similarly, firewall policies allow or block traffic based on port numbers.

Use of port numbers in IPsec selectors and firewalls may assume that the numbers correspond to well-known services. It is useful to note that there is no such requirement; any service may run on any port, subject to mutual agreement between the endpoint hosts. Use of SNO may interfere with this assumption both within IPsec and in other firewalling systems, but it does not add a new vulnerability. New implementations of IPsec and firewall systems may want to support interpreting SNO in these policy rules, but again should not rely on either port numbers to indicate a specific service.

TCP SNO occupies space in the TCP SYN segment. Such space is severely limited in cases where TCP-level security is present, as noted in detail in Section 5.

>> TCP SNO MUST be protected in the same way that the existing SYN destination port is protected.

For IPsec, this is not an issue because the entire TCP header and payload are protected by all IPsec modes. None of the TCP header is

protected by application-layer security, e.g., TLS, so again this is not an issue [RFC5246].

The resulting primary concern is TCP-level security, e.g., legacy TCP/MD5 and its successors TCP-AO [RFC2385][RFC5925]. TCP/MD5 always excludes TCP options in its hash calculation; this it fails to protect current critical TCP options such as alternate checksums, window scale, and timestamp options [RFC793] [RFC7323]. TCP-AO allows options to be included or excluded, depending on per-connection parameter. This document recommends, as per above, that SNO, as all options, be included in TCP-level protection. Note that it may be difficult to use SNO together with any of these TCP-layer protection mechanisms unless the TCP option space is extended, as with TCP EDO and/or EDO-SYN [To2018a][To2018b].

A number of protocols exchange port numbers in-band, notably to coordinate separate concurrent connections, e.g., FTP (file transfer) and SIP (teleconferencing) [RFC959][RFC3261]. Because these protocols coordinate the specific port numbers in advance, there is no need for SNO to indicate the desired service. As a result, it is unlikely that it would be useful to augment these protocols to support SNO in their creation of subordinate connections. SNO could still be useful in establishing the primary (first) connection for these services.

Network address and port translators, known collectively as NATs, not only read TCP ports, but may also translate them [RFC2993]. This interferes with the use of ports for service identification [RFC3234]. SNO may allow services to be identified behind NATs if NATs are not further extended to translate SNO. It is thus unknown whether SNO will help restore service identification in the presence of NATs.

TCP connections using SNO continue to use IP addresses and ports, although both port numbers are typically set arbitrarily. Translation of these ports should not interfere with the operation of NATs, though this has not been verified and is not a design requirement.

5.2. Potential use in other transport protocols

As noted earlier, SNO may be a useful addition to a variety of other transport protocols, such as UDP, SCTP and DCCP [RFC768] [RFC4960] [RFC4340]. Adding SNO support to SCTP and DCCP should be straightforward because both already have an option space. These are not addressed further in this document, because this focuses on TCP only.

DCCP already includes a Service Code that provides a similar way to separately identify services, but these codes are 32 bits and use a separate IANA registered space. DCCP does not use Service Codes as a way to expand the number of concurrent connections to a given IANA transport service.

UDP lacks options, so adding support for SNO is not feasible.

5.3. Discussion of alternative approaches

The current proposal assumes that the source TCP selects both source and destination port numbers randomly, that SNO occurs only in SYN and SYN-ACKs. A number of alternative approaches were considered during the development of the approach presented herein. These include:

- A portmapper-like service that returns a specific port number
- Continued demuxing based on SNO
- Dynamic overwriting of the destination port

The first approach, of returning a specific port number for a service, requires a separate round trip and messages to initiate a connection. We avoid both the additional time and messages in the proposed solution which integrates the lookup in the SYN.

Continued demultiplexing based on SNO would violate TCP connection semantics, which indicate that a connection be uniquely identified by the 4-tuple: <src addr><src port><dst addr><dst port>. Although SNO demuxing would increase the connection tuple space, this seems unnecessary as it is already over 2^32 concurrent connections between a single pair of host addresses. Finally, this variant incurs the SNO option overhead on every message, which seems unnecessarily inefficient. The proposed solution is more efficient and sufficiently increases the utility of the entire current connection name space.

