TCPM Working Group
Internet-Draft
Intended status: Experimental
Expires: August 25, 2008

S. Schuetz
NEC
N. Koutsianas
L. Eggert
Nokia
W. Eddy
Verizon
Y. Swami
Nokia
K. Le
NSN

February 22, 2008

TCP Response to Lower-Layer Connectivity-Change Indications
draft-schuetz-tcpm-tcp-rlci-03

Status of this Memo

Copyright Notice

Abstract

When the path characteristics between two hosts change abruptly, TCP can experience significant delays before resuming transmission in an efficient manner or TCP can behave unfairly to competing traffic. This document describes TCP extensions that improve transmission behavior in response to advisory, lower-layer connectivity-change indications. The proposed TCP extensions modify the local behavior of TCP and introduce a new TCP option to signal locally received connectivity-change indications to remote peers. Performance gains result from a more efficient transmission behavior and there is no difference in aggressiveness in comparison to a newly-started connection.

Table of Contents

1. Introduction

The Transmission Control Protocol (TCP) [RFC0793] generally assumes that the end-to-end path between two hosts has characteristics that are relatively stable over the lifetime of a connection. Although TCP's congestion control algorithms [RFC2581] can adapt to changes to the path characteristics after several round-trip times, they fail to support efficient operation in the few round-trip times immediately after a significant path change. This is due to the granularity of TCP's sampling mechanisms. Significant changes to path connectivity include loss or reestablishment of connectivity, and drastic, abrupt changes in round-trip time (RTT) or available bandwidth. Connectivity changes that occur on such short time-scales are becoming more common, due to host mobility or intermittent network attachment.

This document describes a set of complementary TCP extensions that improve behavior when transmitting over paths whose characteristics can change on short time-scales. TCP implementations that support these extensions respond to receiving generic, link-technology-independent, per-connection connectivity-change indications from lower layers. A connectivity-change indication signals that the characteristics of the end-to-end path between the local node and its peer have changed in some undefined way. The response mechanisms proposed for TCP act on this information in a conservative fashion. The specific response depends on the current state of a connection when a connectivity-change indication is received.

It is important to note that this addition of response mechanisms to lower-layer information is following an established precedent. TCP and other transport protocols already react to information and signals from lower layers; the proposed connectivity-change indications thus extend an established interface between layers in the protocol stack. TCP measures the end-to-end path to implicitly derive network-layer information. TCP also directly reacts to network-layer signals delivered via ICMP, for example, "Port Unreachable" or the now-deprecated "Source Quench" [RFC1122]. Explicit Congestion Notification (ECN) [RFC3168] and Quick-Start [RFC4782] are other sources of network-layer information for which response mechanisms for TCP have been defined. Connectivity-change indications are yet another source of lower-layer information that TCP can use to improve its operation.

A second important point to note is that the TCP response mechanisms to connectivity-change indications are purely optional efficiency improvements. In the absence of connectivity-change indications, a TCP that implements these changes behaves identically to an unmodified TCP. When lower layers provide connectivity-change

indications that trigger the response mechanisms, they enhance TCP operation based on the explicit lower-layer information that is signaled. These response mechanisms do not increase the aggressiveness of TCP.

Note that the IAB has recently described architectural issues of "link indications" [RFC4907]. The authors feel that this term is not quite accurate in this environment, because transport mechanisms should remain link-technology-agnostic. However, transport protocols have always acted on network-layer information and signals, such as measured path characteristics or ICMP-signaled conditions. Because of the growing proliferation of shim layers between the traditional network and transport layers, this document uses the term "lower-layer indication" to remain independent of specific network or shim layers.

Note that it is currently an open question as to whether additional lower-layer indications can provide further information to transport protocols. Also, this document only describes response mechanisms for TCP, although other transport protocols may benefit from similar response mechanisms to react to connectivity-change indications.

2. Terminology

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].

The following abbreviations are used throughout the document:

CCI	Connectivity-Change Indication
RLCI	Response to Lower-layer Connectivity-change Indications

Table 1: Abbreviations

3. Motivation and Overview

Several proposed network-layer extensions support host mobility, including Mobile IPv4 [RFC3344], Mobile IPv6 [RFC3775] and HIP [I-D.ietf-hip-mm]. Typically, they shield transport-layer protocols from mobility events and enable them to sustain established connections across mobility events. However, the path characteristics that established connections experience after a mobility event may have changed drastically and on short time-scales.

Congestion control, RTT and path-MTU state gathered over an old path before the move generally have no meaning for the new path. Because TCP uses stale information when resuming transmission over the new path, it can be either too aggressive or highly inefficient. Similar conditions may be found when fail-overs occur for multihomed hosts through the shim6 protocol. Some background on the types of scenarios that the technology described in this document is designed to work within is found in Appendix A.

TCP already forces a slow-start restart in some cases where the network state becomes unknown, such as after an idle period or heavy losses. A first part of the response specified in this document involves a similar return to initial slow-start state in response to connectivity-change indications that are received while a connection is transmitting in steady-state. Note that this behavior is more conservative than the standard TCP response or lack of response. Some performance gains with the proposed mechanisms are due to either avoiding overloading the new path, which typically incurs an RTO, or using slow-start to quickly detect new capacity far above the point where steady-state had previously been near.

A second response component improves TCP operation in the presence of temporary connectivity disruptions. These disruptions can occur independently of mobility events and, for example, may be due to insufficient wireless access coverage or nomadic computer use. Connectivity disruptions can severely decrease TCP performance. The main reason for this decrease is TCP's retransmission behavior after a connectivity disruption [SCHUETZ]. TCP uses periodic retransmission attempts in exponentially increasing intervals, which can unnecessarily delay retransmissions after connectivity returns. In the extreme case, TCP connections can even abort, if the disruption is longer than the TCP "user timeout". (Connection aborts are out of scope for this document but can be prevented by the TCP User Timeout Option [I-D.ietf-tcpm-tcp-uto].)

This second response action executes when receiving a connectivity-change indication while a connection is stalled in exponential back-off. It improves TCP retransmission behavior after connectivity is restored through an immediate speculative retransmission attempt [footnote-1]. Similar to the first response component, the second one also increases TCP performance through a more intelligent transmission behavior that uses periods of connectivity more efficiently. In comparison to startup of a new connection, it does not cause significant amounts of additional traffic and it does not change TCP's congestion control algorithms.

