Network Working Group
INTERNET-DRAFT
Expires: May 2002

Richard Price, Siemens/Roke Manor
Robert Hancock, Siemens/Roke Manor
Stephen McCann, Siemens/Roke Manor
Mark A West, Siemens/Roke Manor
Abigail Surtees, Siemens/Roke Manor
Paul Ollis, Siemens/Roke Manor

Qian Zhang, Microsoft Research Asia
Hongbin Liao, Microsoft Research Asia
Wenwu Zhu, Microsoft Research Asia
Ya-Qin Zhang, Microsoft Research Asia

21 November, 2001

# TCP/IP Compression for ROHC
<draft-ietf-rohc-tcp-epic-02.txt>

## Status of this Memo



This document is a submission to the IETF ROHC WG. Comments should be directed to the mailing list of ROHC, rohc@cdt.luth.se.

## Abstract

This draft describes a ROHC profile for the robust compression of TCP/IP.

The RObust Header Compression [ROHC] scheme is designed to compress packet headers over error prone channels. It is built around an extensible core framework that can be tailored to compress new protocol stacks by adding additional ROHC profiles.

The new profile for TCP/IP compression is provided by the Efficient Protocol Independent Compression (EPIC-LITE) scheme.

Table of contents



## 1. Introduction

This document describes a method for compressing TCP/IP headers within the [ROHC] framework.

The new profile for TCP/IP compression is provided by the Efficient Protocol Independent Compression (EPIC) scheme. EPIC takes as its input a BNF description of the protocol stack to be compressed, and derives a set of packet formats that can be used to quickly and efficiently compress and decompress headers.

A TCP-Aware RObust Header Compression Control scheme, TAROC-C, is also introduced in this draft. The key point of TAROC-C is the TCP congestion window tracking mechanism, which can be used to improve the efficiency of the window-based encoding and the performance of the overall header compression scheme without sacrificing the robustness. With the dynamic congestion window tracking, our scheme can achieve good performance even when the feedback channel is not available.

## 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 [RFC-2119].

## 3. ROHC Profile for compression of TCP/IP

This chapter describes a simple ROHC profile for the compression of TCP/IP.

Note that the current TCP/IP profile is designed specifically to test implementations of [EPIC]. The profile is not designed to compress TCP/IP with a high level of efficiency.

The profile supports all TCP options (it does not compress the options, but instead passes them through transparently as part of the payload).

The profile for TCP/IP compression is given below:

```
profile_identifier      0xFFFF
max_formats             200
max_sets                1
bit_alignment           8
npatterns               224
CO_packet               TCP-IP

; The profile identifier is a placeholder.

; The IR-DYN_packet and IR_packet parameters are not specified. This
; means that the IR-DYN and IR packets are generated using the same
; encoding method "TCP/IP" as for the CO packets.

; The encoding methods used by the TCP/IP profile are given below:

TCP-IP          =    IPv4-header
                     TCP-header
                     msn

msn             =    C(MSN-LSB(4,-1,90%)) | C(MSN-LSB(7,-1,9%)) |
                     MSN-IRREGULAR(16,1%)

IPv4-header     =    version
                     header_len
                     tos
                     ecn
                     length
                     ip-id
                     rf_flag
```

```
                        df_flag
                        mf_flag
                        offset
                        ttl
```

```
                        protocol
                        ip_chksum
                        src_address
                        dst_address

   version          =   STATIC-KNOWN(4,4)

   header_len       =   STATIC-KNOWN(4,5)

   tos              =   C(STATIC(99%)) | IRREGULAR(6,1%)

   ecn              =   IRREGULAR(2,100%)

   length           =   IRREGULAR(16)

   ip-id            =   C(LSB(4,-1,90%)) | C(LSB(6,-1,8%)) |
                        C(LSB(8,-1,1%))  | IRREGULAR(16,1%)

   rf_flag          =   VALUE(1,0,100%)

   df_flag          =   IRREGULAR(1,100%)

   mf_flag          =   VALUE(1,0,99%) | VALUE(1,1,1%)

   offset           =   C(STATIC(99%)) | IRREGULAR(13,1%)

   ttl              =   C(STATIC(99%)) | IRREGULAR(8,1%)

   protocol         =   STATIC-KNOWN(8,6)

   ip_chksum        =   IRREGULAR(16,100%)

   src_address      =   STATIC-UNKNOWN(32)

   dst_address      =   STATIC-UNKNOWN(32)

   TCP-header       =   source_port
                        dest_port
                        seqno
                        ackno
                        data_offset
                        flags
                        window
                        tcp_chksum
                        urg_ptr

   source_port      =   STATIC-UNKNOWN(16)

   dest_port        =   STATIC-UNKNOWN(16)
```

```
   seqno           =    C(LSB(8,63,80%))  | C(LSB(14,127,10%)) |
                        C(LSB(20,1023,5%)) | IRREGULAR(32,5%)

   ackno           =    C(LSB(8,-1,80%))  | C(LSB(14,-1,10%))  |
```

```
                        C(LSB(20,-1,5%))   | IRREGULAR(32,5%)

   data_offset      =    IRREGULAR(4,100%)

   window           =    C(STATIC(80%)) | C(LSB(12,63,10%)) |
                         IRREGULAR(16,10%)

   tcp_chksum       =    IRREGULAR(16,100%)

   urg_ptr          =    C(STATIC(99%)) | IRREGULAR(16,1%)

   flags            =    reserved
                         cwr
                         ece
                         urg
                         ack
                         psh
                         rst
                         syn
                         fin

   reserved         =    C(STATIC(90%)) | IRREGULAR(4,10%)

   cwr              =    VALUE(1,0,80%) | VALUE(1,1,20%)

   ece              =    VALUE(1,0,80%) | VALUE(1,1,20%)

   urg              =    VALUE(1,0,99%) | VALUE(1,1,1%)

   ack              =    VALUE(1,1,99%) | VALUE(1,0,1%)

   psh              =    IRREGULAR(1,100%)

   rst              =    VALUE(1,0,99%) | VALUE(1,1,1%)

   syn              =    VALUE(1,0,99%) | VALUE(1,1,1%)

   fin              =    VALUE(1,0,95%) | VALUE(1,1,5%)
```

## 4. The concept and framework of TAROC-C

This section first describes the concept of the TCP-aware robust header compression control (TAROC-C) mechanism and then discusses how this concept leads to a better performance when used over unreliable links.

