TCP Maintenance and Minor
Extensions (tcpm)
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2007

L. Eggert
Nokia
F. Gont
UTN/FRH
March 5, 2007

TCP User Timeout Option
draft-ietf-tcpm-tcp-uto-05

Status of this Memo

Copyright Notice

Abstract

The TCP user timeout controls how long transmitted data may remain unacknowledged before a connection is forcefully closed. It is a local, per-connection parameter. This document specifies a new TCP option - the TCP User Timeout Option - that allows one end of a TCP connection to advertise its current user timeout value. This information allows the other end to adapt its user timeout accordingly. Increasing the user timeouts on both ends of a TCP

connection allows it to survive extended periods without end-to-end connectivity. Decreasing the user timeouts allows busy servers to explicitly notify their clients that they will maintain the connection state only for a short time without connectivity.

Table of Contents

1. Introduction

The Transmission Control Protocol (TCP) specification [RFC0793] defines a local, per-connection "user timeout" parameter that specifies the maximum amount of time that transmitted data may remain unacknowledged before TCP will forcefully close the corresponding connection. Applications can set and change this parameter with OPEN and SEND calls. If an end-to-end connectivity disruption lasts longer than the user timeout, no acknowledgments will be received for any transmission attempt, including keep-alives, and the TCP connection will close when the user timeout occurs.

In the absence of an application-specified user timeout, the TCP specification [RFC0793] defines a default user timeout of 5 minutes. The Host Requirements RFC [RFC1122] refines this definition by introducing two thresholds, R1 and R2 (R2 > R1), that control the number of retransmission attempts for a single segment. It suggests that TCP should notify applications when R1 is reached for a segment, and close the connection when R2 is reached. [RFC1122] also defines the recommended values for R1 (three retransmissions) and R2 (100 seconds), noting that R2 for SYN segments should be at least 3 minutes. Instead of a single user timeout, some TCP implementations offer finer-grained policies. For example, Solaris supports different timeouts depending on whether a TCP connection is in the SYN-SENT, SYN-RECEIVED, or ESTABLISHED state [SOLARIS-MANUAL].

Although some TCP implementations allow applications to set their local user timeout, TCP has no in-protocol mechanism to signal changes to the local user timeout to the other end. This causes local changes to be ineffective in allowing a connection to survive extended periods without connectivity, because the other end will still close the connection after its user timeout expires.

The ability to inform the other end about the local user timeout for the connection can improve TCP operation in scenarios that are currently not well supported. One example of such scenarios are

mobile hosts that change network attachment points based on current location. Such hosts, maybe using Mobile IP [RFC3344], HIP [RFC4423] or transport-layer mobility mechanisms [I-D.eddy-tcp-mobility], are only intermittently connected to the Internet. In between connected periods, mobile hosts may experience periods without end-to-end connectivity. Other factors that can cause transient connectivity disruptions are high levels of congestion or link or routing failures inside the network. In these scenarios, a host may not know exactly when or for how long connectivity disruptions will occur, but it might be able to determine an increased likelihood for such events based on past mobility patterns and thus benefit from using longer user timeouts. In other scenarios, the time and duration of a

connectivity disruption may even be predictable. For example, an orbiting node on a non-geostationary satellite might experience connectivity disruptions due to line-of-sight blocking by other planetary bodies. The timing of these events may be computable from orbital mechanics.

This document specifies a new TCP option - the TCP User Timeout Option - that allows one end of a TCP connection to advertise its current user timeout value. This information allows the other end to adapt its user timeout accordingly. Increasing the user timeouts on both ends of a TCP connection allows it to survive extended periods without end-to-end connectivity. Decreasing the user timeouts allows busy servers to explicitly notify their clients that they will maintain the connection state only for a short time without connectivity.

2. Conventions

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 [RFC2119].

3. Operation

Use of the TCP User Timeout Option can be enabled either on a per-application basis, e.g., through a socket option, or controlled by a system-wide setting. TCP maintains three per-connection state

variables to control the operation of UTO options, two of which (ENABLED and CHANGEABLE) are new:

ENABLED (Boolean)
: Flag that controls whether UTO options are enabled for a connection. Defaults to false.

LOCAL_UTO
: Local user timeout in effect for this connection. This is either the system-wide default or an application-specified value. Defaults to the system-wide default.

