

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2015

C. Huitema
Microsoft
E. Rescorla
Mozilla
J. Iyengar
Google
March 5, 2015

DTLS as Subtransport protocol
draft-huitema-tls-dtls-as-subtransport-00.txt

Abstract

The developers of "user level" transports will benefit from a standard implementation of authentication and encryption. This can be achieved using DTLS as a sub-transport. Using DTLS enables developers to benefit from the investment in TLS, and removes the burden and the risks of re-creating similar technology.

There are several requirements to ensure that DTLS is a suitable sub-transport: zero RTT setup, low overhead, and DOS prevention. The IAB SEMI workshop outlined other potential requirements: start/stop indication, and ability to accept indications from the network. The draft presents guidelines for meeting these requirements in a new version of DTLS.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on September 6, 2015.

Internet-Draft

DTLS as Subtransport

March 2015

Copyright Notice

Copyright (c) 2015 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.

Table of Contents

| | | |
|-----------------------|-------------------------------------|--------------------|
| 1. | Introduction | 2 |
| 1.1. | Requirements | 3 |
| 2. | DTLS as a sub transport | 3 |
| 3. | Efficient retransmissions | 4 |
| 4. | Zero-RTT with TLS/1.3 | 5 |
| 5. | Overhead reduction | 5 |
| 6. | DOS resilience | 6 |
| 7. | Connection-id option | 7 |
| 8. | Start/stop indication | 7 |
| 9. | Indication verification | 8 |
| 10. | Security Considerations | 9 |
| 11. | IANA Considerations | 10 |
| 12. | Acknowledgments | 10 |
| 13. | References | 10 |
| 13.1. | Normative References | 10 |
| 13.2. | Informative References | 10 |
| | Authors' Addresses | 11 |

[1.](#) Introduction

There is a growing demand to develop "user level" transport, motivated by "innovation" and "deployment." The innovation part is the desire to get better performance than TCP, or especially the combination of TCP and TLS, addressing such issues as zero-RTT setup or head of queue blocking. The deployment part is motivated by observation that platform upgrades are slow, and typically only reach

a fraction of the deployed systems. The proposed solution is to execute the transport functions in user space, so the transport innovation can be distributed with application updates.

Any transport innovation has to work on top of an encryption layer. This is good security practice, and it also prevent middleboxes from interfering with the transport functions. This interference with TCP is widespread and effectively blocks innovation, making it hard to deploy even something as simple as ECN. Encryption prevents the middle boxes from twiddling bits in the header. For example, Google's QUIC [[QUICBLOG](#)]. protocol uses an encryption system that is tightly integrated with the transport layer in order to optimize performance and reduce the protocol's accessibility to middleboxes.

QUIC uses a specially designed security layer, but there was a consensus in the IAB SEMI workshop [[IABSEMI](#)] that we don't want multiple applications each designing their specific key exchange and encryption algorithms. The natural solution is to base the end-to-end transports on a standard security layer, allowing transport specialists can worry about efficient retransmission, congestion and multiplexing, while security specialists can implement the security layer.

The obvious candidate is DTLS [[RFC6347](#)], as the general idea of "TLS over UDP" allows us to reuse the TLS experience and the TLS implementations. Of course, we may need to work on a new features to meet transport requirements.

[1.1](#). Requirements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [[RFC2026](#)].

[2](#). DTLS as a sub transport

Examination of DTLS to the requirements for a subtransport layer reveals some areas for improvement.

Efficient retransmissions: Part of the rationale for doing new

transports is to explore efficient retransmission strategies, but this conflicts with the existing retransmission procedures that are embedded in standard DTLS.

Zero-RTT setup: DTLS/1.2's minimum connection setup requires 1-RTT. One of the major performance targets for new transports is low-latency, motivating a 0-RTT connection setup.

Low overhead: DTLS/1.2 record headers include elements like version number, epoch, sequence number or clear text length that cause a fair bit of overhead in a small UDP datagram. Using some form of header compression would reduce that overhead.

