Network Working Group
INTERNET-DRAFT
Expires: May 2001

Reiner Ludwig
Ericsson Research

November 17, 2000

# The Eifel Algorithm for TCP
# <draft-ludwig-tsvwg-tcp-eifel-alg-00.txt>

## Status of this memo



## Abstract

TCP's intertwined error and congestion control is not robust against spurious timeouts nor is it robust against packet re-orderings. A packet that is delayed in the network beyond the expiration of TCP's retransmission timer, is mistaken for a packet loss by a TCP sender. Also, a packet that is re-ordered in the network beyond TCP's duplicate acknowledgment threshold, is eventually mistaken for a packet loss by a TCP sender. Both situations lead to a spurious retransmit of the oldest outstanding segment, and an unnecessary reduction of the congestion window at the sender. Moreover, a spurious timeout forces the sender into a go-back-N retransmission mode leading to spurious retransmits of all outstanding segments.

We propose the "Eifel algorithm" as a way to make TCP robust against spurious timeouts and packet re-orderings. The Eifel algorithm uses extra information in the ACKs to reliably detect (a posteriori) a spurious retransmit of the oldest outstanding segment at the TCP sender. In response to such a detection, the Eifel algorithm restores the congestion window, and prevents the spurious go-back-N retransmits following a spurious timeout. As extra information in the ACKs, the Eifel algorithm allows for two alternatives: the timestamp option [RFC1323] and/or two new flags in the Reserved field of the TCP header.

## 1. Introduction

In this document, we use the terms 'valid ACK' as defined in [RFC793], and the terms 'duplicate ACK' (DUPACK), 'Congestion Window' (cwnd), and 'Slow Start Threshold' (ssthresh) as defined in [RFC2581]. Further, our use of the term 'retransmit' includes both fast retransmits triggered by the third DUPACK and retransmits triggered by a timeout.

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 [RFC2119].

TCP's intertwined error and congestion control is not robust against spurious timeouts nor is it robust against packet re-orderings. A packet that is delayed in the network beyond the expiration of TCP's retransmission timer, is mistaken for a packet loss by a TCP sender. This results in a so-called spurious timeout, i.e., a timeout that would not have occurred had the sender "waited longer". Also, a packet that is re-ordered in the network beyond TCP's DUPACK threshold of 3, is eventually mistaken for a packet loss by a TCP sender. This is because the fast retransmit algorithm uses the arrival of 3 DUPACKs as an indication that a segment has been lost [RFC2581]. Both situations lead to a spurious retransmit of the oldest outstanding segment, and an unnecessary reduction of the congestion window at the sender. Moreover, a spurious timeout forces the sender into a go-back-N retransmission mode leading to spurious retransmits of all outstanding segments. A detailed explanation of these effects using trace plots is found in [LK00].

We propose the "Eifel algorithm" as a way to make TCP robust against spurious timeouts and packet re-orderings. The Eifel algorithm is based on the observation that the spurious go-back-N retransmits following a spurious timeout and the unnecessary reduction of the congestion window caused by packet re-ordering have the same root: the retransmission ambiguity. The retransmission ambiguity problem

[KP87] is the TCP sender's inability to distinguish an ACK for the original transmit of a segment from the ACK for its retransmit. The Eifel algorithm uses extra information in the ACKs to reliably detect

(a posteriori) a spurious retransmit of the oldest outstanding segment at the TCP sender. In response to such a detection, the Eifel algorithm restores the congestion window, and prevents the spurious go-back-N retransmits following a spurious timeout.

Spurious timeouts have not generally been a concern in the past since they are rare [Pax97]. This can be credited to the conservativeness of TCP's retransmission timer [LS00]. Yet, there is benefit in avoiding the detrimental effects that spurious timeouts have on TCP performance. This is since those effects create a strong incentive for keeping a conservative - potentially too conservative - retransmission timer. However, a retransmission timer that is too conservative may cause long idle times before a lost packet is retransmitted. This can degrade performance. This is obvious for interactive request/response-style connections. But it also affects bulk data transfers whenever the sender has exhausted its send window before the retransmission timer has expired. The Eifel algorithm opens the door to the development of a more optimistic retransmission timer as it ensures that the penalty for underestimating the round-trip time is minimal. In the common case, the only penalty is a single spurious retransmit.