Finally, this draft specifies a third response component, which is a new TCP option that notifies the connection's remote peer of a

connectivity-change event detected locally. This is useful because connectivity-change indications typically require appropriate responses at both ends of a connection, but may only be received or detected by one end. The other parts of the response to a connectivity-change indication are independent of the indication's source (locally notified or remotely signaled) and depend only on the specific indication and the state of the connection for which it was received.

4. Connectivity-Change Indications

The focus of this document is on specifying TCP response mechanisms to lower-layer connectivity-change indications. This section briefly describes how different network- and shim-layer mechanisms underneath the transport layer may provide these connectivity-change indications to TCP. This section is included for clarification only; details on connectivity indication sources are out of scope of this document.

When lower layers detect a connectivity-change event, they generate corresponding connectivity-change indications. Lower-layer events that could trigger such an indication include (but are not limited to):

o the IP address of the local outbound interface used for a given connection has changed, e.g., due to DHCP [RFC2131] or IPv6 router advertisements [RFC2460];

o link-layer connectivity of the local outbound interface used for a given connection has changed, e.g., link-layer "link up" event [RFC4957];

o the local outbound interface used for a given connection has changed, due to routing changes or link-layer connectivity changes at other interfaces (including tunnel establishment or teardown, e.g., in response to IKE events [RFC4306]);

o a Mobile IP binding update has completed [RFC3775];

o a HIP readdressing update has completed [I-D.ietf-hip-mm];

o a path-change signal from the network has arrived (possible in theory, depends on network capabilities);

o other notifications as defined by the IETF's Detecting Network Attachment (DNA) working group have occurred [RFC4957].

Note that the list above only describes some potential sources for

connectivity-change events. Other sources exist, but the details on when to generate such events are out of the scope of this document, which focuses on the TCP response mechanisms when such events are received.

5. TCP Response to Connectivity-Change Indications (CCIs)

A TCP connection can receive a connectivity-change indication (CCI) either from its local stack ("local CCI") or through a new "connectivity-change indication TCP option" from its peer ("remote CCI"). Section 5.1 specifies this new TCP option. In either case, upon reception of a CCI, the TCP RLCI (Response to Lower-layer Connectivity-change Indications) mechanisms defined in this document immediately re-probe path characteristics. They do this by either performing a speculative retransmission or by sending a single segment of new data or a pure ACK, depending on whether the connection is currently stalled in exponential back-off or transmitting in steady-state, respectively. A connection is "stalled in exponential back-off", if at least one segment was retransmitted due to a RTO expiration but has not been ACK'ed yet.

The remainder of this section first defines the format of the new CCI TCP option in Section 5.1 and its processing in Section 5.2. After that, the two TCP response mechanisms triggered by receiving CCIs - re-probing path characteristics and speculative retransmission - are described in Section 5.3 and Section 5.4.

The TCP RLCI mechanisms defined in this document depend on the TCP Timestamps option (TSopt) [RFC1323]. Consequently, it is REQUIRED that an end host that wishes to use the RLCI mechanisms for a TCP connection negotiate the use of TCP Timestamps options with its peer. If this negotiation fails, a host MUST NOT use the RLCI mechanisms for a connection. TCP Timestamps options are needed by the RLCI mechanisms during the following operations:

- To re-probe the path characteristics after a connectivity-change indication. A host uses the TS Echo Reply (TSecr) field of a TCP Timestamps option to distinguish whether incoming ACKs are for segments that have been transmitted before or after CCI.

- To identify a new remote CCI. A host uses the TS Value (TSval) field of an incoming TCP Timestamps option to distinguish a new remote CCI from the delayed reception of an old one. As a result, last remote CCI is defined as the one received with the highest TS Value.

Section 5.2 and Section 5.3 give more details about how the RLCI

mechanisms use TCP Timestamps options.

An implementation of the RLCI mechanisms defined in this document maintains nine new state variables per TCP connection. [footnote-2]

LOCAL_CCI
It is a 1-bit counter, having an initial value of 0. It is used for distinguishing the existence of a new local CCI. It changes its value every time a new local CCI received from the local stack starts being processed.

REMOTE_CCI
It holds a copy of the last CCI value advertised by the peer through a CCI TCP option. This is a 1-bit counter initialized to 0 and gets updated in response to remote CCIs according to the rules defined in Section 5.2.

LOCAL_CCI_STATUS
It holds the status of the processing of local CCIs. It can have three possible values: LOCAL_CCI_IDLE (0), LOCAL_CCI_NEW (1), LOCAL_CCI_ECHO_ACK (2). The initial value is LOCAL_CCI_IDLE.

REMOTE_CCI_STATUS
It holds the status of the processing of the last remote CCI advertised by the peer through a CCI TCP option. It can have two possible values: REMOTE_CCI_IDLE (0), REMOTE_CCI_ECHO (1). The initial value is REMOTE_CCI_IDLE.

LAST_CCI_TIME
It holds the local time when the last CCI (either local or remote) was received. It is updated every time either LOCAL_CCI or REMOTE_CCI is modified.

REMOTE_CCI_PEER_TIME
This variable is used in order to distinguish new remote CCIs from the retransmissions of the past ones. It holds the TS Value (TSval) of the Timestamps option of the segment advertising the last remote CCI. It is initialized when receiving the first segment from the peer and it is updated every time REMOTE_CCI is modified.

LOCAL_CCI_PEER_ECHO_TIME
This variable is used in order to distinguish the echo of a new local CCI from delayed retransmissions of echoes of older local CCIs. It holds the TS Value (TSval) of the Timestamps option of the segment that echoed the last local CCI. It is initialized when receiving the first segment from the peer and it is updated every time LOCAL_CCI_STATUS changes from LOCAL_CCI_NEW to

LOCAL_CCI_ECHO_ACK.

CCI_SNDMAX
Retains the highest sequence number transmitted when the most recent CCI (either local or remote) was received.

CCI_CONTROLLED_CWND
It is a Boolean variable that sets an additional condition controlling the increment of TCPs congestion window (CWND). Having an initial value of false, it is updated according to the rules defined in Section 5.2.

