Multimedia & Wireless Networking Laboratory

Session 8. 헤더 및 어플리케이션 생성

이강현
Multimedia & Wireless Networking Laboratory, SNU
khlee@mwnl.snu.ac.kr

Contents

Multimedia & Wireless
Networking Laboratory

Application in ns-3

Application

- Class ns3::Application can be used as a base class for ns3 applications
  - Applications are associated with individual nodes
  - Each node holds a list of references (smart pointers) to its applications
  - The main purpose of the base class application public API is to provide a uniform way to start and stop applications

- Conceptually, an application has zero or more ns3::Socket objects associated with it, which are created using the Socket API of the Kernel capability

Application Helper

- Application helper can make it easy to install application to node

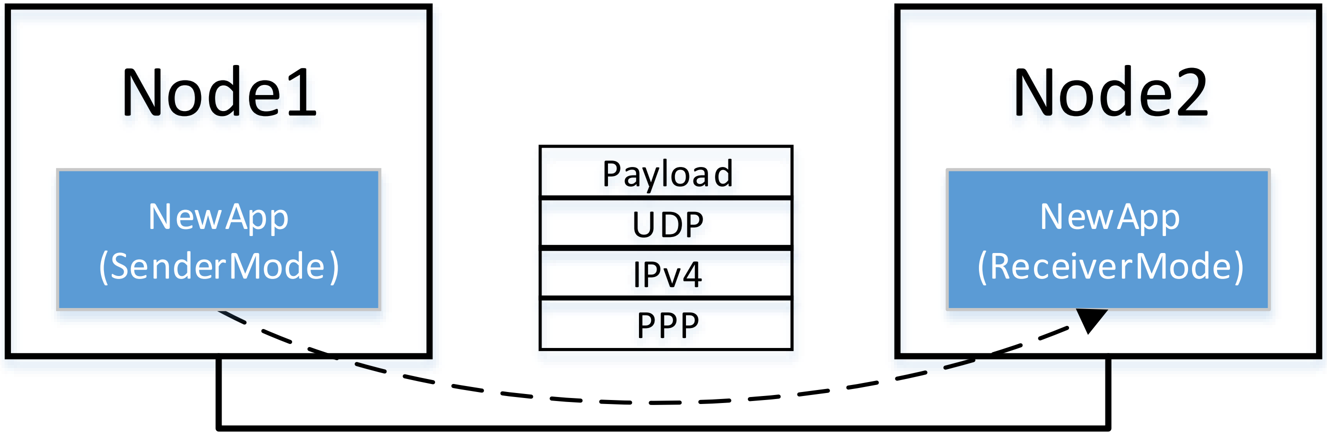
- Using application helper, simulation script can become more simpler
  - Only “Install” method

Multimedia & Wireless
Networking Laboratory

Example1:
New Application into ns-3

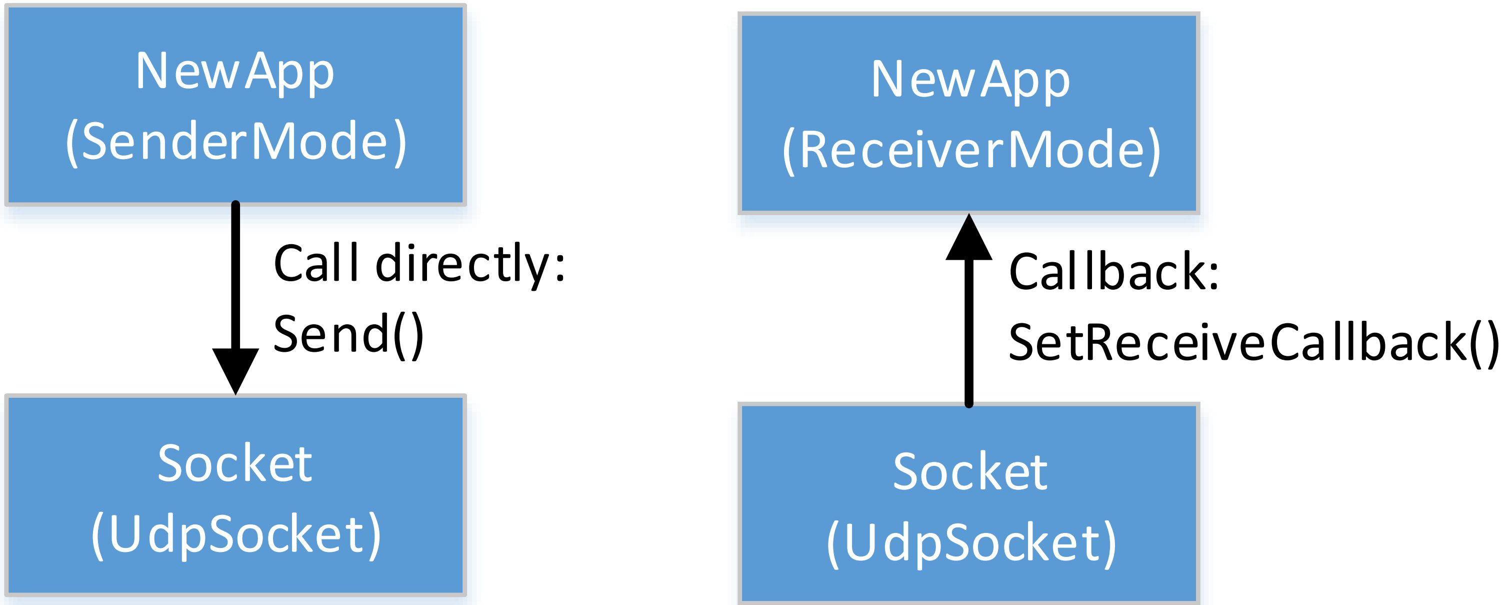
Overview

- Creating new application called “NewApp”
  - UDP traffic generator
  - Sender mode / Receiver mode
  - # of packets, data rate can be prescribed

