Master of Science in Communications and Computer Security

École Nationale Supérieure des Télécommunications
Institut Eurécom

Internship Report

Nikolaos Koutsianas

TCP Response to Lower-Layer Connectivity-Change Indications

Company: Nokia
Industrial supervisors: Dr. Lars Eggert, Dr. Pasi Sarolahti
Academic supervisor: Prof. Ernst Biersack

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 freshly-started connection.

Preface

This document presents the work of a six months internship accomplished in Nokia Research Center (NRC), Helsinki from July 2007 to December 2007 as part of my studies at École Nationale Supérieure des Télécommunications - Institut Eurécom for obtaining the Master of Science degree in “Communications and Computer Security”.

This work results the second update of the related Internet-Draft ‘draft-schuetz-tcpm-tcp-rlci-02’ (work in progress). The Internet-Draft is available at: http://tools.ietf.org/html/draft-schuetz-tcpm-tcp-rlci-02

Acknowledgments

I would like to thank Dr. Lars Eggert and Dr. Pasi Srolahti, my industrial supervisors, for providing me with the great opportunity of working in Nokia Research Center, their guidance, precious advice, availability and excellent collaboration throughout my internship.

Special thanks to Ismo Kangas, leader of the Ubiquitous Internet Connectivity Team of Nokia Research Center, for his support and help.

Furthermore, I would like to thank my academic supervisor, Prof. Ernst Biersack, for his follow up and supervision.

Last but not least, I would like to thank my MSc supervisor, Prof. Refik Molva and my annual project supervisor, Prof. Marc Dacier, for sharing their knowledge with me during my studies at Eurécom.

Table of contents

1 Introduction 1
1.1 Motivation and Overview 1
1.2 Contribution 2
2 Design 5
2.1 Connectivity-Change Indications 5
2.2 Response to Connectivity-Change Indications 6
3 Specification 7
3.1 Connectivity-Change Indication TCP Option 8
3.2 Processing of Connectivity-Change Indications 10
3.2.1 Initiator Mode 11
3.2.2 Responder Mode 12
3.3 Re-Probing Path Characteristics 13
3.4 Speculative Retransmission 14
3.5 Discussion 14
3.5.1 Triggered Transmission during Steady-State 14
3.5.2 Impact of Packet Loss 14
3.5.3 Use of Limited Transmit with RLCI 15
3.5.4 Simultaneous Connectivity-Change Indications 16
3.6 Security Considerations 16
4 Experimental Evaluation 17
4.1 Implementation of RLCI Mechanism 17
4.2 Bulk Transfer of a Single Connection 17
4.3 Bulk Transfer of Multiple Connections 23
5 Related Work 29
6 Conclusion 31
6.1 Recapitulation 31
6.2 Future Work 32
7 Bibliography 33
Appendix 35
A. Internship Timetable 35
B. Company Profile 37

1

Introduction

The Transmission Control Protocol (TCP) [1] 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 [2] 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 (CCI) 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 state of a connection.

An important point to note is that the TCP response mechanisms to lower-layer 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.

1.1 Motivation and Overview

Several proposed network-layer extensions support host mobility, including Mobile IPv4 [3], Mobile IPv6 [4] and HIP [5]. 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.

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 [6]. 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 [7].)

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. 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 document 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.

1.2 Contribution

This document improves the existing ‘Response to Lower Layer Connectivity-Change Indications’ (RLCI) mechanism already published at draft-schuetz-tcpm-tcp-rlci-01 (work in progress). It presents the work of a six months internship accomplished in Nokia Research Center (NRC), Helsinki from July 2007 to December 2007 as part of my studies at École Nationale Supérieure des Télécommunications - Institut Eurécom for obtaining the Master of Science degree in “Communications and Computer Security”.

The main changes compared to the previous version of RLCI mechanism include:

- Addition of a reliable transmission of connectivity-change indications between peers, based on a 3-way handshake.

- Modification of the related TCP state machines implemented by hosts support RLCI (Section 3.2) in order to support the reliable transmission of connectivity-change indications.

- Enrichment of the related TCP Option format, described at Section 3.1, in order to support the reliable transmission of connectivity-change indications.

- Use of Timestamps TCP Option for distinguishing the ACKs for segments sent through the path after CCI.

- Improvements on the processing of multiple simultaneous connectivity-change indications (Section 3.5.4).

- Suggestions for modification of Limited Transmit [8] for use with RLCI (Section 3.5.3).

The improved RLCI mechanism is implemented at NS2 network simulation. The right operation of the modified mechanism is evaluated through simulation analysis.

RLCI mechanism depends on Timestamps TCP Option. It is expected (but not needed) that the peers implementing RLCI will also support TCP Sack. NS2 TCP Sack implementation does not follow the related RFC [19] causing decreased performance of RLCI. A significant part of the work included the modification of NS2 TCP Sack implementation according to the related RFC for use with RLCI.

Another important part of this work is the performance evaluation of RLCI TCP described at Section 4. Using appropriate simulation scenarios, it is evaluated:

- The performance improvement of TCP implementing RLCI, compared to Standard TCP (Section 4.2 and4.3) for:

    - Mobile host having a single active connection.

    - Mobile host having multiple active connections.

- The impact of packet loss to RLCI TCP performance (Section 3.5.2).

This work resulted the second update of the related Internet-Draft -schuetz-tcpm-tcp-rlci-02 (work in progress) available at http://tools.ietf.org/html/draft-schuetz-tcpm-tcp-rlci-02

2

Design

This chapter gives a high level description of TCP Response to Lower-Layer Connectivity-Change Indications (RLCI) description.

2.1 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):

- the IP address of the local outbound interface used for a given connection has changed, e.g., due to DHCP [9] or IPv6 router advertisements [10].
- link-layer connectivity of the local outbound interface used for a given connection has changed, e.g., link-layer "link up" event [11].
- 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 [12]).
- a Mobile IP binding update has completed [4].
- a HIP readdressing update has completed [5].
- a path-change signal from the network has arrived (possible in theory, depends on network capabilities)
- other notifications as defined by the IETF's Detecting Network Attachment (DNA) working group have occurred [11].

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.

2.2 Response to Connectivity-Change Indications

TCP response to lower-layer connectivity-change indications (RLCI) is an optional TCP mechanism that improves TCP performance based on signals coming from lower layers. As TCP already reacts to information and signals from lower layers (e.g. ICMP signals), the proposed connectivity-change indications thus extend an established interface between layers in the protocol stack.

Lower-layer connectivity-change indications are received only from one peer, while appropriate responses are required from both peers. As a result, a new CCI TCP option is defined having appropriate fields in order to inform the other peer for the connectivity-change indication. The CCI TCP Option is described at Section 3.1.

Under the reception of a connectivity-change indication a host supporting RLCI takes the following actions:

- If the connectivity-change indication comes from the local stack, the host immediately sends a TCP segment including the CCI TCP Option initiating a 3-way handshake that ensures that the other peer becomes informed about that connectivity-change indication.

- If the connectivity-change indication comes through the CCI TCP Option, the host immediately responds to its peer by sending a TCP segment including the CCI TCP Option that continues the 3-way handshake.

- TCP starts re-probing the new path in order to prevent causing congestion by transmitting based on stale path state. In principle, this occurs similarly to the initial slow-start: The sender must not transmit more than the default initial window (INIT_WINDOW) of data after a 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.

  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. RLCI TCP depends on the TCP Timestamps option (TSopt) [13] in order to achieve that.

- When a TCP connection receives a connectivity-change indication 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. That action takes resume immediately a stalled connection.

A detailed description of the RLCI mechanism and the related CCI TCP is given at the next chapter of this document.

3

Specification

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 3.1 specifies this new TCP option. In either case, upon reception of a CCI, the TCP response 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 RTO expiration but has not been ACK'ed yet.

The remainder of this section first defines the format of the new CCI option in Section 3.1 and then describes the two TCP response mechanisms triggered by receiving CCIs - re-probing path characteristics and speculative retransmission - in Section 3.3 and Section 3.4.

The RLCI mechanisms defined in this document depend on the TCP Timestamps option (TSopt) [13]. Consequently, it is required that an end host that wishes to use the RLCI mechanisms for TCP connection negotiate the use of TCP Timestamps options with its peer. If this negotiation fails, a host must not use 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 3.2 and Section 3.3 give more details about how RLCI mechanisms use TCP Timestamps options.

