

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

T. Pauly
E. Kinnear
Apple Inc.
July 02, 2018

TCP Encapsulation Considerations
draft-pauly-tsvwg-tcp-encapsulation-00

Abstract

Network protocols other than TCP, such as UDP, are often blocked or suboptimally handled by network middleboxes. One strategy that applications can use to continue to send non-TCP traffic on such networks is to encapsulate datagrams or messages within in a TCP stream. However, encapsulating datagrams within TCP streams can lead to performance degradation. This document provides guidelines for how to use TCP for encapsulation, a summary of performance concerns, and some suggested mitigations for these concerns.

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 <https://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 January 3, 2019.

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 (<https://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

Internet-Draft

TCP Encapsulation

July 2018

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
2.	Motivations for Encapsulation	3
2.1.	UDP Blocking	3
2.2.	UDP NAT Timeouts	3
3.	Encapsulation Formats	3
3.1.	Multiplexing Flows	4
4.	Deployment Considerations	5
5.	Performance Considerations	5
5.1.	Loss Recovery	6
5.1.1.	Concern	6
5.1.2.	Mitigation	6
5.2.	Bufferbloat	7
5.2.1.	Concern	7
5.2.2.	Mitigation	8
5.3.	Head of Line Blocking	8
5.3.1.	Concern	8
5.3.2.	Mitigation	9
6.	Security Considerations	9
7.	IANA Considerations	9
8.	Informative References	9
	Authors' Addresses	10

[1.](#) Introduction

TCP streams are sometimes used as a mechanism for encapsulating datagrams or messages, which is referred to in this document as "TCP encapsulation". Encapsulation may be used to transmit data over networks that block or suboptimally handle non-TCP traffic. The current motivations for using encapsulation generally revolve around the treatment of UDP packets ([Section 2](#)).

Implementing a TCP encapsulation strategy consists of mapping datagram messages into a stream protocol, often with a length-value record format ([Section 3](#)). While these formats are described here as applying to encapsulating datagrams in a TCP stream, the formats are equally suited to encapsulating datagrams within any stream abstraction. For example, the same format may be used for both raw

[2.](#) Motivations for Encapsulation

The primary motivations for enabling TCP encapsulation that will be explored in this document relate mainly to the treatment of UDP packets on a given network. UDP can be used for real-time network traffic, as a mechanism for deploying non-TCP transport protocols, and as a tunneling protocol that is compatible with Network Address Translators (NATs).

[2.1.](#) UDP Blocking

Some network middleboxes block any IP packets that do not appear to be used for HTTP traffic, either as a security mechanism to block unknown traffic or as a way to restrict access to whitelisted services. Network applications that rely on UDP to transmit data will be blocked by these middleboxes. In this case, the application can attempt to use TCP encapsulation to transmit the same data over a TCP stream.

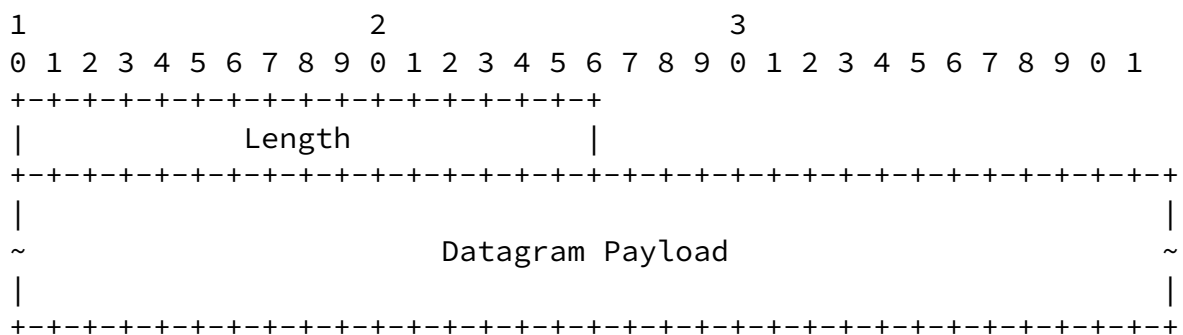
[2.2.](#) UDP NAT Timeouts

Other networks may not altogether block non-TCP traffic, but instead make other protocols unsuitable for use. For example, many Network Address Translation (NAT) devices will maintain TCP port mappings for long periods of time, since the end of a TCP stream can be detected by the NAT. Since UDP packet flows do not signal when no more packets will be sent, NATs often use short timeouts for UDP port mappings. Thus, applications can attempt to use TCP encapsulation when long-lived flows are required on networks with NATs.