CHANGEABLE (Boolean)
: Flag that controls whether the local user timeout may be changed based on UTO options received from the other end. Defaults to true and becomes false when an application explicitly sets LOCAL_UTO.

Note that an exchange of UTO options between both ends of a

connection is not a binding negotiation. Transmission of a UTO option is a suggestion that the other end consider adapting its user timeout. This adaptation only happens if the the other end has explicitly enabled it (CHANGEABLE is true).

Before opening a connection, an application that wishes to use UTO options SHOULD enable their use by setting ENABLED to true. It MAY pick an appropriate local UTO by setting LOCAL_UTO, which is otherwise set to the system default. Finally, the application should determine whether it will allow the local UTO to change based on received UTO options from the other end. The default is to allow this for connections that do not have a specific user timeout concerns, i.e., connections that operate with the default LOCAL_UTO. If an application explicitly sets LOCAL_UTO, CHANGEABLE MUST become false, to prevent UTO options from the other end to override local application requests. Alternatively, applications MAY set or clear CHANGEABLE directly.

Performing these steps before an active or passive open causes UTO options to be exchanged in the SYN and SYN-ACK packets and is a reliable way to initially exchange and potentially adapt to UTO values. Systems MAY provide system-wide default settings for the

ENABLED, LOCAL_UTO and CHANGEABLE connection parameters when applications do not initialize them themselves.

In addition to exchanging UTO options in the SYN segments, a connection that has enabled UTO options SHOULD include a UTO option in the first packet that does not have the SYN flag set. This helps to minimize the amount of state information TCP must keep for connections in non-synchronized states, and is particularly useful when mechanisms such as "SYN cookies" [I-D.ietf-tcpm-syn-flood] are implemented, allowing a newly-established TCP connection to benefit from the information advertised by the UTO option, even if the UTO contained in the initial SYN segment was not recorded.

A host that supports UTO options SHOULD include it in the next possible outgoing segment whenever it starts using a new user timeout for the connection. This allows the other end to adapt its local user timeout for the connection accordingly.

A TCP implementation that does not support UTO options MUST silently ignore them [RFC1122], thus ensuring interoperability.

Hosts MUST impose upper and lower limits on the user timeouts they use for a connection. Section 3.1 discusses user timeout limits and discusses potentially problematic effects of user timeout settings.

3.1. Changing the Local User Timeout

When a host receives a TCP User Timeout Option, it must decide whether to change the local user timeout of the corresponding connection. If the CHANGEABLE flag is false, LOCAL_UTO MUST NOT be changed, regardless of the received UTO option. Without this restriction, UTO would modify TCP semantics, because application-requested UTOs could be overridden by peer requests. In this case, they SHOULD, however, notify the application about the user timeout value received from the other end.

In general, unless the application on the local host has requested a specific LOCAL_UTO for the connection, CHANGEABLE will be true and hosts SHOULD adjust the local user timeout in response to receiving a UTO option, as described in the remainder of this section.

The UTO option specifies the user timeout in in seconds or minutes, rather than in number of retransmissions or round-trip times (RTTs). Thus, the UTO option allows hosts to exchange user timeout values from 1 second to over 9 hours at a granularity of seconds, and from 1 minute to over 22 days at a granularity of minutes.

Very short user timeout values can affect TCP transmissions over high-delay paths. If the user timeout occurs before an acknowledgment for an outstanding segment arrives, possibly due to packet loss, the connection closes. Many TCP implementations default to user timeout values of a few minutes. Although the UTO option allows suggestion of short timeouts, applications advertising them should consider these effects.

Long user timeout values allow hosts to tolerate extended periods without end-to-end connectivity. However, they also require hosts to maintain the TCP state information associated with connections for long periods of time. Section 5 discusses the security implications of long timeout values.

To protect against these effects, implementations MUST impose limits on the user timeout values they accept and use. The remainder of this section describes a RECOMMENDED scheme to limit user timeouts based on upper and lower limits.

Under the RECOMMENDED scheme, and when CHANGEABLE is true, each end SHOULD compute LOCAL_UTO for a connection according to this formula:

LOCAL_UTO = min(U_LIMIT, max(LOCAL_UTO, REMOTE_UTO, L_LIMIT))

Each field is to be interpreted as follows:

U_LIMIT
: Current upper limit imposed on the user timeout of a connection by the local host.

L_LIMIT
: Current lower limit imposed on the user timeout of a connection by the local host.