Packet re-orderings can occur due to the connection-less nature of IP [RFC791] which does not guarantee an in-order delivery of packets. However, it is difficult to evaluate how serious this is in the Internet today. Early studies [Pax97] conclude that this occurs rarely, while recent studies [BPS99] find this problem to be more serious. Clearly, this depends on the paths underlying such studies, e.g., whenever routers are inter-connected via multiple links/paths (for fault tolerance) and load balancing is performed across those links/paths on the aggregate traffic, packet re-orderings will occur more frequently.

## 2. Detecting a Spurious Retransmit in TCP

Detecting the retransmission ambiguity requires extra information in the ACKs that the sender can use to unambiguously distinguish an ACK for the original transmit of a segment from that of a retransmit. This in turn requires that every segment and the corresponding ACK carry the extra information to allow the sender to avoid the spurious go-back-N retransmits described in Section 1. Waiting for the receiver to signal in DUPACKs that is has correctly received duplicate segments, as proposed in [RFC2883], would be too late, and is thus not an alternative.

As extra information in the ACKs, the Eifel algorithm allows for two alternatives: the timestamp option [RFC1323] and/or two new flags in

the Reserved field of the TCP header. Both alternatives are specified in the following two subsections. In Section 2.3, we specify precedence rules among those two alternatives. We speak of the

timestamp-based Eifel algorithm to emphasize that timestamps are being used to detect spurious retransmits. Likewise, we speak of the flag-based Eifel algorithm.

### 2.1. Detection based on the Xmit-Echo Flag

We define Bit 6 and Bit 7 in the Reserved field of the TCP header as the "Xmit flag" and "Xmit-Echo flag", respectively. The sender uses the Xmit flag to mark retransmits while the receiver sets the Xmit-Echo flag in the ACKs it sends in response to a segment with the Xmit flag set. Since, TCP can be a sender and receiver at the same time, two separate bits need to be used for those flags. It is worth noting that the two flags are similar to the sub-sequence field proposed in [ISO8073]. The location of the 6-bit Reserved field in the TCP header is shown in Figure 3 of [RFC793]. Bit 8 and 9 of the Reserved field have been assigned to the Explicit Congestion Notification (ECN) [RFC2481].

#### 2.1.1. TCP Initialization

The text in this subsection has been derived from Section 6.1.1 of [RFC2481]. This is because the same initialization semantics also apply to the flag-based Eifel algorithm. Thus, TCP's that support ECN, and wish to support the flag-based Eifel algorithm, should be able to re-use most of the initialization code implemented for ECN.

When a TCP sends a SYN packet, it MAY set (i.e., equal to 1) the Xmit and Xmit-Echo flag. For a SYN packet, the setting of both flags is defined as an indication that the sending TCP whishes to use the flag-based Eifel algorithm, rather than as an indication that the SYN packet is a retransmit. More precisely, such a SYN packet indicates that the TCP transmitting the SYN packet will participate in the flag-based Eifel algorithm as both a sender and receiver.

Only if a TCP receives a SYN packet with both the Xmit and Xmit-Echo flags set, MAY it respond with a SYN-ACK packet in which it sets the Xmit-Echo flag, but unsets (i.e., sets equal to 0) the Xmit flag. For a SYN-ACK packet, the pattern of the Xmit-Echo flag set and the Xmit flag unset is defined as an indication that the TCP transmitting the SYN-ACK packet agrees to participate in the flag-based Eifel algorithm as both a sender and receiver.

This asymmetry is necessary for the robust negotiation of the use of the flag-based Eifel algorithm with deployed TCP implementations (see section 6.1.1 of [RFC2481] for details).

For the TCP transmitting the SYN packet with both the Xmit and Xmit-Echo flags set, the flag-based Eifel algorithm has been successfully negotiated, if it receives a SYN-ACK packet in which the Xmit-Echo

flag is set, but the Xmit flag is unset. For the TCP transmitting the SYN-ACK packet, the flag-based Eifel algorithm has been successfully

negotiated, if it has received a SYN packet with both the Xmit and Xmit-Echo flags set, while it has set the Xmit-Echo flag but unset the Xmit flag in its SYN-ACK.