5.1. Connectivity-Change Indication (CCI) TCP Option

Connectivity-change indications (CCIs) are generally asymmetric, i.e., they may occur or be detected by one end but not the other. The basic idea behind the CCI option is to signal the occurrence of local CCIs to the other end, in order to allow also the other end to respond appropriately. Note that this assumes that paths will generally be symmetric, meaning that a CCI received by one end for its path to the other end will imply that the characteristics of the reverse path have changed, too.

```
                            1                   2
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
         +---------------+---------------+-----+-+-+---+-+
         |               |               |  R  | | |   |E|
         |   Kind = X    |  Length = 3   |  E  |C|E| C |C|
         |               |               |  S  | |C| S |S|
         +---------------+---------------+-----+-+-+---+-+
```

Figure 1: Format of the connectivity-change indication TCP option.

Figure 1 shows the format of the CCI option. It contains these fields:

Kind (8 bits)
The TCP option number X [RFC0793] allocated by IANA upon publication of this document (see Section 8).

Length (8 bits)
Length of the TCP option in octets [RFC0793]; its value MUST be 3.

RES (3 bits)
Reserved bits. The sender SHOULD set these to zero and the receiver MUST ignore them.

C (1 bit)
Current value of LOCAL_CCI of the end sending the option.

EC (1 bit)
Echoed value of C, i.e., the current value of REMOTE_CCI of the end sending the option.

CS (2 bit)
Current value of LOCAL_CCI_STATUS of the end sending the option.

ECS (1 bit)
Current value of REMOTE_CCI_STATUS of the end sending the option.

The CCI option contains two single-bit fields (C and EC) used to distinguish new CCIs from delayed retransmissions of past ones. It also contains some flags representing the status of each CCI processing. These flags are used for a 3-way handshake ensuring that both parties have been informed of a new CCI. At the beginning of a connection, LOCAL_CCI and REMOTE_CCI MUST be set to 0. LOCAL_CCI_STATUS and REMOTE_CCI_STATUS MUST be set to LOCAL_CCI_IDLE and REMOTE_CCI_IDLE, respectively.

A host actively opening a connection and wishing to use the CCI option for that connection MUST include a CCI option in its SYN segment with C := 0, CS := LOCAL_CCI_IDLE, EC := 0 and ECS := REMOTE_CCI_IDLE in order to advertise support for the TCP CCI option. A host receiving a SYN segment MUST NOT include a CCI option in its SYN-ACK or any subsequent segment, unless it has received a CCI option in the corresponding SYN. In case a host has received a CCI option in the SYN segment, it MUST echo that CCI option in its SYN-ACK segment, i.e., it MUST set C := 0, CS := LOCAL_CCI_IDLE, EC := 0 and ECS := REMOTE_CCI_IDLE. A host MUST NOT process any following CCI options unless one was included in both the SYN and SYN-ACK and both peers have enabled TCP Timestamps for the connection. Section 5.2.1 and Section 5.2.2 describe the processing rules in detail.

A host MUST send a CCI option in all outgoing segments whenever LOCAL_CCI_STATUS is not LOCAL_CCI_IDLE or REMOTE_CCI_STATUS is not REMOTE_CCI_IDLE (or both). A host MUST NOT send a CCI option when LOCAL_CCI_STATUS is LOCAL_CCI_IDLE and REMOTE_CCI_STATUS is REMOTE_CCI_IDLE, i.e., when the host is not currently processing any CCI. The only exceptions to that rule are SYN and SYN-ACK segments. Whenever sending any CCI option, C MUST be set to the current LOCAL_CCI, EC MUST be set to the current REMOTE_CCI, CS MUST be set to LOCAL_CCI_STATUS and ECS MUST be set to REMOTE_CCI_STATUS, respectively.

5.2. Generation and Processing of Connectivity-Change Indication TCP Options

Processing of a connectivity-change indication can be separated into two parts:

1. Processing in "initiator" mode, i.e., when a host receives a local CCI and (reliably) forwards it to the other end through a CCI option.

2. Processing in "responder" mode, i.e., when a host that receives a remote CCI in a CCI option from the other end.

Section 5.2.1 and Section 5.2.2 describe the state machines at an initiator and a responder, respectively. Note that a single host can be both - initiator and responder - at the same time. This can happen if a local CCI occurs while processing for a remote CCI is ongoing, or vice versa.

The following events, conditions and actions are used in the definition of the two state machines:

Events:

E_LOCAL_CCI
   Local end received a local CCI.

E_REMOTE_CCI
   Local end received information about a remote CCI, i.e., received a TCP segment that includes a CCI option.

E_SEGMENT_SENT
   Local end sent a TCP segment that includes the CCI option.

Conditions:

C_NEW_REMOTE_CCI
   A received CCI option signals a new remote CCI, i.e., C != REMOTE_CCI, CS == LOCAL_CCI_NEW and the TSval of the Timestamps option of the received segment is greater than the current REMOTE_CCI_PEER_TIME (TSval > REMOTE_CCI_PEER_TIME).

C_ECHOED_LOCAL_CCI
   A received CCI option echoes the last local CCI, i.e., EC == LOCAL_CCI, ECS == REMOTE_CCI_ECHO and the TSval of the Timestamps option of the received segment is greater than the current LOCAL_CCI_PEER_ECHO_TIME (TSval > LOCAL_CCI_PEER_ECHO_TIME).

C_ECHOED_REMOTE_CCI
A received CCI option acknowledges that the peer has received the echo of its last local CCI, i.e., C == REMOTE_CCI, CS == LOCAL_CCI_ECHO_ACK and the TSval of the Timestamps option of the received segment is greater than the current REMOTE_CCI_PEER_TIME (TSval > REMOTE_CCI_PEER_TIME).

Actions:

A_TGL_LOCAL_CCI
Toggle LOCAL_CCI.

A_TGL_REMOTE_CCI
Toggle REMOTE_CCI.

A_REPROBE_PATH
TCP discards all congestion control information gathered on the current path, initializes them to the defaults and re-probes path characteristics based only on the segments transmitted after this event, as described in Section 5.3. In other words, CCI_CONTROLLED_CWND := 1, LAST_CCI_TIME := current local time, CCI_SNDMAX := highest sequence number transmitted so far and the congestion control state (CWND and SS_THRESH), round-trip time measurement (RTTM) state and RTO timer are reset to the initial values for a new connection. Additionally, if the connection is stalled in exponential back-off, TCP MUST act as if RTO had expired and start the speculative retransmission procedure described in Section 5.4.