An implementation of the RLCI mechanism defined in this document maintains nine new state variables per TCP connection.

LOCAL_CCI: It is a 1-bit counter, having an initial value of 0, used for distinguishing the existence of a new local CCI. It changes 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 is updated in response to remote CCIs according to the rules defined in Section 3.2.

LOCAL_CCI_STATUS: It holds the status of the local CCI. It can have 3 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 last remote CCI advertised by the peer through a CCI TCP option. It is a Boolean variable that can have 2 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 the 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 the delayed retransmissions of echoes of older local CCIs. It holds the TS Value (TSval) of the Timestamps option of the segment 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 the congestion window. Having an initial value of false, it is updated according to the rules defined in Section 3.2.

3.1 Connectivity-Change Indication 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 TCP option is to signal the occurrence of local CCIs to the other end, in order to allow it 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 peer will imply that the characteristics of the reverse path have also changed.

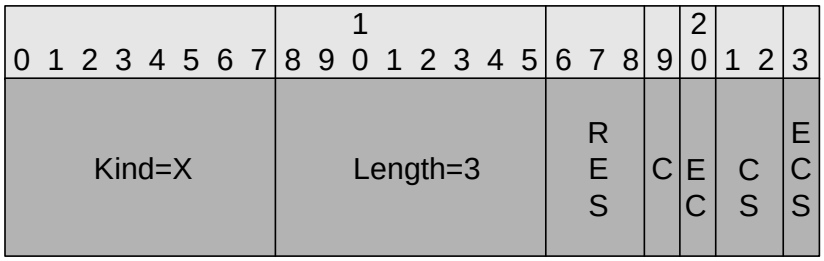


Figure 3.1: Format of the connectivity-change indication TCP option

Figure 3.1 shows the format of the CCI TCP option. It contains these fields:

Kind (8 bits): The TCP option number X [14] allocated by IANA upon publication of this document.

Length (8 bits): Length of the TCP option in octets [14]; 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.

CCI TCP option contains two single-bit fields (C and EC) used for distinguishing new CCIs from delayed retransmissions of the old ones. It also contains some flags representing the status of each CCI. These flags are used for a 3-way handshake that ensures 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 opening a connection includes 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 CCI mechanism. A host receiving a SYN segment must not include a CCI option in its SYN-ACK, 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 the same 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.

After the SYN exchange, a host should send a CCI option only if receiving a new local CCI, or in response to receiving a new CCI option from the other end. Section 3.2.1 and Section 3.2.2 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. 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.

3.2 Processing of Connectivity-Change Indications

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 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 3.2.1 and Section 3.2.2 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, if a local CCI happens to occur 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 TCP 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 3.3. 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 at Section 3.4.

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 at Section 3.4. 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. If there is no new data to be sent, TCP must immediately send a pure ACK.

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.

3.2.1 Initiator Mode

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 3.2 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 toggle LOCAL_CCI, because it is already processing another local CCI.

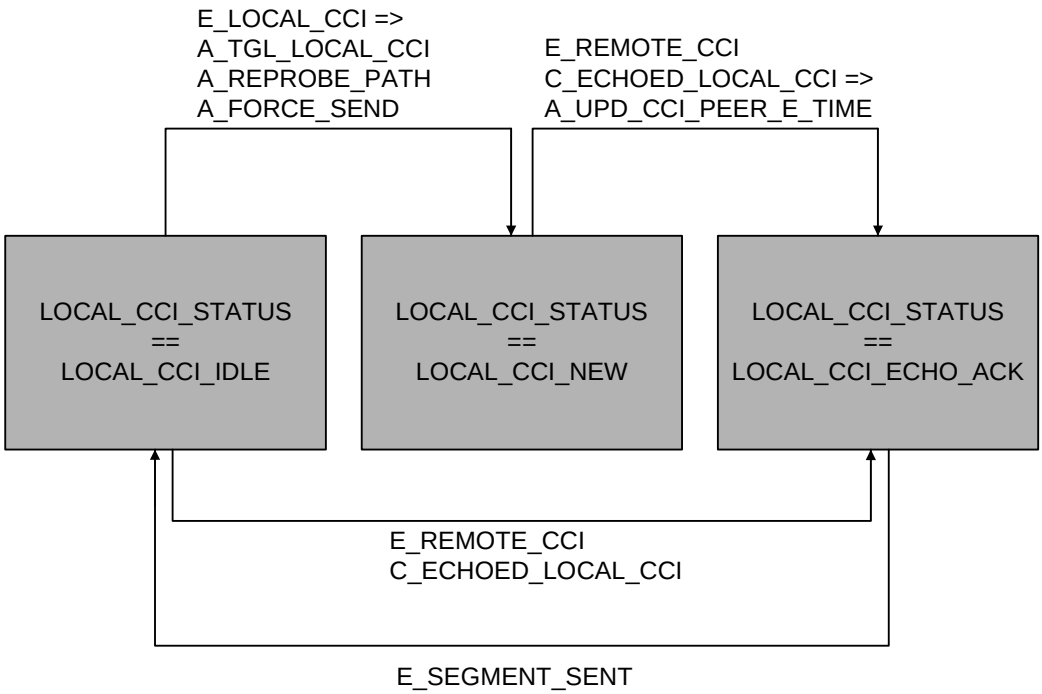


Figure 3.2: State machine for initiator processing

3.2.2 Responder Mode

This section describes the responder mode processing of CCIs for a TCP host implementing the CCI TCP 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 3.3 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 TCP option (E_REMOTE_CCI) signaling a new remote CCI (C_NEW_REMOTE_CCI), it toggles 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 TCP 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).

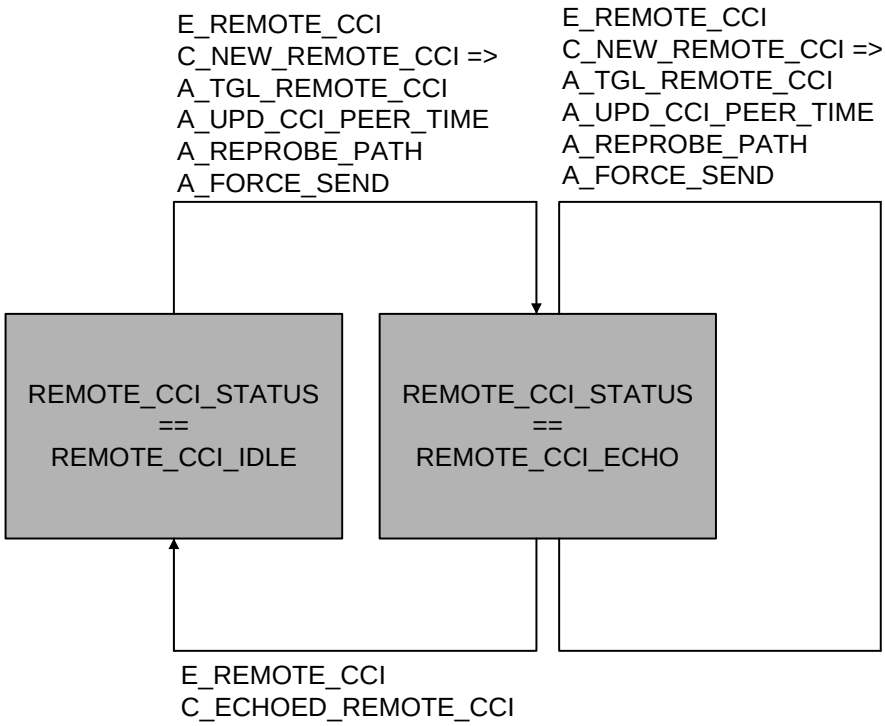


Figure 3.3: State machine for responder processing

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.

3.3 Re-Probing Path Characteristics

When a TCP connection receives a CCI, it must re-probe path characteristics in order to prevent causing congestion by transmitting based on stale path state. In principle, this occurs similarly to the initial slow-start: The sender must not transmit more than the default initial window (INIT_WINDOW) of data after a 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 [2][15]. If Path MTU Discovery (PMTUD) is in use, the PMTUD state MUST also be reset [1] [16] [17].

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.

3.4 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 connectivity-change indication - either from the local stack or in a connectivity-change 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.