To design suitable mechanisms for efficient compression of all TCP/IP header fields, it would be important to analyze their change patterns first. It is known that the change patterns of several TCP fields

(for example, Sequence Number, Acknowledgement Number, Window, etc.) are completely different from the ones of RTP, which had already discussed in detail in [ROHC], and are very hard to predict. Thus, it is hard to encode these fields with k-LSB both efficiently and

robustly. On the other hand, Window-based LSB encoding [ROHC], which does not assume the linear changing pattern of the target header fields, is more suitable to encode those TCP fields both efficiently and robustly.

The main idea of TAROC-C, the control mechanism of TAROC, is the combination of the Window-based LSB encoding (W-LSB encoding) and dynamically TCP congestion window tracking. In W-LSB encoding, a sliding window (VSW), which equals to value r mentioned in Section 6.4, is maintained on the compressor side. The compressor gets inconsistent with the decompressor only when the reference value on the decompressor side is out of this VSW. By keeping the sliding window large enough, the compressor rarely gets out of synchronization with the decompressor.

However, the larger the sliding window is, the less the header compression gains. To shrink the window size, the compressor needs some form of feedback to get sufficient confidence that a certain value will not be used as a reference by the decompressor. Then the window can be advanced by removing that value and all other values older than it. Obviously, when a feedback channel is available, confidence can be achieved by proactive feedback in the form of ACKs from the decompressor. A feedback channel, however, is unavailable or expensive in some environments. In this Internet draft, a mechanism based on dynamically tracking TCP congestion window is proposed to explore such feedbacks from the nature feedback-loop of TCP protocol itself.

Since TCP is a window-based protocol, a new segment cannot be transmitted without getting the acknowledgment of segment in the previous window. Upon receiving the new segment, the compressor can get enough confidence that the decompressor has received the segment in the previous window and then shrink the sliding window by removing all the values older than that segment.

As originally outlined in [CONG1] and specified in [CONG2], TCP is incorporated with four congestion control algorithms: slow-start, congestion-avoidance, fast retransmit, and fast recovery. The effective window of TCP is mainly controlled by the congestion window and may change during the entire connection life. TAROC-C designs a mechanism to track the dynamics of TCP congestion window, and control the sliding window of W-LSB encoding by the estimated congestion window. By combining the W-LSB encoding and TCP congestion window tracking, TAROC can achieve better performance over high bit-error-rate links.

Note that in one-way TCP traffic, only the information about sequence number or acknowledgment number is available for tracking TCP congestion window. TAROC-C does not require that all one-way TCP

traffics must cross the same compressor. The detail will be described in the following sections.

The TAROC scheme achieves its compression gain by establishing state information at both ends of the link, i.e., at the compressor and at

the decompressor. Header compression with TAROC can be characterized as an interaction between two state machines, one compressor machine and one decompressor machine, each instantiated once per context.

In the rest of this session, the TCP congestion window tracking algorithm, the state machines in the TCP/IP header compression framework, and the logics of the compressor/decompressor, are described in detail.

## 4.1. TCP congestion window tracking

### 4.1.1. General principle of congestion window tracking

The general principle of congestion window tracking is as follows. The compressor imitates the congestion control behavior of TCP upon receiving each segment, in the meantime, estimates the congestion window (cwnd) and the slow start threshold (ssthresh). Besides the requirement of accuracy, there are also some other requirements for the congestion window tracking algorithms:

- Simplex link. The tracking algorithm SHOULD always only take Sequence Number or Acknowledgment Number of a one-way TCP traffic into consideration. It SHOULD NOT use Sequence Number and Acknowledgment Number of that traffic simultaneously.

- Misordering resilience. The tracking algorithm SHOULD work well while receiving misordered segments.

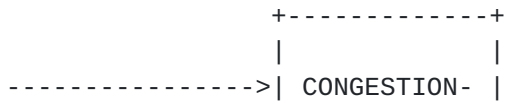
- Multiple-links. The tracking algorithm SHOULD work well when not all the one-way TCP traffics are crossing the same link.

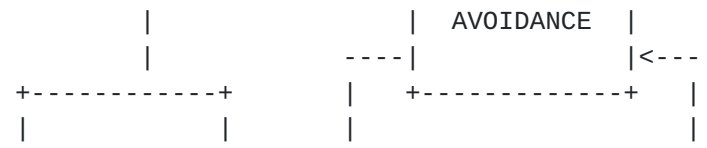
- Slightly overestimation. If the tracking algorithm cannot guarantee the accuracy of the estimated cwnd and ssthresh, it is RECOMMANDED that it produces a slightly overestimated one.