A_FORCE_SEND
Force transmission of a segment that MUST include a CCI option, in order to inform the other peer about the local CCI. If the connection is stalled in exponential back-off, this is taken care of by the speculative retransmission procedure described in Section 5.4. If the connection is in steady-state and there is new data to be sent, TCP MUST immediately send a single segment of new data including a CCI option. If there is no new data to be sent, TCP MUST immediately send a pure ACK including a CCI option.

A_UPD_CCI_PEER_TIME
Set REMOTE_CCI_PEER_TIME to the TSval value of the TCP Timestamps option of the received segment.

A_UPD_CCI_PEER_E_TIME
Set LOCAL_CCI_PEER_ECHO_TIME to the TSval value of the TCP Timestamps option of the received segment.

5.2.1. Initiator Mode Processing

This section describes the initiator mode processing of a TCP host implementing RLCI. In initiator mode, a host signals the occurrence of a local CCI to its peer, until the peer echoes reception of that CCI. After receiving the echo, the host needs to acknowledge the echo reception, resulting in a 3-way handshake. Figure 2 shows the corresponding state machine.

At the beginning of a connection, i.e., before the first local CCI occurs, LOCAL_CCI is 0 and LOCAL_CCI_STATUS is LOCAL_CCI_IDLE. This remains the case until TCP receives a local CCI (E_LOCAL_CCI).

When that happens, TCP toggles LOCAL_CCI (A_TGL_LOCAL_CCI), sets LOCAL_CCI_STATUS := LOCAL_CCI_NEW, starts re-probing the new path (A_REPROBE_PATH) and forces a segment to be sent to the peer (A_FORCE_SEND).

Note that all subsequently transmitted segments MUST contain a CCI option until LOCAL_CCI_STATUS becomes LOCAL_CCI_IDLE. After the host receives the echo of the local CCI (C_ECHOED_LOCAL_CCI), it updates LOCAL_CCI_PEER_ECHO_TIME (A_UPD_CCI_PEER_E_TIME) and sets LOCAL_CCI_STATUS := LOCAL_CCI_ECHO_ACK. The initiator remains in this state until it can send a segment with the CCI option (E_SEGMENT_SENT) that acknowledges reception of the CCI echo. At that time, it sets LOCAL_CCI_STATUS := LOCAL_CCI_IDLE.

The transition from LOCAL_CCI_IDLE to LOCAL_CCI_ECHO_ACK occurs if a segment acknowledging the reception of a CCI echo is lost, and the initiator retransmits the echo acknowledgment.

When a local CCI occurs (E_LOCAL_CCI) while LOCAL_CCI_STATUS != LOCAL_CCI_IDLE, the host MUST ignore it and MUST NOT alter LOCAL_CCI, because it is already processing another local CCI.

```
                 E_LOCAL_CCI =>
                 A_TGL_LOCAL_CCI        E_REMOTE_CCI
                 A_REPROBE_PATH         C_ECHOED_LOCAL_CCI=>
                 A_FORCE_SEND           A_UPD_CCI_PEER_E_TIME
                 +----------------+    +----------------+
                 |                |    |                |
                 |                |    |                |
                 |                |    |                |
                 |                V    |                V
         +----------------+  +----------------+  +----------------+
         |                |  |                |  |                |
         |LOCAL_CCI_STATUS|  |LOCAL_CCI_STATUS|  |LOCAL_CCI_STATUS|
         |       ==       |  |       ==       |  |       ==       |
         |LOCAL_CCI_IDLE  |  |LOCAL_CCI_NEW   |  |LOCAL_CCI_ECHO_ |
         |                |  |                |  |ACK             |
         +----------------+  +----------------+  +----------------+
                ^  |                                   ^  |
                |  |                                   |  |
                |  +-----------------------------------+  |
                |           E_REMOTE_CCI                  |
                |           C_ECHOED_LOCAL_CCI            |
                |                                         |
                |                                         |
                +-----------------------------------------+
                               E_SEGMENT_SENT
```

Figure 2: State machine for initiator processing.

5.2.2. Responder Mode Processing

This section describes the responder mode processing of CCIs for a TCP host implementing the CCI option. In responder mode, a host echoes the last received remote CCI to its peer, until it can be sure that the peer correctly received the echo. Figure 3 shows the corresponding state machine.

At the beginning of a connection, REMOTE_CCI is 0 and REMOTE_CCI_STATUS is REMOTE_CCI_IDLE, i.e., the local host is not processing any remote CCIs.

When TCP receives a segment with a CCI option (E_REMOTE_CCI) signaling a new remote CCI (C_NEW_REMOTE_CCI), it increments REMOTE_CCI (A_TGL_REMOTE_CCI), changes REMOTE_CCI_STATUS to REMOTE_CCI_ECHO, updates REMOTE_CCI_PEER_TIME according to TSval (A_UPD_CCI_PEER_TIME), starts re-probing the new path (A_REPROBE_PATH) and forces a segment to be sent to the peer

(A_FORCE_SEND).

Note that all subsequently transmitted segments MUST contain a CCI option until REMOTE_CCI_STATUS is again REMOTE_CCI_IDLE. This transition occurs when the peer acknowledges the reception of the CCI echo (C_ECHOED_REMOTE_CCI).

```
                   E_REMOTE_CCI             E_REMOTE_CCI
                   C_NEW_REMOTE_CCI =>      C_NEW_REMOTE_CCI =>
                   A_TGL_REMOTE_CCI         A_TGL_REMOTE_CCI
                   A_UPD_CCI_PEER_TIME      A_UPD_CCI_PEER_TIME
                   A_REPROBE_PATH           A_REPROBE_PATH
                   A_FORCE_SEND             A_FORCE_SEND
                   +-----------------+      +--------------+
                   |                 |      |              |
                   |                 V      |              |
            +-----------------+  +-----------------+       |
            |REMOTE_CCI_STATUS|  |REMOTE_CCI_STATUS|       |
            |        ==       |  |        ==       |       |
            |REMOTE_CCI_IDLE  |  |REMOTE_CCI_ECHO  |       |
            +-----------------+  +-----------------+       |
                    ^                 |     ^              |
                    |                 |     |              |
                    +-----------------+     +--------------+
                     E_REMOTE_CCI
                     C_ECHOED_REMOTE_CCI
```

Figure 3: State machine for responder processing.