3.5 Discussion

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

3.5.1 Triggered Transmission during Steady-State

A TCP stack that implements RLCI and receives a local connectivity-change indication immediately sends a TCP segment (A_FORCE_SEND) in order to inform the peer of the CCI, after resetting 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 connectivity-change indication.

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 the 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.

3.5.2 Impact of Packet Loss

If a connection is in exponential back-off when a connectivity-change indication 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 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 path delay 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.

3.5.3 Use of Limited Transmit with RLCI

As described at the previous section, when 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 segments loss before CCI, the new data segment transmitted after 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 [8] can be used in case of packet loss in order to cause the transmission of 3 duplicate ACKs and trigger the fast retransmission procedure. As it must not cause an amount of outstanding data more than the congestion window plus 2 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 2 segments before CCI, resetting of CWND to the initial value after CCI causes an amount of outstanding data greater than the new CWND plus 2 segments and disables Limited Transmit.

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

- If CCI_CONTROLLED_CWND is true: Limited Transmit Algorithm as described at [8] should be followed but without checking the amount of outstanding data, i.e., if 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: Limited Transmit Algorithm as described at [8] should be followed.

When 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.

3.5.4 Simultaneous Connectivity-Change Indications

As mentioned in Section 3.2.1, 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, it cannot be processed more than one local CCI at each end any time. As every remote CCI at one end is triggered by a local CCI at the other end, it cannot be processed more than one remote CCI at each end any 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.

3.6 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 options that seemingly indicate connectivity changes, or an attacker's ability to generate bogus connectivity-change indications 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 connectivity indicators (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 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 [18] for protection against spoofed segments.

4

Experimental Evaluation

This section experimentally analyzes the protocol enhancements described in Section 3.

4.1 Implementation of RLCI Mechanism

NS2 network simulator is selected as the appropriate platform for the experimental analysis of RLCI performance. TCP implementation of NS2 is modified according to section 3 in order to support RLCI mechanism. The changes mainly include the implementation of the two state machines that process the local and remote connectivity-change indications (CCIs). Furthermore, TCP packet header is enriched with the appropriate CCI option.

These changes affect only the TCP stack of NS2 and mainly tcp.h, tcp.cc, tcp-sink.h and tcp-sink.cc files located at tcp directory. As NS2 uses an object-oriented architecture, only minor changes are needed at tcp/tcp-reno.cc, tcp/tcp-newreno.cc and tcp/tcp-sack1.cc files in order for RLCI to be supported by the related TCP versions.

As the main topic of interest of this work is the processing of connectivity-change indications for achieving increased TCP performance, there is no focusing on the way that lower layers detect the connectivity-change indications. As a result, lower layers of NS2 remain unmodified and connectivity-change indications are received externally by TCP using the Tcl language at the scripts that model our simulation scenarios.

Even if it is not needed, it is expected that the hosts supporting RLCI support also Sack based loss-recovery. During loss-recovery, TCP Sack uses a ‘pipe’ variable that estimates the number of the segments that are on the flight, traveling along the path. According to the related RFC [19], TCP SACK uses the IsLost() function in order to distinguish the lost segments from the ones traveling along the path and calculate the value of the ‘pipe’. NS2 does not implement this function. As a result, during loss recovery, it considers all the unsacked segments as traveling along the path, causing an overestimation of the pipe. This overestimation of pipe, combined with the initialization of the CWND after a CCI, it may result to decreased performance, as TCP Sack can send new segments only if pipe < CWND. As a result, the modification of Sack based loss recovery of NS2 according to the related RFC [19] was a significant part of this work.

4.2 Bulk Transfer of a Single Connection

The first experiment analyzes the performance of a single bulk data transfer in the presence of a single disconnection. A mobile host (M) transmits a fixed amount of data to its immobile correspondent host (C) over a single TCP connection. The network topology consists of 2 access points (AP1 and AP2) that are connected to a router (R). Host C is also connected to R through a fixed link. The links between the router (R) and the access points represent the links of internet core. They have much higher capacity than the access points (not the bottleneck)

and propagation delay between 10ms and 200ms. The fixed link between R and C is a virtual link used for connecting C with both access point and has perfect characteristics (very high bandwidth and almost no delay).

The mobile host (M) is first connected with C through access point 1 (AP1) and at time t_0=0 it starts transmitting. During the connection duration, M detaches from AP1 at time t_1 and remains detached for a certain period of time until t_2. Afterwards, it attaches to access point 2 (AP2) at time t_2 and remains there by the time t_3 that the data transmission ends. After that, it is assumed that the TCP connection ends and the host detaches permanently from the network. This assumption does not affect our experimental results. The following picture gives the scenario topology.

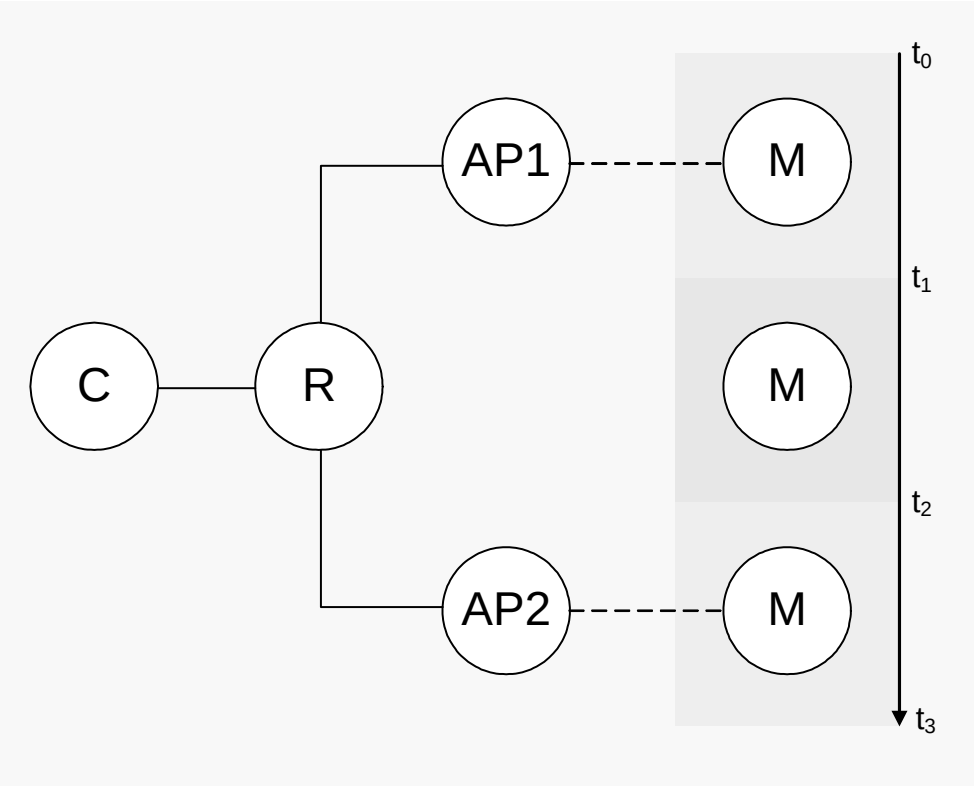


Figure 4.1: Disconnection Scenario Modelled

The first phase of the experiment between t_0 and t_1, when M is attached to AP1, is the initial connected period and it has duration of t_1. The second phase, between t_1 and t_2, is the disconnection duration and takes ($t_2 - t_1$) time. The third phase, between t_2 and t_3, when M is connected to AP2, is the final connected period and it has duration of ($t_3 - t_2$). The total connection duration is equal to t_3. We define as net connection duration the sum of the initial connected period duration and final connected period duration that is equal to ($t_3 + t_1 - t_2$).

As NS2 does not use IP addresses and ports for the TCP connection identification, the connection remains active after the change of the attach point to the network. NS2 identifies each connection using the flow ID of each connection that remains unmodified during all the connection duration. As result, there is no need of any shim layer, like Mobile IP or HIP, for mobility support.

The parameters of the simulation scenario are the following:

- The link between the AP1 and M has 6 Mbps capacity and 1ms delay.
- The link between the AP2 and M has 6 Mbps capacity and 1ms delay.
- The link between AP1 and R has 1Gbps capacity and 40ms delay.
- The link between AP2 and R has 1Gbps capacity and 40ms delay.
- The amount of transferred data is 3 Mbytes.
- TCP Sack implementing the modified Limited Transmit proposed at Section 3.5.3 is used.
- The Segment size is 1500 bytes.
- The initial connected period duration (t_1) is 2.0 sec.
- The disconnection duration ($t_2 - t_1$) is 8.0 sec.