[3.](#) Encapsulation Formats

The simplest approach for encapsulating datagram messages within a TCP stream is to use a length-value record format. That is, a header consisting of a length field, followed by the datagram message itself.

For example, if an encapsulation protocol uses a 16-bit length field (allowing up to 65536 bytes of datagram payload), it will use a format like the following:



The format of the length header field could be longer or shorter depending on the needs of the protocol. 16 bits is most appropriate when encapsulating datagrams that would otherwise be sent directly in IP packets, since the payload length field for an IP header is also 16 bits.

The length field must be specified to either include itself in the length of the entire record, or to only describe the length of the payload field. The protocol used for encapsulating IKE and ESP packets in TCP [[RFC8229](#)] does include the length field itself in the length of the record. This may be slightly easier for implementations to parse out records, since they will not need to add the length of the length field when finding record offsets within a stream.

3.1. Multiplexing Flows

Since TCP encapsulation is used to avoid failures caused by NATs or firewalls, some implementations re-use one TCP port or one

established TCP stream for multiple kinds of encapsulated traffic. Using a single port or stream allows re-use of NAT bindings and reduces the chance that a firewall will block some flows, but not others.

If multiple kinds of traffic are multiplexed on the same listening TCP port, individual streams opened to that port need to be differentiated. This may require adding a one-time header that is sent on the stream to indicate the type of encapsulated traffic that will follow. For example, TCP encapsulated IKE [[RFC8229](#)] uses a stream prefix to differentiate its encapsulation strategy from proprietary Virtual Private Network (VPN) protocols.

Multiplexing multiple kinds of datagrams, or independent flows of datagrams, over a single TCP stream requires adding a per-record type field or marker to the encapsulation record format. For ease of parsing records, this value should be placed after the length field of the record format. For example, various ESP packet flows are identified by the four-byte Security Parameter Index (SPI) that

comprises the first bytes of the datagram payload, while IKE packets in the same TCP encapsulated stream are differentiated by using all zeros for the first four bytes.

[4.](#) Deployment Considerations

In general, any new TCP encapsulation protocol should allocate a new TCP port. If TCP is being used to encapsulate traffic that is normally sent over UDP, then the the most obvious port choice for the TCP encapsulated version is the equivalent port value in the TCP port namespace.

Simply using TCP instead of UDP may be enough in some cases to mitigate the connectivity problems of using UDP with NATs and other middleboxes. However, it may be useful to also add a layer of encryption to the stream using TLS to obfuscate the contents of the stream. This may be done for security and privacy reasons, or to prevent middleboxes from mishandling encapsulated traffic or ossifying around a particular format for encapsulation.

[5.](#) Performance Considerations

Many encapsulation or tunnelling protocols utilize an underlying transport like UDP, which does not provide stateful features such as loss recovery or congestion control. Because encapsulation using TCP involves an additional layer of state that is shared among all traffic inside the tunnel, there are additional performance considerations to address.

Even though this document describes encapsulating datagrams or messages inside a TCP stream, some protocols, such as ESP, themselves often encapsulate additional TCP streams, such as when transmitting data for a VPN protocol [[RFC8229](#)]. This introduces several potential sources of suboptimal behavior, as multiple TCP contexts act upon the same traffic.

For the purposes of this discussion, we will refer to the TCP encapsulation context as the "outer" TCP context, while the TCP context applicable to any encapsulated protocol will be referred to as the "inner" TCP context.

The use of an outer TCP context may cause signals from the network to be hidden from the inner TCP contexts. Depending on the signals that the inner TCP contexts use for indicating congestion, events that would otherwise result in a modification of behavior may go unnoticed, or may build up until a large modification of behavior is necessary. Generally, the main areas of concern are signals that

inform loss recovery, Bufferbloat and delay avoidance, and head of line blocking between streams.

