Implementation of Robust Header Compression for Network SImulator-3

Introduction

In streaming applications, the overhead of IP, UDP and RTP is 40 bytes for IPv4 or 60 bytes for IPv6. For VoIP, this corresponds to around 60% of the total amount of data sent. Such large overheads may be tolerable in local wired links where capacity is often not an issue, but are excessive for wide area networks and wireless systems where bandwidth is scarce.
RoHC compresses these 40 bytes or 60 bytes of overhead typically into only one or three bytes, by placing a compressor before the link that has limited capacity, and a decompressor after that link. The compressor converts the large overhead to only a few bytes, while the decompressor does the opposite. The current implementation of NS-3 does not support Robust Header Compression at the Pdcp Layer.

Installation of RoHC library

We are currently using a free and open source library for RoHC https://rohc-lib.org/ . Follow the instruction given in the documentation for the installation of the library.

Patch the RoHC library with NS3

Open "ABSOLUTE-PATH-TO-NS3/src/lte/wscript" and add the dependencies for the RoHC library.

```
bld.env.append_value("CXXFLAGS", "-I/usr/include")
bld.env.append_value("LINKFLAGS", ["-L/usr/lib", "-L/usr/lib/lrohc_decomp", "-L/usr/lib/lrohc_comp",
"-L/usr/lib/lrohc_common"])
bld.env.append_value("LIB", ["rohc", "rohc_comp", "rohc_decomp", "rohc_common"])
```

Using the RoHC Library Functionalities

- Include the RoHC header files.
  Add the following header files in "ABSOLUTE-PATH-TO-NS3/src/lte/model/lte-pdcp.h"
  ```
  #include <rohc/rohc_comp.h>
  #include <rohc/rohc_decomp.h>
  #include <rohc/rohc.h>
  ```

- Create a Compressor and Decompressor Object:
  Declare the objects in "ABSOLUTE-PATH-TO-NS3/src/lte/model/lte-pdcp.h" in "LtePdcp" Class.
  ```
  struct rohc_comp *compressor;
  struct rohc_decomp *decompressor;
  ```

- <u>Initialise the Constructor and Destructor with the RoHC functionalities</u>:
  In our case, we used the ***IP-Compression Profile Only***. The Library also provides other functionalities like TCP, UDP, RTP EPS Header compression etc.
  Add the below code in the LtePdcp Contructor in the “ABSOLUTE-PATH-TO-NS3/src/lte/model/lte-pdcp.cc” file.

```
static int gen_random_num(const struct rohc_comp *const comp, void *const user_context){
        return rand();
}
compressor = rohc_comp_new2(ROHC_SMALL_CID, ROHC_SMALL_CID_MAX,
gen_random_num, NULL);
decompressor = rohc_alloc_decompressor(compressor);
```

- <u>Compress the IP Packet received at the PDCP layer from the sending side</u>:
  The RoHC compressor function expects a Packet of type “struct rohc_buf” which is defined in the rohc header file. So create a rohc_buf packet corresponding to the Packet received at the PDCP layer and compress the packet using
  <u>The following changes are made in DoTransmitPdu()</u>

```
unsigned char ip_buff[BUFFER_SIZE];
struct rohc_buf ip_packet = rohc_buf_init_empty(ip_buff, BUFFER_SIZE);
unsigned char rohc_buffer[BUFFER_SIZE];
struct rohc_buf rohc_packet = rohc_buf_init_empty(rohc_buffer, BUFFER_SIZE);
struct iphdr * ip_header;
ip_header = (struct iphdr *) rohc_buf_data(ip_packet);
ip_header->version = 4;
ip_header->ihl = 5;
ip_packet.len += ip_header->ihl * 4;
ip_header->tos = p_ip.GetTos();
ip_header->tot_len = htons(p -> GetSize());
ip_header->id = p_ip.GetIdentification();
ip_header->frag_off = p_ip.GetFragmentOffset();
ip_header->ttl = (unsigned char)p_ip.GetTtl();
ip_header->protocol = p_ip.GetProtocol();
ip_header->saddr = htonl(p_ip.GetSource().Get());
ip_header->daddr = htonl(p_ip.GetDestination().Get());
ip_header -> check = p_ip.GetChecksum();
sum = ip_checksum(m_tempbuff2, 20);
rohc_buf_append(&ip_packet, m_tempbuff1, p1 -> GetSize()); //m_tempbuff1 contains the payload
rohc_compress4(compressor, ip_packet, &rohc_packet);
```

