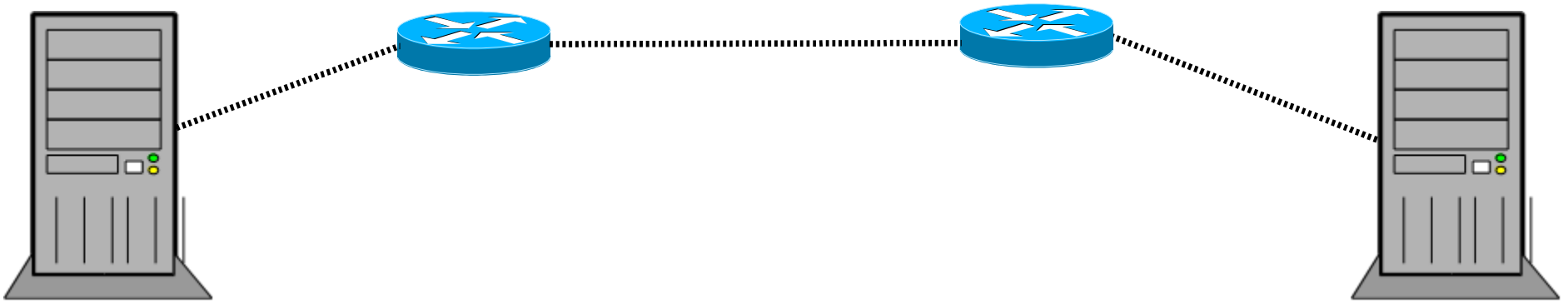
# Scenario 2
# My Application



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

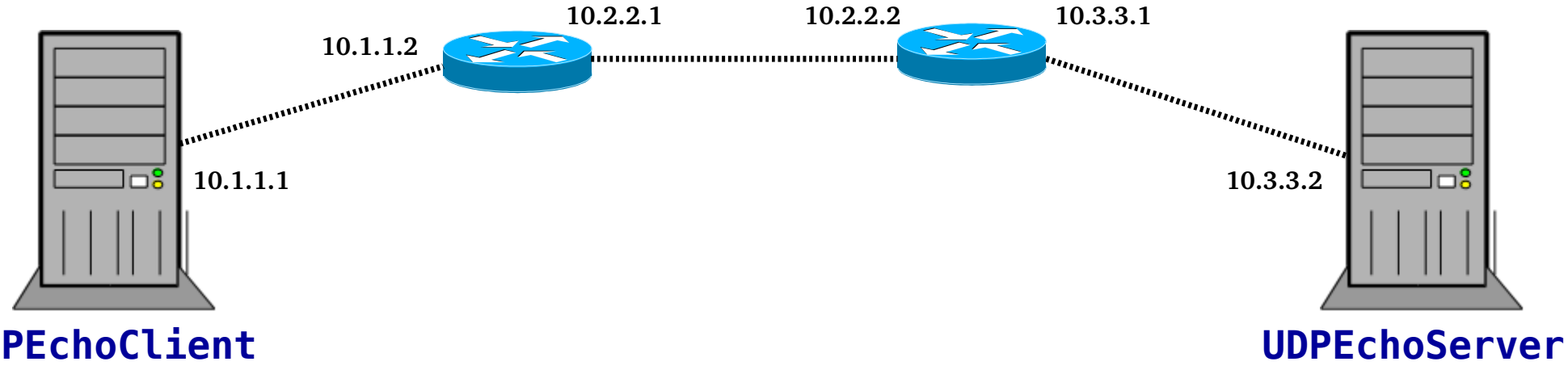
Rahul Hada
hada.rahul@gmail.com

# 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

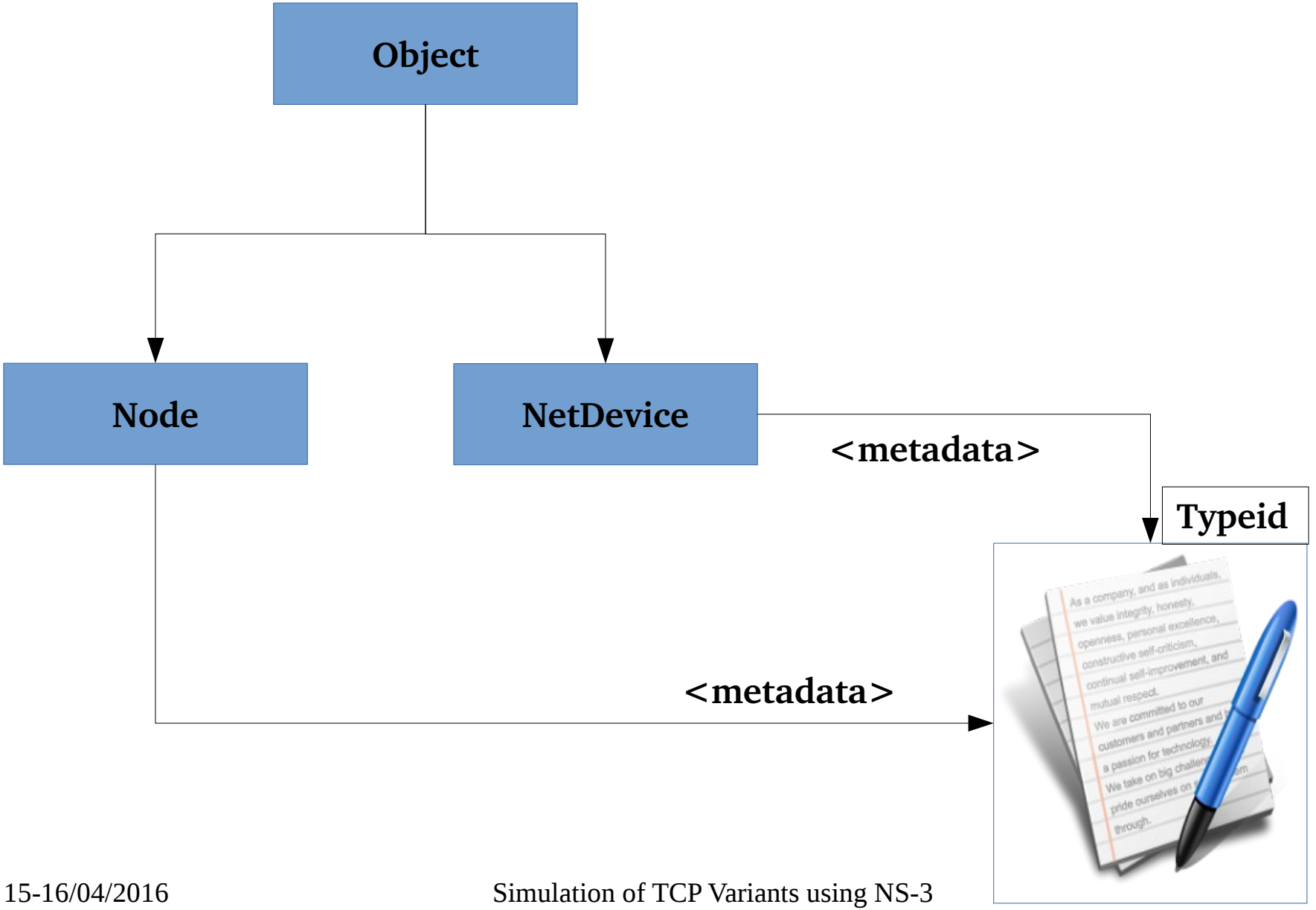
- This class provide a unique identifier for an interface
- This is a class that records a lot of meta-information about the ***subclass of Object classes*** :-
  - the ***base class*** of the subclass
  - the set of ***accessible constructor*** in the subclass
  - the set of ***attributes accessible*** 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.