Each link has a DropTail queue with size equal to link_bandwidth · 100ms (bits).

The following graph gives the sequence numbers of the segments sent and the ACKs received of a standard TCP connection that does not support RLCI. The two vertical lines represent t_1 and t_2 respectively.

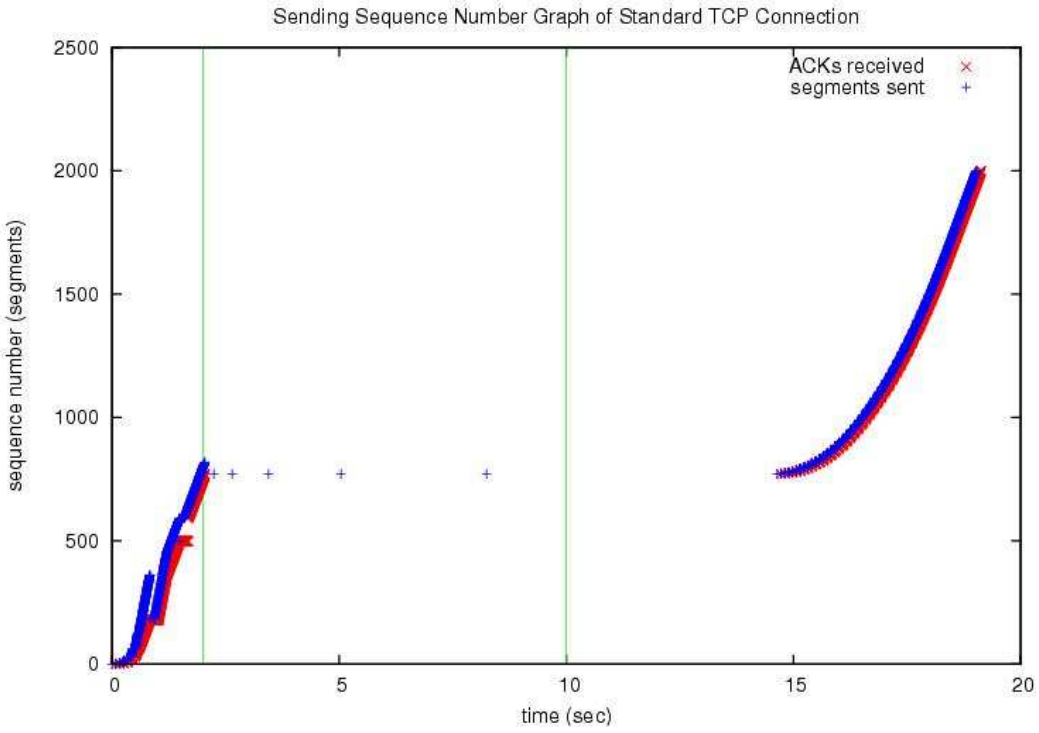


Figure 4.2: Sending Sequence Number Graph of Standard TCP Connection

After detaching from AP1 at t_1 = 2.0 sec, TCP connection stalls in exponential back-off. Due to that, even if the mobile host (M) attaches to AP2 at t_2 = 10.0 sec, the connection starts retransmitting at t = 14.6 sec. The total connection duration is 19.1 sec and the net connection duration is 11.1 sec.

The repetition of the previous simulation scenario for a TCP connection that implements RLCI gives the following sending sequence number graph:

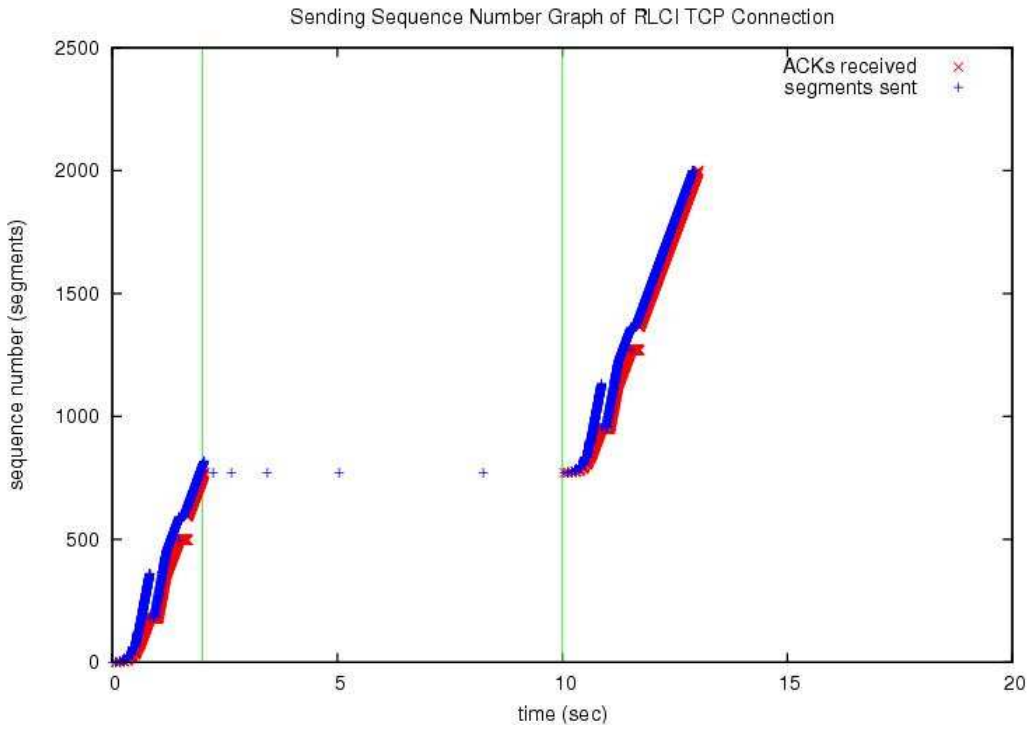


Figure 4.3: Sending Sequence Number Graph of RLCI TCP Connection

In that case, RCLI mechanism triggers the retransmission procedure immediately after the host attaches to AP2 and TCP starts re-probing the new path. The total connection duration is 13.0 sec and the net connection duration is 5.0 sec.

In order to investigate how RLCI affects TCP performance, we repeat the previous experiment multiple times using the same network topology but transferring a larger amount of data (5 Mbytes). Each time, we modify the initial connected period duration and the disconnection duration. The new parameters of the simulation scenario are summarized below:

- The link between the AP1 and M has 6 Mbps capacity and 1ms delay.
- The link between the AP2 and M has 6 Mbps capacity and 1ms delay.
- The link between AP1 and R has 1Gbps capacity and 40ms delay.
- The link between AP2 and R has 1Gbps capacity and 40ms delay.
- The amount of transferred data is 5 Mbytes.

- TCP Sack implementing the modified Limited Transmit proposed at Section 3.5.3 is used.

- The Segment size is 1500 bytes.

- The initial connected period duration (t_1) varies from 0.5 to 5 sec having a uniform distribution.

- The disconnection duration ($t_2 - t_1$) varies from 1 ms to 50 sec.

Each link has a DropTail queue with size equal to link_bandwidth · 100ms (bits).

The following graphs give the mean value and standard deviation of the net connection duration as a function of the disconnection duration. For every value of disconnection duration, the simulation scenario is repeated 15 times with different initial connected period duration and the mean value and standard deviation of the net connection duration is calculated. The initial connected period duration values are produced by a pseudorandom generator and they follow a uniform distribution between 0.5 and 5 sec.