Dynamic overwriting of the destination port complicates the connection establishment on the source side, because the SYN-ACK would have a different port pair than the SYN. It would further interfere with NAT traversal. The primary utility for overwriting the port number would be to facilitate demultiplexing at the receiver, but this is should already include the entire 4-tuple anyway. Overall, this variant seems unnecessarily complex for no real benefit.

5.4. Implementation Issues

Prototypes underway in both FreeBSD and Linux indicate substantial challenges with implementing SNO due to errors in option processing as well as optimizations that interfere with SNO's decoupling of service and connection identifiers.

Option processing has never been sufficiently described to ensure interoperable implementation. Both FreeBSD and Linux assume that TCP options can be processed at a single location in both incoming and outgoing TCP header processing, but this has never been true. In particular, options that determine whether a segment is valid (TCP MD5, TCP-AO, header checksum, etc.) must be processed before any other header fields are interpreted, whereas options that are interpreted in the context of header fields (e.g., SACK, etc.) must be interpreted afterwards.

Keeping track of TCP state can require multiple data structures on both endpoints, but these structures are currently optimized assuming that port numbers are overloaded as both service and connection identifiers. Connections can be in any of the following 11 states: CLOSED, LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT. CLOSED is fictional because no connection context exists. The remaining states are often grouped as follows:

- LISTEN - no connection state yet; tracking which ports are bound

- Active - SYN-SENT, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, and LAST-ACK (sometimes also TIME-WAIT), in which full connection state is kept

There are two states that are typically not kept in detail - SYN-RECEIVED and TIME-WAIT. SYN-RECEIVED keeps track of a connection that has received a SYN but not yet the final ACK of the three-way handshake; it state is typically not kept in detail to avoid DOS attacks that overload a server with half-open connections [RFC4987]. Similarly, the TIME-WAIT state is often ignored or kept in aggregate to avoid state accumulation on busy servers [Fa99].

The challenge implementing SNO involves using the LISTEN queue for SYN-RECEIVED states. Connections in the LISTEN state are indexed by the service number: for legacy TCP connections, this is the SYN destination port, and for SNO connections this is the SNO service number. In the SYN-RECEIVED state, connections always need to be indexed by the receive port number of the incoming ACK segment. As a result, SNO implementations need a distinct SYN-RECEIVED queue; they

cannot reuse the LISTEN queue to keep track of pending half-open connections.

The additional state needed for the SYN-RECEIVED queue is the same regardless of whether it shares space with the LISTEN queue - each receive port for half-open connections needs to be listed. The key difference is the index to the queue.

6. SNO impact on TCP option space

SNO needs to fit inside the available TCP option space, which provides 40 bytes for options. It is useful to consider that TCP SYN segments may include other options, notably:

- 4 bytes of MSS [RFC793]
- 10 bytes of timestamp [RFC7323]
- 3 bytes of window scale [RFC7323]
- 2 + 8N bytes of SACK, for N SACK blocks [RFC2018][RFC6675]

This leaves only 13 bytes for the SNO option (assuming 1 SACK block), which is more than sufficient. The experimental variant described herein uses 6 bytes; a standards-track variant would use only 4 bytes.

7. Security Considerations

There are four areas of security which the SNO option raises:

1. Interaction with IPsec and firewalls
2. Interaction with TCP/MD5 and TCP-AO security
3. Increased DOS impact

The impact on IPsec and firewalls is discussed in detail in Section 5.1. As noted there, SNO defeats the assumption that port numbers correspond to specific services, an assumption that was already defeated between consenting hosts. The SNO option thus raises no new vulnerability.

The impact of SNO on TCP/MD5 and TCP-AO is also discussed in Sections 5.1. Use of these services without inclusion of TCP options makes all options vulnerable, including SNO.

The additional resources incurred by parsing the SNO option are minimal.

8. IANA considerations

This document specifies a new TCP option that uses the shared experimental options format, with ExID = 0x5323 in network-standard byte order (representing ASCII "S#") [RFC6994]. This ExID has already been registered with IANA.