```
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

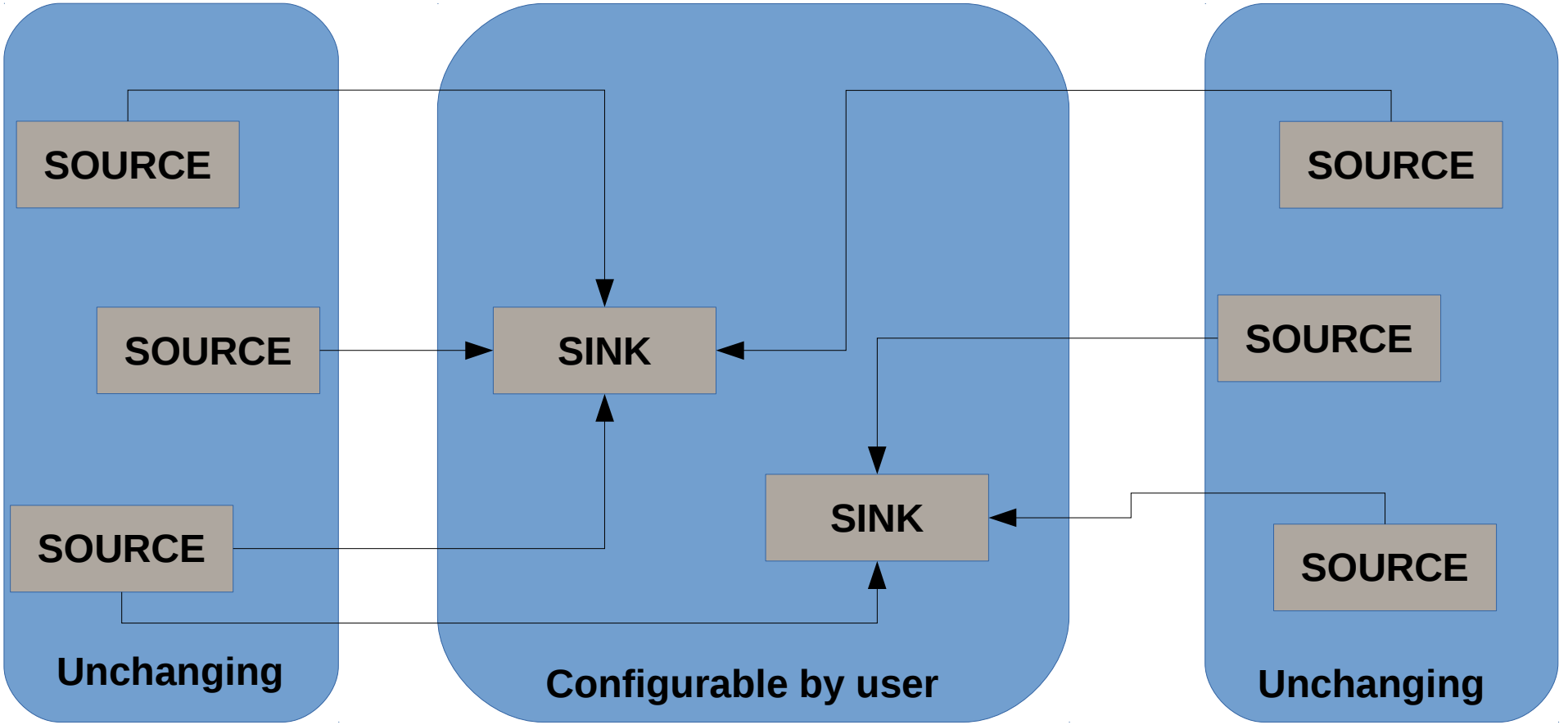


# Lets Trace

- 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**



# 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)**

```
80
81      //
82      // Trace sources at the "top" of the net device, where packets transition
83      // to/from higher layers.
