**네트워크 프로토콜 설계 및 실습 과제 8**

전기정보공학부 2016-10769 이권형

**1. Objective of the experiment**

본 실험에서는 직접 application을 정의한다. 또한, header field를 제작하여 sequence number와 time 등의 value를 교환하고, trace value를 추가하여 header에 있는 sequence number를 통해 error를 측정한다.

**2. Implementation process**

1) Make new helper for MyApp

PacketSinkHelper를 참고하여 다음과 같이 구현하였다.

(1) constructor

constructor의 경우 MyApp에 존재하는 private variable중 초기화가 필요한 m\_factory에 대 attribute를 setting한다.   
(2) Install method

Install method는 application을 node에 설치하도록 도와준다. 3개의 override된 function을 갖고 있으며, 각각은 argument로 node, node name, node container를 갖는다. 최종적으로는 InstallPriv함수를 call하는데, 이 함수에서는 m\_factory에서 application을 create하여, node에 application을 추가한 뒤에 app을 return한다.

Install함수에서 node를 argument로 받으면 InstallPriv함수에 이를 전하여 return하고, name을 받으면 Names::Find<Node> 함수를 이용하여 node를 name으로부터 얻은 뒤 해당 node에 대해 InstallPriv를 return한다. container를 받으면 container를 iterate하면서 각 node에 대해 InstallPriv를 통해 app을 설치한 뒤 application container를 return한다.

2) Add sequence field into Week4Header

In class work에서 time field를 추가한 것과 같은 과정을 거친다. GetSerializedSize()함수에서 10을 return하도록 하여 2byte의 sequence filed를 추가한다. 이후 SetSeq(), GetSeq()함수를 추가하여 m\_seq변수를 Week4Header class 외부에서 set하고 get할 수 있도록 하였다.

Serialize()함수는 iterator buffer에 대해 m\_time과 같이 m\_seq 변수를 16bit으로 header field에 설정한다. 이는 WriteHtonU16함수로 구현하였다. sequence number가 time보다 앞에 위치하도록 하였다. Deserialize()함수는 iterator buffer에 대해 header에서 16, 64 bit을 읽어 각각 m\_seq, m\_time변수에 저장한다. 이후 start로부터 10bytes 떨어진 위치를 return하여 payload를 읽을 수 있도록 한다.

이렇듯 sender에서는 m\_seq을 SetSeq()를 통해 set하여 packet을 보낼 수 있고, receiver에서는 m\_seq를 GetSeq()를 통해 get하여 header에 있는 sequence number를 확인할 수 있다.

myapp.cc에서, 다음과 같이 header를 주고받도록 구현하였다.

StartApplication에서, m\_mode가 true이면 sender이고, false이면 receiver이다. 각각에 대해 seq\_num을 -1로 초기화하고, socket을 만든 뒤에 SendPacket()을 통해 packet을보낸다. receiver의 경우 RecvCallback함수를 통해 callback함수를 HandleRead로 설정한다. SendPacket함수에서는 Week4Header class의 instance인 header에 대해 sequence number를 1 더하여 set하고(초기화가 -1이고, 이후 1씩 더하므로), 이후 time value도 set하여 AddHeader함수를 통해 header를 설치한다. 보낸 packet의 개수가 nPackets보다 작으면 ScheduleTx를 호출한다. ScheduleTx는 일정 시간 이후에 m\_sendEvent를 set하여 다시 SendPacket이 호출되도록 한다.

HandleRead에서는 socket에서 packet을 받았고 이 size가 0 이상이면 다음과 같은 code를 실행한다. header에서 RemoveHeader을 호출하여 Deserialize함수를 통해 m\_time과 m\_seq값을 header의 값으로 설정하고, 이후 m\_seq값을 받아 seq\_num에 저장한다. m\_errTrace에서 packet과 header를 넘기면서 sequence number에 대한 trace를 하도록 한다.