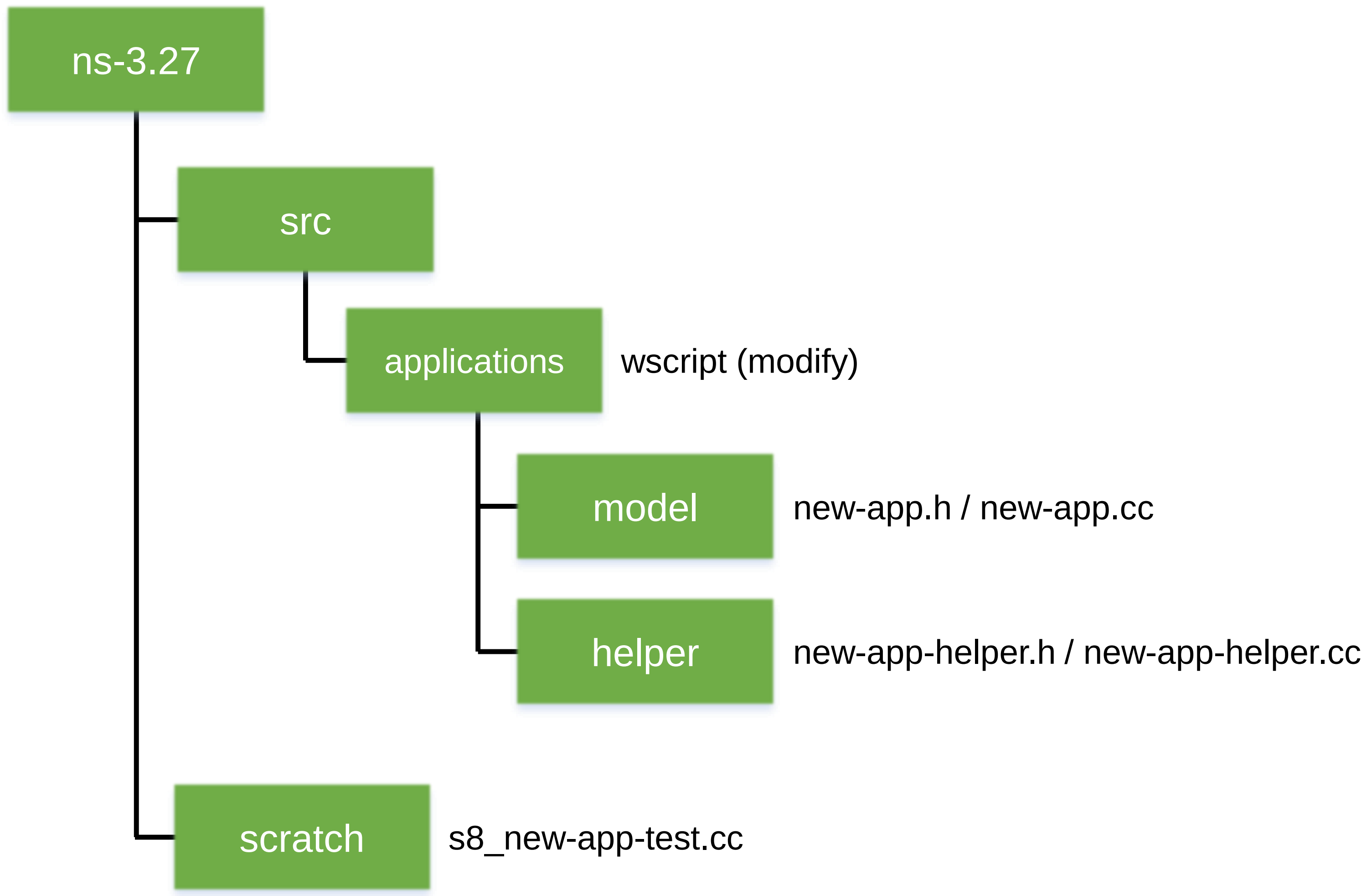


Interface

- Using UDP socket



Source codes



wscript

- All ns-3 modules depend on the core module and usually on other modules.
- This dependency is specified in wscript files.
- To include new source files, the wscript file in the corresponding directory should be modified.