  <u>**Note**:</u> IP Checksum needs to be enabled. The RoHC library expects an IP packet with the calculated Checksum. By default, IP checksum is disabled. Also, the below function can be used to calculate the checksum externally.

```
uint16_t
ip_checksum(void* vdata,size_t length)
{
 char* data=(char*)vdata;
 uint32_t acc=0xffff;
 size_t i;
 for (i=0;i + 1 < length; i+=2)
   {
     uint16_t word;
     memcpy(&word,data+i,2);
     acc+=ntohs(word);
     if (acc>0xffff)
     {
       acc-=0xffff;
     }
   }

 if (length&1)
   {
     uint16_t word=0;
     memcpy(&word,data+length-1,1);
     acc+=ntohs(word);
     if (acc>0xffff)
       {
         acc-=0xffff;
       }
   }

 return htons(~acc);
}
```

- Send the Compressed packet to RLC Layer.

  p = new Packet(buff, rohc_packet.len);
  Here, buff is the byte stream for the compressed RoHC Packet.

- Decompress the RoHC Packet at the PDCP layer from the receiving side.
  The de-compressor function converts the received packet into byte array.
  Decompress function is called to convert into the original IP packet.
  The following changes are made in DoReceivePdu()
  `

  ```
  status = rohc_decompress3(decomp, rohc_packet, &ip_packet,
               rcvd_feedback, feedback_send);
  ```

  The IP packet we receive from the library is converted into NS-3 packet by assigning Individual fields.

```
p_ip.SetProtocol(ip_header -> protocol);
p_ip.SetTtl(ip_header -> ttl);
p_ip.SetTos(ip_header -> tos);
p_ip.SetPayloadSize(20);
p_ip.SetFragmentOffset((uint16_t) ip_header -> frag_off);
p_ip.SetDestination(Ipv4Address(ntohl(ip_header -> daddr)));
p_ip.SetSource(Ipv4Address(ntohl(ip_header -> saddr)));
p_ip.SetIdentification((uint16_t) ip_header -> id);
```

- Send the de-compressed IP Packet to the above IP Layer.

```
Ptr <Packet> p2 = new Packet(temp_ip_buff, x);
p = p2 -> Copy();
p -> AddHeader(p_ip);
uint8_t m_tempbuff1[BUFFER_SIZE];
p -> CopyData(m_tempbuff1, p -> GetSize());
```

Implementation of In-Sequence Delivery at PDCP layer for Network SImulator-3 (NS-3)

NS-3 does not support In-sequence delivery at the PDCP layer. The implementation contain timer based re-ordering queues. The threshold time for re-ordering can be decided by upper layers.

❏To implement we used a variable m_lastRxSequenceNumber (last received sequence number) , a timer and a pdcp_queue (i.e. a map in which key is the sequence number and value is the PTR <packet>) in the header file.

❏Whenever a new packet comes, peek it’s header and compare the sequence number of the packet to the next sequence number. If they are equal , we transmit it otherwise store it in our queue.

❑ do_timer() function will determine that how long the packet stays in the queue before it is assumed to be lost. If the timer (threshold time) for a packet expires, it is assumed that the packet is lost and the next received in-sequence packet is transmitted.

Receive Function :

```
if (next_seq > m_maxPdcpSn)
                    next_seq = 0;
          while(pdcp_queue.find(next_seq)!=pdcp_queue.end()){
                    DoReceivePduActual(pdcp_queue[next_seq]);
                    m_timer = m_qtimer
                    pdcp_queue.erase(next_seq);
                    m_lastRxSequenceNumber=next_seq;
                    next_seq++;
                    if (next_seq > m_maxPdcpSn)
                              next_seq = 0;
          }
```

Timer Function:

```
if (pdcp->pdcp_queue.size() != 0){
                                        uint16_t next_seq = pdcp->m_lastRxSequenceNumber + 1;
                                        if (next_seq > pdcp->m_maxPdcpSn)
                                                  next_seq = 0;
                                        while(pdcp->pdcp_queue.find(next_seq) == pdcp->pdcp_queue.end()){
                                                  pdcp->m_lastRxSequenceNumber = next_seq;
                                                  next_seq++;
                                                  if (next_seq > pdcp->m_maxPdcpSn)
                                                            next_seq = 0;
                                        }
                                        while(pdcp->pdcp_queue.find(next_seq) != pdcp->pdcp_queue.end()){
                                                  pdcp->DoReceivePduActual(pdcp->pdcp_queue[next_seq]);
                                                  pdcp->m_timer = m_qtimer;
                                                  pdcp->pdcp_queue.erase(next_seq);
                                                  pdcp->m_lastRxSequenceNumber=next_seq;
                                                  next_seq++;
                                                  if (next_seq > pdcp->m_maxPdcpSn)
                                                            next_seq = 0;
                                        }
                              }
                              else{
                                        pdcp->m_timer= m_qtimer;
                              }
```