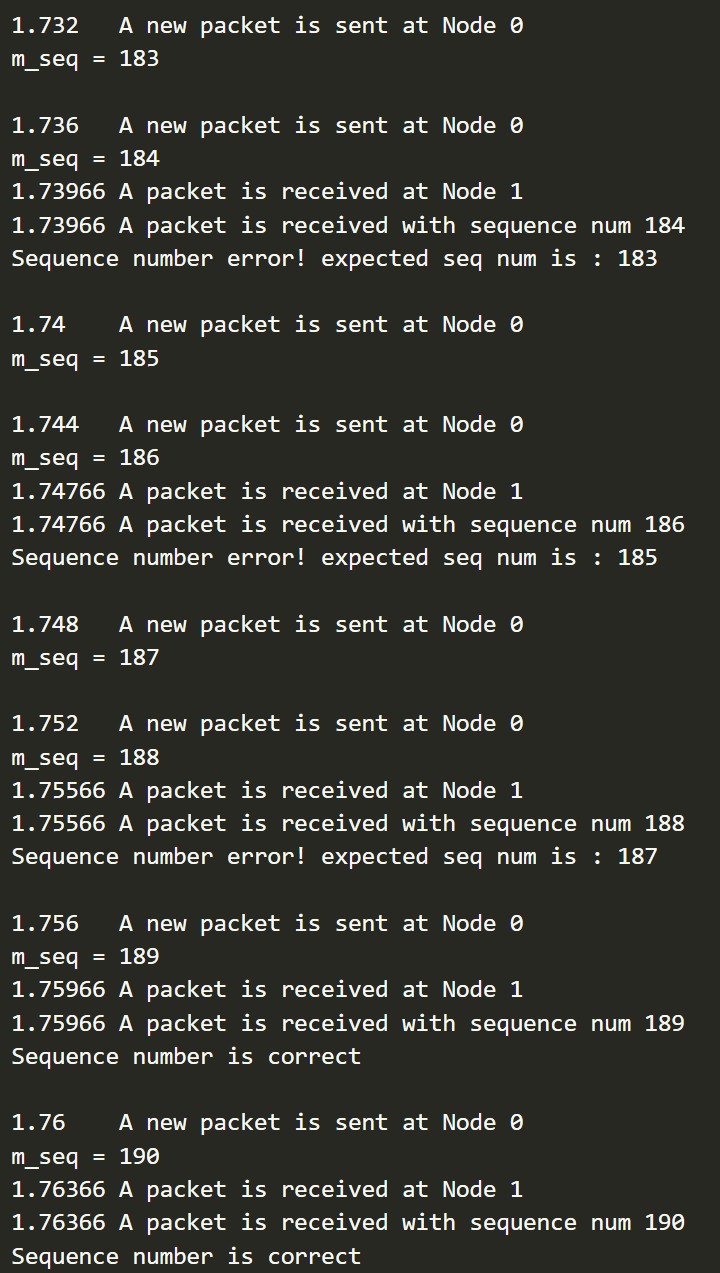
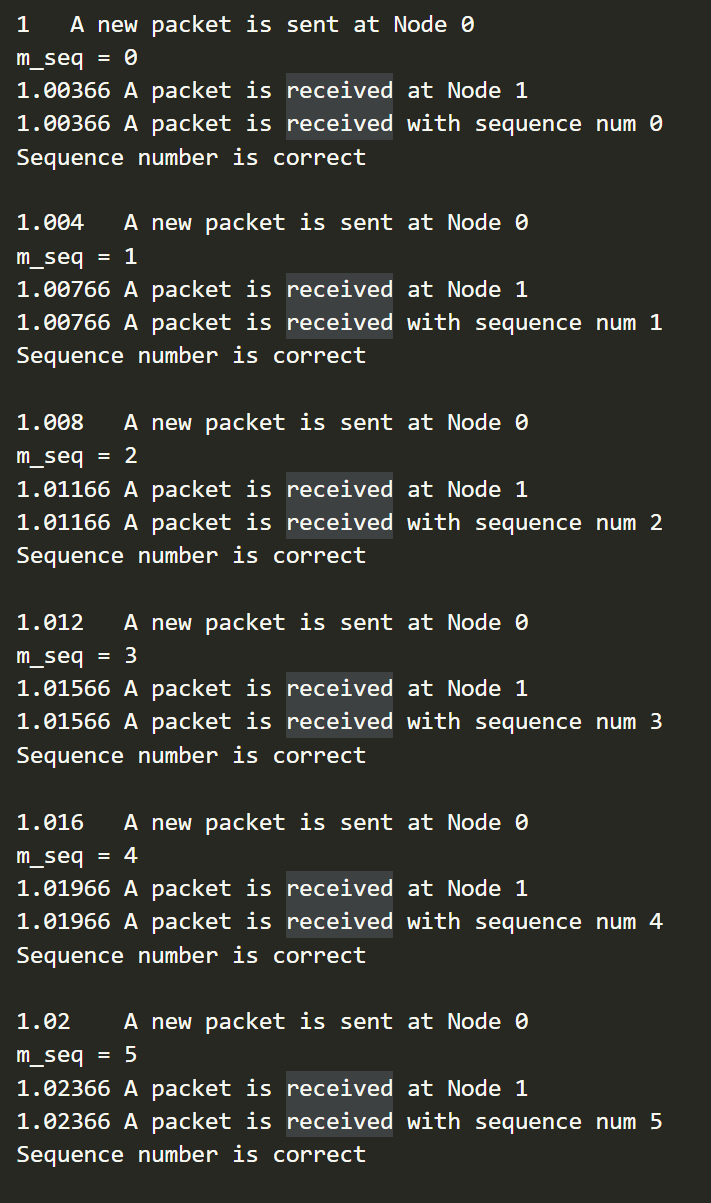
3) Check Packet sequence