Module dependency

- Modify “wscript” file in src/applications folder

```
def build(bld):
    module = bld.create_ns3_module('applications', ['internet', 'config-store', 'tools'])
    module.source = [
        …
        'model/new-app.cc',
        'helper/new-app-helper.cc',
       ]
    …
    headers.source = [
        …
        'model/new-app.h',
        'helper/new-app-helper.h',
        ]
```

NewApp

- Class definition (new-app.h)

```
class NewApp : public Application
{
  public:
    static TypeId GetTypeId (void);
    NewApp ();
    virtual ~NewApp ();

  private:
    virtual void StartApplication (void);
    virtual void StopApplication (void);

    void ScheduleTx (void);
    void SendPacket (void);
    void HandleRead (Ptr<Socket> socket);
```

Functions to handle packet transmission and reception

NewApp

- Class definition (new-app.h)

```
    bool m_mode; // Tx: true, Rx: false
    Address m_address;
    uint32_t m_nPackets;
    DataRate m_dataRate;

    Ptr<Socket> m_socket;
    uint32_t m_packetSize;
    uint32_t m_packetsSent;
    EventId m_sendEvent;
    bool m_running;

    TracedCallback<Ptr<const Packet> > m_txTrace;
    TracedCallback<Ptr<const Packet> > m_rxTrace;
};
```

NewApp

- Constructor (new-app.cc)

```
NewApp::NewApp ()
  : m_socket (0),
    m_packetSize (1000),   ← Packet size = 1000
    m_packetsSent (0),
    m_running (false)
{
  NS_LOG_FUNCTION (this);
}
```

TypeId

- ns-3 classes can include a metadata class called “TypeId” that records meta-information about the class, i.e., a unique identifier for an interface
  - A unique string identifying the class
  - Base class of the subclass
  - Set of accessible constructors in the class
  - List of publicly accessible properties (“attributes”) in the class

GetTypeId

- TypeId Application::GetTypeId (void)
  - Get the TypeId
  - TypeId ns3::TypeId::SetParent (TypeId tid)
    - Record in this TypeId which TypeId is the TypeId of the base class of the subclass.
  - TypeId ns3::TypeId::AddConstructor (void)
    - Record in this TypeId the fact that the default constructor is accessible
  - TypeId ns3::TypeId::AddAttribute (…)
    - Record in this TypeId the fact that a new attribute exists
  - TypeId ns3::TypeId::AddTraceSource (…)
    - Record in this TypeId the fact that a new trace source exists

GetTypeId

- TypeId NewApp::GetTypeId (void) (new-app.cc)

```
TypeId NewApp::GetTypeId (void) {
  static TypeId tid = TypeId ("ns3::NewApp")
    .SetParent<Application> ()
    .AddConstructor<NewApp> ()
    .AddAttribute ("Mode", "The mode : Sender(true), Receiver(false)",
                BooleanValue (false),
                MakeBooleanAccessor (&NewApp::m_mode),
                MakeBooleanChecker ())
    .AddAttribute ("Address", "The address",
                AddressValue (),
                MakeAddressAccessor (&NewApp::m_address),
                MakeAddressChecker ())
```

Name

Help text

Intial value

Associated value

GetTypeId

- TypeId NewApp::GetTypeId (void) (new-app.cc)

```
    .AddAttribute ("NPackets", "The total number of packets to send",
                  UintegerValue (10000),
                  MakeUintegerAccessor (&NewApp::m_nPackets),
                  MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("DataRate", "The data rate",
                  DataRateValue (DataRate ("500kb/s")),
                  MakeDataRateAccessor (&NewApp::m_dataRate),
                  MakeDataRateChecker ())
    .AddTraceSource("Tx", "A new packet is created and is sent",
 MakeTraceSourceAccessor(&NewApp::m_txTrace), "ns3::Packet::TracedCallback")
    .AddTraceSource("Rx", "A packet has been received", MakeTraceSourceAccessor
 (&NewApp::m_rxTrace), "ns3::Packet::TracedCallback")
 ;
  return tid;
}
```

Trace source

Name

Help text

StartApplication

- void NewApp::StartApplication (void) (new-app.cc)

```
void NewApp::StartApplication (void)
{
  NS_LOG_FUNCTION (this);

  if(m_mode == true)          ← Sender Mode
  {
    if(!m_socket){
      TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
      m_socket = Socket::CreateSocket(GetNode(), tid);   ┐
      m_socket->Bind();                                    │ Create UDP socket
      m_socket->Connect(m_address);                        ┘ & Connect to receiver
    }
    m_running = true;
    SendPacket();             ← Packet transmission
  }
```

StartApplication

- void NewApp::StartApplication (void) (new-app.cc)

```
else
{
  if(!m_socket){
    TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
    m_socket = Socket::CreateSocket(GetNode(), tid);
    m_socket->Bind(m_address);
    m_socket->Listen();
    m_socket->ShutdownSend();
    m_socket->SetRecvCallback (MakeCallback (&NewApp::HandleRead, this));
  }
 }
}
```

Receiver Mode

Create UDP socket
& Prepare to receive

Register callback function for packet reception

Packet Transmission

- void NewApp::SendPacket () (new-app.cc)

```
void NewApp::SendPacket (void)
{
  NS_LOG_FUNCTION (this);
  Ptr<Packet> packet = Create<Packet> (m_packetSize);        <- Create a packet
  m_txTrace(packet);          <- Tracing

  m_socket->Send(packet);          <- Send a packet

  if(++m_packetsSent < m_nPackets)
  {
    ScheduleTx();        <- Schedule next packet transmission
  }
}
```

