ROHC WG HongBin Liao, Microsoft Research Asia
Internet Draft Qian Zhang, Microsoft Research Asia
Expires: May 2002 Wenwu Zhu, Microsoft Research Asia
Ya-Qin Zhang, Microsoft Research Asia

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

November 21, 2001

TCP-Aware RObust Header Compression (TAROC)

draft-ietf-rohc-tcp-taroc-04.txt

Status of this Memo

1. Abstract

As a major transport protocol of current Internet, TCP has the problem of the large header overhead on bandwidth-limited links. Header compression has been proven to be efficient for using TCP over bandwidth-limited reliable links. Unfortunately, existing

TCP/IP header compression schemes do not work well on noisy links, especially the one with high bit error rate and long roundtrip time. In addition, existing schemes [2,3] have not addressed some TCP options such as SACK [4,5] and Timestamps [6].

A robust and efficient header compression scheme for TCP/IP, called TAROC, is presented in this document. TAROC is composed of a behavior-aware control mechanism, called TAROC-C, and a detailed header encoding scheme. In this draft, the Efficient Protocol Independent Compression (EPIC-LITE) scheme is used as the compressed header encoding framework. The window-based LSB encoding is introduced in our scheme for compressing redundant fields and reducing error propagation. The key point of TAROC-C is the TCP congestion window tracking approach, which can be used to improve the efficiency of the window-based encoding and the performance of the overall header compression scheme. With the dynamical congestion window tracking, our scheme can achieve good performance even when the feedback channel is not available.

Table of Contents

Document History

```
04  Nov. 21, 2001     Separate the control mechanism, TAROC-C, with
                      the detailed compressed packet formats
                      generation approach;
                      TAROC-C does not have an IPR-statement;
```

```
                         Introduce the simple TCP/IP profile;
                         Use EPIC-LITE as coding framework to simplify
                         the creation of new TCP/IP compressed header
                         format.
   03  Oct. 26, 2001     Modify our TCP congestion window estimation
                         scheme with the MAX and MIN boundary;
                         Clarify the initialization and state
                         transition process in compressor state
                         management;
                         Add the CRC option in our compressed header.
   02  July 20, 2001     Integrate TAROC with ROHC framework;
                         Add a second order (SO) state on compressor
                         side for fixed-payload packets compression;
                         Modify the coding method for type
                         identification and adjust corresponding packet
                         format to improve compression efficiency;
                         Update the simulation results.
   01  March 01, 2001    Improve congestion window tracking algorithm
                         to handle the special cases where congestion
                         indications are lost;
                         Improve the compression efficiency by adding
                         fixed-payload encoding;
                         Change in header format accordingly.
   00  November 17, 2000 First release.
```

2. Conventions used in this document

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 [7].

Other terminologies, such as Profile, Context, Compressed header format, Encoding method, Indicator flags, Set of compressed header formats, Library of encoding methods, Input language, Control field, are defined in [19].

3. Introduction

The necessity and importance of doing TCP/IP header compression on low- or medium-speed links have been discussed in [3]. For conciseness, the general background information on header compression has not been discussed in detail in this draft. Detailed information can be found in RFC2507 [3]. Existing header compression schemes, such as VJHC [2] and IPHC [3], rely on transmitting only the difference from the previous header in order to reduce the large overhead of TCP/IP header.

Although VJHC works well over reliable links, when used over unreliable link, such as wireless links, it induces many additional errors due to inconsistent contexts between the compressor and the decompressor. Considering the high bit error rate in wireless channel, if a packet gets lost, the compressed header of next packet cannot be correctly decompressed. Then the decompressor must send the request for resynchronization and in the meanwhile discard all compressed header. A fatal result of this effect is that it prevents TCP Fast Retransmit algorithm [8] from being fired and always causes TCP retransmission timeout. This effect is shown in detail in [9].