myapp.cc에서, 37번 줄에 AddTraceSource를 이용하여 Err이라는 trace source를 추가한다. tracesource accessor를 m\_errTrace 변수로 설정한다. HandleRead에서 m\_errTrace를 통해 trace source에 access할 수 있다.

myapp-ex1.cpp에서, PacketErr()라는 callback function을 새로 정의하였다. 이 함수는 Packet과 Week4Header를 argument로 받는다. latest\_seq라는 변수를 통해 가장 최근에 받은 packet의 sequence number를 저장하고, header에 기록된 sequence number가 latest\_seq에 1을 더한 값과 같지 않으면 Error message를 출력한다. 즉, sequence number가 1이 증가하지 않고 2 이상 증가한 것으로, packet loss event가 발생한 것을 알린다. 이후 latest\_seq를 header에서의 sequence number로 update한다. Tx, Rx와 같이 TraceConnecWithoutContext함수를 이용해 receiverApp에서 callback함수를 연결해준다.

ErrorRate는 0.0002로 설정하였다. RateErrorModel을 이용하여 receiver node의 device에서 error rate attribute을 설정하였다. Packet의 개수를 200개로 늘려서 error가 충분히 일어날 수 있도록 설정하였다.

**3. Result & Discussion**



**그림1.** Simulation 결과.

그림1은 위 code를 통해 simulation을 실행한 결과이다. 가장 처음에, Node 0에서 sequence number를 0으로 하여 보내고 이를 receiver node인 Node 1에서 받아 확인했음을 알 수 있다. 오른쪽 그림에서 sequence number가 183, 185, 187인 packet에 대해 receiver에서 message를 표시하지 않는 것을 볼 수 있다. 즉, 이 때는 packet loss가 발생하여 receiver application이 이를 처리하지 못한 것이다. 따라서 이후 따라오는 184, 186, 188인 packet에 대해 expected Sequence number와 error message를 출력하는 것을 확인할 수 있다.

packet loss는 총 30번 발생하였다. 또한 error는 연속하여 발생하는 경우도 있었다. 즉, 직접 제작한 application과 trace source, header를 통해 sequence number를 구현할 수 있었고 성공적으로 packet loss event를 추적할 수 있었다.