Multimedia & Wireless
Networking Laboratory

Seoul National
University

Packet Transmission

- void NewApp::ScheduleTx () (new-app.cc)

```
void NewApp::ScheduleTx (void)
{
  if(m_running)
  {
    Time tNext (
      Seconds (m_packetSize*8/static_cast<double>(m_dataRate.GetBitRate())));
    m_sendEvent = Simulator::Schedule(tNext, &NewApp::SendPacket, this);
  }
}
```

Calculate next packet transmission time & Schedule transmission event

Packet Reception

- void NewApp::HandleRead (Ptr<Socket> socket) (new-app.cc)

```
void NewApp::HandleRead (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  Address from;
  while ((packet = m_socket->RecvFrom(from)))
  {
    if(packet->GetSize() > 0)
    {
      m_rxTrace(packet);
    }
  }
}
```

Receive a packet

Tracing

StopApplication

- void NewApp::StopApplication(void) (new-app.cc)

```
void NewApp::StopApplication ()
{
  NS_LOG_FUNCTION (this);
  m_running = false;
  if(m_sendEvent.IsRunning())
  {
    Simulator::Cancel (m_sendEvent);
  }
  if(m_socket)
  {
    m_socket->Close();
  }
}
```

Cancel next packet transmission

Close a socket

NewAppHelper

- Class definition (new-app-helper.h)

```
class NewAppHelper {
    public:
      NewAppHelper (bool mode, Address address);
      void SetAttribute (std::string name, const AttributeValue &value);
      ApplicationContainer Install (Ptr<Node> node) const;
      ApplicationContainer Install (std::string nodeName) const;
      ApplicationContainer Install (NodeContainer c) const;

    private:
      Ptr<Application> InstallPriv (Ptr<Node> node) const;
      ObjectFactory m_factory;
  };
```

NewAppHelper

- Helper functions (new-app-helper.cc)

```
NewAppHelper::NewAppHelper (bool mode, Address address) {
  m_factory.SetTypeId ("ns3::NewApp");
  m_factory.Set ("Mode", BooleanValue (mode));
  m_factory.Set ("Address", AddressValue (address));
}
```

Set “Mode” and “Address” attribute for NewApp

```
void NewAppHelper::SetAttribute (std::string name, const AttributeValue &value) {
  m_factory.Set (name, value);
}
```

Set attribute for NewApp

```
ApplicationContainer NewAppHelper::Install (Ptr<Node> node) const {
  return ApplicationContainer (InstallPriv (node));
}
```

Install NewApp (omit details)

Simulation Script

- Sender part

```
int main (int argc, char *argv[]) {
  …
  uint16_t port = 8080;
  Address destination (InetSocketAddress (interfaces.GetAddress (1), port));

  NewAppHelper sender (true, destination);
  sender.SetAttribute("NPackets", UintegerValue(10));
  sender.SetAttribute("DataRate", DataRateValue(DataRate("2Mb/s")));
  ApplicationContainer senderApp = sender.Install(nodes.Get(0));

  senderApp.Start (Seconds(1.0));
  senderApp.Stop (Seconds(5.0));
  …
}
```

Setup destination address

Make a NewAppHelper, Set Attribute for NewApp, Install NewApp to a node

Setup time to start and stop

Simulation Script

- Receiver part

```
int main (int argc, char *argv[]) {
  …
  Address any (InetSocketAddress (Ipv4Address::GetAny(), port));

  NewAppHelper receiver (false, any);
  ApplicationContainer receiverApp = receiver.Install(nodes.Get(1));

  receiverApp.Start (Seconds(0.5));
  receiverApp.Stop (Seconds(7.0));
  …
}
```

Setup address to receive

Make a NewAppHelper,
Install NewApp to a node

Setup time to start and stop

Simulation Script

- Connect trace sources to callback function

```
int main (int argc, char *argv[]) {
  …
  senderApp.Get(0)->TraceConnectWithoutContext("Tx", MakeCallback (&PacketTx));

  receiverApp.Get(0)->TraceConnectWithoutContext("Rx", MakeCallback (&PacketRx));
  …
}
```

Simulation Script

- Callback functions

```
static void PacketTx (Ptr<const Packet> p)
{
  NS_LOG_UNCOND (Simulator::Now().GetSeconds() << "\t“
                                << "A new packet is sent at Node 0");
}
```

Print tx time

```
static void PacketRx (Ptr<const Packet> p)
{
  NS_LOG_UNCOND (Simulator::Now().GetSeconds() << "\t"
                                << "A packet is received at Node 1");
}
```