REMOTE_UTO
   Last "user timeout" value received from the other end in a TCP User Timeout Option.

This means that, provided they are within the upper and lower limits, the maximum of current LOCAL_UTO and the last user timeout value received from the other end will become the new LOCAL_UTO for the connection. The rationale is that choosing the maximum of the two values will let the connection survive longer periods without end-to-end connectivity. If the end that announced the lower of the two user timeout values did so in order to reduce the amount of TCP state information that must be kept on the host, it can, nevertheless, close or abort the connection whenever it wants. [anchor3]

It must be noted that the two endpoints of the connection will not necessarily adopt the same user timeout.

Enforcing a lower limit (L_LIMIT) prevents connections from closing due to transient network conditions, including temporary congestion, mobility hand-offs and routing instabilities.

An upper limit (U_LIMIT) can reduce the effect of resource exhaustion attacks. Section 5 discusses the details of these attacks.

Note that these limits MAY be specified as system-wide constants or at other granularities, such as on per-host, per-user or even per-connection basis. Furthermore, these limits need not be static. For example, they MAY be a function of system resource utilization or attack status and could be dynamically adapted.

The Host Requirements RFC [RFC1122] does not impose any limits on the length of the user timeout. However, a time interval of at least 100 seconds is RECOMMENDED. Consequently, the lower limit (L_LIMIT) SHOULD be set to at least 100 seconds when following the RECOMMENDED scheme described in this section. Adopting a user timeout smaller than the current retransmission timeout (RTO) for the connection would likely cause the connection to be aborted unnecessarily. Therefore, the lower limit (L_LIMIT) MUST be larger than the current retransmission timeout (RTO) for the connection. It is worth noting that an upper limit may be imposed on the RTO, provided it is at

least 60 seconds [RFC2988].

3.2. UTO Option Reliability

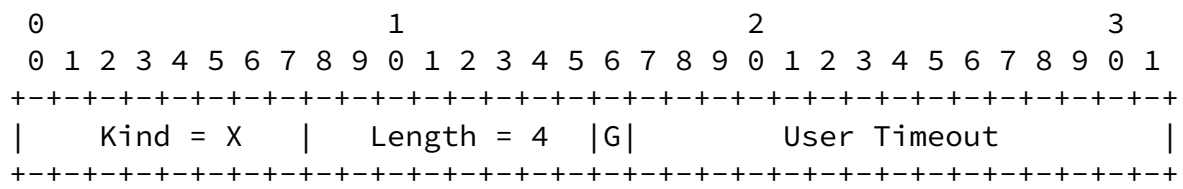
The TCP User Timeout Option is an advisory TCP option that does not change processing of subsequent segments. Unlike other TCP options, it need not be exchanged reliably. Consequently, the specification does not define a reliability handshake for UTO option exchanges. When a segment that carries a UTO option is lost, the other end will simply not have the opportunity to update its local UTO.

Implementations MAY implement local mechanisms to improve delivery reliability, such as retransmitting a UTO option when they retransmit a segment that originally carried it, or "attaching" the option to a byte in the stream and retransmitting the option whenever that byte or its ACK are retransmitted.

It is important to note that although these mechanisms can improve transmission reliability for UTO options, they do not guarantee delivery (a three-way handshake would be required for this). Consequently, implementations MUST NOT assume that UTO options are transmitted reliably.

3.3. Option Format

Sending a TCP User Timeout Option informs the other end of the current local user timeout for the connection and suggests that the other end adapt its user timeout accordingly. The user timeout value included in a UTO option contains the local user timeout (LOCAL_UTO) used during the synchronized states of a connection (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, or LAST-ACK). Connections in other states MUST use the default timeout values defined in [RFC0793] and [RFC1122].



(One tick mark represents one bit.)

Figure 1: Format of the TCP User Timeout Option

Figure 1 shows the format of the TCP User Timeout Option. It contains these fields:

Kind (8 bits)
A TCP option number [RFC0793] to be assigned by IANA upon publication of this document (see Section 6).

Length (8 bits)
Length of the TCP option in octets [RFC0793]; its value MUST be 4.

Granularity (1 bit)
Granularity bit, indicating the granularity of the "User Timeout" field. When set (G = 1), the time interval in the "User Timeout" field MUST be interpreted as minutes. Otherwise (G = 0), the time interval in the "User Timeout" field MUST be interpreted as seconds.