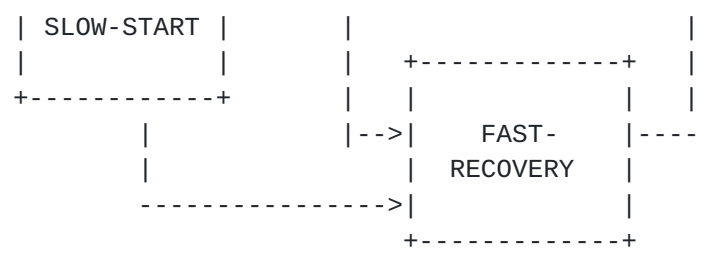
The following sections will describe two congestion window tracking algorithms, which use Sequence Number and Acknowledgment Number of a one-way TCP traffic, respectively.

### 4.1.2. Congestion window tracking based on Sequence Number

This algorithm (Algorithm SEQ) contains 3 states: SLOW-START, CONGESTION-AVOIDANCE, and FAST-RECOVERY, which are equivalent to the states in TCP congestion control algorithms. It maintains 2 variables: cwnd and ssthresh.




AVOIDANCE




Initially, this algorithm starts in state SLOW-START with ssthresh set to ISSTHRESH and cwnd set to IW.

Upon receiving a segment, if it is the first segment, which is not necessary to be the SYN segment, the algorithm sets the current maximum Sequence Number (CMAXSN) and the current minimum Sequence Number (CMINSN) to this segment's sequence number; otherwise, the algorithm takes a procedure according to the current state.

- SLOW-START

  * If the new Sequence Number (NSN) is larger than CMAXSN, increase cwnd by the distance between NSN and CMAXSN, and update CMAXSN and CMINSN based on the following rules:

```
CMAXSN = NSN
if (CMAXSN - CMINSN) > cwnd)
    CMINSN = cwnd - CMAXSN;
```

    If the cwnd is larger than ssthresh, the algorithm transits to CONGESTION-AVOIDANCE state;

  * If the distance between NSN and CMAXSN is less than or equal to 3*MSS, ignore it;

  * If the distance is larger than 3*MSS, halve the cwnd and set ssthresh to MAX(cwnd, 2*MSS). After that, the algorithm transits into FAST-RECOVERY state.

- CONGESTION-AVOIDANCE

  * If NSN is larger than CMAXSN, increase cwnd by ((NSN-CMAXSN)*MSS)/cwnd and then update CMAXSN and CMINSN based on the following rules:

```
CMAXSN = NSN
if (CMAXSN - CMINSN) > cwnd)
    CMINSN = cwnd - CMAXSN;
```

  * If the distance between NSN and CMAXSN is less than or equal to 3*MSS, ignore it;

  * If the distance is larger than 3*MSS, halve the cwnd and set ssthresh to MAX(cwnd, 2*MSS). After that, the algorithm

transits into FAST-RECOVERY state.

- FAST-RECOVERY

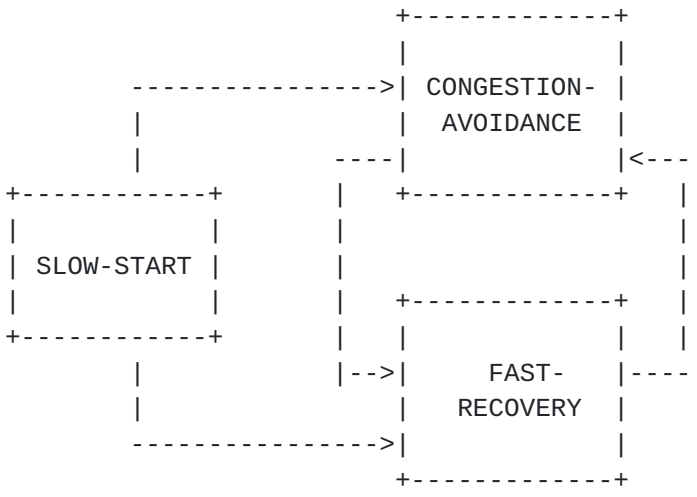
* If NSN is larger than or equal to CMAXSN + cwnd, the algorithm transits into CONGESTION-AVOIDANCE state;

* Otherwise, ignore it.

In this algorithm, MSS is denoted as the estimated maximum segment size. The implementation can use the MTU of the link as an approximation of this value. ISSHRESH and IW are the initial values of ssthresh and cwnd, respectively. ISSTHRESH MAY be arbitrarily high. IW SHOULD be set to 4*MSS.

### 4.1.3. Congestion window tracking based on Acknowledgment Number

This algorithm (Algorithm ACK) maintains 3 states: SLOW-START, CONGESTION-AVOIDANCE and FAST-RECOVERY, which are equivalent to the states in TCP congestion control algorithms. It also maintains 2 variables: cwnd and ssthresh.




Initially, this algorithm starts in state SLOW-START with ssthresh set to ISSTHRESH and cwnd set to IW.

Upon receiving a segment, if it is the first segment, which is not necessary to be the SYN segment, the algorithm sets the current maximum Acknowledgment Number (CMAXACK) to this segment's acknowledgment number; otherwise, the algorithm takes a procedure according to the current state.

- SLOW-START

  * If the new Acknowledgment Number (NEWACK) is larger than CMAXACK, increase cwnd by the distance between NEWACK and CMAXACK, set duplicate ack counter (NDUPACKS) to 0, and update CMAXACK accordingly; If the cwnd is larger than ssthresh, the algorithm transits to CONGESTION-AVOIDANCE state;

* If NEWACK is equal to CMAXACK, increase the NDUPACKS by 1. If NDUPACKS is greater than 3, halve the cwnd and set ssthresh to

MAX(cwnd, 2*MSS). Consequently, the algorithm transits into FAST-RECOVERY state;

* Otherwise, set NDUPACKS to 0.