IPHC proposes two simple mechanisms, the TWICE algorithm and the full header request mechanism, to reduce the errors due to the inconsistent contexts between the compressor and the decompressor. The TWICE algorithm assumes that only the Sequence Number field of TCP segments are changing during the connection and the deltas among consecutive packets are constant in most cases. However, these assumptions are not always true, especially when TCP Timestamp and SACK options are used. The full header request mechanism needs a feedback channel, which is unavailable in some circumstances. Even when the feedback channel is available, this mechanism still cannot perform well enough if the roundtrip time of this link is very long. Once a packet is corrupted on the noisy link, there are still several consecutive packets dropped due to the inconsistency between

the compressor and the decompressor.

This Internet draft describes a new header compression scheme (TAROC, or TCP-Aware RObust header Compression), which consists of two components, TAROC-C (TCP-Aware RObust Header Compression Control mechanism) and EPIC-LITE (Efficient Protocol Independent Compression

scheme). By combining them together, our scheme is more robust against packet loss and hence achieves better performance over wireless links.

4. The concept and components of TCP-Aware RObust Header compression and Efficient Protocol Independent Compression (EPIC-LITE) scheme

This section first describes the concept of the TCP-aware robust header compression (TAROC) proposal 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 [10], 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 [10], 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 the value r mentioned in the Section 6.4 in EPIC-LITE [19], 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 [11] and specified in [12], 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 topology assumption of TAROC is the same as the one in VJHC.

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.

The Efficient Protocol Independent Compression (EPIC-LITE) scheme, which had been discussed in detail in [19], is used to generate new ROHC profiles. This scheme takes as its input a list of fields in the protocol stack to be compressed, and for each field a choice of one or more compression techniques. Using this input EPIC-LITE derives a set of compressed header formats that can be used to quickly and efficiently compress and decompress headers.

A TCP/IP profile is proposed to describe the behaviors of each field in TCP/IP header.

In the rest of this draft, the control mechanism, TAROC-C, and the detailed compressed packet header format will be discussed in detail respectively. More specifically, the TCP congestion window tracking algorithm, the state machines in the header compression framework, and the logics of the compressor/decompressor, are described in TAROC-C.

5. The framework of TAROC-C

5.1. TCP congestion window tracking

5.1.1. 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.

5.1.2. 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.

```
                         +-------------+
                         |             |
        ---------------->| CONGESTION- |
        |                |  AVOIDANCE  |
        |            ----|             |<---
+------------+       |   +-------------+   |
|            |       |                     |
| SLOW-START |       |                     |
|            |       |   +-------------+   |
+------------+       |   |             |   |
        |            |-->|    FAST-    |----
        |                |  RECOVERY   |
        ---------------->|             |
                         +-------------+
```

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.

5.1.3. Congestion window tracking based on Acknowledgment Number


```
                                   +-------------+
                                   |             |
                  ---------------->| CONGESTION- |
                  |                |  AVOIDANCE  |
                  |            ----|             |<---
          +------------+       |   +-------------+   |
          |            |       |                     |
          | SLOW-START |       |                     |
          |            |       |                     |
          +------------+       |   +-------------+   |
                  |            |   |             |   |
                  |            |-->|     FAST-   |----
                  |                |   RECOVERY  |
                  ---------------->|             |
                                   +-------------+
```

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.

5.1.4. 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 [18]; 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 [13] and [14], 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.

5.2. Compressor/decompressor state management with TAROC-C

5.2.1. 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.

```
                         +----------+
                         |          |
 +----------+            | FO State |             +----------+
 |          |  <-------->|          |  <-------->  |          |
 | IR State |            +----------+             | SO State |
 |          |  <---------------------------------> |          |
 +----------+                                     +----------+
```

5.2.1.1. 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 [3]. 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.

5.2.1.2. 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.

5.2.1.3. 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.

5.2.2. 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.

```
      +--------------+         +--------------+
      |  No Context  |  <--->  | Full Context |
      +--------------+         +--------------+
```

5.3. 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.

5.3.1. 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.

5.3.2. 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) 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.