### 2.1.2. The TCP Receiver

If the flag-based Eifel algorithm has been successfully negotiated, the following rules apply.

The receiver SHOULD send an immediate ACK with the Xmit-Echo flag set in response to an incoming data segment that has the Xmit flag set. The immediate ACK is RECOMMENDED because of the range check that the sender performs on incoming ACKs after a retransmit (see Section 2.1.2).

In all other cases, the receiver SHOULD unset (i.e., set equal to 0) the Xmit-Echo flag in all ACKs it sends.

### 2.1.3. The TCP Sender

If the flag-based Eifel algorithm has been successfully negotiated, the following rules apply.

The sender MUST set the Xmit flag in the TCP header of retransmits. This is REQUIRED since otherwise the Eifel algorithm might get (falsely) triggered in response to a genuine packet loss (see Section 5). Recall, that our use of the term 'retransmit' includes both fast retransmits triggered by the third DUPACK and retransmits triggered by a timeout.

The sender MUST store in "cwnd_prev" the value that the sender's cwnd had before it is reduced when the retransmit occurs. Likewise, the sender MUST store in "ssthresh_prev" the value that the sender's ssthresh had before it is reduced when the retransmit occurs. This is REQUIRED since the sender will use cwnd_prev and ssthresh_prev to restore its cwnd and ssthresh after it has detected that the retransmit was spurious (see Section 3).

When a retransmit is sent, the sender MUST store the next (previously unsent) sequence number to be sent in "sent_high", i.e., sent_high is the highest outstanding sequence number transmitted so far plus 1, or alternatively, the ACK number of a valid ACK that would ack all outstanding data. This is REQUIRED since the sender will use sent_high to detect a spurious retransmit as described below. Note that the definition of "send_high" (spelled with a 'd') in [RFC2582] is different.

When the first valid ACK that is not a DUPACK arrives after a retransmit was sent, the sender detects that the retransmit was

spurious if all of the following conditions are true:

- the sender was expecting an ACK for a retransmit, and
- the ACK number of that ACK is less than or equal to sent_high, and
- that ACK does not have the Xmit-Echo bit set.

The range check implied by the second condition prevents that the Eifel algorithm is triggered in situations where a series of ACKs is lost and a cumulative ACK beyond sent_high acks the retransmit.

### 2.2. Detection based on Timestamps

The timestamp-based Eifel algorithm requires that both the sender and receiver have correctly implemented the timestamp option as specified in [RFC1323]. In addition, the TCP sender implementation needs to be enhanced as specified in Section 2.2.2. No change to the TCP protocol is required nor any change to the TCP receiver implementation.

#### 2.1.1. TCP Initialization

The timestamp-based Eifel algorithm has been successfully negotiated, if use of the timestamp option has been successfully negotiated during connection setup (see [RFC1323]).

#### 2.1.2. The TCP Receiver

No change is required beyond the implementation of the timestamp option as specified in [RFC1323].

#### 2.1.3. The TCP Sender

If the timestamp-based Eifel algorithm has been successfully negotiated, the following rules apply.

The sender MUST store in "ts_first_xmit" the timestamp of the first retransmit for a data segment. This is REQUIRED since otherwise the Eifel algorithm might get (falsely) triggered in response to a genuine packet loss (see Section 5). For the same reason, any subsequent retransmit for the same oldest outstanding sequence number MUST NOT overwrite ts_first_xmit. Recall, that our use of the term 'retransmit' includes both fast retransmits triggered by the third DUPACK and retransmits triggered by a timeout.

As with the flag-based Eifel algorithm, the sender MUST store in cwnd_prev the value that the sender's cwnd had before it is reduced when the retransmit occurs. Likewise, the sender MUST store in ssthresh_prev the value that the sender's ssthresh had before it is reduced when the retransmit occurs.