- CONGESTION-AVOIDANCE

* If NEWACK is larger than CMAXACK, increase cwnd by ((NEWACK-CMAXACK)*MSS)/cwnd, set NDUPACKS to 0 and update CMAXACK accordingly;

* If NEWACK is equal to CMAXACK, increase NDUPACKS by 1. If NDUPACKS is greater than 3, halve the cwnd and set ssthresh to MAX(cwnd, 2*MSS). After that, the algorithm transits into FAST-RECOVERY state;

* Otherwise, set NDUPACKS to 0.

- FAST-RECOVERY

* If NEWACK is larger than CMAXACK, set NDUPACKS to 0. Consequently, the algorithm transits into CONGESTION-AVOID state;

* Otherwise, ignore it.

In this algorithm, MSS is denoted as the estimated maximum segment size. The implementation can use the MTU of the link as an approximation of this value. ISSHRESH and IW are the initial values of ssthresh and cwnd, respectively. ISSTHRESH MAY be arbitrarily high. IW SHOULD be set to 4*MSS.

### 4.1.4. Further discussion on congestion window tracking

In some cases, it is inevitable for the tracking algorithms to overestimate the TCP congestion window. Also, it SHOULD be avoided that the estimated congestion window gets significantly smaller that the actual one. For all of these cases, TAROC simply applies two boundaries on the estimated congestion window size. One of the two boundaries is the MIN boundary, which is the minimum congestion window size and whose value is determined according to the [INITWIN]; the other boundary is the MAX boundary, which is the maximum congestion window size. There are two possible approaches to setting this MAX boundary. One is to select a commonly used maximum TCP socket buffer size. The other one is to use the simple equation W=sqrt(8/3*l), where W is the maximum window size and l is the typical packet loss rate.

If ECN mechanism is deployed, according to [RFC-2481] and [ECN], the TCP sender will set the CWR (Congestion Window Reduced) flag in the

TCP header of the first new data packet sent after the window reduction, and the TCP receiver will reset the ECN-Echo flag back to 0 after receiving a packet with CWR flag set. Thus, the CWR flag and the ECN-Echo flag's transition from 1 to 0 can be used as another

indication of congestion combined with other mechanisms mentioned in the tracking algorithm.

## 4.2. Compressor/decompressor state machine with TAROC-C

### 4.2.1. Compressor states

There are three compressor states in TAROC: Initialization and Refresh (IR) state, First Order (FO), and Second Order (SO) states. The compressor starts in the lowest compression state (IR) and transits gradually to the higher compression state. The compressor will always operate in the highest possible compression state, under the constraint that the compressor is sufficiently confident that the decompressor has the information necessary to decompress a header, which is compressed according to the state.


#### 4.2.1.1. Initialization and Refresh (IR) state

The purpose of IR state is to initialize or refresh the static parts of the context at the decompressor. In this state, the compressor sends full header periodically with an exponentially increasing period, which is so-called compression slow-start [RFC-2507]. The compressor leaves the IR state only when it is confident that the decompressor has correctly received the static information.

To compress short-lived TCP transfers more efficiently, the compressor should speed up the initial process. The compressor enters the IR state when it receives the packet with SYN bit set and sends IR packet. When it receives the first data packet of the transfer, it should transit to FO state because that means the decompressor has received the packet with SYN bit set and established the context successfully at its side. Using this mechanism can significantly reduce the number of context initiation headers.

#### 4.2.1.2. First Order (FO) State

The purpose of FO state is to efficiently transmit the difference

between the two consecutive packets in the TCP stream. When operating in this state, the compressor and the decompressor should have the same context. Only compressed packet is transmitted from the compressor to the decompressor in this state. The compressor transits back to IR state only when it finds that the context of decompressor

may be inconsistent, or there are remarkable changes in the TCP/IP header.

#### 4.2.1.3. Second Order (SO) State

The purpose of SO state is to efficiently transmit the fixed-payload data. The compressor enters this state when it is sufficiently confident that the decompressor has got the constant payload size of the data transferring.

The compressor leaves this state and transits to the FO state when the current payload size no longer conforms to the constant payload. The compressor transits back to IR state only when it finds that the context of decompressor may be inconsistent, or there are remarkable changes in the TCP/IP header.

### 4.2.2. Decompressor states

The decompressor starts in its lowest compression state, "No Context" and gradually transits to higher state, "Full Context". The decompressor state machine normally never leaves the "Full Context" state once it has entered this state.


## 4.3. Compressor logic in TAROC-C

In TAROC-C, the compressor will start in the IR state and perform different logics in different states. The following sub-sections will describe the logic for each compressor sate in detail.

### 4.3.1. IR state

The operations of compressor in IR state can be summarized as follows:

a) Upon receiving a packet, the compressor sends IR or IR-DYN packet on the following conditions: 1) if it is the turn to send full header packet according to compression slow-start, i.e. after sending F_PERIOD compressed packets; 2) if the packet to be sent is a retransmission of the packet in VSW and it was sent as IR or IR-DYN packet previously. Otherwise, the compressor compresses the packet using W-LSB encoding. If the compressor enters the IR state for the first time or the static part of the TCP flow has changed, it will send IR packet. Otherwise, it will send IR-DYN packet because the decompressor has known the static part.

b) The packet is added into VSW as a potential reference after it has been sent out. The compressor then invokes the Algorithm SEQ

and Algorithm ACK to track the congestion windows of the two one-way traffics with different directions in a TCP connection. Suppose that the estimated congestion windows are cwnd_seq and cwnd_ack, while the estimated slow start thresholds are

ssthresh_seq and ssthresh_ack, respectively. Let W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack) = K*MAX(MAX(cwnd_seq, 2*ssthresh_seq), MAX(cwnd_ack, 2*ssthresh_ack)). If the size of VSW is larger than W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack), the VSW can be shrunk. K is an implementation parameter that will be further discussed in Section 5.6.

c) After sending F_PERIOD compressed packets, F_PERIOD SHOULD be doubled. If it gets larger than W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack), the compressor transits to FO or SO state. If the compressor finds that the payload size of consecutive packets is a constant value and one of such packets is removed from the VSW, which means the decompressor has known the exact value of the constant size, it may transit to SO state. Otherwise it will transit to the FO state.