Print rx time

Multimedia & Wireless Networking Laboratory

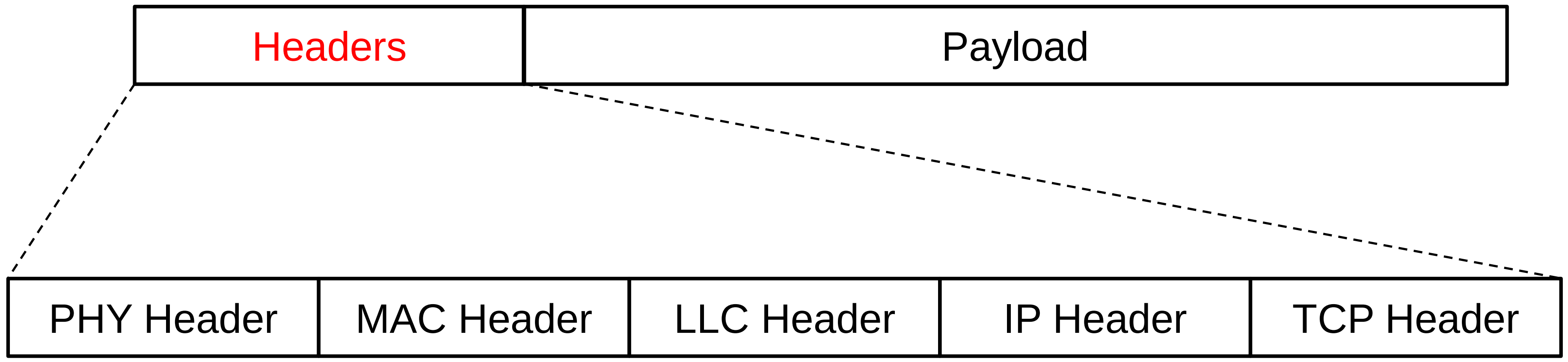
Header in ns-3

Packet Structure

- A packet has two types of fields
  - Header fields & Payload field
- Header
  - Supplemental data placed at the beginning of a payload
- Payload
  - A block of data being transmitted

Packet Structure

- Example



Header Class

- Using Header class
  - Base class for every protocol headers

  - Define pure virtual methods
    - Serialization, Deserialization, Print, etc.
    - Each protocol header class implement this methods

  - Used in tandem with the Packet class
    - Attach header: AddHeader method
    - Detach header: RemoveHeader method

Function of Header Class

- TypeId Header::GetTypeId(void)
  - Get the type ID
- virtual void Header::Serialize(Buffer::Iterator start) const
  - Used by Packet::AddHeader to store a header into the byte buffer of a packet
  - start: an iterator which points to where the header should be written
  - Convert value into byte stream
    - Using Buffer Iterator functions
      - WriteU8, WriteHtonU32, WriteHtonU64, etc.

Function of Header Class

- virtual uint32_t Header::GetSerializedSize(void) const
  - Used by Packet::AddHeader to store a header into the byte buffer of a packet
  - Return the expected size of the header by Serialize

Function of Header Class

- virtual uint32_t Header::Deserialize(Buffer::Iterator start)
    - Used by Packet::RemoveHeader to re-create a header from the byte buffer of a packet
    - start: an iterator which points to where the header should be read from
    - Return the number of bytes read
    - Convert byte stream into values
        - Using Buffer Iterator functions
            - ReadU8, ReadNtohU32, ReadNtohU64, etc.

Function of Header Class

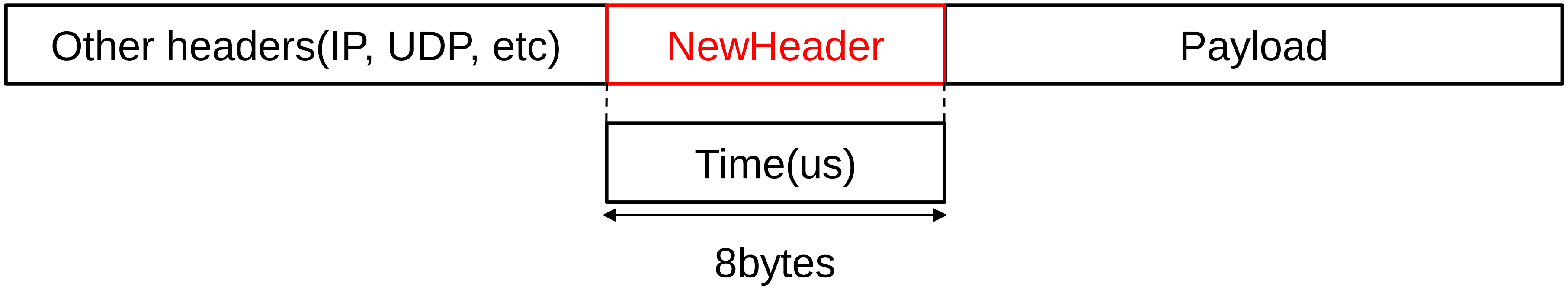
- virtual void Header::Print(std::ostream& os) const
  - Used by Packet::Print to print the content of a header as ascii data to a c++ output stream
  - os: output stream to print