If TCP receives a new remote CCI while REMOTE_CCI_STATUS == REMOTE_CCI_ECHO, this indicates that the acknowledgment of a previous CCI echo may have been lost and that the peer had a new CCI occur. In this case, TCP MUST perform the same actions as if REMOTE_CCI_STATUS == REMOTE_CCI_IDLE.

5.3. Re-Probing Path Characteristics

When a TCP connection receives a new CCI, it MUST re-probe path characteristics in order to prevent causing congestion by transmitting based on stale path state information. In principle, this is similar to the initial slow-start: The sender MUST NOT transmit more than the default initial window (INIT_WINDOW) of data after a new CCI is received and it MUST reset the congestion control state (CWND and SS_THRESH), round-trip time measurement (RTTM) state and RTO timer, as if this were a new connection [RFC2581][RFC2988].

If Path MTU Discovery (PMTUD) is in use, the PMTUD state MUST also be reset [RFC1191][RFC1981][RFC4821].

One difference to an initial slow-start is that after a CCI, the connection may have segments in flight towards the destination along a previous path. Therefore, after a CCI, TCP MUST ignore any ACKs received for data that was sent before the CCI and it MUST update the congestion window solely based on ACKs for data that was sent after the CCI occurred.

The mechanism used for distinguishing ACKs for data sent after a CCI occurred from ACKs for data sent before a CCI occurred uses TCP Timestamps options. When a host receives a new CCI (either local or remote), LAST_CCI_TIME MUST be set to the current local time, CCI_SNDMAX MUST be set to the highest sequence number transmitted so far and CCI_CONTROLLED_CWND MUST be set to true.

While CCI_CONTROLLED_CWND == true, TCP MUST update the congestion window based only on inbound ACKs that contain a TS Echo Reply (TSecr) value greater than or equal to LAST_CCI_TIME. Any inbound ACK with a TS Echo Reply (TSecr) value less than LAST_CCI_TIME MUST NOT cause an update to the congestion window, even if it advances the window. If CCI_CONTROLLED_CWND is true and the host receives an ACK with a sequence number greater than or equal to CCI_SNDMAX, CCI_CONTROLLED_CWND MUST be set to false and the congestion control algorithm MUST begin to process all ACKs normally, without checking their Timestamps options.

5.4. Speculative Retransmission

The basic idea behind the speculative retransmission is to allow TCP to resume stalled connections as soon as it receives an indication that connectivity to previously unreachable peers may have returned.

When a TCP connection receives a new CCI - either from the local stack or in a CCI TCP option from the peer - and is currently stalled in exponential back-off, it MUST immediately initiate the standard retransmission procedure, just as if the RTO for the connection had expired.

6. Discussion

This section discusses some design choices of the RLCI mechanisms that can affect TCP performance under certain circumstances.

6.1. Triggered Segment Transmission during Steady-State

A TCP stack that implements RLCI mechanisms and receives a local CCI immediately sends a TCP segment (A_FORCE_SEND) in order to inform the other end of the CCI and resets all path information (A_REPROBE_PATH). When TCP is stalled in exponential back-off, this is taken care of by the speculative retransmission procedure that is triggered by the CCI.

On the other hand, when TCP is in steady-state, it sends a new segment (A_FORCE_SEND) if there is any new data queued for transmission. As usual, the number of unacknowledged segments is limited by CWND. However, CWND has just been reset to its initial value. This means that there is a possibility that the transmission sends a segment that is outside the current congestion window. Although this behavior may appear to be aggressive, it is in fact as conservative as a newly starting connection, because only a single unacknowledged segment is sent along the path after CCI.

6.2. Impact of Packet Loss

If a connection is in exponential back-off when a CCI occurs, TCP considers all unacknowledged segments to be lost and the speculative retransmission procedure immediately starts.

On the other hand, if the connection is in steady-state when a CCI occurs, TCP considers all unacknowledged segments to still be in flight and continues sending new data. Depending on what caused a CCI, four scenarios are possible that differ in what happens to segments and ACKs in flight:

1. All (or at least the vast majority of) segments and ACKs in flight reach their respective destinations, i.e., there are no losses. In this case, TCP acts as if a new connection had started and re-probes the new path.

2. Some of the ACKs in flight from the receiver to the sender are lost. In this case, TCP behaves exactly as above, because a cumulative ACK for the new segment sent along the path after the CCI acknowledges all the previous unacknowledged segments.

3. Some of the data segments in flight from the sender to the receiver are lost. In this case, the new data segment transmitted after the CCI causes a duplicate ACK. As this duplicate ACK does not cause TCP to send another data segment, the connection stalls and a RTO occurs. After RTO, the standard retransmission procedure takes place with SS_THRESH equal to INITIAL_WINDOW/2 (i.e., the minimum allowed). This disables slow

start and causes a severely decreased performance. A possible solution is to execute the speculative retransmission procedure after receiving a CCI even if the connection is in steady-state.

4. Some of the data segments and some of the ACKs that are in flight are lost. This case is similar to the previous one.

In all these cases, it is also possible that the round-trip time changes significantly after the CCI, reordering data segments and ACKs that are still in flight with ones sent after the CCI. These reorderings appear to TCP as losses, and may result in the connection experiencing one of the above cases even if there was no actual packet loss.

6.3. Use of Limited Transmit with RLCI

As described in the previous section, when a connection is in steady-state, a connectivity-change indication (CCI) resets all path information of TCP and causes one new data segment to be sent. In case of significant data segment loss before a CCI, the new data segment transmitted after a CCI causes a duplicate ACK. As this duplicate ACK does not trigger TCP to send another data segment, the connection stalls and an RTO occurs.