When the first valid ACK that is not a DUPACK arrives after a

retransmit was sent, the sender detects that the retransmit was spurious if all of the following conditions are true:

- the sender was expecting an ACK for a retransmit, and
- the value of the Timestamp Echo Reply field in the timestamp option field of that ACK is less than ts_first_xmit.

Using the comparison operator "less than" in the second condition is conservative. In theory, when the timestamp clock is slow or the network is fast, ts_first_xmit could (at most) also be equal to the value of the Timestamp Echo Reply field in the timestamp option field of the first or a subsequent ACK that acks the retransmit. Thus, in such a case the sender assumes that the retransmit was a genuine retransmit, i.e., that it was not spurious.

### 2.3. Timestamps or the Xmit-Echo Flag?

The advantage of using the timestamp-based over the flag-based Eifel algorithm is that it does not require changes to the TCP protocol nor to the TCP receiver implementation. Also, [RFC1323] is already a standards track document, and the timestamp option has already been widely deployed in TCP implementations [???].

The disadvantage of using the timestamp-based over the flag-based Eifel algorithm is that including the 12 bytes TCP timestamp option field in every segment and the corresponding ACKs introduces extra protocol overhead. Moreover, current TCP/IP header compression schemes [RFC1144], [RFC2507] do not compress timestamp option fields. For those reason, a sender might not choose to negotiate the timestamp option. Note, that timestamps are only required for the PAWS mechanism (Protect Against Wrapped Sequences) [RFC1323], since for the RTTM mechanism (Round Trip Time Measurement) [RFC1323] there exist implementation alternatives that work without the timestamp option field [LS00].

The flag-based Eifel algorithm has none of the above mentioned disadvantages, but instead requires changes to the TCP protocol (two new flags in the Reserved field of the TCP header) and to the TCP receiver implementation. Hence, we define the following precedence rules that would allow for an incremental deployment of the Eifel algorithm.

- A TCP sender that implements the timestamp option SHOULD also implement the timestamp-based Eifel algorithm.

- A TCP sender SHOULD implement the flag-based Eifel algorithm and SHOULD try to negotiate its use during connection setup. There are situations where it might be advisable to deviate from this rule (see Section 5).

- If a receiver correctly sets the Xmit-Echo flag (see Section5),

the operation of the Eifel algorithm SHOULD be based on the Xmit-Echo flag independent of whether timestamps are also being used.

In case timestamps are also being used, a sender MAY use timestamps as an additional check to verify whether a retransmit was spurious. This rule implies that the negotiation to use the Xmit-Echo flag has succeeded.

- If timestamps are being used but the Xmit-Echo flag is not being used for a particular TCP connection, the Eifel algorithm SHOULD be operated based on timestamps. This rule implies that either the negotiation to use the Xmit-Echo flag has failed or it's use has been turned off due to a broken receiver (see Section 5).

## 3. Responding to a Spurious Retransmit in TCP

If the flag- or timestamp-based Eifel algorithm has been successfully negotiated, the following rules apply.

When a sender has detected that it has performed a spurious retransmit, the sender resumes transmission with the next unsent segment. In addition, it performs one of the following two actions based on the definition of cwnd_prev and ssthresh_prev provided in Section 2.1.2 and 2.2.2.

- If only a single retransmit had been sent, the sender SHOULD restore cwnd with cwnd_prev and ssthresh with ssthresh_prev.

- If two retransmits of the same oldest outstanding sequence number had been sent, the sender SHOULD restore both cwnd and ssthresh with one half the value of cwnd_prev.

If more than two retransmits of the same oldest outstanding sequence number had been sent, the Eifel algorithm has no effect on cwnd and ssthresh.

Note that in the case of packet re-ordering, the ACK that was used to detect that the retransmit was spurious (see Section 2.1.2 and 3.2.2), will usually clock out a burst of segments. The size of that burst is equal to the number of DUPACKs that did not clock out a new segment during the first phase of the fast recovery phase when cwnd is inflated [RFC2581].

Responding to a spurious retransmit as specified above is visualized in [LK00] using trace plots. This might aid a better understanding.

## 4. Avoiding Competition between Timeout- and DUPACK-based Error Recovery

In the spirit of the Eifel algorithm, although unrelated to spurious retransmits, we propose the following rule, based on the definition

of cwnd_prev and ssthresh_prev provided in Section 2.1.2 and 2.2.2.