5.3.3. 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.

5.4. Decompressor logic in TAROC-C

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

5.4.1. 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 [10], only IR or IR-DYN packets may be decompressed in No Context state.

5.4.2. 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 5.6.

5.5. 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.

5.5.1. 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 5.3 and 5.4.

5.5.2. 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.

5.5.2.1. Compressor states and logic (O-mode)

Following rules should be combined with the action defined in section 5.3.

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.

5.5.2.2. Decompressor states and logic (O-mode)

The decompression states and the state transition logic are the same as in the Unidirectional case (see section 5.5.1.). 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).

5.5.3. 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.

5.5.3.1. Compressor states and logic (R-mode)

Following rules should be reparation to the action defined in section 5.3.

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.

5.5.3.2. 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 5.6.

5.6. Implementation issues

5.6.1. 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 [15] 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.

5.6.2. 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.

5.6.3. 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.

6. Coding scheme and compressed packet header format

Following the requirement of TCP/IP header compression [15], TAROC should fit into the ROHC framework. Thus, TAROC will conform to the general format and the reserved packet types defined in [10]. A compressed header format had been discussed in [20] in our past work. As stated in [19], EPIC-LITE is a generic encoding scheme which can automatically generate efficient packet format for the compressed header. In this draft, TAROC adopts EPIC-LITE as the coding framework. To use the EPIC-LITE coding framework, a suitable TCP/IP profile is also needed as the input. In the following of this section, we will discuss that in detail.

6.1. Window-based LSB encoding and fixed-payload encoding

As stated above, the change patterns of several TCP fields (for example, Sequence Number, Acknowledgement Number, Window, etc.) are completely different from the ones of RTP, and are very hard to predict. Thus, Window-based LSB encoding, which does not assume the linear changing pattern of the target header fields, is used in TAROC to encode those TCP fields both efficiently and robustly.

The Window-based LSB encoding (W-LSB encoding) used in TAROC is a slightly modified version of [10]. The major modifications can be summarized as follows:

- For reference selection, the decompressor always choose the one which is the last received non-retransmission value or uncompressed value that had passed the TCP checksum successfully.

- After sending a value v (compressed or uncompressed), the compressor always adds v into the VSW since each TCP segment is protected by the TCP checksum.

The W-LSB encoding will be applied to several fields, such as IP-ID, Sequence Number, Acknowledgment Number, Window fields, TCP Timestamp option, etc.

For some applications, such as bulk data transferring, etc., the payload size of each packet is usually a constant value, e.g. 1460 bytes. In such a case, the sequence number and acknowledgment number can be represented as the following equation:

```
SEQ (or ACK) = m * MTU + n.
```

If all the packets in VSW have the same 'n', only 'm' need be transmitted to the decompressor. The decompressor can obtain the sequence number or acknowledgment number after correctly decoding 'm', and use them as the reference values. This encoding method is called fixed-payload encoding.

6.2. The framework of EPIC-LITE scheme

The detailed information about EPIC-LITE, include the structure of the EPIC-LITE compressed headers, the overview of the input language for EPIC-LITE, the packet types available to EPIC-LITE, the library of EPIC-LITE encoding methods, and how to create a new ROHC profile, are described in [19].

6.3. ROHC Profile for compression of TCP/IP

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

Note that the probabilities listed for each encoding method are initial estimates only. These need to be refined with more accurate values from genuine TCP/IP streams.