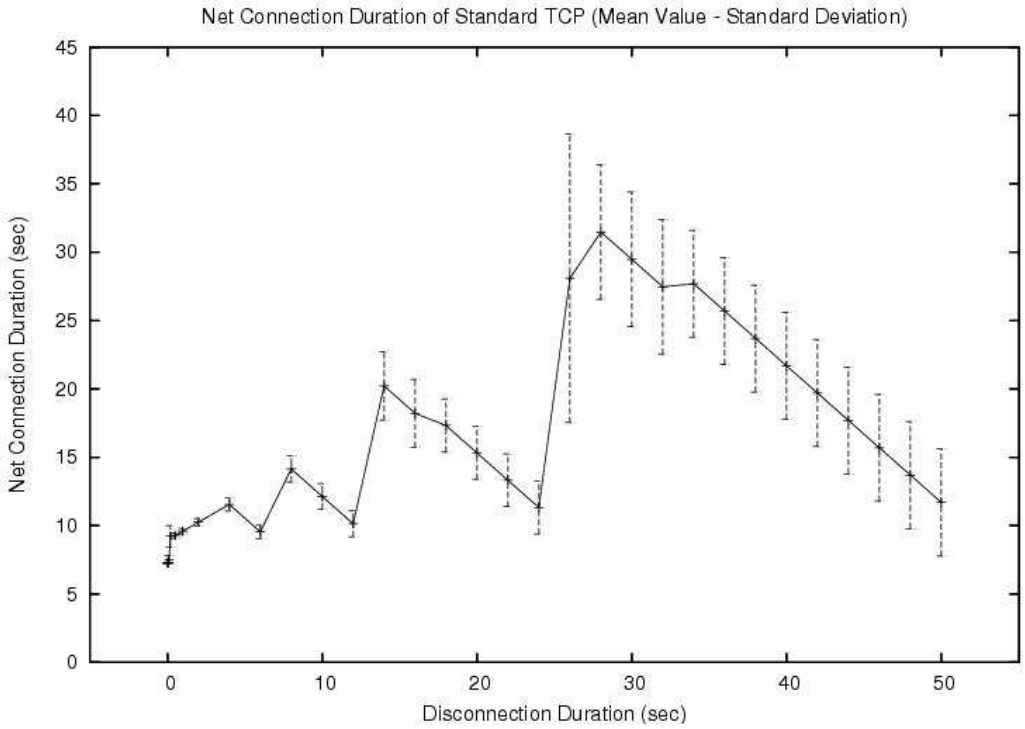


Figure 4.4: Net Connection Duration of Standard TCP

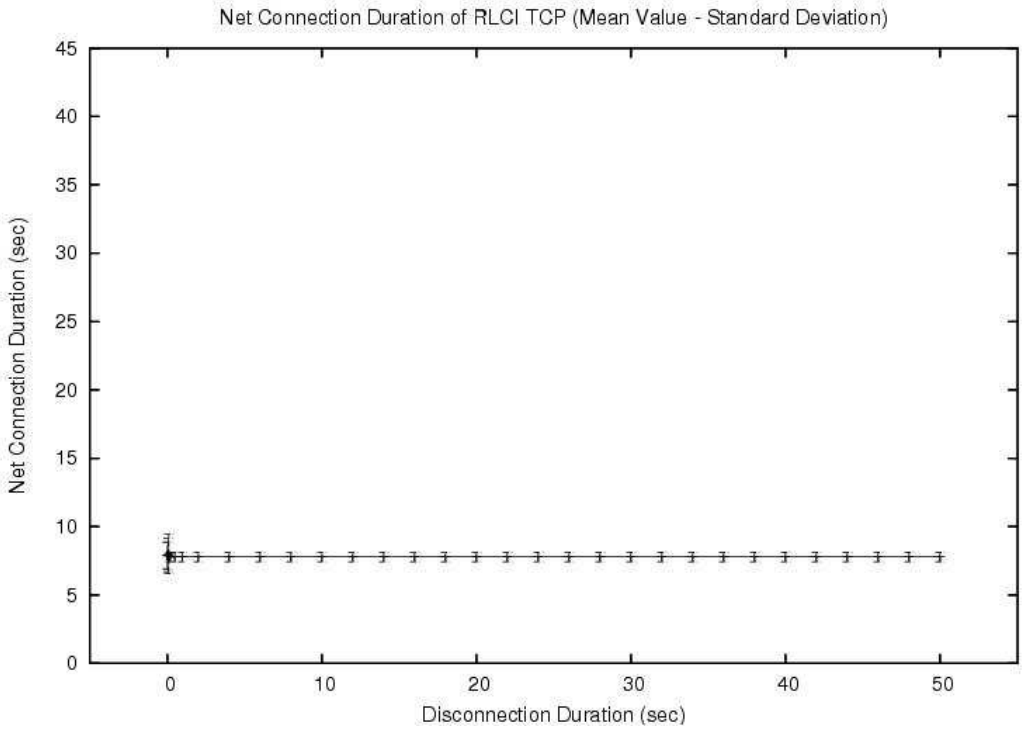


Figure 4.5: Net Connection Duration of RLCI TCP

According to graph 4.4, net connection duration of standard TCP increases in a sea saw pattern as disconnection duration increases. Furthermore, peak size increases as disconnection duration increases. This pattern is caused by the exponential back-off that TCP retransmission procedure follows. When RTO expires, back-off procedure doubles it and forces TCP to stall for that period. If connectivity recovers after last RTO expires, there is a significant period of unused connectivity before TCP starts retransmitting. This creates the peaks at the net connection duration graph. The bigger is the disconnection period, the bigger can be the unused connectivity period and as a result the net connection duration and the related peak. If connectivity recovers before last RTO expires, the period of unused connectivity is limited. In that case, net connection duration is small, causing increased TCP performance.

On the other hand the net connection duration of RLCI TCP needed for the transmission of a fixed data amount remains almost constant and independent of the disconnection duration. As RLCI mechanism immediately starts re-probing the path after a CCI, it uses more efficiently the new path than standard TCP does.

An interesting point to notice is that RLCI TCP has the longer net connection duration for small disconnection duration values. This happens when the disconnection duration is long enough to cause significant segments loss but smaller than RTO. In this case, the initialization of the congestion window (CWND) after CCI decreases the effectiveness of the fast recovery procedure that needs more time to retransmit the lost segments.

4.3 Bulk Transfer of Multiple Connections

This experiment analyzes the performance of multiple bulk data transfers in the presence of a single disconnection. A mobile host (M) having n active unidirectional TCP connections of the same direction with immobile correspondent hosts C_i (i=0,…,n-1) moves from one access point to another. Each connection transmits a fixed amount of data to the correspondent host C_i. The network topology consists of 2 access points (AP1 and AP2) that are connected to a router (R). All hosts C_i are also connected to R through fixed links. The links between the router (R) and the access points represent the links of internet core. They have much higher capacity than the access points (not the bottleneck) but significant propagation delay (40ms-100ms). The fixed links between R and C_i have also very high capacity (not the bottleneck) and a propagation delay that is specific for each connection.

The mobile host (M) is first connected to the network through access point 1 (AP1). At t=0, the mobile host (M) is attached to AP1. While the mobile host is attached to AP1, all TCP connections start transmitting with starting times that follow a uniform distribution between t=0 and $t=t_1$. At $t=t_2$, M detaches from AP1 and remains detached for a certain period of time until t_3. Afterwards, it attaches to access point 2 (AP2) at time t_3 and remains there by the time all data transmissions end. After each transmission ends, the related connection terminates. After all connection terminate, the host detaches permanently from the network. This assumption does not affect our experimental results. The following picture gives the scenario topology:

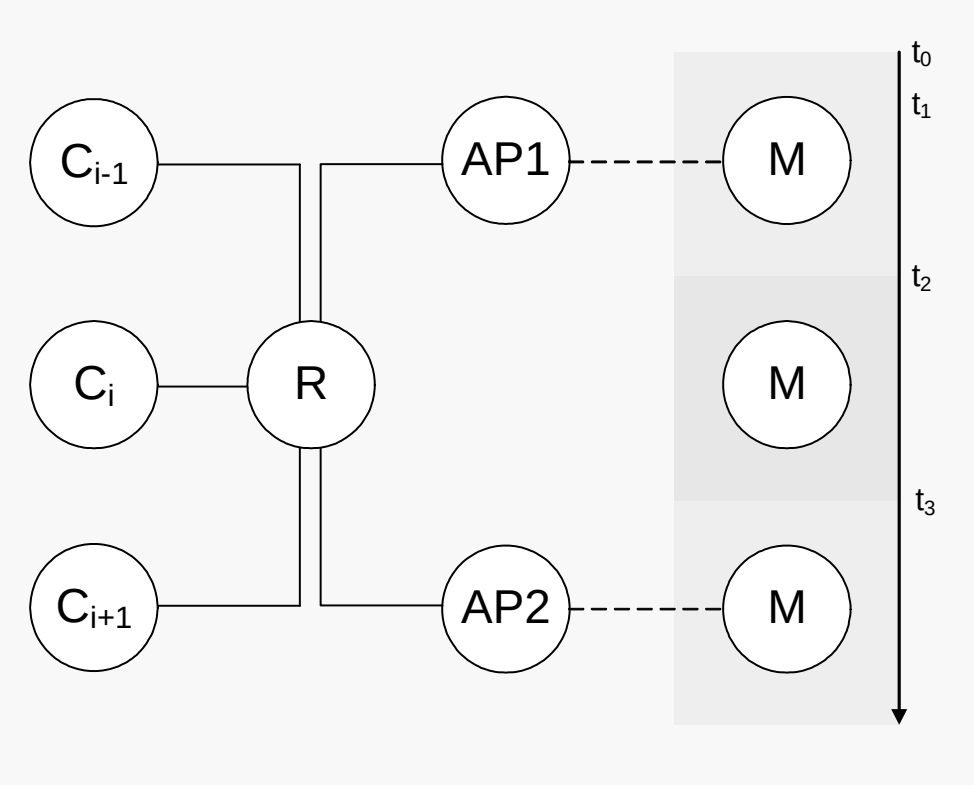


Figure 4.6: Disconnection scenario having multiple active connections

The parameters of the simulation scenario are the following:

- The link between the AP1 and M has 6 Mbps capacity and 1ms delay
- For the link between AP2 and M, there are 3 cases:
  - 1 Mbps / 100ms
  - 6 Mbps / 1ms
  - 20 Mbps /1ms
- The link between AP1 and R has 1Gbps capacity and 40ms delay.
- The link between AP2 and R has 1Gbps capacity and 40ms delay
- The links between R and C_i have 1Gbps capacity and delay that follows a uniform distribution between 0 and 50ms.
- Each link has a DropTail queue with size equal to link_bandwidth • 100ms (bits). The only exception is the 1 Mbps / 100ms link that has queue size equal to link_bandwidth • 200ms (bits).
- The number n of the mobile connections is: 1, 2, 5, 10
- TCP Sack implementing the modified Limited Transmit proposed at Section 3.5.3 is used.
- The Segment size is 1500 bytes.
- The connections starting time varies from 0 to 1 sec having a uniform distribution.
- The disconnection time varies from 4 to 6 sec having a uniform distribution.
- The disconnection duration varies from 1 sec to 30 sec.
- The amount of transferred data is calculated using the following formula in order to cause a final connected period of approximately 20 sec:

$$data = \frac{AP1\_bandwidth \cdot (t_2 - \frac{t_0 + t_1}{2}) + AP2\_bandwidth \cdot 20\,\text{sec}}{n}$$

For each set of parameters the experiment runs 10 times using random connection starting times and disconnection times and all the connection duration times are collected and sorted.

The pervious scenario is repeated twice, for standard and RLCI TCP. For each set of parameters, the mean value and standard deviation of the net connection duration difference between standard and RLCI TCP are calculated (net connection duration difference = standard TCP net connection duration – RLCI TCP net connection duration).

The following graphs present the experimental results.

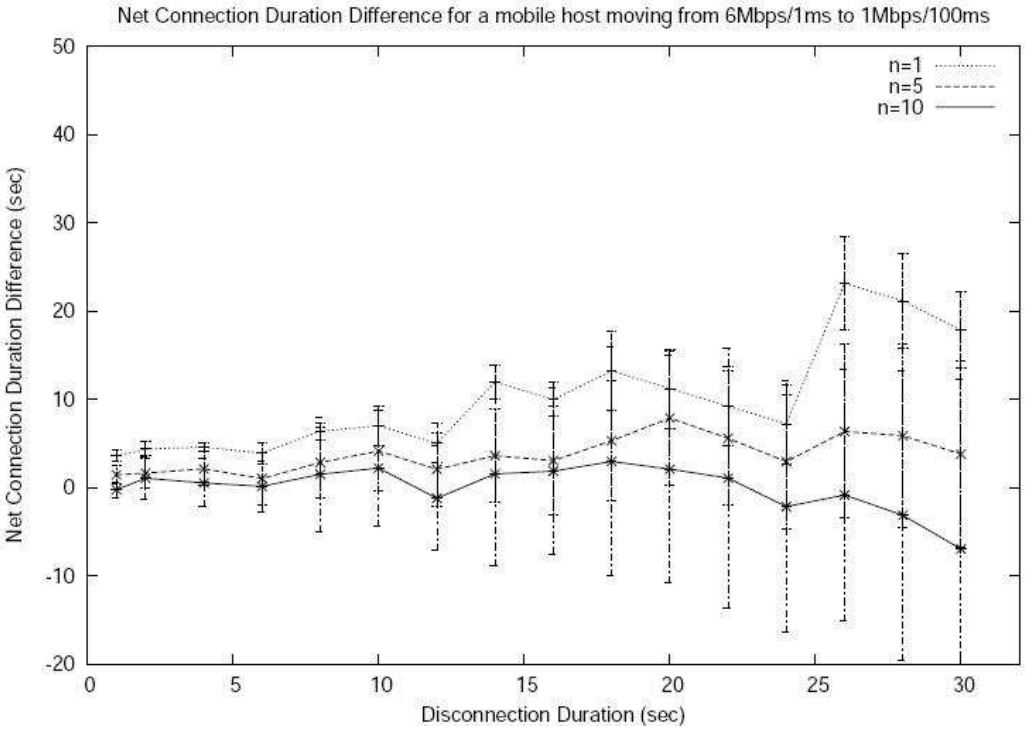


Figure 4.7: Net Connection Duration Difference between standard and RLCI TCP for a host moving from an 6Mbps/1ms AP to 1Mbps/100ms AP

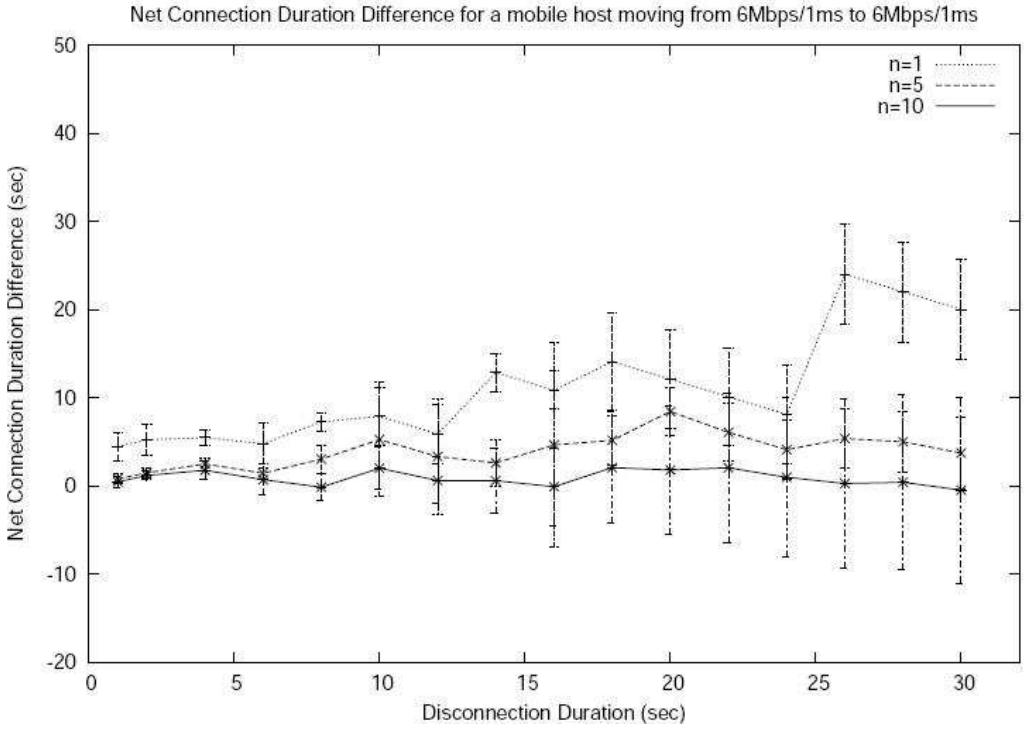


Figure 4.8 Net Connection Duration Difference between standard and RLCI TCP for a host moving from an 6Mbps/1ms AP to 6Mbps/1ms AP