#### 4.3.2. FO state

The operations of the compressor in the FO state can be summarized as follows:

a) Upon receiving a packet, if it falls behind the VSW, i.e. it is older than all the packets in VSW; the compressor transits to IR state. Otherwise, the compressor compresses it using W-LSB encoding and sends it.

b) The packet is added into VSW as a potential reference after it has been sent out. The compressor then invokes the Algorithm SEQ and Algorithm ACK to track the congestion windows of the two one-way traffics with different directions in a TCP connection. Suppose that the estimated congestion windows are cwnd_seq and cwnd_ack, while the estimated slow start thresholds are ssthresh_seq and ssthresh_ack, respectively. Let W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack) = K*MAX(MAX(cwnd_seq, 2*ssthresh_seq), MAX(cwnd_ack, 2*ssthresh_ack)). If the size of VSW is larger than W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack), the VSW can be shrunk. K is also an implementation parameter, which can be set to the same value as in the IR state.

c) If the VSW contains only one packet, which means there is a long jump in the packet sequence number or acknowledge number, the compressor will transit to the IR state and re-initialize the algorithm for tracking TCP congestion window. Here, a segment causes a long jump when the distance between its sequence number (or acknowledgment number) and CMAXSN (or CMAXACK) is larger than the estimated congestion window size, i.e.,
|sequence number (acknowledgement number) û CMAXSN (CMAXACK)| > estimated congestion window size.

d) If the compressor finds that the payload size of consecutive packets is a constant value and one of such packets has been removed from the VSW, which means the decompressor has known the exact value of the constant size, it may transit to the SO state.

e) If the static context of transfers changed, the compressor will transit to the IR state and re-initialize the algorithms for tracking TCP congestion window.

#### 4.3.3. SO state

The operations of the compressor in the SO state can be summarized as follows:

a) Upon receiving a packet, if it falls behind the VSW, i.e. it is older than all the packets in VSW; the compressor transits to IR state. Otherwise, the compressor compresses it using fixed-payload encoding and sends it.

b) The packet is added into VSW as a potential reference after it has been sent out. The compressor then invokes the Algorithm SEQ and Algorithm ACK to track the congestion windows of the two one-way traffics with different directions in a TCP connection. Suppose that the estimated congestion windows are cwnd_seq and cwnd_ack, while the estimated slow start thresholds are ssthresh_seq and ssthresh_ack, respectively. Let W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack) = K*MAX(MAX(cwnd_seq, 2*ssthresh_seq), MAX(cwnd_ack, 2*ssthresh_ack)). If the size of VSW is larger than W(cwnd_seq, ssthresh_seq, cwnd_ack, ssthresh_ack), the VSW can be shrunk. K is an implementation parameter, which can be set to the same value as in the IR state.

c) If the VSW contains only one packet, which means there is a long jump in the packet sequence number or acknowledge number, the compressor will transit to the IR state and re-initialize the algorithms for tracking TCP congestion window.

d) If the payload size of the packets in VSW doesn't keep constant, the compressor transits to the FO state.

e) If the static context of transfers changed, the compressor will transit to the IR state and re-initialize the algorithms for tracking TCP congestion window.

### 4.4. Decompressor logic in TAROC-C

The logic of the decompressor is simpler compared to the compressor.

#### 4.4.1. No Context State

The decompressor starts in this state. Upon receiving an IR or IR-DYN packet, the decompressor should verify the correctness of its header by TCP checksum. If the verification succeeds, the decompressor will update the context and use this packet as the reference packet. After

that, the decompressor will pass it to the system's network layer and transit to Full Context State. Conformed to ROHC framework [ROHC], only IR or IR-DYN packets may be decompressed in No Context state.

### 4.4.2. Full Context State

The operations of decompressor in Full Context state can be summarized as follows:

a) Upon receiving an IR or IR-DYN packet, the decompressor should verify the correctness of its header by TCP checksum. If the verification succeeds, the decompressor will update the context and use this packet as the reference packet. Consequently, the decompressor will convert the packet into the original packet and pass it to the network layer of the system.

b) Upon receiving the other type of packet, the decompressor will decompress it. After that, the decompressor MUST verify the correctness of the decompressed packet by the TCP checksum. If the verification succeeds, the decompressor passes it to the system's network layer. Then the decompressor will use it as the reference value if this packet is not older than the current reference packet.

c) If consequent N packets fail to be decompressed, the decompressor should transit downwards to No Context State. N is an implementation parameter that will be further discussed in Section 8.6.

## 4.5. Modes of operation

There are three modes in ROHC framework, called Unidirectional, Bi-directional Optimistic, and Bi-directional Reliable mode, respectively. The mode transitions are conformed to ROHC framework. However, the operations of each mode are different.

### 4.5.1. Unidirectional mode -- U-mode

When in U-mode, packets are sent in one direction only: from compressor to decompressor. Therefore, feedbacks from decompressor to the compressor are unavailable under this mode.