Limited Transmit [RFC3042] can be used in case of packet loss in order to cause the transmission of three duplicate ACKs and trigger the fast retransmission procedure. As it must not cause an amount of outstanding data more than the congestion window plus two segments, it cannot always be used after a CCI due to the initialized CWND. If the connection has more outstanding data than INITIAL_WINDOW plus two segments before a CCI, resetting of CWND to the initial value after CCI causes an amount of outstanding data greater than the new CWND plus two segments and disables Limited Transmit.

A modified Limited Transmit algorithm can be used in combination with RLCI:

If CCI_CONTROLLED_CWND is true:
   The Limited Transmit Algorithm as described in [RFC3042] should be followed, but without checking the amount of outstanding data, i.e., if a TCP sender has previously unsent data queued for transmission it should transmit new data upon the arrival of the first two consecutive duplicate ACKs when the receiver's advertised window allows this transmission.

If CCI_CONTROLLED_CWND is false:
   The Limited Transmit Algorithm as described in [RFC3042] should be followed unmodified.

When the fast retransmission procedure is triggered by the modified Limited Transmit after a CCI, SS_THRESH is set to INITIAL_WINDOW/2 (i.e., the minimum allowed) as CWND before fast retransmission was equal to INITIAL_WINDOW. As a result, slow-start is disabled causing decreased TCP performance.

A minor modification can keep SS_THRESH unmodified in the previous case, i.e., if CCI_CONTROLLED_CWND == true and CWND == INITIAL_WINDOW, keep SS_THRESH unmodified (having its initial value) upon the reception of the third duplicate ACK that triggers the fast retransmission procedure.

6.4. Simultaneous Processing of Connectivity-Change Indications

As mentioned in Section 5.2.1, if a local CCI occurs (E_LOCAL_CCI) while LOCAL_CCI_STATUS != LOCAL_CCI_IDLE, the host MUST ignore it, because it is already processing another local CCI. As a result, only one local CCI at each end can be processed at the same time. Consequently, as every remote CCI at one end is triggered by a local CCI at the other end, only one remote CCI at each end can be processed at the same time.

On the other hand, if both hosts receive connectivity-change indications from their local stacks (local CCIs) at almost the same time, there is a possibility of simultaneous processing of local and remote CCIs at both ends. In that case, path re-probing is triggered twice at each end in a very short time that can be lower than RTT. As this does not improve TCP performance, it can be avoided by triggering the A_REPROBE_PATH action only if CCI_CONTROLLED_CWND == false.

7. Security Considerations

The only foreseen security considerations with the techniques presented in this document result from either an attacker's ability to spoof valid TCP segments with CCI options that seemingly indicate connectivity changes, or an attacker's ability to generate bogus CCIs locally. An attacker might produce a stream of such false indicators that could keep a connection in slow-start at the initial window. One possible defense against this type of attack is to rate-limit the response to CCIs (whether local or remote). This is also probably less serious than other attacks such an empowered adversary could perform, like resetting the connection or injecting data. A similar

effect could be achieved without the new CCI option by forging duplicate ACKs that would keep a sender in loss recovery. If both sets of IP addresses, port numbers, and sequence numbers are guessable for a connection, then the connection should employ other measures [RFC4953] for protection against spoofed segments.

8. IANA Considerations

This section is to be interpreted according to [I-D.narten-iana-considerations-rfc2434bis].

This document does not define any new namespaces. It requests that IANA allocate a new 8-bit TCP option number for the CCI option from the registry maintained at http://www.iana.org/assignments/tcp-parameters.

9. Acknowledgments

This draft combines and obsoletes [I-D.swami-tcp-lmdr] and [I-D.eggert-tcpm-tcp-retransmit-now]. The authors would like to thank Mark Allman, Marcus Brunner, Alfred Hoenes, Shashikant Maheshwari, Kacheong Poon, Juergen Quittek, Stefan Schmid and Joe Touch for their comments and suggestions on this draft as well as the two original drafts.

Simon Schuetz and Lars Eggert are partly funded by the Trilogy project, a research project supported by the European Commission under its Seventh Framework Program.

Wesley Eddy's work on this document was performed at NASA's Glenn Research Center, while in support of the NASA Space Communications Architecture Working Group (SCAWG), and the FAA/Eurocontrol Future Communications Study (FCS).

10. References

10.1. Normative References

[I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-08 (work in progress), October 2007.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7,

RFC 793, September 1981.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

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

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

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

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

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

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

10.2. Informative References

[DUKE]   Duke, M., Henderson, T., and J. Meegan, "Experience with ``Link-UP Notification'' Over a Mobile Satellite Link", ACM Computer Communication Review, Vol. 34, No. 3, July 2004.

[EDDY]   Eddy, W. and Y. Swami, "Adapting End-host Congestion Control for Mobility", NASA Glenn Research Center Technical Report, CR-2005-213838, July 2005.

[I-D.dawkins-trigtran-linkup]
Dawkins, S., "End-to-end, Implicit 'Link-Up' Notification", draft-dawkins-trigtran-linkup-01 (work in progress), October 2003.

[I-D.eggert-tcpm-tcp-retransmit-now]
Eggert, L., "TCP Extensions for Immediate Retransmissions", draft-eggert-tcpm-tcp-retransmit-now-02 (work in progress), June 2005.

[I-D.ietf-hip-mm]
Henderson, T., "End-Host Mobility and Multihoming with the Host Identity Protocol", draft-ietf-hip-mm-05 (work in progress), March 2007.

[I-D.ietf-tcpimpl-restart]
Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP Slow-Start Restart After Idle", draft-ietf-tcpimpl-restart-00 (work in progress), March 1998.

[I-D.ietf-tcpm-tcp-uto]
Eggert, L. and F. Gont, "TCP User Timeout Option", draft-ietf-tcpm-tcp-uto-08 (work in progress), November 2007.

[I-D.swami-tcp-lmdr]
Swami, Y., "Lightweight Mobility Detection and Response (LMDR) Algorithm for TCP", draft-swami-tcp-lmdr-07 (work in progress), March 2006.

[KOODLI] Koodli, R. and C. Perkins, "Fast Handovers and Context Transfers in Mobile Networks", ACM Computer Communication Review, Vol. 31, No. 5, October 2001.

[OTT] Ott, J. and D. Kutscher, "OTT Internet: IEEE 802.11b for Automobile Users", Proc. Infocom 2004, March 2004.