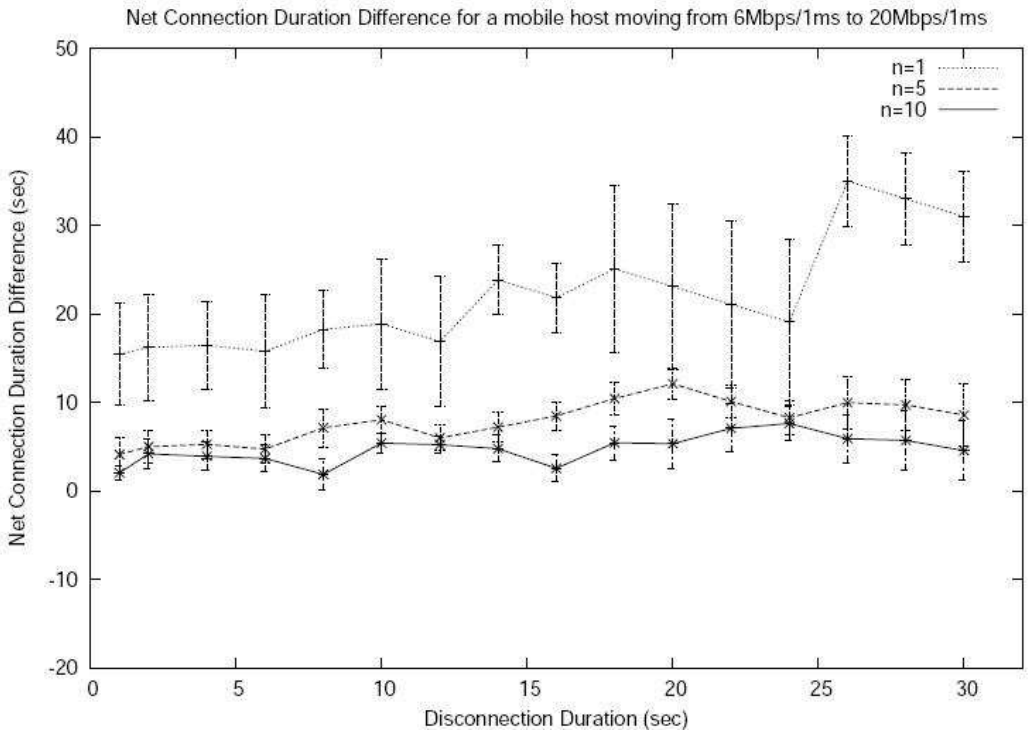


Figure 4.9: Net Connection Duration Difference between standard and RLCI TCP for a host moving from an 6Mbps/1ms AP to 20Mbps/1ms AP

From the graphs above, we can observe that if the number of the connections of a mobile host increases, RLCI performance decreases. In the extreme case, it is even worse than standard TCP. RLCI triggers all connections start retransmitting immediately after a connectivity-change indication. As all the connections begin simultaneous slow-starts, they may overshoot the new path causing significant packet drops. The increased packet drops force the TCP congestion control algorithm to transmit in a lower rate, causing decreased performance. It takes multiple round-trip times before TCP transmits in full rate. On the other hand, if standard TCP is used, even if connectivity recovers, TCP connections wait until RTO expires before start retransmitting. As RTO of each connection expires at different time, slow starts begin at different times and packet drops are fewer.

Another point to notice is that RLCI is more effective when switching to a higher capacity path. Standard TCP slow start has an SS_THRESH value equal to the half of the CWND value when last ACK received. If the mobile host moves to a higher capacity path, SS_THRESH value does not let TCP to re-probe the new path. TCP starts transmitting in a rate lower than the available capacity and it takes multiple round-trip times before TCP manages to transmit in full-rate. After a CCI, RLCI sets SS_THRESH value as in initial slow start enabling path re-probing. On the other hand, if the mobile host moves to a lower or equal capacity path, SS_THRESH not only allows transmitting in full rate, but it also may help in order to avoid new path overshooting.

On the graphs above there is a high standard deviation on the net connection duration difference between standard and RLCI TCP. After a connectivity change, RLCI immediately triggers all connections start transmitting. As a result, all of them start competing for available

bandwidth at the same time. In general terms, all the connections share equally the available capacity of the new path. On the other hand, when standard TCP is used, each connection starts transmitting on different time, when its RTO expires. For some of the connections RTO expires just after the connectivity recovery and for some others it takes longer time. The first ones are benefited by that behavior of standard TCP, as they share the available bandwidth with fewer connections for some period of time. For these connections, RLCI causes performance decrease as it forces them to share the available bandwidth with all the active TCP connections for all the time. For the standard TCP connections that wait for a long time before RTO expires, RLCI causes performance increase. As a result the standard deviation of the net connection duration difference between standard and RLCI TCP is high.

5

Related Work

This chapter discusses related work in the area of Transport Layer Performance Enhancements for Intermittent Connectivity.

Implicit Link-Up Notification [20] and Smart Link Layer [21] are solutions that attempt to avoid the unused connectivity period after a reconnection. by sending duplicates that trigger the fast recovery procedure. Both operate at network layer but they capture and buffer TCP segments for resending after reconnection. The main drawbacks include the Layering violation due to buffering of TCP segments at network layer and the potential danger of retransmitting duplicates of buffered segments after the maximum segment lifetime violating TCP semantics. Furthermore, these mechanisms become useless under the use of network layer encryption mechanisms that encrypt TCP headers.

Explicit Link Failure Notification [22] and TCP-F [23] are two protocols that base their operation to messages received from intermediate routers and inform TCP senders about disrupted paths. Explicit Link Failure Notification adds an additional “stand-by” state at TCP state machine during which all standard retransmission timers are disabled. Instead, TCP re-probes regularly the path for connectivity recovery with the potential danger of causing significant extra traffic. TCP-F completely suspends ongoing connections until receiving route reestablishment notifications that indicate peer reachability. Both designed for ad-hoc networks and base their operation on changes on intermediate routers and not at network edge. ATCP [24] uses a similar approach as the Explicit Link Failure Notification, but discovers link failures through ICMP Destination Unreachable messages.

All previous solutions attempt to detect connectivity interruptions in order to take advantage of the connectivity recovery as soon as possible and increase the path utilization. Similarly, RLCI takes advantage of the connectivity-change indications and starts transmitting immediately when receiving them, but additionally it takes into account that the new path may have different characteristics and starts re-probing the new path.

6

Conclusion

This chapter summarizes the work described on the previous chapters, concludes about 'Response to Lower-Layer Connectivity-Change Indications' benefits and proposes future work on the subject.

6.1 Recapitulation

The work described on this document is the result of a six months internship accomplished in Nokia Research Center, Helsinki. It consists of two parts:

- Design improvement of the RLCI mechanism.
- NS2 implementation and performance evaluation through simulation analysis.

The first part of the work mainly includes:

- Enrichment of RLCI with a reliable transmission of connectivity-change indications between peers based on a 3-way handshake
- Re-design of the related state machines implemented by RLCI TCP in order to support the reliable transmission of connectivity-change indications.
- Enrichment of the related CCI TCP Option, in order to support reliable transmission of connectivity-change indications.
- Use of Timestamps TCP Option for distinguishing the ACKs for segments sent through the path after CCI.
- Suggestions for modification of Limited Transmit [8] for use with RLCI.
- Simulation evaluation of the right operation of the modified mechanism using NS2 platform.

The second part mainly investigates the benefits of use of RLCI mechanism and it consists of simulation evaluation of the performance gain of mobile hosts supporting RLCI:

- Having a single active connection
- Having multiple active connections

The conclusion of this work is that in cases of mobile hosts having single active connection, RLCI can improve performance under connectivity interruptions. This mainly due to two factors:

- RLCI immediately triggers TCP start retransmitting after connectivity recovers. In case of long time interruptions that cause the connection to stall in exponential back-off, this behavior makes better use of the available connectivity.

- RLCI TCP uses slow start to re-probe the new path after a connectivity-change indication. This behavior increases TCP performance when the new path has different capacity than old one.

On the other hand, in cases of mobile hosts having multiple active connections, RLCI forces multiple simultaneous slow-starts after a connectivity recovery. This is a potential danger, as the parallel slow-starts may overshoot the new path causing increased packet drop. This impact needs further investigation.

6.2 Future Work

The work done consists of the RLCI mechanism re-design and the NS2 implementation used for simulation evaluation.