[5.1](#). Loss Recovery

[5.1.1](#). Concern

The outer TCP context experiences packet loss on the network directly, while any inner TCP contexts present observe the effects of that loss on the delivery of their packets by the encapsulation layer. Furthermore, inner TCP contexts still observe direct network effects for any network segments that are traversed outside of the encapsulation, as is common with a VPN.

In this way, the outer TCP context masks packet loss from the inner

contexts by retransmitting encapsulated segments to recover from those losses. An inner context observes this as a delay while the packets are retransmitted rather than a loss. This can lead to spurious retransmissions if the recovery of the lost packets takes longer than the inner context's retransmission timeout (RTO). Since the outer context is retransmitting the packets to make up for the losses, the spurious retransmissions waste bandwidth that could be used for packets that advance the progress of the flows being encapsulated. A RTO event on an inner TCP context also hinders performance beyond generating spurious retransmissions, as many TCP congestion control algorithms dramatically reduce the sending rate after an RTO is observed.

When recovery from a loss event on the outer TCP context completes, the network or endpoint on the other end of the encapsulation will receive a potentially large burst of packets as the retransmitted packets fill in any gaps and the entire set of pending data can be delivered.

If content from multiple inner flows is shared within a single TCP packet in the outer context, the effects of lost packets from the outer context will be experienced by more than one inner flow at a time. However, this loss is actually shared by all inner flows, since forward progress for the entire encapsulation tunnel is generally blocked until the lost segments can be filled in. This is discussed further in [Section 5.3](#).

[5.1.2](#). Mitigation

Generally, TCP congestion controls and loss recovery algorithms are capable of recovering from loss events very efficiently, and the inner TCP contexts observe brief periods of added delay without much penalty.

A TCP congestion control should be selected and tuned to be able to gracefully handle extremely variable RTT values, which may already be the case for some congestion controls, as RTT variance is often greatly increased in mobile and cellular networks.

Additionally, use of a TCP congestion control that considers delay to be a sign of congestion may help the coordination between inner and outer TCP contexts. LEDBAT [[RFC6817](#)] and BBR

[\[I-D.cardwell-iccr-g-bbr-congestion-control\]](#) are two examples of delay based congestion control that an inner TCP context could use to properly interpret loss events experienced by the outer TCP context. Care must be taken to ensure that any TCP congestion control in use is also appropriate for an inner context to use on any network segments that are traversed outside of the encapsulation.

Since any losses will be handled by the outer TCP context, it might seem reasonable to modify the the inner TCP contexts' loss recovery algorithms to prevent retransmissions, there are often network segments outside of the encapsulated segments that still rely on the inner contexts' loss recovery algorithms. Instead, spurious retransmissions can be reduced by ensuring that RTT values are tuned such that the outer TCP context will fully time out before any inner TCP contexts.

[5.2.](#) Bufferbloat

[5.2.1.](#) Concern

"Bufferbloat", or delay introduced by consistently full large buffers along a network path [\[TSV2011\]](#) [\[BB2011\]](#), can increase observed RTTs along a network path, which can harm the performance of latency sensitive applications. Any spurious retransmissions sent on the network take place in queues that would otherwise be filled by useful data. In this case, any retransmission sent by an inner TCP context for a loss or timeout along the network segments also covered by the outer TCP context is considered to be spurious. This can pose a performance problem for implementations that rely on interactive data transfer.

Additionally, because there may be multiple inner TCP contexts being multiplexed over a single outer TCP context, even a minor reduction in sending rate by each of the inner contexts can result in a dramatic decrease in data sent through the outer context. Similarly, an increase in sending rate is also amplified.

[5.2.2.](#) Mitigation

Great care should be taken in tuning the inner TCP congestion control to avoid spurious retransmissions as much as possible. However, in order to provide effective loss recovery for the segments of the network outside the tunnel, the set of parameters used for tuning needs to be viable both inside and outside the tunnel. Adjusting the retransmission timeout (RTO) value for the TCP congestion control on the inner TCP context to be greater than that of the out TCP context will often help to reduce the number of spurious retransmissions generated while the outer TCP context attempts to catch up with lost or reordered packets.

