Network Working Group	A. Langley	
Internet-Draft	Google Inc	
Expires: February 7, 2009	August 06, 2008	

TOC

Faster application handshakes with SYN/ACK payloads
draft-agl-tcpm-sadata-01

Status of this Memo

Abstract

This document advocates the usage of small, mostly constant payloads in the SYN+ACK frame of the 3-way TCP (Postel, J., “Transmission Control Protocol,” September 1981.) [RFC0793] handshake. We show how this can have immediate benefits for some protocols. Additionally, we describe a new TCP option that enables a wider range of protocols to gain from it.

Table of Contents

---

1. Requirements Notation

TOC

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 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.) [RFC2119].

---

2. Changes since 00

TOC

To be removed by the RFC Editor before publication.

*Greatly expanded on the introduction

*Fixed the wording around retransmissions which mistakenly suggested that no packets of any type could be transmitted without payloads.

*Renamed the flag to SYNACK Payloads Processed.

*Required that the flag be echoed in resulting SYNACK frames.

*Added discussion of simultaneous open.

*Added discussion of SYNACKs with payloads that are nothing to do with this spec, noting that they are still permitted.

*Changed the option to a standard, 2 byte, flags option.

TOC

3. Introduction

At the current time, almost no stacks will send payloads in the SYNACK frame of a TCP handshake even though RFC 793 (Postel, J., “Transmission Control Protocol,” September 1981.) [RFC0793] permits it. This springs from a handful of reasons:

1. Processing time for a SYN must be minimal to mitigate the effects of SYN floods. Even waking up an application to process a SYN would greatly increase the costs.

2. Replies to SYNs must be small, otherwise it provides a way to amplify a DDoS attacks using false source IP addresses.

3. The ubiquitous sockets API doesn't make it easy to do so.

This document proposes that a semi-constant payload (a payload such that it's trivial for the kernel to compute) overcomes the first and third reasons. Additionally, limiting that payload to 64-bytes overcomes the second.
There are protocols that could immediately benefit from a gradual deployment of hosts which supported a `setsockopt` to set a constant payload and hosts that would ACK and enqueue such a payload. SMTP would be one such protocol: clients wait for a 200 code banner from the server before starting their part of the exchange and the banner is small and constant. SMTP is also a protocol which ends up making many, short lived connections.
Since such behaviour is already permitted by TCP it requires no standards work. It would also be easy to deploy. Active open hosts that don't enqueue payloads in SYNACK frames will ACK only the SYN flag and the passive open host then knows to retransmit the payload immediately after.
However, this common lack of carrying data in SYNACK frames, and the sockets API which reflects it, has guided the design of many application layer protocols. These protocols are often designed such that:

1. The client starts the exchange. For example, the first application layer bytes sent on an HTTP connection are the client's request.

2. The exchange is large since there is little space pressure. SSH algorithm agreement uses strings like "diffie-hellman-group14-sha1" (28 bytes) because of this.

In these cases we suggest that, had a general ability to send payloads in SYNACK frames existed at the time that these protocols were written, they may have ended up differently. However, the ability for a passive open host to send a payload with no latency overhead is of value: we outline three motivating examples in next sections.

Modifications to take advantage of SYNACK payloads would then require changes to the application level protocol. This could be managed by assigning new ports, trying connections on the new ports first, backing off etc. However, given that SYNACK payloads are partly a latency optimisation, that would utterly negate any gains.
Because of this, we also describe a TCP option that lets the application layer on both sides know that their respective stacks support at least the limited SYNACK payloads described herein, and also to agree to use an alternative protocol which takes advantage of it.
Fundamentally, any protocol which used payloads in SYNACK frames could achieve the same effect without them, at the cost of an extra round trip. Thus, this should only be used where latency is important. None the less, the advantage of avoiding a round trip should not be discounted. Round trip times are often in excess of 100ms for distant hosts, or in poorly networked areas of the world.