The profile for TCP/IP compression is given below:

only uses the following toolbox methods:
- STATIC-KNOWN
- STATIC-UNKNOWN
- STATIC
- IRREGULAR
- LSB
- VALUE
- MSN-IRREGULAR
- MSN-LSB
- C

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


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%)
```

7. Conclusions

Based on the requirements proposed in [16] and [17], a robust header compression scheme should be of transparency, ubiquity, and efficiency. It must be able to support both IPv4 and Ipv6 packet and tolerate error propagation. Different types of link delay and the

misordering of packets should be addressed. In addition, multiple links and unidirectional link should be supported in the proposed header compression scheme. Particularly for TCP/IP, the header compression scheme should compress TCP SACK and Timestamp options.

From the above analysis, it can be seen that all these requirements can be satisfied in our proposed TAROC.

Considering the behavior of TCP protocol itself, even the packets misordering occurs between the compressor and the decompressor, a good performance can still be achieved in TAROC.

Note that in our scheme, we need to select a packet with correct checksum of the whole packet as a reference. In this way, it does not require link layer to treat the header and payload of the packet separately.

Simulations results (Appendix A) demonstrate the effectiveness of control mechanism TAROC-C and corresponding header compression scheme, TAROC of TAROC.

8. Acknowledgments

When designing this protocol, earlier header compression ideas described in [2], [3] and [10] have been import sources of knowledge.

This draft also benefited from discussion on the ROHC mailing list about the TAROC-C mechanism.

9. Security considerations

Security issues are not considered in this memo.

10. Authors' addresses

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

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

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

11. References

1 S. Bradner, "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

2 V. Jacobson, "Compressing TCP/IP headers for low-speed serial links", RFC 1144, February 1990.

3 M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression", RFC 2507, February 1999.

4 M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

5 S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

6 V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

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

8 V. Jacobson, "Fast Retransmit", Message to the end2end-interest mailing list, April 1990.

9 M. Degermark, M. Engan, B. Nordgren, and Stephen Pink, " Low-loss TCP/IP header compression for wireless networks", In the Proceedings of MobiCom, 1996.

10 Bormann (ed.), et al., "RObust Header Compression (ROHC)", RFC 3095, July 2001.

11 V. Jacobson, "Congestion avoidance and control", In ACM SIGCOMM '88, 1988.

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

13 K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, January 1999.

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

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

16 M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", Internet Draft (work in progress), August 2000. <draft-ietf-tsvwg-limited-xmit-00.txt>

17 M. Degermark, "Requirements for robust IP/UDP/RTP header compression", RFC 3096, July 2001.

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

19 Richard Price et al, "Framework for EPIC-LITE", <draft-ietf-rohc-epic-lite-00.txt>, Internet Draft (work in progress), Oct. 2001.

20 H. Liao, Q. Zhang, W. Zhu, and Y.-Q. Zhang, TCP-Aware RObust Header Compression (TAROC)÷, Internet Draft (work in progress), Nov. 2001. <draft-ietf-rohc-taroc-03.txt>

12. Intellectual property considerations

The TCP-Aware Robust header Compression Control mechanism, TAROC-C, and the Efficient Protocol Independent Compression scheme, EPIC-LITE, do not have an IPR statement.

Appendix A - Simulation results

To study the performance of various TCP/IP header compression schemes, we have simulated VJHC, IPHC and TAROC schemes on NS-2 network simulator. The simulation result in gained by the TAROC coding scheme discussed in [20].

A.1. Simulation topology