DOS prevention: TLS over UDP offers a big surface area for DOS attacks, as unauthenticated clients can ask a server to perform expensive crypto or produce large responses. This is especially true if we support 0-RTT. While DTLS has some defenses against DoS attacks, they may need to be strengthened.

connection-id: DTLS/1.2 identifies connections using the 5 tuple. Having an independent ID would allow for functionalities similar to "TCP multipath." It would also facilitate the work of load balancers in front of a server farm.

Discussions in the IAB SEMI workshop also pointed out that middleboxes interaction would be easier to manage if the UDP transport had some specific properties:

Start/stop: Many middleboxes need to assign state to UDP flows. For example, NATs need to assign and maintain port mappings. UDP flows do not have explicit beginning and end markers similar to TCP SYN/FIN/RST flags. In the absence of such flags middleboxes have to resort to timer based management. This in turn forces applications to use periodic "keep alive" traffic, which is inefficient.

Indication verification: Middleboxes may wish to send informative messages similar to ICMP, providing for example indications about MTU size or congestion state. Application that receive these messages need to differentiate between legitimate data coming from network elements "on the path" and fake signals coming from attackers. This is easier if the messages coming from the network

can copy hard to predict header elements like connection-id or sequence numbers.

It is not yet clear whether these features are feasible or deployable, but we document them here as an outcome of the IAB SEMI discussion.

[3.](#) Efficient retransmissions

Protocols like QUIC implement innovative retransmission strategies, combining Forward Error Correction with selective acknowledgements and selective retransmissions. DTLS implements a minimalist retransmission strategy for the messages that are part of the handshake protocol, as explained in [section 3.2 of \[RFC6347\]](#). This creates a tension between adhering to the standard and efficient retransmission:

- o One could keep the QUIC retransmission for the handshake packets and switch to an innovative transport for the reminder of the

connection. The downside is that using less efficient transport during the handshake risk can cause additional latency, which is contrary to the objective of transport innovation.

- o One could design an innovative transmission as a layer under the TLS framing, effectively redesign the layering of DTLS. This solves the efficiency issues, but expose the clear-text transmission controls to interference by middle-boxes, which may ultimately prevent innovation.
- o One could consider a hybrid design that allows clear text innovation for the initial handshake and encrypted innovation for data retransmissions, but no such design is available yet.

To put it simply, there is no consensus yet on a good solution to this problem.

[4.](#) Zero-RTT with TLS/1.3

Probably the biggest requirement is to have a 0-RTT connection setup, meaning that the initiator (typically the "client") can start sending protected upper-level data in its initial flight of datagrams. In

general, a 0-RTT handshake requires that both the client and server retain state:

- o The client must retain the server's parameters, including a long-term cryptographic key.
- o The server must retain enough state to detect replays of the client's initial flight.

In DTLS 1.2 and before, the client and server are both assumed to be naive and so the first round-trip is used to establish this state. This is still necessary for situations where the client and server have never talked before and have no out-of-band communications channel, but if both sides are primed, it is possible to define a 0-RTT handshake as well. Such a mode will be part of (D)TLS 1.3 and is currently under development in the TLS WG.

[5.](#) Overhead reduction

DTLS is not generally very aggressive about conserving per-packet overhead. The minimum DTLS record adds 13 bytes of header to the packet and the common AES-GCM cipher suites add another 8 bytes or a total of 21 bytes of header overhead (plus the authentication tag, which is required). While these header bytes are not entirely redundant (for instance, the sequence number allows the receiver to deal with reordered packets) they are largely redundant in the common

case where the network mostly delivers packets in order essentially every record is application data.

For maximum efficiency, it is desirable to have a mechanism for compressing this data. [[I-D.modadugu-dtls-short](#)] describes one set of techniques for doing so, though research into the optimal method is still required.

[6.](#) DOS resilience

Our principal DoS concerns are:

- o Preventing resource over-consumption on the server.
- o Preventing the server from being used as a traffic amplifier.

Because TLS runs over TCP, it inherits TCP's DoS resistance properties: an attacker must first establish a TCP connection before he can talk to the TLS implementation. This generally means demonstrating that he can receive traffic at the IP address he is sending from. This significantly reduces the risk of amplification and allows the server to differentially throttle traffic from clients which appear to be sending bogus handshakes. The result is partial protection against resource consumption attacks, but an attacker can still mount such attacks if they control a large number of IP addresses.