---

4. Example One: Opportunistic HTTP encryption

TOC

Here we assume that both HTTP client and server implement this specification.
The client, before calling connect, calls setsockopt to instruct the kernel to include an option to advertise support for this specification. The server has already configured its listening socket to include a Diffie-Hellman public value in the SYNACK payloads elicited from SYN frames carrying this option. Additionally, the server's stack generates an 8-byte random nonce and includes it in the payload.
The client is aware that the server implements this specification because the advertising option is echoed back in the SYNACK frame. Thus, it expects to read the nonce and public value from the connection. It then sends its own nonce and public value to the server. Both sides can calculate a shared key and use a cipher to encrypt the remaining data in both directions.
Choosing the correct cryptographic primitives can make this particularly cheap. Curve25519 (Bernstein, D., “Curve25519: new Diffie-Hellman speed records,” .) [curve25519] is an elliptic curve Diffie-Hellman function that can be calculated in 240 microseconds on a 2.33GHz Intel Core2. Salsa20/8 (Bernstein, D., “Salsa20/8 and Salsa20/12,” .) [salsa20] is a stream cipher that can encrypt data in 2 cycles/byte on the same hardware.
The resulting key could also be used to establish integrity using the forthcoming TCP Auth Option specification.

This example demonstrates a number of salient features of this specification:

*By using the correct primitive (curve25519), a constant payload can be used to establish cryptographic connections.

*We can add significant extensions to latency sensitive protocols without affecting latency. Previous attempts (Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1," May 2000.) [RFC2817] to do the same have required an extra round trip weather the server side supported the protocol or not.

*We can do in a backwards compatible fashion, affording a gradual deployment.

The above is cursorary in order not to distract from the topic of this document, however enquiring readers are welcome to continue reading this section if they still have questions.

Why Elliptic Curves?: The payload must be short otherwise SYN-floods could use this as an amplification to backscatter DDoS another host. The reduced computation cost (as compared to Diffie-Hellman over a multiplicative finite field) is very nice.

Most importantly, curve25519 is specifically designed to allow a constant public value to be used for multiple key agreements. If a new public value had to be generated for every SYN, not only would the stack have to be able to perform that operation, a SYN flood would be very effective.

Can't the client's public value fit in the SYN?: A SYN generally has twenty bytes of free option space these days. (We can't use the payload space in a SYN). Since we wouldn't want to define the last option ever, we need to leave four bytes spare. Two bytes for the option header means fourteen bytes (or 112 bits) for the public value. The closest prime is then 2^112-75.

The best, general algorithm currently known for breaking the Diffie-Hellman problem on elliptic curves is Pollard's Rho. The work involved in this attack is sqrt(n), which is 2^56 in this case. Critically, once you have solved a single instance you can precompute tables to speed up breaking more instances. With a petabyte of storage, you could break 112-bit curves in only 2^12 operations.

Can't a smaller field be used?: Some speedup could be gained by using an elliptic curve with a field size around 200 bits. However the effort of defining such a curve is pretty huge. The standard NIST curves around that size are slower than curve25519.

What about man-in-the-middle attacks: All opportunistic schemes are open to man-in-the-middle and downgrade attacks. This is no exception, it's a trade off and for real security, TLS (Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," April 2006.) [RFC4346] should be used. It has been suggested that the HTTP server include a header in replies giving a URL on the same

domain, using the https scheme, which contains the server's public value and an expiry time.

5. Example Two: Faster SSH connections

TOC

SSH (Ylonen, T. and C. Lonvick, “The Secure Shell (SSH) Transport Layer Protocol,” January 2006.) [RFC4253] connection latency is a small, but quotidian frustration for those who use it. Current efforts to address it involve multiplexing interactive sessions over long-term, persistent connections.