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

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

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

[RFC3775] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

[RFC3819] Karn, P., Bormann, C., Fairhurst, G., Grossman, D.,

Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4782] Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, January 2007.

[RFC4907] Aboba, B., "Architectural Implications of Link Indications", RFC 4907, June 2007.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC4957] Krishnan, S., Montavont, N., Njedjou, E., Veerepalli, S., and A. Yegin, "Link-Layer Event Notifications for Detecting Network Attachments", RFC 4957, August 2007.

[SCHUETZ] Schuetz, S., Eggert, L., Schmid, S., and M. Brunner, "Protocol Enhancements for Intermittently Connected Hosts", ACM Computer Communication Review, Vol. 35, No. 3, July 2005.

[SCOTT] Scott, J. and G. Mapp, "Link layer-based TCP optimization for disconnecting networks", ACM Computer Communication Review, Vol. 33, No. 5, October 2003.

Editorial Comments

[footnote-1] The authors have heard the idea of triggering retransmits based on connectivity events of directly-connected links being attributed to Phil Karn ("kick" operation in the KAQ9 TCP stack). A thread from the PILC mailing list in 2000 discusses some thoughts on this (http://www.isi.edu/pilc/list/archive/0691.html).

[footnote-2] Although this specification introduces eight new per-connection state variables, a preliminary implementation of an earlier revision of this mechanism [I-D.swami-tcp-lmdr] only required around a hundred lines of kernel code.

Appendix A. **Background: Classification of Connectivity Disruptions**

Connectivity disruptions can occur in many different situations.

They can be due to wireless interference, movement out of a wireless coverage area, switching between access networks, or simply due to unplugging an Ethernet cable. Depending on the situation in which they occur, the implications of connectivity disruptions are different and must be handled appropriately. This section attempts to classify different types of connectivity disruptions and discusses their implications and impact on TCP.

Two main properties of connectivity disruptions affect how TCP reacts to them: their duration and whether the path characteristics have significantly changed after they end. This document distinguishes between "short" and "long" disruptions and "changed" and "unchanged" path characteristics. Note that these two categories are orthogonal to each other, i.e., four types of connectivity disruptions exist.

Connectivity disruptions are "short" for a given TCP connection, if connectivity returns before the RTO fires for the first time, i.e., when TCP is still in steady-state. In this case, standard TCP recovers lost data segments through Fast Retransmit and lost ACKs through successfully delivered later ACKs. Appendix A.1 briefly describes this case.

Connectivity disruptions are "long" for a given TCP connection, if the RTO fires at least once before connectivity returns, i.e., when TCP is in exponential back-off. In this case, TCP can be inefficient in its retransmission scheme, as described in Appendix A.2.

Whether or not path characteristics change when connectivity returns is a second important factor for TCP's retransmission scheme. Standard TCP implicitly assumes that path characteristics remain unchanged across short disruptions by performing Fast Retransmit using the path parameters collected before the disruption. For long disruptions, standard TCP is more conservative and performs slow-start, re-probing the path characteristics from scratch. However, the standard behavior can be inefficient due to when it is initiated.

These implicit assumptions can cause standard TCP to misbehave or perform inefficiently in some scenarios. Figure 4 illustrates the standard TCP behavior.

Short Duration < RTO	Fast Retransmit using currently collected path characteristics	Fast Retransmit using currently collected path characteristics
Long Duration >= RTO	Slow-start	Slow-start
	Unchanged Path Characteristics	Changed Path Characteristics

Figure 4: Standard TCP behavior.

A.1. Short Connectivity Disruptions

One common cause of short connectivity disruptions that result in a change of the end-to-end path characteristics is transparent network layer mobility, via protocols such as Mobile IP, NEMO, or HIP. These protocols generally hide mobility events from the transport layer, but cannot mask the resulting changes to the end-to-end path that established TCP connections transmit over.

Consider a Mobile IP scenario as shown in Figure 5. At time T, a mobile node MN attaches to access network Net-1, connected to the Internet through access router AR-1 and has the care-of address <Net-1, MN>. It establishes a TCP connection to the correspondent node CN. While MN attaches to AR-1, packets between CN and <Net-1, MN> follow PATH-1 (via Cloud-1 and AR-1). Assume that at some time T+1, MN moves and then attaches to Net-2, which is reachable through AR-2 with the care-of address <Net-2, MN>. While MN attaches to AR-2, all packets between CN and <Net-2, MN> follow PATH-2 (through Cloud-2 and AR-2).

```
                      <---------PATH-1---------->

                       /---------\   +------+
                       |         |   |      | Net-1
                   +---+ Cloud-1 +---+ AR-1 +-----> MN (time=T)
                   |   |         |   |      |
                   |   \----+----/   +---+--+        |
                   |        |                        |
         CN <------+        | PATH-3                 |
                   |        |                        |
                   |   /----V----\   +-------+       V
                   |   |         |   |       |
                   +---+ Cloud-2 +---+ AR-2  +-----> MN (time=T+1)
                       |         |   |       | Net-2
                       \---------/   +-------+

                      <--------PATH-2----------->
```

Figure 5: Mobility example.

During a transient disconnected period, MN may have disconnected from Net-1 and not yet attached to Net-2. Consequently, AR-1 may not be able to deliver packets to MN. This could result in a burst of packet losses. Several approaches for "fast" or "seamless" handovers exist that involve adding machinery to the ARs to buffer and redirect packets originally sent to Net-1 towards Net-2, rather than dropping them (e.g., [KOODLI]).

As long as MN remains in Net-1, standard congestion control algorithms [RFC2581] are sufficient. However, once MN moves from Net-1 to Net-2, two different scenarios are possible depending on network topology:

o In the first scenario, with standard Mobile IPv4, all packets destined to <Net-1, MN> are dropped by AR-1 once MN has moved. Since the latency involved in establishing a new tunnel to the HA is on the order of the RTT (2*RTT in case of Mobile IPv6), roughly an entire window's worth of data and ACKs will be dropped by AR-1. Because of this burst loss, CN and MN are likely to incur expensive retransmission timeouts.

o In the second scenario, with a fast handover mechanism in place, losses are masked through buffering and tunneling between routers AR-1 and AR-2. The exact sequence of buffering and forwarding between the ARs is not guaranteed to occur in a manner consistent with the available bandwidth of PATH-3 or conformant to TCP's clocking expectations. This can cause TCP's behavior over PATH-2 to be based on the unrelated properties of PATH-1 and PATH-3.