84      //
85      .AddTraceSource ("MacTx",
86                       "Trace source indicating a packet has arrived "
87                       "for transmission by this device",
88                       MakeTraceSourceAccessor (&PointToPointNetDevice::m_macTxTrace),
89                       "ns3::Packet::TracedCallback")
90      .AddTraceSource ("MacTxDrop",
91                       "Trace source indicating a packet has been dropped "
92                       "by the device before transmission",
93                       MakeTraceSourceAccessor (&PointToPointNetDevice::m_macTxDropTrace),
94                       "ns3::Packet::TracedCallback")
95      .AddTraceSource ("MacPromiscRx",
96                       "A packet has been received by this device, "
97                       "has been passed up from the physical layer "
98                       "and is being forwarded up the local protocol stack.  "
99                       "This is a promiscuous trace,",
100                      MakeTraceSourceAccessor (&PointToPointNetDevice::m_macPromiscRxTrace),
101                      "ns3::Packet::TracedCallback")
102     .AddTraceSource ("MacRx",
103                      "A packet has been received by this device, "
104                      "has been passed up from the physical layer "
105                      "and is being forwarded up the local protocol stack.  "
106                      "This is a non-promiscuous trace,",
107                      MakeTraceSourceAccessor (&PointToPointNetDevice::m_macRxTrace),
108                      "ns3::Packet::TracedCallback")
109 #if 0
110     // Not currently implemented for this device
111     .AddTraceSource ("MacRxDrop",
112                      "Trace source indicating a packet was dropped "
113                      "before being forwarded up the stack",
114                      MakeTraceSourceAccessor (&PointToPointNetDevice::m_macRxDropTrace),
115                      "ns3::Packet::TracedCallback")
116 #endif
117     //
118     // Trace souces at the "bottom" of the net device, where packets transition
119     // to/from the channel.
120     //
```

# 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]

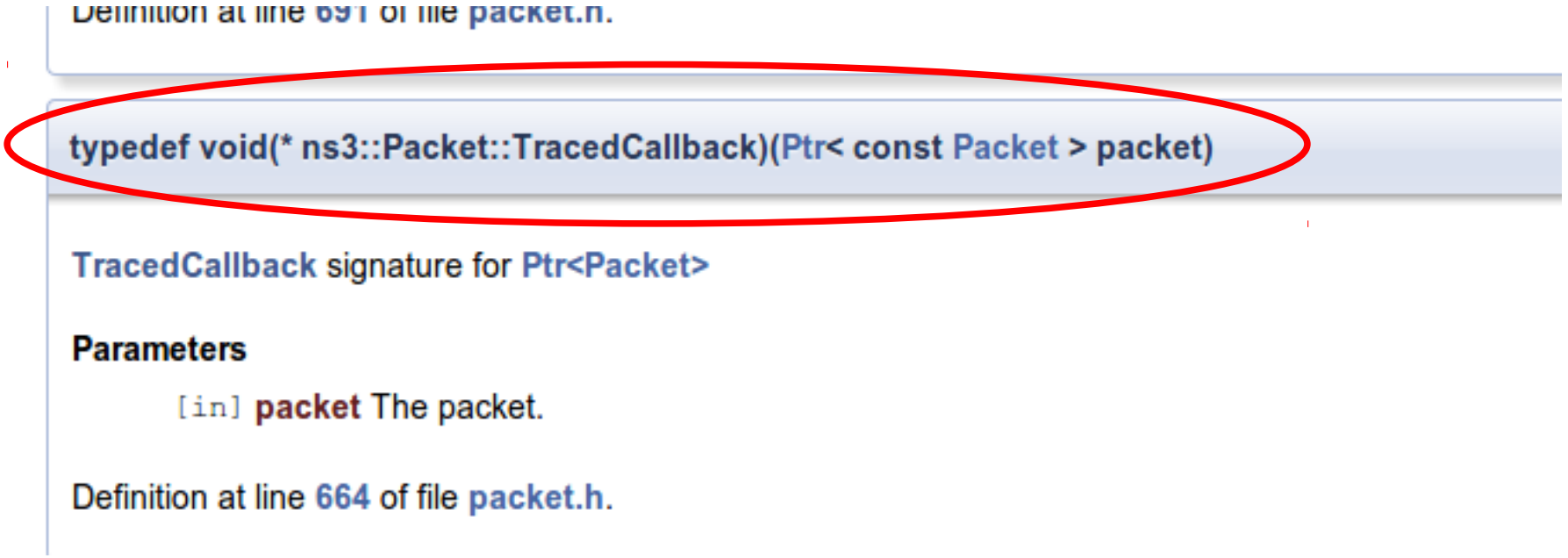
- **Step-2** Create an Object in which the trace source lives.
  - **Ptr<Object> traceSource = Object_of_TraceSource.GetXXX(X);**
  - **Example PointToPointNetDevice Trace Source**

**Ptr<Object> traceSource = devices.Get(0);**

# Tracing
# [Mid Level]

## Callback Signature

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



# Tracing
# [Mid Level]

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