In the U-mode, the compressor and decompressor logic is the same as the discussion in section 8.3 and 8.4.

### 4.5.2. Bi-directional Optimistic mode -- O-mode

When in O-mode, a feedback channel is used to send error recovery requests and (optionally) acknowledgments of significant context updates from the decompressor to the compressor. In this mode, the VSW will be shrunk more efficiently.

#### 4.5.2.1. Compressor states and logic (O-mode)

Following rules should be combined with the action defined in section

8.3.

In the IR state, the compressor can transit to the FO or SO state once it receives a valid ACK(O) for an IR packet sent (an ACK(O) can

only be valid if it refers to a packet sent earlier). If the packet referred by the feedback is in the VSW, the compressor will remove the packets older than the referred packet from the VSW window. Because ACK(O) means that the packet referred by ACK(O) has been the reference of the decompressor, the compressor doesn't need to keep older packets.

If the compressor is in the FO or SO state, it will remove the packets older than the referred packet from the VSW window.

Upon receiving an NACK(O), the compressor transits back to IR state.

#### 4.5.2.2. Decompressor states and logic (O-mode)

The decompression states and the state transition logic are the same as in the Unidirectional case (see section 8.5.1.). What differs is the feedback logic.

Below, rules are defined stating which feedback to use when.

When an IR packet passes the verification, send an ACK(O). When an IR-DYN packet or other packet is correctly decompressed, optionally send an ACK(O). When any packet fails the verification, send an NACK(O).

### 4.5.3. Bi-directional Reliable mode -- R-mode

The R-mode are a more intensive usage of the feedback channel and a stricter logic at both the compressor and the decompressor that prevents loss of context synchronization between the compressor and decompressor except for very high residual bit error rates. Feedback is sent to acknowledge all context updates. In this mode, the VSW will be shrunk with the highest efficiency.

#### 4.5.3.1. Compressor states and logic (R-mode)

Following rules should be reparation to the action defined in section 8.3.

In IR state, the compressor should transit to the FO or SO state only when it receives a valid ACK(R) for an IR or IR-DYN packet sent (an ACK(R) can only be valid if it refers to an packet sent earlier). If the packet referred by the feedback is in the VSW, the compressor will remove the packets older than the referred packet from the VSW window. Because ACK(R) means that the packet referred by ACK(R) has been the reference of the decompressor; the compressor doesn't need to keep older packets.

If the compressor is in the FO or SO state, when it receives a valid ACK(R), it will remove the packets older than the referred packet

from the VSW window. In this mode, the compressor need not use window tracking, because feedback can shrink VSW efficiently and robustly.

Upon receiving an NACK(O), the compressor transits back to IR state.

#### 4.5.3.2. Decompressor states and logic (R-mode)

Below, rules are defined stating which feedback to use when.

@When a packet is correctly decompressed and updates the context, send an ACK(R).

@When any packet fails the verification, send a NACK(R).

The frequency of updating context will be discussed in section 8.6.

## 4.6. Implementation issues

### 4.6.1. Determine the value K

As mentioned above, the VSW SHOULD be shrunk when its size gets larger than K*MAX(MAX(cwnd_seq, 2*ssthresh_seq), MAX(cwnd_ack, 2*ssthresh_ack)). Since the Fast Recovery algorithm was introduced in TCP, several TCP variants had been proposed, which are different only in the behaviors of Fast Recovery. Some of them need several RTTs to be recovered from multiple losses in a window. Ideally, they may send L*W/2 packets in this stage, where L is the number of lost packets and W is the size of the congestion window where error occurs. Some recent work [TCPREQ] on improving TCP performance allows to transmit packets even when receiving duplicate acknowledgments. Due to the above concerns, it'd better keep K large enough so as to prevent shrinking VSW without enough confidence that corresponding packets had been successfully received.

Considering the bandwidth-limited environments and the limited receiver buffer, a practical range of K is around 1~2. From the simulation results, K=1 is good enough for most cases.

### 4.6.2. Determine the value N

We should distinguish out of synchronization from the packet errors cause by the link. So considering the error condition of the link, N should be higher than the packet burst error length, a practical range of N is around 8~10.

### 4.6.3. Determine the frequency of updating context

The choice of the frequency of updating context, ACK(R), is a balance between the efficiency and robustness, i.e. sending ACK(R) more frequently improves the compression robustness but adds more system overhead, and the vice versa. From a practical view, the ACK(R) SHOULD be sent for every 4~8 successfully decompressed packets.

## 4.7. Performance of TAROC-C

The Simulations results (see Appendix B in [TAROC-3]) demonstrate the effectiveness of control mechanism TAROC-C and corresponding header compression scheme.

## 5. Security considerations

EPIC-LITE generates compressed header formats for direct use in ROHC profiles. Consequently the security considerations for EPIC-LITE match those of [ROHC].

## 6. Acknowledgements

Header compression schemes from [ROHC] have been important sources of ideas and knowledge. Basic Huffman encoding [HUFF] was enhanced for the specific tasks of robust, efficient header compression.

Thanks to

Carsten Bormann (cabo@tzi.org)
Christian Schmidt (christian.schmidt@icn.siemens.de)
Max Riegel (maximilian.riegel@icn.siemens.de)

for valuable input and review.

