Scenario 2
My Application

STVN-2016
Jointly organized by
Poornima University, Jaipur & IIIT-Kota

Rahul Hada
hada.rahul@gmail.com
Scenario 2

- Scenario 2
  - i) Create four nodes architecture client, server and two router.
  - ii) Create your own application (TestApp) binds with TCP socket.
Modify Scenario 1

• Scenario 2 i)
Add Routing

- **Ipv4GlobalServerHelper::PopulateRoutingTables()**
- **GlobalRouting**
  - To provide global routing information
  - It is an interface by which the router advertise its connections to neighboring routers.
  - It uses Link State Routing Algorithm
Tracing
Tracing

- To verify our idea/algorithm.
- To generate output for further study.
- It is a ns3 subsystem enables researchers to see how the new idea/algorithm's behaves, by gathering statistics that capture the behavior of the idea/algorithm.
Tracing: Prerequisite

• Prerequisites for Tracing subsystem:

  – **Attributes**
    • To organize the access of internal member objects of a simulation
  
  – **TypeId**
    • A class that records a lot of meta-information about the subclass of Object classes
  
  – **Callbacks**
    • To allow one piece of code to call a function/method without any specific inter-module dependency
Typeid

• This class provide a unique identifier for an interface

• This is a class that records a lot of meta-information about the **subclass of Object classes** :-
  – the **base class** of the subclass
  – the set of **accessible constructor** in the subclass
  – the set of **attributes accessible** in the subclass
  – Tracing mechanism
TypeId GetTypeId(void)

* Return TypeId-object to identify and characterize object.
* Contain object type, constructor and parent object.
* Define the object attribute and trace source.

```cpp
TypeId RoutingProtocol::GetTypeId (void)
{
    static TypeId tid = TypeId ("ns3::olsr::RoutingProtocol")
        .SetParent<ipv4RoutingProtocol> ()
        .AddConstructor< RoutingProtocol > ()
        .AddAttribute ("HelloInterval", "HELLO messages emission interval.",
                        TimeValue (Seconds (2)),
                        MakeTimeAccessor (&RoutingProtocol::m_helloInterval),
                        MakeTimeChecker ()
                ...
        .AddTraceSource ("Rx", "Receive OLSR packet.",
                        MakeTraceSourceAccessor (&RoutingProtocol::m_rxPacketTrace))
    ;
    return tid;
}
```
Typeid

Object

Node

NetDevice

<metadata>

<metadata>

Typeid

<metadata>
Lets Trace

• It provide pre-configured *TRACE SOURCE*

• Users provide *TRACE SINKS* and attach to the *TRACE SOURCE*

• Multiple *TRACE SOURCES* can connect to a *TRACE SINK*
**NS3-Tracing Model**

- Decouple **TRACE SOURCE** from **TRACE SINK**

![Diagram of NS3-Tracing Model]

- **SOURCE**
- **SINK**

Unchanging

Configurable by user

Unchanging

15-16/04/2016

Simulation of TCP Variants using NS-3
Multiple Levels of Tracing

• **High-level**
  − Use a *helper to hook* a predefined trace source to an existing trace sink (Ascii, pcap)

• **Mid-Level**
  − Hook an existing trace source to a *custom trace sink*

• **Low-level**
  − Add a *new trace source* and connect it to a *special trace sink*
Trace Helper
[High Level]

• It provide a rich environment for configuring and selection different *trace events and write them to files*.

• Two types of tracing helpers:
  - **Device Helper** – enable trace on node + device pair
  - **Protocol Helper** – enable trace on protocol + interface
Tracing

[Mid Level]

- Custom Track Sink
  - Step 1 Find the trace source (**PointToPointNetDevice**)

```c
// Trace sources at the "top" of the net device, where packets transition
// to/from higher layers

//
.AddTraceSource ("MacTx",
   "Trace source indicating a packet has arrived "
   "for transmission by this device",
   MakeTraceSourceAccessor (&PointToPointNetDevice::m_macTxTrace),
   "ns3::Packet::TracedCallback")

.AddTraceSource ("MacRx",
   "A packet has been received by this device, "
   "has been passed up from the physical layer",
   "and is being forwarded up the local protocol stack. "
   "This is a non-promiscuous trace."
   MakeTraceSourceAccessor (&PointToPointNetDevice::m_macRxTrace),
   "ns3::Packet::TracedCallback")

#if 0

// Not currently implemented for this device
.AddTraceSource ("MacRxDrop",
   "Trace source indicating a packet was dropped "
   "before being forwarded up the stack",
   MakeTraceSourceAccessor (&PointToPointNetDevice::m_macRxDropTrace),
   "ns3::Packet::TracedCallback")
#endif

// Trace sources at the "bottom" of the net device, where packets transition
// to/from the channel.
```
TraceSource: PointToPointNetDevice

- **Trace Source** on **Point-To-PointNetDevice** few are as follows:
  - **MacTx** – a packet has arrived for transmission by this device
  - **MacRx** – a packet has been received by this device and is being forwarded up the local protocol stack
  - **PhyTxBegin** – a packet has begun transmitting over the channel
  - **PhyTxEnd** – a packet has been completely transmitted over the channel
  - **PhyRxBegin** – a packet has begun being received by the device.
  - **PhyRxEnd** – a packet has been completely received by the device.
Tracing
[Mid Level]

• **Step-2** Create an Object in which the trace source lives.
  
  – `Ptr<Object> traceSource = Object_of_TraceSource.GetXXX(X);`
  
  – Example **PointToPointNetDevice** Trace Source

  ```cpp
  Ptr<Object> traceSource = devices.Get(0);
  ```
Tracing [Mid Level]

Callback Signature

- **Step-3** Find the signature of the callback function

```c
typedef void(* ns3::Packet::TracedCallback)(Ptr<const Packet> packet)
```

*TracedCallback signature for *Ptr*<Packet>*

**Parameters**

*packet* The packet.

Definition at line 664 of file packet.h.
Tracing [Mid Level]

- Step-4 Connect trace SOURCE with SINK
  - We need to write a callback function to serve as a trace sink