In most cases, fast retransmit will be sufficient to recover from losses on network segments after the inner flows leave the tunnel, although loss events that trigger a full RTO on those last-mile segments will carry a higher penalty with such tuning. However, in many deployments, the last-mile segments will often observe lower loss rates than the first-mile segments, leading to a balance that often favors spurious retransmission avoidance on the first-mile over loss recovery speed on the last-mile.

[5.3.](#) Head of Line Blocking

[5.3.1.](#) Concern

Because TCP provides in-order delivery and reliability, even if there are multiple flows being multiplexed over the encapsulation layer, loss events, spurious retransmissions, or other recovery efforts will cause data for all other flows to back up and not be delivered to the client. In deployments where there are additional network segments to traverse beyond the encapsulation boundary, this may mean that flows are not delivered onto those segments until recovery for the outer TCP context is complete.

With UDP encapsulation, packet reordering and loss did not necessarily prevent data from being delivered, even if it was delivered out of order. Because TCP groups all data being encapsulated into one outer congestion control and loss recovery context, this may cause significant delays for flows not directly impacted by a recovery event.

Reordering on the network will also cause problems in this case, as it will often trigger fast retransmissions on the outer TCP context, blocking all inner contexts from being able to deliver data until the retransmissions are complete. However, a well behaved TCP will reorder the data that arrived out of order and deliver it before the

retransmissions arrive, reducing the detrimental impact of such reordering.

[5.3.2.](#) Mitigation

One option to help address the head of line blocking would be to run multiple tunnels, one for throughput sensitive flows and one for latency sensitive flows. This can help to reduce the amount of time that a latency sensitive flow can possibly be blocked on recovery for any other flow. Latency sensitive flows should take extra care to ensure that only the necessary amount of data is in flight at any given time.

Explicit Congestion Notification (ECN) ([\[RFC3168\]](#), [\[RFC5562\]](#)) could also be used to communicate between outer and inner TCP contexts during any recovery scenario. In a strategy similar to that taken by tunnelling of ECN fields in IP-in-IP tunnels [\[RFC6040\]](#), if an implementation supports such behavior, any ECN markings communicated to the outer TCP context by the network could be passed through to any inner TCP contexts transported by a given packet. Alternately, an implementation could elect to pass through such markings to all inner TCP contexts if a greater reduction in sending rate was deemed to be necessary.

[6.](#) Security Considerations

Any attacker on the path that observes the encapsulation could potentially discard packets from the outer TCP context and cause significant delays due to head of line blocking. However, an attacker in a position to arbitrarily discard packets could have a similar effect on the inner TCP context directly or on any other encapsulation schemes.

[7.](#) IANA Considerations

This document has no request to IANA.

[8.](#) Informative References

[BB2011] "Bufferbloat: Dark Buffers in the Internet", n.d..

[I-D.cardwell-iccr-g-bbr-congestion-control]
Cardwell, N., Cheng, Y., Yeganeh, S., and V. Jacobson,
"BBR Congestion Control", [draft-cardwell-iccr-g-bbr-congestion-control-00](#) (work in progress), July 2017.

Internet-Draft

TCP Encapsulation

July 2018

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", [RFC 3168](https://www.rfc-editor.org/info/rfc3168), DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", [RFC 5562](https://www.rfc-editor.org/info/rfc5562), DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", [RFC 6040](https://www.rfc-editor.org/info/rfc6040), DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", [RFC 6817](https://www.rfc-editor.org/info/rfc6817), DOI 10.17487/RFC6817, December 2012, <<https://www.rfc-editor.org/info/rfc6817>>.
- [RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", [RFC 8229](https://www.rfc-editor.org/info/rfc8229), DOI 10.17487/RFC8229, August 2017, <<https://www.rfc-editor.org/info/rfc8229>>.
- [TSV2011] "Bufferbloat: Dark Buffers in the Internet", March 2011.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

Eric Kinnear
Apple Inc.

One Apple Park Way
Cupertino, California 95014
United States of America

Email: ekinnear@apple.com

Pauly & Kinnear

Expires January 3, 2019

[Page 10]