After attaching to Net-2, reception of stale ACKs (for data sent on PATH-1) will cause MN to incorrectly inflate its congestion window. These stale ACKs do not provide any indication of the congestion along PATH-2. CN's congestion window becomes similarly inflated by ACKs that MN sends for data segments redirected over PATH-3. If the congestion windows from PATH-1 are already too big for PATH-2, this can overload Net-2 or PATH-2, causing packet loss and timeouts.

On the other hand, if the available bandwidth along PATH-2 is greater than along PATH-1, and if the sender is in congestion avoidance, it will need potentially many RTTs before utilizing the available path capacity. This is due to relatively slow bandwidth increase during congestion avoidance caused by a stale SS_THRESH. (See [EDDY] for details.)

A.2. Long Connectivity Disruptions

For long disruptions, standard TCP performs slow-start after connectivity returns, because the retransmission timeout (RTO) has expired. This conservative strategy avoids overloading the new path. However, TCP's general exponential back-off retransmission strategy can time these slow-starts such that performance decreases.

When a long connectivity disruption occurs along the path between a host and its peer while the host is transmitting data, it stops receiving ACKs. After the RTO expires, the host attempts to retransmit the first unacknowledged segment. TCP implementations that follow the recommended RTO management proposed in [RFC2988] double the RTO after each retransmission attempt until it exceeds 60 seconds. This scheme causes a host to attempt to retransmit across established connections roughly once a minute. (More frequently during the first minute or two of the connectivity disruption, while the RTO is still being backed off.)

When the long connectivity disruption ends, standard TCP implementations still wait until the RTO expires before attempting retransmission. Figure 6 illustrates this behavior. Depending on when connectivity becomes available again, this can waste up to a minute of connectivity for TCPs that implement the recommended RTO management described in [RFC2988]. For TCP implementations that do not implement [RFC2988], even longer connectivity periods may be wasted. For example, Linux uses 120 seconds as the maximum RTO by default.

```
          Sequence
          number      X = Successfully transmitted segment
           ^          O = Lost segment
           |     :                     :              : X
           |     :                     :              :X
           |     OO O  O    O        O :              X
           |    X:                     :              :
           |   X :                     :<------------>:
           |  X  :                     :    Wasted    :
           | X   :                     :  connection  :
           |X    :                     :     time     :
           +-----:---------------------:--------------:-------->
                 :                     :              :       Time
            Connectivity          Connectivity       TCP
               gone                  back         retransmit
```

Figure 6: Standard TCP behavior in the presence of disrupted connectivity.

This retransmission behavior is not efficient, especially in scenarios where connectivity periods are short and connectivity disruptions are frequent [OTT]. Experiments show that TCP performance across a path with frequent disruptions is significantly worse, compared to a similar path without disruptions [SCHUETZ].

In the ideal case, TCP would attempt a retransmission as soon as connectivity to its peer was re-established. Figure 7 illustrates the ideal behavior.

```
          Sequence
          number      X = Successfully transmitted segment
           ^          O = Lost segment
           |     :                     : X            :
           |     :                     :X             :
           |     OO O  O    O        O X              :
           |    X:                     :              :
           |   X :                     :<------------>:
           |  X  :                     :  Efficiency  :
           | X   :                     :  improvement :
           |X    :                     :              :
           +-----:---------------------:--------------:-------->
                 :                     :              :       Time
            Connectivity          Connectivity      Next
               gone             back := immediate  scheduled
                                 TCP retransmit   retransmit
```

Figure 7: Ideal TCP behavior in the presence of disrupted connectivity

The ideal behavior is difficult to achieve for arbitrary connectivity disruptions. One obviously problematic approach would use higher-frequency retransmission attempts to enable earlier detection of whether connectivity has returned. This can generate significant amounts of extra traffic. Other proposals attempt to trigger faster retransmissions by retransmitting buffered or newly-crafted segments from inside the network [SCOTT][I-D.dawkins-trigtran-linkup][DUKE][RFC3819].

Note that scenarios exist where path characteristics remain unchanged after long connectivity disruptions. In this case, even an intelligently scheduled slow-start is inefficient, because TCP could safely resume transmitting at the old rate instead of slow-starting. Although originally developed to avoid line-rate bursts, techniques for the well-known "slow-start after idle" case [I-D.ietf-tcpimpl-restart] may be useful to further improve performance after a disruption ends in such a scenario. This document does not currently describe this additional optimization, and an open question remains on how unchanged path characteristics after long connectivity disruptions could be validated by an end host.

Appendix B. Document Revision History

Revision	Comments
03	Mainly editorial and textual changes according to feedback received since last version.
02	Major modification to the RLCI mechanism for implementing a 3-way handshake that ensures that both peers are informed about a connectivity-change indication. CCI option format, RLCI variables maintained by the TCP peers and the related state machines are affected by that modification.
01	Major revision of the description of the connectivity-change indication TCP option and its processing in Section 5. Other formatting changes to the document include moving some background material to the appendix.
00	Initial version. This document is a merge of and obsoletes [I-D.eggert-tcpm-tcp-retransmit-now] and [I-D.swami-tcp-lmdr].

Authors' Addresses

Simon Schuetz
NEC Laboratories Europe
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 165
Email: simon.schuetz@nw.neclab.eu
URI: http://www.nw.neclab.eu

Nikolaos Koutsianas
Nokia Research Center

Email: nkout@mobile.ntua.gr

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/

Wesley M. Eddy
Verizon Federal Network Systems
NASA Glenn Research Center
21000 Brookpark Road, MS 54-5
Cleveland, OH 44135
USA

Email: weddy@grc.nasa.gov

Yogesh Prem Swami
Nokia Research Center, Dallas
955 Page Mill Road
Palo Alto, California 94304
USA

Phone: +1 972 374 0669
Email: yogesh.swami@nokia.com

Khiem Le
Nokia Siemens Networks
6000 Connection Drive
Irving, TX 75039
USA

Phone: +1 972 342 3502
Email: khiem.le@nsn.com

Full Copyright Statement

Intellectual Property

Acknowledgment

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