User Timeout (15 bits)
Specifies the user timeout (LOCAL_UTO) used for this connection. It MUST be interpreted as a 15-bit unsigned integer. The granularity of the timeout (minutes or seconds) depends on the "G" field.

3.4. Reserved Option Values

An empty TCP User Timeout Option, i.e., one with a "User Timeout" field of zero and a "Granularity" bit of either minutes (1) or seconds (0), is reserved for future use. TCP implementations MUST NOT send it and MUST ignore it upon reception.

4. Interoperability Issues

This section discusses interoperability issues related to introducing the TCP User Timeout Option.

4.1. Middleboxes

A TCP implementation that does not support the TCP User Timeout Option MUST silently ignore it [RFC1122], thus ensuring interoperability. In a study of the effects of middleboxes on transport protocols, Medina et al. have shown that the vast majority of modern TCP stacks correctly handle unknown TCP options [MEDINA]. In this study, 3% of connections failed when an unknown TCP option appeared in the middle of a connection. Because the number of failures caused by unknown options is small and they are a result of incorrectly implemented TCP stacks that violate existing requirements to ignore unknown options, they do not warrant special measures. Thus, this document does not define a mechanism to negotiate support

of the TCP User Timeout Option during the three-way handshake.

Stateful firewalls usually reset connections after a period of inactivity. If such a firewall exists along the path, it may close or abort connections regardless of the use of the TCP User Timeout Option. In the future, such firewalls may learn to parse the TCP User Timeout Option and modify their behavior - or the option contents - accordingly.

4.2. TCP Keep-Alives

Some TCP implementations, such as those in BSD systems, use a different abort policy for TCP keep-alives than for user data. Thus, the TCP keep-alive mechanism might abort a connection that would otherwise have survived the transient period without connectivity. Therefore, if a connection enables keep-alives that is also using the TCP User Timeout Option, then the keep-alive timer MUST be set to a value larger than that of the adopted USER TIMEOUT.

5. Security Considerations

Lengthening user timeouts has obvious security implications. Flooding attacks cause denial of service by forcing servers to commit resources for maintaining the state of throw-away connections. However, TCP implementations do not become more vulnerable to simple SYN flooding by implementing the TCP User Timeout Option, because user timeouts exchanged during the handshake only affect the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK), which simple SYN floods never reach.

However, when an attacker completes the three-way handshakes of its throw-away connections it can amplify the effects of resource exhaustion attacks, because the attacked server must maintain the connection state associated with the throw-away connections for longer durations. Because connection state is kept longer, lower-frequency attack traffic, which may be more difficult to detect, can already exacerbate resource exhaustion.

Several approaches can help mitigate this issue. First, implementations can require prior peer authentication, e.g., using

IPsec [RFC4301], before accepting long user timeouts for the peer's connections. Similarly, a host can start to accept long user timeouts for an established connection only after in-band authentication has occurred, for example, after a TLS handshake across the connection has succeeded [RFC4346]. Although these are arguably the most complete solutions, they depend on external mechanisms to establish a trust relationship.

A second alternative that does not depend on external mechanisms

would introduce a per-peer limit on the number of connections that may use increased user timeouts. Several variants of this approach are possible, such as fixed limits or shortening accepted user timeouts with a rising number of connections. Although this alternative does not eliminate resource exhaustion attacks from a single peer, it can limit their effects. Reducing the number of high-UTO connections a server supports in the face of an attack turns that attack into a denial-of-service attack against the service of high-UTO connections.

Per-peer limits cannot protect against distributed denial of service attacks, where multiple clients coordinate a resource exhaustion attack that uses long user timeouts. To protect against such attacks, TCP implementations could reduce the duration of accepted user timeouts with increasing resource utilization.

TCP implementations under attack may be forced to shed load by resetting established connections. Some load-shedding heuristics, such as resetting connections with long idle times first, can negatively affect service for intermittently connected, trusted peers that have suggested long user timeouts. On the other hand, resetting connections to untrusted peers that use long user timeouts may be effective. In general, using the peers' level of trust as a parameter during the load-shedding decision process may be useful. Note that if TCP needs to close or abort connections with a long TCP User Timeout Option to shed load, these connections are still no worse off than without the option.

Finally, upper and lower limits on user timeouts, discussed in Section 3.1, can be an effective tool to limit the impact of these sorts of attacks.