The rule applies to the case when the third DUPACK arrives after the first timeout for the same oldest outstanding sequence number has already occurred. In that case, the sender SHOULD suppress the fast retransmit and SHOULD restore both cwnd and ssthresh with one half the value of cwnd_prev. I.e., the sender restores cwnd and ssthresh as if the timeout had not occurred, but instead goes into congestion avoidance [RFC2581].

## 5. Security Considerations

There is a considerable risk when implementing the Eifel algorithm in a naive fashion. This is since a misbehaving receiver can severely upset congestion control at the sender. The risk is that the Eifel algorithm is (falsely) triggered in response to a genuine packet loss. In that case the Eifel algorithm would mistake a genuine retransmit as a spurious retransmit. As a consequence the sender would effectively not reduce its congestion window in response to the lost packet. However, there are reliable sender-side mechanisms to protect against this case as outlined below.

One needs to distinguish between broken receivers that misbehave due to an implementation mistake versus malicious receivers that deliberately misbehave. We first describe two mechanism to protect against broken receivers, followed by a different mechanism to protect against malicious receivers. We only recommend that protection against broken receivers SHOULD be implemented at the sender. This is motivated by the fact that the current TCP implicitly assumes a trust relationship between sender and receiver, i.e., it can be assumed that receivers are not malicious. Note that even without the Eifel algorithm, there are ways a misbehaving receiver can upset congestion control at the sender [SCWA99].

We do not discuss problems with respect to misbehaving senders assuming that the implementation of the sender-side Eifel algorithm complies with the specifications in this text.

Protection Against Broken Receivers

There is no risk to falsely trigger the timestamp-based Eifel algorithm as long as the receiver correctly implements the timestamp option [RFC1323]. However, the flag-based Eifel algorithm can be falsely triggered when the receiver has agreed to set the Xmit-Echo flag in ACKs for retransmits but then "forgets" to do so. The ACK for a genuine retransmit would then falsely trigger the Eifel algorithm once it arrives at the sender. To protect against this case the sender SHOULD implement the following mechanism if it uses the flag-based Eifel algorithm.

After the sender has detected a spurious retransmit and in response restores cwnd and ssthresh with cwnd_prev and ssthresh_prev,

respectively (see Section 3), it also saves the former values of cwnd and ssthresh in cwnd_prev and ssthresh_prev, respectively (e.g., help = cwnd; cwnd = cwnd_prev; cwnd_prev = help;). Until an ACK arrives at the sender that acks beyond sent_high, the sender checks for a valid ACK that arrives with the Xmit-Echo flag set. If such an ACK arrives, the sender assumes that the Eifel algorithm was rightful triggered and does nothing further. Otherwise, when an ACK arrives at the sender that acks beyond sent_high, the sender assumes that the Eifel algorithm was falsely triggered and reverses the effects of the Eifel algorithm. I.e. cwnd is (re-)restored to cwnd_prev and ssthresh is (re-)restored to ssthresh_prev. Since, also the ACKs with the Xmit-Echo flag set can get lost, it would be too conservative to completely disable the Eifel algorithm for the rest of the connection in this situation.

Another risk stems from receivers that set the Xmit-Echo flag for segments that have not been retransmitted. A TCP sender SHOULD implement an appropriate detection mechanism, since in this case, it cannot reliably use the flag-based Eifel algorithm. If a sender detects such misbehavior, it SHOULD disable the flag-based Eifel algorithm for the rest of the connection.

Protection Against Malicious Receivers

There are number of ways in which a malicious receiver could falsely trigger the Eifel algorithm at the sender. A sender is particularly vulnerable to this if it operates the flag-based Eifel algorithm. Hence, the sender MAY choose to only use the timestamp-based Eifel algorithm, and in addition implement the following mechanism. The mechanism combines the idea of a "singular nonce" proposed in [SCWA99] with the timestamp option specified in [RFC1323].