9. References

9.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, Sep. 1981 (STANDARD).

[RFC1122] Braden, R. (ed.), "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, Oct. 1989 (STANDARD).

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, Mar. 1997 (BEST CURRENT PRACTICE).

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC6994, Aug. 2013 (PROPOSED STANDARD).

9.2. Informative References

[Fa99] T. Faber, J. Touch, and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", in Proc. IEEE Infocom, 1999, pp. 1573-1583.

[Fr2008] Freire, S., A. Zuquete, "A TCP-layer name service for TCP ports", Proc. Usenix, 2008.

[IANA] Internet Assigned Numbers Authority, www.iana.org

[RFC768] Postel, J., "User Datagram Protocol", RFC768, Aug. 1980 (STANDARD).

[RFC814] Clark, D., "NAME, ADDRESSES, PORTS, AND ROUTES", RFC 814, Jul. 1982 (UNKNOWN).

[RFC959] Postel, J., J. Reynolds, "FILE TRANSFER PROTOCOL (FTP)", STD 9, RFC 959, Oct. 1985 (STANDARD).

[RFC1078] Lottor, M., "TCP Port Service Multiplexer (TCPMUX)", RFC1078, Nov. 1988 (UNKNOWN).

[RFC1833] Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, Aug. 1995 (PROPOSED STANDARD).

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, Aug. 1998 (PROPOSED STANDARD).

[RFC2780] Bradner, S., V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, Mar. 2000 (BEST CURRENT PRACTICE).

[RFC2782] Gulbrandsen, A., P. Vixie, L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, Feb. 2000 (PROPOSED STANDARD).

[RFC2993] Hain, T., "Architectural Implications of NAT", RFC 2993, November 2000 (INFORMATIONAL).

[RFC3234] Carpenter, B., S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234 Feb. 2002 (INFORMATIONAL).

[RFC3261] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, Jun. 2002 (PROPOSED STANDARD).

[RFC4301] Kent, S., K. Seo, "Security Architecture for the Internet Protocol", RFC4301, Dec. 2005 (PROPOSED STANDARD).

[RFC4340] Kohler, E., M. Handley, S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, Mar. 2006 (PROPOSED STANDARD).

[RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol", RFC 4960, Sep. 2007 (PROPOSED STANDARD).

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, Aug. 2007.

[RFC5246] Dierks, T., E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, Aug. 2008 (PROPOSED STANDARD).

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2006 (DRAFT STANDARD).

[RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option", RFC5925, Jun. 2010 (PROPOSED STANDARD).

[RFC6335] Cotton, M., L. Eggert, J. Touch, M. Westerlund, S. Cheshire "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Transport Protocol Port Number and Service Name Registry", RFC 6335 / BCP 165, Aug. 2011.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

[RFC7323] D. Borman, R. Braden, Jacobson, V., R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, May 1992 (PROPOSED STANDARD).

[To1999] Touch, J., T. Faber, "The TIME-WAIT state in TCP and its Effect on Busy Servers", Proc. Infocom, 1999.

[To2006] Touch, J., "A TCP Option for Port Names", draft-touch-tcp-portnames-00.txt (work in progress), Apr. 2006.

[To2018a] Touch, J., W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo (work in progress), Jan. 2018.

[To2018b] Touch, J., T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt (work in progress), Jan. 2018.

10. Acknowledgments

This work was inspired by discussions on the IETF mailing list, notably by suggestions by Keith Moore and Noel Chiappa. Bob Braden noted some of the origins of the named service concept.

This document is based on an earlier version based on using strings rather than IANA port numbers, where the receiving host used the

strings to directly identify services [To2006]. A similar approach was proposed that also used strings was implemented in Linux, except that the strings were resolved by a separate server and transmitted in the TCP segment as data (e.g., as with TCPMUX) [Fr2008].

This work is partly supported by USC/ISI's Postel Center.

This document was initially prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch

Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com
URL: http://www.strayalpha.com/