The new design is publicly available as an Internet-Draft for studding, understanding and commenting from the Internet Society. Possible future work includes implementation of the mechanism on different simulation platforms (e.g. OPNET) for cross-checking of its right operation and the investigation for potential corner-cases. Implementation of TCP RLCI mechanism on Linux kernel for use in cooperation with IP Mobile or HIP can also provide very interesting results under real conditions.

Regarding the simulation analysis, the results presented on that document are mainly based on the difference of net connection duration between the cases that RLCI mechanism is supported or not. Another very interesting approach of measuring the mechanism aggressiveness can be based on the packet drops on the routers and on the mean throughput of the connections after a connectivity-change indication.

Furthermore, in all previous scenarios it assumed that there is no background traffic, neither at the old path, nor at the new one. It is very interesting to evaluate the fairness of RLCI TCP to background traffic consisting of standard TCP. It is also very interesting to evaluate the impact of RLCI TCP to existing real-time traffic.

7

Bibliography

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

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

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

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

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

[6] 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.

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

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

[9] Droms, R., *"Dynamic Host Configuration Protocol,"* RFC 2131, March 1997 (TXT, HTML, XML).

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

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

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

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

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

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

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

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

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

[19] Blanton, E., Allman, M., Fall, K., and L. Wang, *"A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP,"* RFC 3517, April 2003.

[20] S. Dawkins, C. Williams, *"End-to-End, Implicit Link-Up Notification"*, Work in Progress (draft-dawkinstrigtran- linkup-01), October 2003.

[21] J. Scott, G. Mapp, *"Link Layer-Based TCP Optimisation for Disconnecting Networks"*, ACM SIGCOMM Computer Communications Review, Vol. 33, No. 5, October 2003

[22] G. Holland, N. Vaidya, *"Analysis of TCP Performance over Mobile Ad Hoc Networks"*, Proc. 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, WA, USA, 1999

[23] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, *"A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks"*, IEEE Personal Communication Systems (PCS) Magazine: Special Issue on Ad Hoc Networks, Vol. 8, No. 1, February 2001

[24] J. Liu, S. Singh, *"ATCP: TCP for Mobile Ad Hoc Networks"*, IEEE Journal on Selected Areas in Communication, Vol. 19, No. 7, July 2001.

Appendix

A. Internship Timetable

The work presented on this document is the result of a six months internship accomplished in Nokia Research Center (NRC), Helsinki from July 2007 to December 2007 as part of my studies at École Nationale Supérieure des Télécommunications - Institut Eurécom for obtaining the Master of Science degree in "Communications and Computer Security".

The timetable of the internship is the following:

July 2007	Studying and understanding the necessary documentation regarding the TCP mechanisms and ns2 operation. The main topics of interest were the TCP congestion control and retransmission algorithms. Furthermore documentation for ns2 TCP implementation was studied in detail.
	Creation of the appropriate network topologies at ns2 that figure the effects of mobility to transport layers.
	Implementation of a variety of automated tools that produce results and diagrams using the ns2 trace files.
	Evaluation of the simulation results and comparison with previous studies.
August 2007	Implementation of the TCP RLCI (Response to Lower-Layer Connectivity-Change Indications) mechanism at ns2 simulator as it is described at the corresponding IETF draft
	Evaluation of the performance gain of the RLCI mechanism using the appropriate simulation scenarios.
	Detection of some corner cases on the protocol operation
September 2007	Improvement of the implementation of the TCP RLCI mechanism
	Implementation of the new RLCI mechanism at ns2
	Experimental evaluation of its right operation
	Update of the appropriate IETF draft according to the current changes

October 2007	Further minor improvements on the implementation of the TCP RLCI mechanism
	Investigation of the effectiveness of RLCI in case of packet loss
	Use of Limited Transmit with RLCI
	Submission of the IETF draft
November 2007	Experimental analysis of RLCI mechanism performance using appropriate simulation scenarios
December 2007	Experimental analysis of RLCI mechanism performance
	Creation of final report

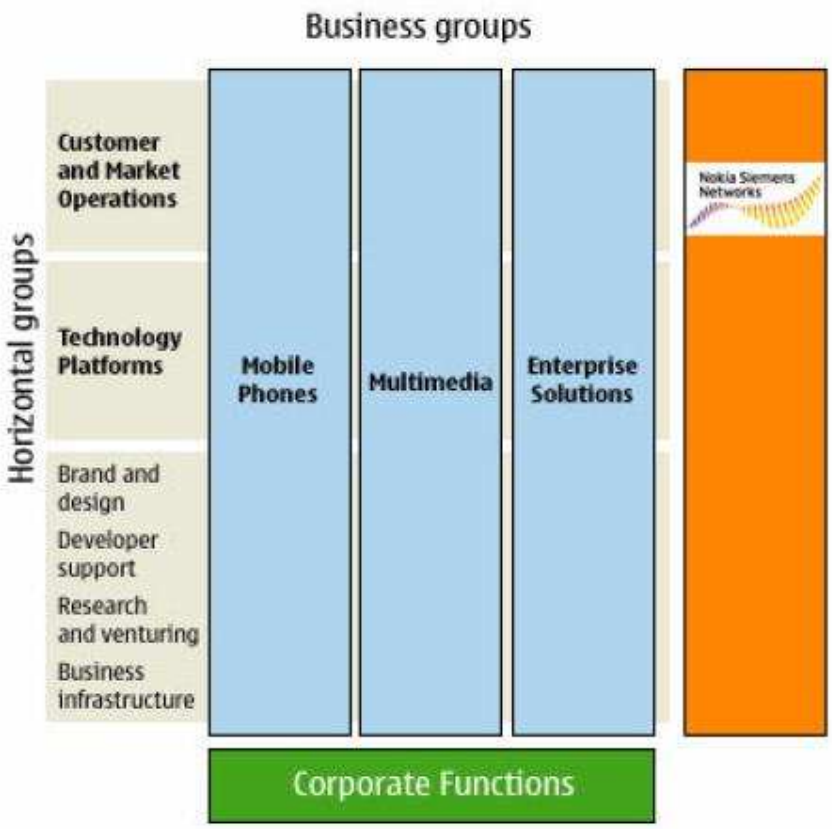
B. Company Profile

Nokia in brief

Nokia is the world leader in mobility, driving the transformation and growth of the converging Internet and communications industries. Nokia makes a wide range of mobile devices and provides people with experiences in music, navigation, video, television, imaging, games and business mobility through these devices. Nokia also provides equipment, solutions and services for communications networks.

Nokia in 2006

- World's leading manufacturer of mobile devices
- An estimated 36% share of the global device market
- Mobile device volume of 347 million units
- Net sales of EUR 41.1 billion
- Operating profit of EUR 5.5 billion
- 68,483 employees of more than 120 nationalities
- Strong R&D presence in 11 countries
- R&D investment of EUR 3.9 billion
- 21,453 people in R&D (approx. 31% of the Nokia workforce)
- 15 production facilities in nine countries
- Sales in more than 150 countries
- World's sixth most-valued brand (ranked by Interbrand)



Nokia comprises three **business groups**:

- **Mobile Phones** connects people by providing expanding mobile voice and data capabilities across a wide range of mobile devices.

- **Multimedia** gives people the ability to create, access, experience and share multimedia in the form of advanced mobile multimedia computers and applications with connectivity over multiple technology standards.

- **Enterprise Solutions** offers businesses and institutions a broad range of products and solutions, including enterprisegrade mobile devices, underlying security infrastructure, software and services.

The business groups are supported by various **horizontal entities**:

- **Customer and Market Operations** is responsible for sales and marketing, manufacturing and logistics, and sourcing and procurement for mobile devices from Mobile Phones, Multimedia and Enterprise Solutions.

- **Technology Platforms** delivers leading technologies and platforms to Nokia's business groups and external customers.

- Many other Nokia-wide horizontal units drive and manage specific Nokia assets. These include **Brand and Design**, **Developer Support**, **Research and Venturing**, and **Business Infrastructure**.

- **Corporate functions** (support Nokia's businesses with company-wide strategies and services)

Nokia Siemens Networks, which started operations on April 1, 2007, combines Nokia's networks business and Siemens' carrier-related operations for fixed and mobile networks into a company owned approximately 50% by each of Nokia and Siemens, and consolidated by Nokia.