```
+------------+            +--------+           +-------------+
|            |------------>|        |------------>|             |
| Fixed Host |  8Mb 100ms  | Router |Wireless link| Mobile Host |
|            |<------------|        |<------------|             |
+------------+            +--------+           +-------------+
```

In this scenario, a fixed host is connected to the router with a WAN link (8Mb, 100ms). The queue size on the router is 6. The

communication channel between the mobile host and the router simulates the wireless link, which has a wide range of bandwidth from 384kb to 9.6kb and a delay of 100ms. The bit error rate (BER) on this wireless link is from 1e-7 to 1e-3. TCP traffic is conveyed from the fixed host to the mobile host.

It is known that, in wireless link under a high bit-error-rate situation, a smaller MTU is better in terms of the increasing chance of successful transmission. So different MTUs are selected under different BER conditions in our simulation.

A.2. Tested header compression schemes

Five header compression schemes in our simulation:

NONE This scheme refers to the situation when no header compression is employed on the wireless link.

VJHC This scheme employs RFC1144 on the wireless link. It assumes that the compressed header size is 4.

IPHC This scheme employs RFC2507 on the wireless link, but without TWICE algorithm. The characteristics of the feedback channel are the same as the forward wireless link. It assumes that the compressed header size is 5.

TAROC It refers to the scheme proposed in this Internet Draft. The compressed header size is determined by the scheme described in this draft.

IDEAL This scheme simulates the situation where header compression does not introduce additional errors. It assumes that the compressed header size is 4, the same one as in the VJHC.

A.3. Simulations and results

Based upon these configurations, enormous simulations have been tested. The followings are the results of several TCP variants, Tahoe, Reno and Sack on the wireless link with wide range of bandwidth, BER and MTU.

Wireless link characteristics:

* Bandwidth: 384kb, 114kb, 64kb, 9.6kb

* Delay: 100ms

* BER: 1e-8, 3e-8, 1e-7, 3e-7, 1e-6, 3e-6, 1e-5, 3e-5, 1e-4, 3e-4

TCP Variants: Tahoe, Reno, Sack

Header compression schemes: NONE, VJHC, IPHC, TAROC, IDEAL

The following lists some of the results: 384kb for Tahoe, 114kb for Sack, 64kb for Reno, and 9.6kb for Sack.

A.3.1. 384kb

Tahoe

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
1e-8	576	Throughput (Byte/s)	25470	25457	25179	25587	25603
		Improvement (%s)	0	-0.05	-1.14	0.46	0.52
3e-8	576	Throughput (Byte/s)	25770	25764	25696	25819	25839
		Improvement (%s)	0	-0.02	-0.29	0.19	0.27

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
1e-7	576	Throughput (Byte/s)	24564	24185	23550	24687	24717

		Improvement (%s)	0	-1.54	-4.12	0.50	0.62
3e-7	576	Throughput (Byte/s)	22256	21240	20216	22365	22407
		Improvement (%s)	0	-4.56	-9.17	0.50	0.68
1e-6	576	Throughput (Byte/s)	16703	14638	13840	16930	17027
		Improvement (%s)	0	-12.36	-17.14	1.36	1.94
3e-6	576	Throughput (Byte/s)	9895	7987	8086	10255	10266
		Improvement (%s)	0	-19.04	-18.03	3.95	4.06
1e-5	296	Throughput (Byte/s)	3531	2803	2950	3825	3826
		Improvement (%s)	0	-20.62	-16.45	8.33	8.35
3e-5	296	Throughput (Byte/s)	1731	1181	1317	1900	1901
		Improvement (%s)	0	-31.77	-23.92	9.76	9.82
1e-4	168	Throughput (Byte/s)	504	342	366	635	636