The mechanism is based on the observation that after a spurious retransmit the sender will at some point receive the ACK that was triggered by the corresponding original transmit assuming that that ACK was not lost. That ACK can be expected to echo the timestamp of the original transmit even if the receiver implements the delayed ACK algorithm. In case of packet re-ordering, this is implied by the rules for generating ACKs for data segments that fill in all or part of a gap in the sequence space (see section 4.2 of [RFC2581]) and by the rules for echoing timestamps in that case (see rule (C) in section 3.4 of [RFC1323]). In case of a spurious timeout, it is quite likely that the delay that has caused the spurious timeout has also caused the receiver's delayed ACK timer [RFC1122] to expire. Hence, to protect against a malicious receiver, the sender should only trigger the Eifel algorithm in response to the ACK for the original transmit and only after it has authenticated that ACK as described below.

To protect against malicious receivers that spoof ACKs, the sender MAY implement the following modification to the timestamp option

specified in [RFC1323]. The sender adds a separate random number to each timestamp to be included in an outgoing segment, and writes the result into the Timestamp Value field in the timestamp option field. A new random number is generated per RTT to avoid that the receiver "learns" the random number. That random number should be carefully chosen to avoid bad interactions with the PAWS mechanism specified in [RFC1323]. Clearly, the random number(s) need to be accounted for in the RTTM mechanism specified in [RFC1323], i.e., it needs to be subtracted from the echoed timestamp before the RTT can be calculated. For this to work, the sender needs to store all "outstanding timestamps" and the corresponding random number. With this modification, the sender only identifies a retransmit as spurious if the ACK for the original transmit echoes the "random timestamp" that was sent. Thus, assuming a receiver has no easy access to the mentioned random numbers, this should provide for a fairly secure protection against malicious receivers that spoof the "right" ACK that would trigger the Eifel algorithm.

## Acknowledgments

Many thanks to Keith Sklower for helping to develop the tools that allowed the study of spurious timeouts and packet re-orderings. Many thanks to Randy Katz, Michael Meyer, Stephan Baucke, Sally Floyd, and Vern Paxson for discussions around the Eifel algorithm.

## References

[RFC2581] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, RFC 2581, April 1999.

[BPS99] J.C.R. Bennett, C. Partridge, N. Shectman, Packet Reordering is Not Pathological Network Behavior, IEEE/ACM Transactions on Networking, December `99.

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

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

[RFC2507] M. Degermark, B. Nordgren, S. Pink, IP Header Compression, RFC 2507, February 1999.

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, A. Romanow, An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883, July 2000.

[ISO8073] ISO/IEC, Information processing systems - Open Systems

Interconnection - Connection oriented transport protocol
specification, International Standard ISO/IEC 8073,

December 1988.

[RFC1323] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992.

[RFC1144] V. Jacobson, Compressing TCP/IP Headers for Low-Speed Serial Links, RFC 1144, February 1990.

[KP87] P. Karn, C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, In Proceedings of ACM SIGCOMM 87.

[LK00] R. Ludwig, R. H. Katz, The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions, ACM Computer Communication Review, Vol. 30, No. 1, January 2000, available at http://www.acm.org/sigcomm/ccr/archive/2000/jan00/ccr-200001-ludwig.html (easier studied when viewed/printed in color).

[LS00] R. Ludwig, K. Sklower, The Eifel Retransmissions Timer, ACM Computer Communication Review, Vol. 30, No. 3, July 2000.

[Pax97] V. Paxson, End-to-End Routing Behavior in the Internet, IEEE/ACM Transactions on Networking, Vol.5, No.5, October 1997.

[RFC791] J. Postel, Internet Protocol, RFC 791, September 1981.

[RFC793] J. Postel, Transmission Control Protocol, RFC793, September 1981.

[SCWA99] S. Savage, N. Cardwell, D. Wetherall, T. Anderson, TCP Congestion Control with a Misbehaving Receiver, ACM Computer Communication Review, Vol. 29, No. 5, October 1999.

Author's Address

Reiner Ludwig
Ericsson Research (EED)
Ericsson Allee 1
52134 Herzogenrath, Germany
Phone: +49 2407 575 719
Fax: +49 2407 575 400
Reiner.Ludwig@Ericsson.com

This Internet-Draft expires in May 2001.