Multimedia & Wireless Networking Laboratory

Example2: New Header into ns-3

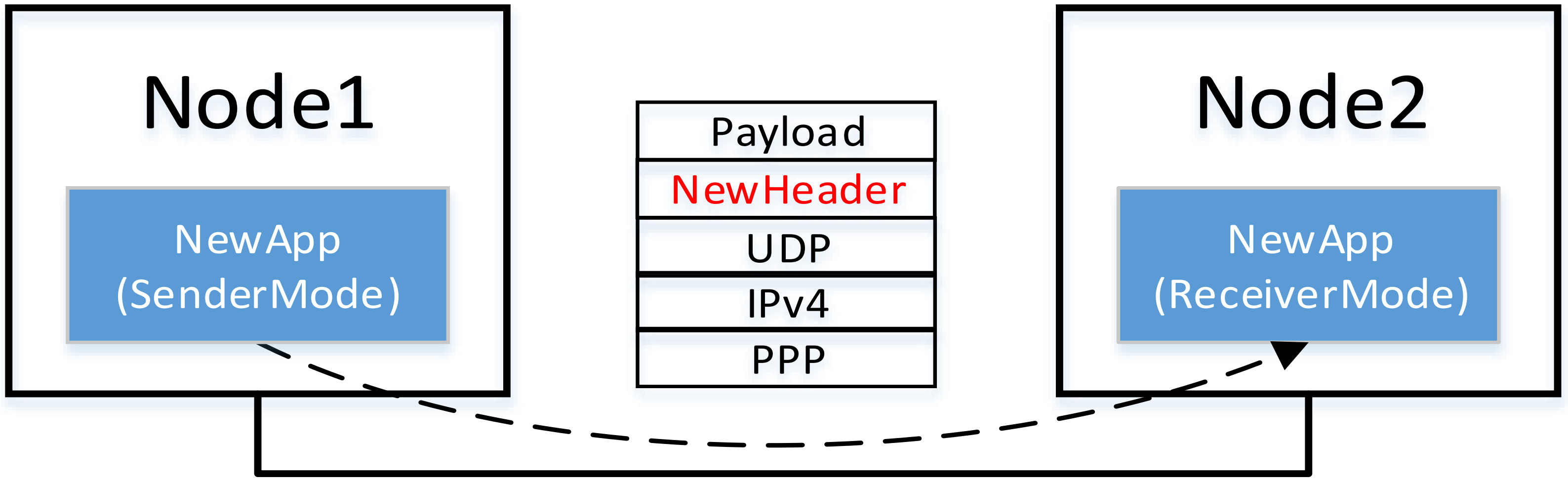
Overview

- Creating a new header called "NewHeader"
  - Application header
  - Attach current time(us) before payload
    - It can be used to measure end-to-end delay