6. IANA Considerations

This section is to be interpreted according to [RFC2434].

This document does not define any new namespaces. It uses an 8-bit TCP option number maintained by IANA at http://www.iana.org/assignments/tcp-parameters.

7. Acknowledgments

The following people have improved this document through thoughtful suggestions: Mark Allman, Caitlin Bestler, David Borman, Bob Braden, Marcus Brunner, Wesley Eddy, Gorry Fairhurst, Abolade Gbadegesin, Ted Faber, Guillermo Gont, Tom Henderson, Joseph Ishac, Jeremy Harris,

Phil Karn, Michael Kerrisk, Dan Krejsa, Jamshid Mahdavi, Kostas Pentikousis, Juergen Quittek, Joe Touch, Stefan Schmid, Simon Schuetz, Tim Shepard and Martin Stiemerling.

Lars Eggert is partly funded by Ambient Networks, a research project supported by the European Commission under its Sixth Framework Program. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Ambient Networks project or the European Commission.

8. References

8.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

8.2. Informative References

[I-D.eddy-tcp-mobility]
Eddy, W., "Mobility Support For TCP", draft-eddy-tcp-mobility-00 (work in progress), April 2004.

[I-D.ietf-tcpm-syn-flood]
Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", draft-ietf-tcpm-syn-flood-01 (work in progress), December 2006.

[MEDINA] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", Proc. 4th ACM SIGCOMM/USENIX Conference on Internet Measurement , October 2004.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC3344] Perkins, C., "IP Mobility Support for IPv4", RFC 3344, August 2002.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4423] Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture", RFC 4423, May 2006.

[SOLARIS-MANUAL]
Sun Microsystems, "Solaris Tunable Parameters Reference Manual", Part No. 806-7009-10, 2002.

Editorial Comments

[anchor3] Lars: With the new CHANGEABLE flag, which prevents changing of LOCAL_UTO when an application has indicated that it cares about the value, I think the formula can become LOCAL_UTO = min(U_LIMIT, max(REMOTE_UTO, L_LIMIT)), i.e., we adopt whatever the other end suggests, given that it is with in acceptable limits. I didn't want to make this change without discussing it first, however.

Appendix A. Document Revision History

To be removed upon publication

Revision	Comments
05	Made behavior on when to change/not change the local UTO in response to incoming options consistent through the document. This required some reshuffling of text and also removed the need for the special "don't care" option value.
04	Clarified the results obtained by Medina et al. Added text to suggest inclusion of the UTO in the first non-SYN segment by the TCP that sent a SYN in response to an active OPEN.

03	Corrected use of RFC2119 terminology. Clarified how use of the TCP UTO is triggered. Clarified reason for sending a UTO in the SYN and SYN/ACK segments. Removed discussion of the SO_SNDTIMEO and SO_RCVTIMEO options. Removed text that suggested that a UTO should be sent upon receipt of an UTO from the other end. Required minimum value for the lower limit of the user timeout. Moved alternative solutions to appendix. Miscellaneous editorial changes.
02	Corrected terminology by replacing terms like "negotiate", "coordinate", etc. that were left from

		pre-WG-document times when the UTO was a more formalized exchange instead of the advisory one it is now. Application-requested UTOs take precedence over ones received from the peer (pointed out by Ted Faber). Added a brief mention of SO_SNDTIMEO and a slightly longer discussion of SO_RCVTIMEO.
01		Clarified and corrected the description of the existing user timeout in RFC793 and RFC1122. Removed distinction between operating during the 3WHS and the established states and introduced zero-second "don't care" UTOs in response to mailing list feedback. Updated references and addressed many other comments from the mailing list.
00		Resubmission of draft-eggert-gont-tcpm-tcp-uto-option-01.txt to the secretariat after WG adoption. Thus, permit derivative works. Updated Lars Eggert's funding attribution. Updated several references. No technical changes.

Authors' Addresses

Lars Eggert
Nokia Research Center
P.O. Box 407
Nokia Group 00045
Finland

Phone: +358 50 48 24461
Email: lars.eggert@nokia.com
URI: http://research.nokia.com/people/lars_eggert/

Fernando Gont
Universidad Tecnologica Nacional / Facultad Regional Haedo
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706

Argentina

Phone: +54 11 4650 8472
Email: fernando@gont.com.ar
URI: http://www.gont.com.ar/

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).