Consider the following, diagrammatic representation of the beginning of an SSH (Ylonen, T. and C. Lonvick, “The Secure Shell (SSH) Transport Layer Protocol,” January 2006.) [RFC4253] connection:

```
        0 SYN     ------------>          0.5
        1         <------------ SYNACK    0.5
        1 Ident   ------------>          1.5
        1 NList   ------------>          1.5
        2         <------------ Ident     1.5
        2         <------------ NList     1.5
        2         <------------ KX        1.5
        2 KX      ------------>          2.5

        Key:
          Ident: A string which contains the SSH implementation name
          NList: Name list: the list of supported algorithms
          KX: Key exchange data, usually Diffie-Hellman
```

Standard SSH protocol

Figure 1

Here, arrows from the left to the right are frames from client to server. Times on the left are the times that the client either transmits or receives a packet (and vice versa). Times are measured in round trip times (RTT), so that it takes 0.5 units for a frame to pass between the hosts.

The above diagram is for a latency tuned implementation of SSH, specifically, the client doesn't wait for the server's identity string to be received. And yet, in this ideal scenario, the client can only start transmitting useful data after 2 RTT and the server can only start transmitting after 2.5 RTT. As a rule of thumb, the RTT from San

Francisco to London is 150ms, so this means a 300ms latency, at least, when setting up this connection.
(To keep the discussion simple, we assume there is no packet loss, that the path is symmetrical and that the client's ACK of the 3-way handshake carries a data payload.)
Now, let us consider the situation when SYNACK payloads are available. First we compact the name-list (which is part of the algorithm negotiation) and put it in the SYNACK.

```
0 SYN     ------------>          0.5
1         <------------ SA+NList 0.5
1 NList   ------------>          1.5
1 KX      ------------>          1.5
2         <------------ KX       1.5
```

SSH protocol with a compact name list carried in the SYN+ACK frame

Figure 2

In this situation, the client knows the results of the algorithm negotiation as soon as the SYNACK comes back and can include the correct key exchange with the first ACK packet. This reduces the server's latency by a full RTT since it can transmit as soon as the 3-way handshake completes.
As a final optimisation, we could assume either that the server takes a successful guess at the key exchange algorithm to use, or that the application level protocol specifies a single key exchange algorithm:

```
0 SYN     ------------>          0.5
1         <------------ SA+KX    0.5
1 KX      ------------>          1.5
```

A protocol which includes key exchange information in the SYN+ACK frame.

Figure 3

Here the client's latency is 1 RTT and the server's is 1.5 RTT, which is equal to the minimum required by the 3-way handshake, saving a full RTT of latency from the initial diagram.

---

6. Example Three: Compressed HTTP headers

TOC

So that all the examples aren't cryptography based, we consider a third example.

There are many HTTP resources that are very small, or even empty. Consider that clicking on Google results involves requesting a resource from the Google server to redirect to the true result. Or OCSP (Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," June 1999.) [RFC2560] revocation servers which serve small ASN.1 documents. For these services the size of HTTP headers might dominate the bandwidth requirements: Firefox 3 transmits over 350 bytes to request the shortest URL possible (/)

HTTP headers, however are highly compressible. They are highly structured, and many strings are very common (such as Keep-Alive). Careful examination of the current patterns in both client requests and server replies would probably yield a range coding (Martin, G., "Range encoding: an algorithm for removing redundancy from a digitized message," Video and Data Recording Conference, July 1979.) [rangecoding] model that achieved significant savings.

However, there is no easy method to deploy such a scheme. Obviously the first client request on a connection could not use a scheme. A server could advertise support in its reply headers for subsequent requests on the same connection, although that could only affect requests that haven't already been pipelined.

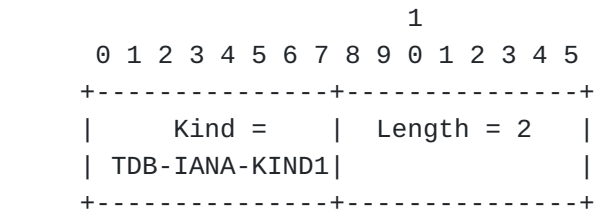
A SYNACK payload could serve to advertise support for this, and any other extensions, allowing every request on a connection to use such a scheme when both ends support it.