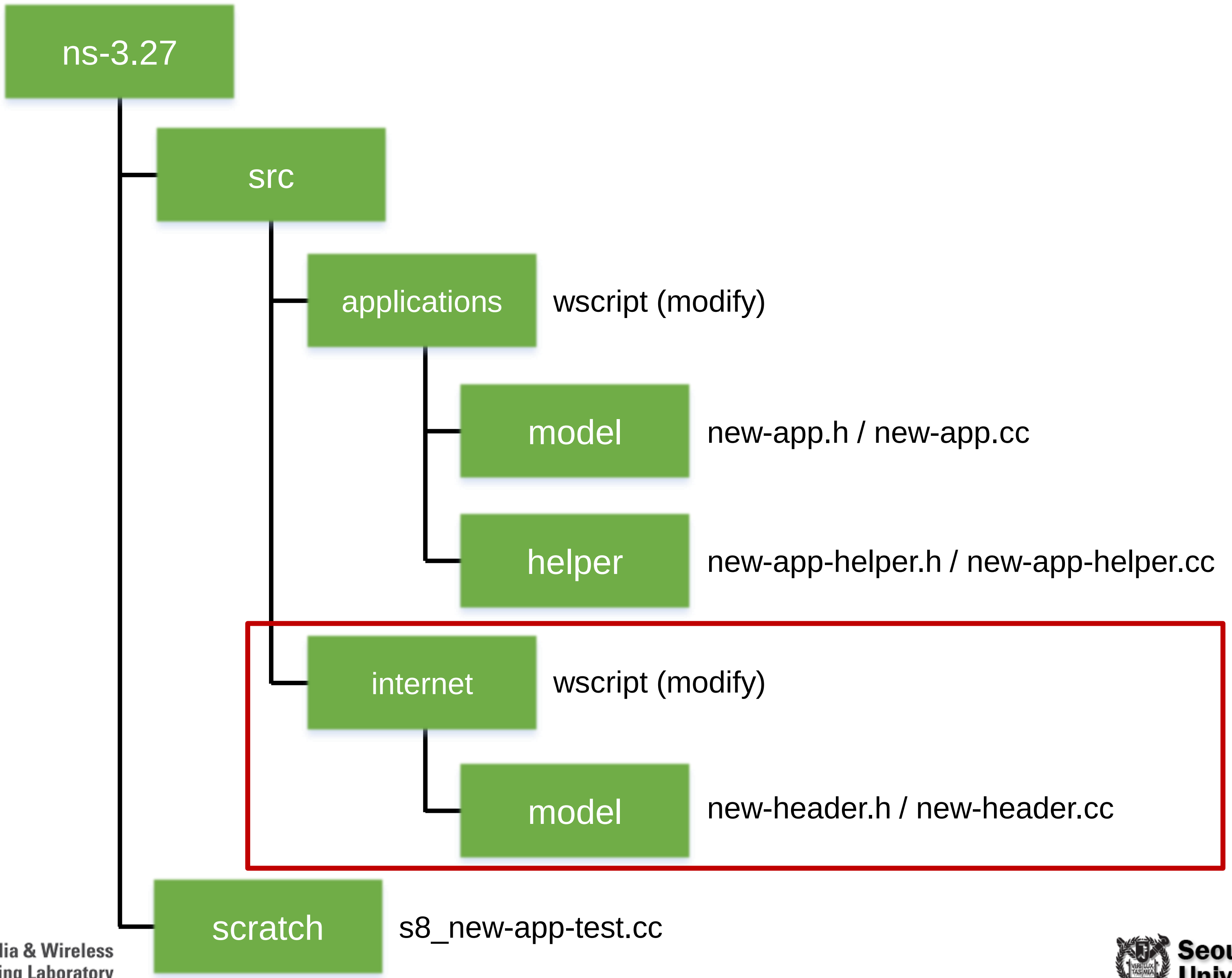


Overview

- Creating a new header called "NewHeader"
  - Used in NewApp



Source codes



Module dependency

- Modify “wscript” file in src/internet folder

```
def build(bld):
    obj = bld.create_ns3_module('internet', ['bridge', 'mpi', 'network', 'core'])
    obj.source = [
        …
        'model/new-header.cc',
       ]
    …
    headers.source = [
        …
        'model/new-header.h',
        ]
```

NewHeader

- Class definition (new-header.h)

```
class NewHeader : public Header
{
  public:
    static TypeId GetTypeId (void);
    virtual TypeId GetInstanceTypeId (void) const;
    virtual void Print (std::ostream &os) const;
    virtual uint32_t GetSerializedSize (void) const;
    virtual void Serialize (Buffer::Iterator start) const;
    virtual uint32_t Deserialize (Buffer::Iterator start);
    void SetTime (void);
    uint64_t GetTime (void) const;
  private:
    uint64_t m_time;
};
```

NewHeader

- GetTypeId & GetInstanceTypeId (new-header.cc)

```
TypeId NewHeader::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::NewHeader")
    .SetParent<Header> ()
    .AddConstructor<NewHeader> ()
    ;
  return tid;
}

TypeId NewHeader::GetInstanceTypeId (void) const
{
  return GetTypeId();
}
```

NewHeader

- SetTime & GetTime (new-header.cc)

```
void NewHeader::SetTime (void)
{
  m_time = Simulator::Now().GetMicroSeconds();
}

uint64_t NewHeader::GetTime (void) const
{
  return m_time;
}
```

Set current time (us)

NewHeader

- Serialize & GetSerializedSize (new-header.cc)

```
void NewHeader::Serialize (Buffer::Iterator start) const
{
  start.WriteHtonU64(m_time);          ← Record time (64bits = 8bytes)
}

uint32_t NewHeader::GetSerializedSize (void) const
{
  return 8;     ← Header size (64bits = 8bytes)
}
```

NewHeader

- Deserialize & Print (new-header.cc)

```
uint32_t NewHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;
  m_time = i.ReadNtohU64();
  return i.GetDistanceFrom(start);
}

void NewHeader::Print (std::ostream &os) const
{
  os << “m_time = " << m_time << "\n";
}
```

Get time from header (64bits = 8bytes)

Distance = 8bytes

Print time

NewHeader in NewApp

- void NewApp::SendPacket () (new-app.cc)

```
void NewApp::SendPacket (void)
{
  …
  Ptr<Packet> packet = Create<Packet> (m_packetSize);
  m_txTrace(packet);

  NewHeader hdr;
  hdr.SetTime();
  hdr.Print(std::cout);
  packet->AddHeader(hdr);

  m_socket->Send(packet);
  …
}
```

Make a NewHeader,
Set time & Print,
Attach header into packet

NewHeader in NewApp

- void NewApp::HandleRead (Ptr<Socket> socket) (new-app.cc)

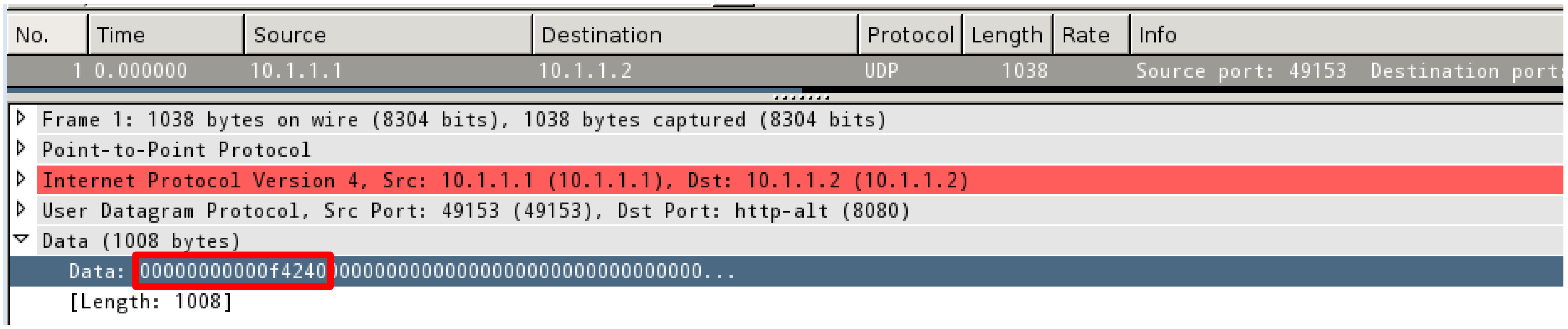
```
void NewApp::HandleRead (Ptr<Socket> socket)
{
  …
  while ((packet = m_socket->RecvFrom(from)))
  {
      …
      NewHeader hdr;
      packet->RemoveHeader(hdr);
      uint64_t hdr_time = hdr.GetTime();
      NS_LOG_INFO("header time at rx = " << hdr_time);

      m_rxTrace(packet);
      …
}
```