		Improvement (%s)	0	-32.14	-27.38	25.99	26.19
3e-4	96	Throughput (Byte/s)	97	80	91	202	203
		Improvement (%s)	0	-17.53	-6.19	108.2	109.3

A.3.2. 114kb

Sack

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
1e-8	576	Throughput (Byte/s)	12105	12636	12605	12660	12662
		Improvement (%s)	0	4.39	4.13	4.58	4.60
3e-8	576	Throughput (Byte/s)	12083	12565	12474	12642	12643
		Improvement (%s)	0	3.99	3.24	4.63	4.63
1e-7	576	Throughput (Byte/s)	12030	12329	12165	12582	12587
		Improvement (%s)	0	2.49	1.12	4.59	4.63
3e-7	576	Throughput (Byte/s)	11856	11687	11326	12392	12411
		Improvement (%s)	0	-1.43	-4.47	4.52	4.68
1e-6	576	Throughput (Byte/s)	11213	9871	9177	11737	11740
		Improvement (%s)	0	-11.97	-18.16	4.63	4.70
3e-6	576	Throughput (Byte/s)	9258	6578	6206	9719	9784
		Improvement (%s)	0	-28.95	-32.97	4.98	5.68
1e-5	296	Throughput (Byte/s)	3883	2622	2587	4236	4239
		Improvement	0	-32.47	-33.38	9.09	9.17

		(%s)					

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
3e-5	296	Throughput (Byte/s)	1786	1111	1214	2000	2012
		Improvement (%s)	0	-37.79	-32.03	11.98	12.65
1e-4	168	Throughput (Byte/s)	489	325	361	640	652
		Improvement (%s)	0	-33.54	-26.18	30.88	33.33
3e-4	96	Throughput (Byte/s)	92	81	88	202	203
		Improvement (%s)	0	-11.96	-4.35	119.6	120.7

A.3.3. 64kb

Reno

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
1e-8	576	Throughput (Byte/s)	7317	7743	7698	7763	7764
		Improvement (%s)	0	5.82	5.21	6.10	6.11

3e-8	576	Throughput (Byte/s)	7312	7716	7672	7756	7757
		Improvement (%s)	0	5.53	4.92	6.07	6.09
1e-7	576	Throughput (Byte/s)	7288	7615	7556	7727	7728
		Improvement (%s)	0	4.49	3.68	6.02	6.04

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
3e-7	576	Throughput (Byte/s)	7213	7351	7222	7648	7649
		Improvement (%s)	0	1.91	0.12	6.03	6.04
1e-6	576	Throughput (Byte/s)	6966	6612	6286	7387	7398
		Improvement (%s)	0	-5.08	-9.76	6.04	6.20
3e-6	576	Throughput (Byte/s)	6206	5070	4746	6562	6580
		Improvement (%s)	0	-18.30	-23.53	5.74	6.03
1e-5	296	Throughput (Byte/s)	3377	2470	2312	3633	3667
		Improvement (%s)	0	-26.86	-31.54	7.58	8.59

3e-5	296	Throughput (Byte/s)	1576	1065	1122	1755	1773
		Improvement (%s)	0	-32.42	-28.81	11.36	12.50
1e-4	168	Throughput (Byte/s)	465	319	340	595	597
		Improvement (%s)	0	-31.40	-26.88	27.96	28.39
3e-4	96	Throughput (Byte/s)	87	79	86	190	194
		Improvement (%s)	0	-9.20	-1.15	118.4	123.0

A.3.4. 9.6kb

Sack

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
3e-8	576	Throughput (Byte/s)	1116	1187	1185	1190	1191
		Improvement (%s)	0	6.36	6.18	6.63	6.72
1e-8	576	Throughput (Byte/s)	1116	1188	1186	1191	1192
		Improvement	0	6.45	6.27	6.72	6.81

		(%s)					
1e-7	576	Throughput (Byte/s)	1116	1183	1181	1190	1191
		Improvement (%s)	0	6.00	5.82	6.63	6.72
3e-7	576	Throughput (Byte/s)	1114	1173	1172	1188	1190
		Improvement (%s)	0	5.30	5.21	6.64	6.82
1e-6	576	Throughput (Byte/s)	1110	1133	1144	1183	1184
		Improvement (%s)	0	2.07	3.06	6.58	6.67
3e-6	576	Throughput (Byte/s)	1089	1036	1070	1164	1167
		Improvement (%s)	0	-4.87	-1.74	6.89	7.16
1e-5	296	Throughput (Byte/s)	979	855	935	1122	1123
		Improvement (%s)	0	-12.67	-4.49	14.61	14.71

BER	MTU (Byte)	Performance	NONE	VJHC	IPHC	TAROC	IDEAL
3e-5	296	Throughput (Byte/s)	759	500	600	900	908

		Improvement (%s)	0	-34.12	-20.95	18.58	19.63
1e-4	168	Throughput (Byte/s)	341	224	252	455	465
		Improvement (%s)	0	-34.31	-26.10	33.43	36.36
3e-4	96	Throughput (Byte/s)	78	67	72	172	173
		Improvement (%s)	0	-14.10	-7.69	120.5	121.8