Any protocol which runs directly over UDP -- as DTLS does -- not inherit these properties. DTLS already has anti-DoS measures in the form of a cookie exchange which allows the server to force the client to prove reachability at a given address. This is the standard technique for addressing resource consumption attacks with such protocols and can be deployed differentially (i.e., only when under attack) to reduce the latency impact at normal times. Other techniques which have been proposed for (D)TLS include computational puzzles.

The DTLS cookie exchange also prevents amplification attacks but because the server does not generally know when it is being used in this fashion, it is harder to know where to set the protection/latency tradeoff. It is currently unclear how important amplification protection is (DNS already has significant amplification vectors) but if so, possible techniques include longer-term cookies and forcing the client to pad its initial flight, thus reducing the impact of amplification.

[7.](#) Connection-id option

Many UDP applications identify the application connection implicitly from the "five tuple" of source and destination addresses and ports, and payload type. There are however several potential advantages to having an explicit "connection-id:"

- o Enabling applications to use several ports and path in parallel,

for performance or resiliency,

- o Enabling seamless continuation of an application over a new port if the preceding port becomes unusable.

The latter problem, ports becoming unusable, is often caused by NAT traversal. NAT are known to forget UDP mappings if they don't see traffic for some period, or for some other reason such as for example hash table collision. Applications must be ready to quickly reestablish their connectivity. Using an explicit connection-id makes this reestablishment straightforward.

The connection-id could be encoded as a header parameter, and its use negotiated during the initial handshake, using techniques similar to the parameters negotiation proposed in [[I-D.modadugu-dtls-short](#)].

8. Start/stop indication

Middleboxes like NAT or firewall assign some state to the UDP flows, such as for example a port mapping in a NAT or an explicit port opening in a firewall. For TCP flows, middleboxes can examine TCP flags and deduce when they see FIN or RST flags that the connection is getting closed. They can then free the state associated with the TCP flow. There are no such flags in UDP packets. The start of a flow can be deduced implicitly from the arrival of a first packet for that flow, but the end cannot. Middleboxes have to resort to timer based management. The timers have to be short, and this drives applications to send frequent keep-alive packets to make sure that port mappings and port opening persists. An explicit indication would enable more efficient management of resource.

TLS and DTLS include an explicit close mechanism, in which the parties use the TLS Alert protocol and exchange "close notify" messages. Sending such an alert message indicates that the sending party is done, and will not send any more messages in the TLS session. When both parties have sent a "close notify" message, the session is effectively terminated.

If a middlebox could monitor the transmission of "close notify" messages, it could effectively decide when resource can be disposed.

However, the alert protocol is part of the encrypted payload, and the

only visible indication in the clear text header is a "Content type" indication set to "Alert", indicating that the encrypted payload contains an Alert message. Closure indication is only one of the usages of the Alert protocol, the other usages being error indication and warning indication. A middlebox that monitors Alert messages will only get an imperfect indication of the connection state:

- o A closure message indicates that one party has finished sending, and waits until a similar closure message from the other end to terminate the session,
- o An error message indicates that one party detected an error, will not send any more data, and will not accept any more data from the other party,
- o A warning message indicates that one party detected an anomaly, but that the session can continue.

The middlebox can gain information about the state of a DTLS connection by monitoring the Alert messages, even if that information is imperfect. Alternatively, we could consider adding an explicit FIN bit in a revised clear-text header.

We should note here that there is a potential tension between the management of resource and the identification of sessions discussed in [Section 7](#). The use of the context identifier allows sessions to spread over multiple addresses and ports, and also allows multiple sessions to share the same addresses and ports. If such multiplexing is in place, the middleboxes would have to allocate resources per context rather than per address and port tuples, but would have no guarantee to see all the alert messages for a specific session.

[9](#). Indication verification

Middleboxes could send messages to applications, using ICMP or perhaps simply sending UDP messages using the same five-tuple as the application. Assuming that such messages can be delivered, the application will have to verify that they come from a legitimate source, for example a middlebox on the path providing an indication about acceptable MTU.