Make a NewHeader,
Detach header from packet
Get time & Print

Header Example

- Result
  - 8byte header, 0xf4240 = 1,000,000



```
No.  Time       Source     Destination  Protocol  Length  Rate  Info
   1 0.000000   10.1.1.1   10.1.1.2     UDP       1038          Source port: 49153  Destination port
```

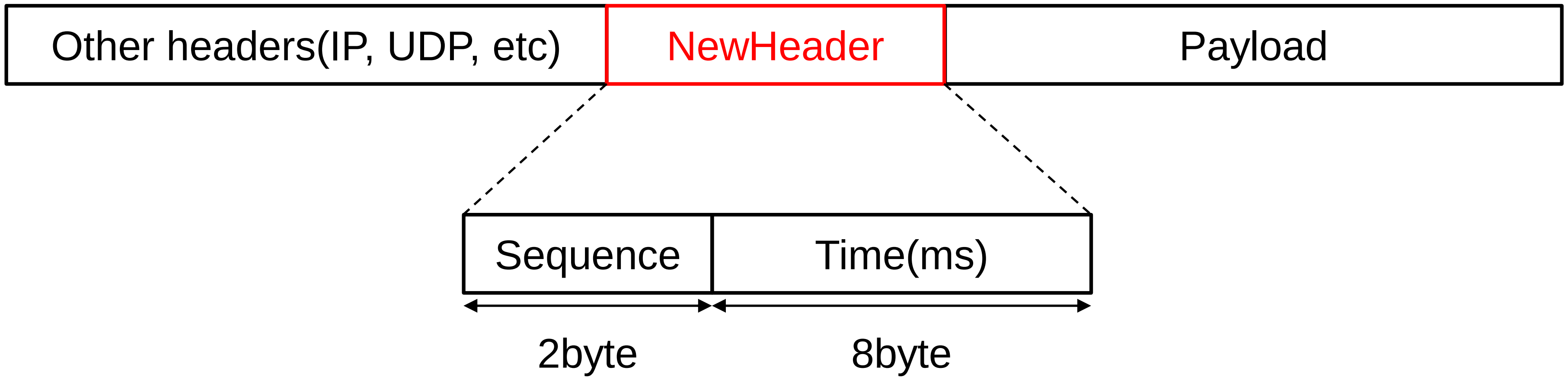
```
▷ Frame 1: 1038 bytes on wire (8304 bits), 1038 bytes captured (8304 bits)
▷ Point-to-Point Protocol
▷ Internet Protocol Version 4, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.1.2 (10.1.1.2)
▷ User Datagram Protocol, Src Port: 49153 (49153), Dst Port: http-alt (8080)
▽ Data (1008 bytes)
    Data: 00000000000f42400000000000000000000000000000000...
    [Length: 1008]
```

Multimedia & Wireless Networking Laboratory

Exercise

Exercise

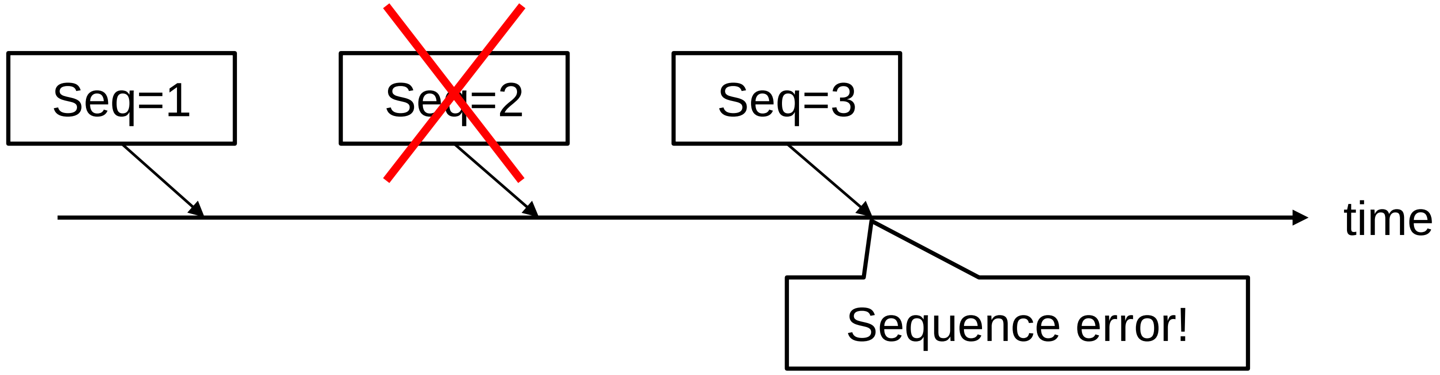
- Add sequence field into NewHeader



- Sequence field: Packet sequence
  - Initial value = 0, Increase 1 for every packet

Exercise

- Check packet sequence



  - Print packet sequences
    - Add trace value for sequence error

Multimedia & Wireless Networking Laboratory

Q & A