## 7. References

[ROHC] "RObust Header Compression (ROHC)", Carsten Bormann et al, RFC3095, Internet Engineering Task Force, July 2001

[EPIC] "Framework for EPIC-LITE", Richard Price et al, <draft-ietf-rohc-epic-lite-00.txt>, Internet Engineering Task Force, October 23, 2001

[HUFF] "The Data Compression Book", Mark Nelson and Jean-Loup Gailly, M&T Books, 1995

[RFC-1144] "Compressing TCP/IP Headers for Low-Speed Serial Links", V. Jacobson, Internet Engineering Task Force, February 1990

[RFC-1951] "DEFLATE Compressed Data Format Specification version 1.3", P. Deutsch, Internet Engineering Task Force, May 1996

[RFC-2026] "The Internet Standards Process - Revision 3", Scott Bradner, Internet Engineering Task Force, October 1996

[RFC-2119] "Key words for use in RFCs to Indicate Requirement Levels", Scott Bradner, Internet Engineering Task

[RFC-2507] M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression", Internet Engineering Task Force, February 1999

[CONG1] "Congestion avoidance and control", V. Jacobson, In ACM SIGCOMM '88, 1988

[CONG2] "TCP Congestion Control", M. Allman, V. Paxson, and W. R. Stevens, RFC 2581, April 1999

[RFC-2481] "A Proposal to add Explicit Congestion Notification (ECN) to IP", K. Ramakrishnan, S. Floyd, Internet Engineering Task Force, January 1999

[ECN] "The Addition of Explicit Congestion Notification (ECN) to IP", K. K. RamaKrishnan, Sally Floyd, D. Black, Internet Draft (work in progress), June, 2001. <draft-ietf-tsvwg-ecn-04.txt>

[TCPREQ] "Requirements for ROHC IP/TCP header compression", L-E. Jonsson, Internet Draft (work in progress), June 20, 2001

[INITWIN] "Increasing TCP's Initial Window", M. Allman, S. Floyd, and C. Partridge, Internet Draft (work in progress), May 2001. <draft-ietf-tsvwg-initwin-00.txt>

[TAROC-4] H. Liao, Q. Zhang, W. Zhu, and Y.-Q. Zhang, ôTCP-Aware RObust Header Compression (TAROC)ö, Internet Draft (work in progress), Nov. 2001. <draft-ietf-rohc-taroc-04.txt>

## 8. Authors' addresses

Richard Price Tel: +44 1794 833681
Email: richard.price@roke.co.uk

Robert Hancock Tel: +44 1794 833601
Email: robert.hancock@roke.co.uk

Stephen McCann Tel: +44 1794 833341
Email: stephen.mccann@roke.co.uk

Mark A West Tel: +44 1794 833311
Email: mark.a.west@roke.co.uk

Abigail Surtees Tel: +44 1794 833131
Email: abigail.surtees@roke.co.uk

Paul Ollis Tel: +44 1794 833168

Email: paul.ollis@roke.co.uk

Roke Manor Research Ltd
Romsey, Hants, SO51 0ZN

United Kingdom

Qian Zhang Tel: +86 10 62617711-3135
Email: qianz@microsoft.com

HongBin Liao Tel: +86 10 62617711-3156
Email: i-hbliao@microsoft.com

Wenwu Zhu Tel: +86 10 62617711-5405
Email: wwzhu@microsoft.com

Ya-Qin Zhang Tel: +86 10 62617711
Email: yzhang@microsoft.com

Microsoft Research Asia
Beijing Sigma Center
No.49, Zhichun Road, Haidian District
Beijing 100080, P.R.C.

## Appendix A. Packet types provided by ROHC framework

In addition to the standard CO (compressed) packets, the [ROHC] framework contains two special packet types designed to help synchronize the context at the compressor and decompressor. An IR (Initialization and Refresh) packet associates a context with a certain ROHC profile, and transmits the value of all fields including those which remain constant throughout the lifetime of the context.

An IR-DYN (Dynamic Initialization and Refresh) packet associates a context with a ROHC profile, and additionally transmits the value of any fields except those that remain constant for the lifetime of the context. An IR-DYN packet cannot be used to completely initialize a new context, but it is usually smaller than a full IR packet.

[ROHC] also defines a general compressed packet that can be used to encapsulate CO, IR and IR-DYN packets. The general packet format includes a CID (Context Identifier) to indicate the context to which the compressed packet belongs. It also includes a packet type indicator to specify whether the packet is a feedback, initialization or general compressed packet, whether it is segmented, and whether it contains padding.

The following packet type indicators are reserved in the overall [ROHC] framework:

```
1110:     Padding or Add-CID octet
11110:    Feedback
11111000: IR-DYN packet
1111110:  IR packet
1111111:  Segment
```

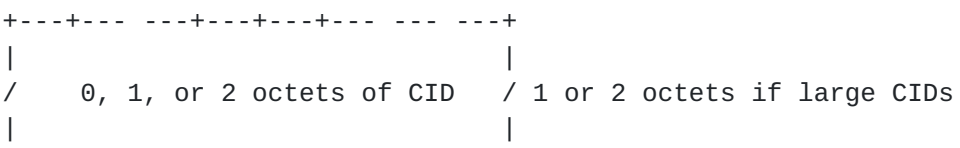
Any packet types not indicated by the bit pattern 111XXXXX can be used by individual [ROHC] profiles such as the TCP/IP profile.

### A.1. CO packet

The compressed (CO) packet type is the basic compressed packet offered by EPIC-LITE. CO packets can be used to transmit data between the compressor and decompressor with a high level of efficiency, and can cope with most irregularities in the packet stream.