---

7. The SYNACK Payload Processed Option

TOC

Alternative application protocols that take advantage of data in a SYNACK frame necessarily require the application level to know when this specification is in effect. To that end, we define an option which signifies compliance with this specification to be carried in the SYN and SYNACK frames:



SYNACK Payload Processed Option

Figure 4

It is required that both endpoints reach agreement about when this option is in effect since it affects the application layer. The next five paragraphs deal with this. This specification considers the option to be an optimisation however, and a valid agreement might be that the option is not in effect even in the case that both endpoints support it. This is to allow implementations to back off in the case of possible middleware interactions and overload.

Hosts MUST NOT include the SYNACK Payload Processed option unless an application has requested it for the current socket. If SYNACK Payload Processed is requested for a socket, the host SHOULD include the SYNACK Payload Processed option. For example, it may choose not to in the case of having to retransmit the SYN frame as middleware may be filtering the extra option.

Upon receipt of a SYN frame with a SYNACK Payload Processed option, to a valid, passive open socket, that socket will have either been configured by an application to take advantage of this specification or not. In the case that it has not, the host MUST NOT include the SYNACK Payload Processed option in any SYNACK. In the case that it has been so configured, the host SHOULD include the configured payload in the SYNACK. Iff it chooses to do so, it MUST include the SYNACK Payload Processed option.

For a given connection, for all resulting SYNACK frames, the presence of the SYNACK Payload Processed option MUST NOT differ.

If a host has alternative mechanisms which involve sending payloads in a SYNACK frame, they MUST NOT be used concurrently with this specification for a given connection. This specification does not prohibit SYNACK frames with payloads generated by other means as long as the SYNACK Payload Processed option is not included.

It's expected that a host will make a best effort to include a SYNACK payload when the application has set one. It may choose not to for a number of reasons including: the SYN frame didn't request it, the host is under heavy SYN load, is using SYN cookies or that the host is having to retransmit the SYNACK.

The next four paragraphs seek to establish a minimal basis for application protocols to build upon. An implementation may allow applications to set arbitrary payloads on a per connection basis, but

we expect that most will wish to expose a more limited scope. Obviously some of these capabilities, such as the inclusion of random bytes, are motivated by the examples above.

In the case that the SYNACK Payload Processed option is in effect: The data payload MUST affect the SEQ/ACK numbers like any other data. Any ACK frame resulting from such a SYNACK frame MUST acknowledge the whole SYNACK frame, including the SYN flag. If a frame is the final ACK in a 3-way handshake, a host MUST reject it unless it acknowledges the whole SYNACK frame.

A host MUST provide a method for applications to set a SYNACK payload, to determine if a passive-open connection sent a SYNACK payload and to determine if an active open connection received the SYNACK Payload Processed option in the SYNACK frame.

A host MUST support configuring passive open sockets with at least 64-bytes of data. (See "Security Considerations", below).

A host SHOULD support including at least 8 random bytes in the SYNACK payload, at any arbitrary (but within range) byte offset. If it does, the random bytes MUST be consistent between retransmissions of the SYNACK frame and the host MUST support a method for the application to learn the value of the random bytes included in any resulting connection.

What follows is clarification on some corner cases:

In the case of a simultaneous open where one or both SYN frames include the SYNACK Payload Processed flag, this specification is not in effect. The connection continues as usual.

In the case of a frame carrying the SYNACK Payload Processed option and with both SYN and FIN flags set, the host MAY support this specification. In practice, many stacks with ignore a FIN flag and any payload in a SYN frame, in which case such a packet is no different from any other SYN frame.

In the case that the MTU makes transmitting the larger SYNACKs problematic, the host MAY choose to fragment the packet or it MAY choose not to echo the SYNACK Payload Processed option, resulting in a smaller SYNACK frame.