There is always a risk that such indications will be misused, and that malevolent third parties would try to provide false indications in order to disrupt the application. The application must thus be able to distinguish between legitimate and spurious indication.

The middlebox could echo some parameters of the clear text header in order to "prove" that they are on path. Typical parameters would be the context ID or the sequence numbers. For this to be effective, these parameters should be "hard to guess," which implies for example unpredictable assignment of context ID or initial sequence numbers. Of course, such desire for unpredictability conflicts with the desire for low overhead, as compressed headers are inherently easier to predict than long numbers.

One question for any indication verification scheme is how much of the connection the middlebox needs to be able to see. For instance, if initial sequence numbers or DTLS handshake nonces are used to demonstrate that the middlebox is on-path, then the middlebox needs to be on-path for the entire connection and maintain connection state.

10. Security Considerations

This document proposes that user level transports use DTLS as a component, instead of inventing their own transport. We believe that this componentized approach will avoid many of the pitfalls of inventing or implementing special purpose security designs. Instead, applications will benefit from the experience accrued in the design and evolution of TLS [[RFC5246](#)] and will be able to reuse already developed TLS and DTLS implementations.

We note that there is a definitive DOS exposure when running a cryptographic protocol over UDP, and that this exposure is increased when we attempt to enable zero RTT setup. The risk and the corresponding mitigations are described in [Section 6](#). Here again, we believe that it is beneficial to reuse the DOS mitigations developed for DTLS and designed for the zero RTT setup options of TLS/1.3 [[I-D.ietf-tls-tls13](#)].

Any start/stop mechanism solving the requirement presented in [Section 8](#) opens the door to an attack is similar to but distinct from TCP RST attacks, where injected RST packets terminate connections. An on path attacker could inject bogus packets with a "Stop" indication, to cause connection state to be torn down at middleboxes, potentially causing the connection to be abruptly terminated. The middleboxes will not be able to separate these injected packets from legitimate "Stop" packets sent by the endpoints, because they cannot verify the end-to-end authentication of packets.

Participants to the SEMI workshop [[IABSEMI](#)] envisage a "path to application" messaging system in which intermediate relays would

provide information to the application, such as for example MTU size or congestion notification. Such messages would not benefit from the

end to end authentication and encryption provided by DTLS. Allowing such messages exposes the application to denial of service attacks. Some potential mitigations are described in [Section 9](#)

[11.](#) IANA Considerations

This draft references [[I-D.modadugu-dtls-short](#)], which proposed four new extensions for DTLS. A future version of this draft will very likely propose the registration of similar extensions, using the mechanisms defined in [[RFC5246](#)] and [[RFC6066](#)].

[12.](#) Acknowledgments

The inspiration for this draft came from discussions in the IAB SEMI workshop, and from studies of the QUIC protocol.

[13.](#) References

[13.1.](#) Normative References

- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", [BCP 9](#), [RFC 2026](#), October 1996.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), August 2008.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", [RFC 6066](#), January 2011.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", [RFC 6347](#), January 2012.

[13.2.](#) Informative References

- [I-D.ietf-tls-tls13]
Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", [draft-ietf-tls-tls13-04](#) (work in progress), January 2015.

[I-D.modadugu-dtls-short]

Modadugu, N. and E. Rescorla, "Extensions for Datagram Transport Layer Security (TLS) in Low Bandwidth Environments", [draft-modadugu-dtls-short-00](#) (work in progress), March 2006.

[IABSEMI] Kuehlewind, M. and B. Trammell, "IAB Workshop on Stack Evolution in a Middlebox Internet (SEMI)", Jan 2015, <<https://www.iab.org/activities/workshops/semi/>>.

Huitema, et al.

Expires September 6, 2015

[Page 10]

Internet-Draft

DTLS as Subtransport

March 2015

[QUICBLOG]

Roskind, J., "Experimenting with QUIC", June 2013, <<http://blog.chromium.org/2013/06/experimenting-with-quic.html>>.

Authors' Addresses

Christian Huitema
Microsoft

Email: huitema@microsoft.com

Eric Rescorla
Mozilla

Email: ekr@rtfm.com

Jana Iyengar
Google

Email: jri@google.com