The location of an EPIC-LITE CO packet within the general ROHC packet is shown below:

```
     0                           7
    --- --- --- --- --- --- --- ---
   |         Add-CID octet         | if for CID 1-15 and small CIDs
   +---+--- --- --- ---+--- --- ---+
   |      EPIC-LITE CO packet      | 1 octet
```


0, 1, or 2 octets of CID
1 or 2 octets if large CIDs

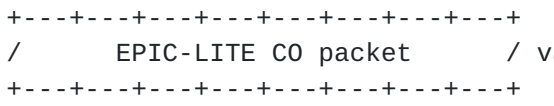



Figure 1 : Format of CO packet generated by EPIC-LITE

Note that CO packets are decompressed relative to the context stored at the decompressor. If the compressor has not yet initialized this context, or suspects that it has become invalidated, then a CO packet cannot be sent.

## A.2. IR-DYN packet

The structure of the IR-DYN packet used by EPIC-LITE is shown below:

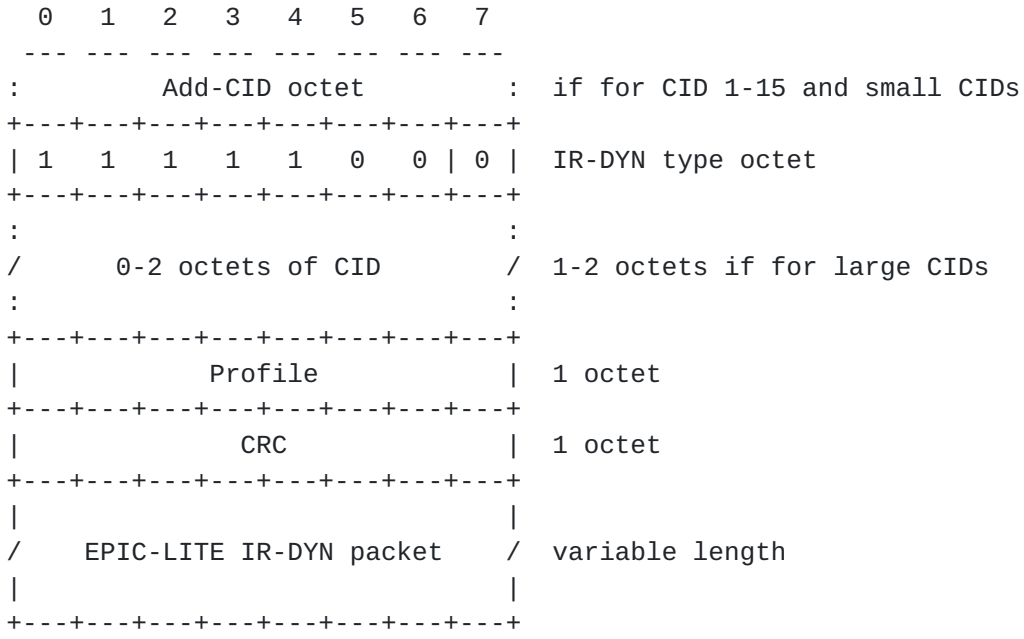
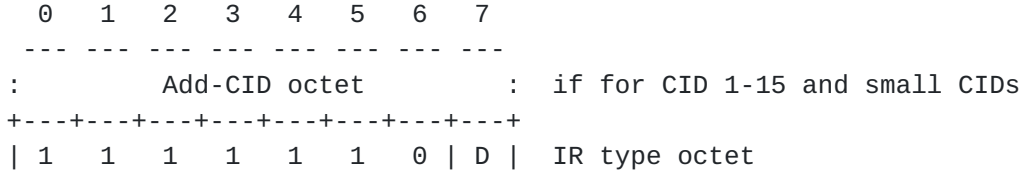


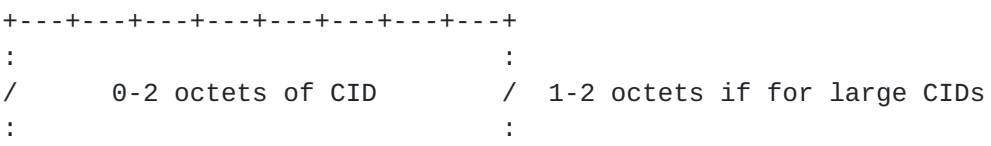

Figure 2 : Format of IR-DYN packet generated by EPIC-LITE

The Profile field associates the context with a certain profile. It transmits the 8 least significant bits of the EPIC-LITE profile_identifier parameter described in Section 7.1. Furthermore, the polynomial used to calculate the CRC is defined in Section 6.12.

## A.3. IR packet

The structure of the IR packet used by EPIC-LITE is shown below:




0-2 octets of CID
1-2 octets if for large CIDs

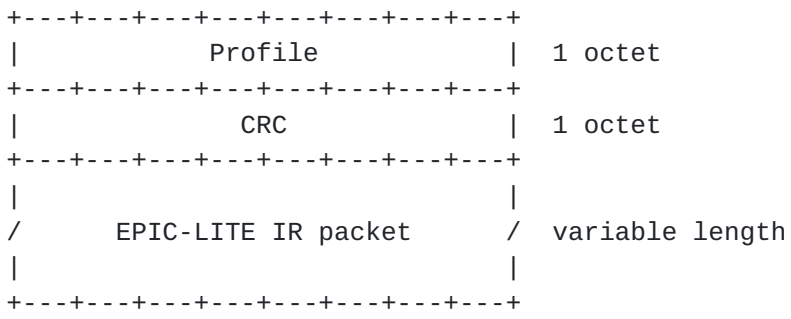



Figure 3 : Format of IR packet generated by EPIC-LITE

Note that the D bit is currently always set to 1 (as specified in [ROHC]), since the IR packet generated by EPIC-LITE always compresses every field in the header. A version of the IR packet that only compresses static fields may be introduced in future.