---

8. Security Considerations

TOC

Any payload in a SYNACK packet must be as frugal as possible since a host will be transmitting it to an unconfirmed address. If a 40 byte frame could elicit a 1500 byte reply to an attacker controlled address, this would be readily used to hide and amplify distributed denial of service attacks.

Thus we specify a maximum size of 64 bytes for the payload. This is sufficient to include a strong elliptic curve key (256 bits), a 64-bit nonce and a small amount of overhead (24 bytes).

9. Comparison to T/TCP

TOC

The idea of including data in frames which also carry a SYN flag isn't new: it was included in the experimental T/TCP RFCs 1379 (Braden, B., "Extending TCP for Transactions -- Concepts," November 1992.) [RFC1379] and 1644 (Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification," July 1994.) [RFC1644]. T/TCP suffered because it broke the assumption that the source address of a new connection from a passive-open socket had been verified by a 3-way handshake. This was a critical security issue for applications like RSH which often used source address whitelists.
This draft doesn't break any such assumptions that applications may be depending on. Source addresses for new connections are still validated by a 3-way handshake for passive-open sockets. Additionally, this draft is dramatically simpler than T/TCP: it doesn't introduce any additional TCP states nor does it deal with the complexity of including payloads in a SYN frame. Nor does this draft apply to any application which is unaware of it since applications are required to explicitly configure SYNACK payloads before they come into effect.

10. Middlebox Interactions

TOC

The large number of middleboxes (firewalls, proxies, protocol scrubbers, etc) currently present in the Internet pose some difficulty for deploying new TCP options. Some firewalls may block segments that carry unknown options. For instance, if the flags option is not understood by a firewall, incoming SYNs advertising SYNACK payload support may be dropped, preventing connection establishment. This is similar to the ECN blackhole problem, where certain faulty hosts and routers throw away packets with ECN bits set [RFC3168] (Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," September 2001.). Some recent results indicate that for new TCP options, this may not be a significant threat, with only 0.2% of web requests failing when carrying an unknown option [transport-middlebox] (Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes," ACM SIGCOMM/USENIX Internet Measurement Conference, October 2004.).

TOC

11. IANA Considerations

This document requires IANA to update values in its registry of TCP options numbers to assign a new entry, referred herein as TBD-IANA-KIND1.

---

12. Acknowledgements

TOC

Wesley Eddy kindly reviewed initial versions of this draft.
Joe Touch provided many helpful comments.

---

13. References

TOC

13.1. Normative References

TOC

[RFC0793]	Postel, J., "Transmission Control Protocol," STD 7, RFC 793, September 1981 (TXT).
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

13.2. Informative References

TOC

[RFC1379]	Braden, B., "Extending TCP for Transactions -- Concepts," RFC 1379, November 1992 (TXT).
[RFC1644]	Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification," RFC 1644, July 1994 (TXT).
[RFC2560]	Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC 2560, June 1999 (TXT).
[RFC2817]	Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1," RFC 2817, May 2000 (TXT).
[RFC3168]	Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, September 2001 (TXT).
[RFC4253]	

	Ylonen, T. and C. Lonvick, “The Secure Shell (SSH) Transport Layer Protocol,” RFC 4253, January 2006 (TXT).
[RFC4346]	Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1,” RFC 4346, April 2006 (TXT).
[curve25519]	Bernstein, D., “Curve25519: new Diffie-Hellman speed records.”
[salsa20]	Bernstein, D., “Salsa20/8 and Salsa20/12.”
[transport-middlebox]	Medina, A., Allman, M., and S. Floyd, “Measuring Interactions Between Transport Protocols and Middleboxes,” ACM SIGCOMM/USENIX Internet Measurement Conference, October 2004.
[rangecoding]	Martin, G., “Range encoding: an algorithm for removing redundancy from a digitized message,” Video and Data Recording Conference, July 1979.

Appendix A. Changes

TOC

Author's Address

TOC

	Adam Langley
	Google Inc
Email:	agl@imperialviolet.org

Full Copyright Statement

TOC

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.