VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY
UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Network Simulation

Lecture 6: Logs and Analyze results

Dr. NGUYEN Minh Huong

Lecture 6:

- Logs
    - Print out method
    - Existing trace sources
    - Create trace source and sink
- Analyze collected data
    - Data processing
    - Analyzing:
        - Embedded script
        - External tools

Logs

- Logs are informational messages that are output when programs is run
  - Warning messages
  - Debug messages
  - Error messages
  - User-added messages
  - Traces

Logging module

- Enable log messages (7 levels):
  - Log error messages: NS_LOG_ERROR
  - Log warning messages: NS_LOG_WARNING
  - Log debugging messages: NS_LOG_DEUBUG
  - Log informational messages about program progress
    NS_LOG_INFO
  - Log messages describing each called function
    NS_LOG_FUNCTION
  - Log messages describing logical flow within a function
    NS_LOG_LOGIC
  - Log everything: NS_LOG_ALL

Logging module

- LOG_LEVEL_TYPE:
  - Enable logging of all level above Type
- Logging messages without association to log levels:

  NS_LOG_UNCOND
- Adding log to your code:
  - Define a log component: NS_LOG_COMPONENT_DEFINE
  - Setting the NS_LOG environment variable to log levels
  - Enable the log component to equal or higher level than NS_LOG variable

Tracing system

- How to collect?
  - Print out
  - Tracing system:
    - Create and connect trace source/sink
    - Collect from existing trace sources

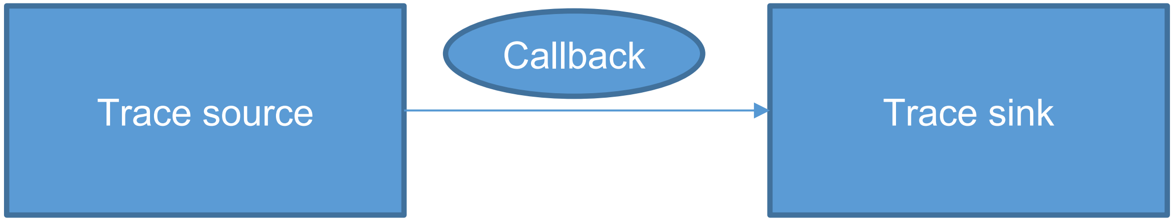
Tracing system (cont)

- Print out:
  - Adding print statements
    - Using logging module
    - Std::cout

→ Quick but hard to organize and control the output format

Tracing system (cont)

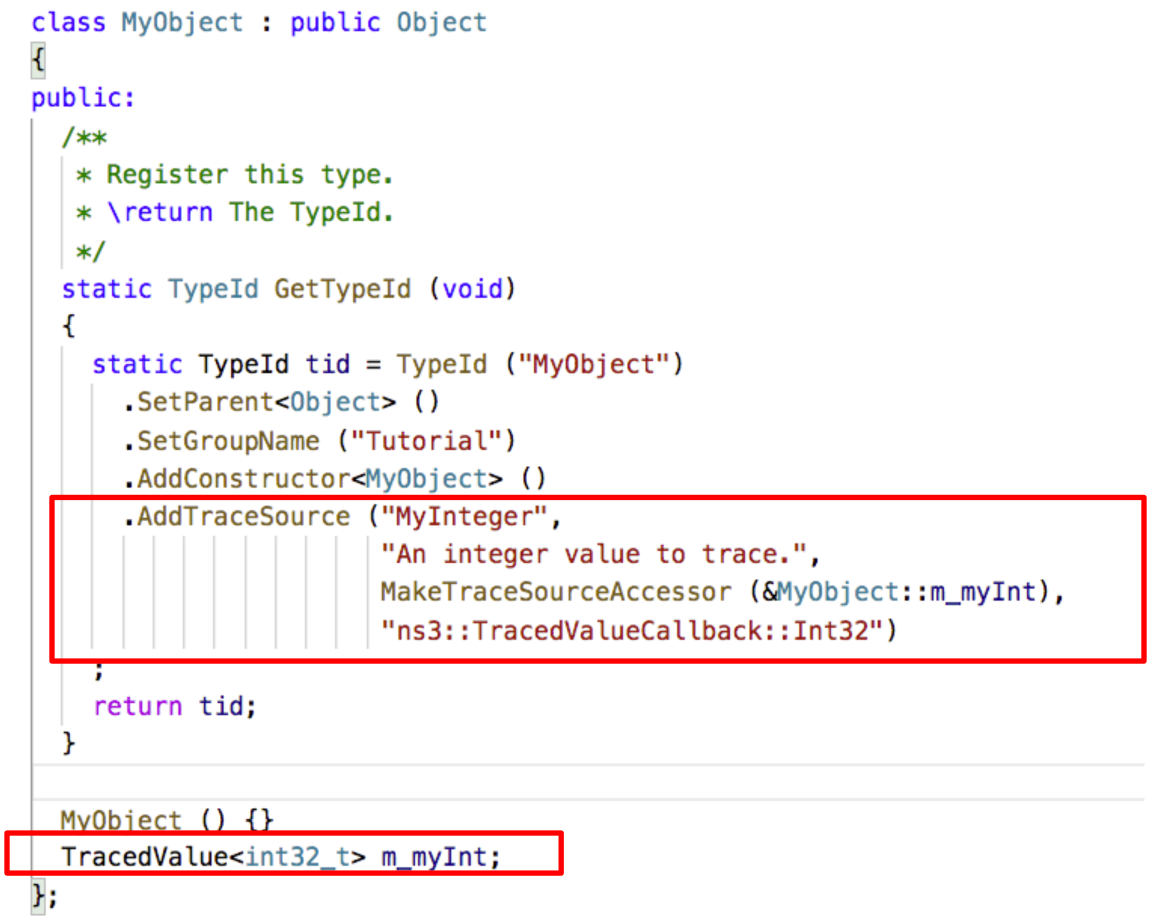
- Tracing system
  - Trace source: notes events that happen in a simulation
  - Trace sink: outputs information from a trace source
  - Connecting source to sink mechanism: callback



Tracing system (cont)

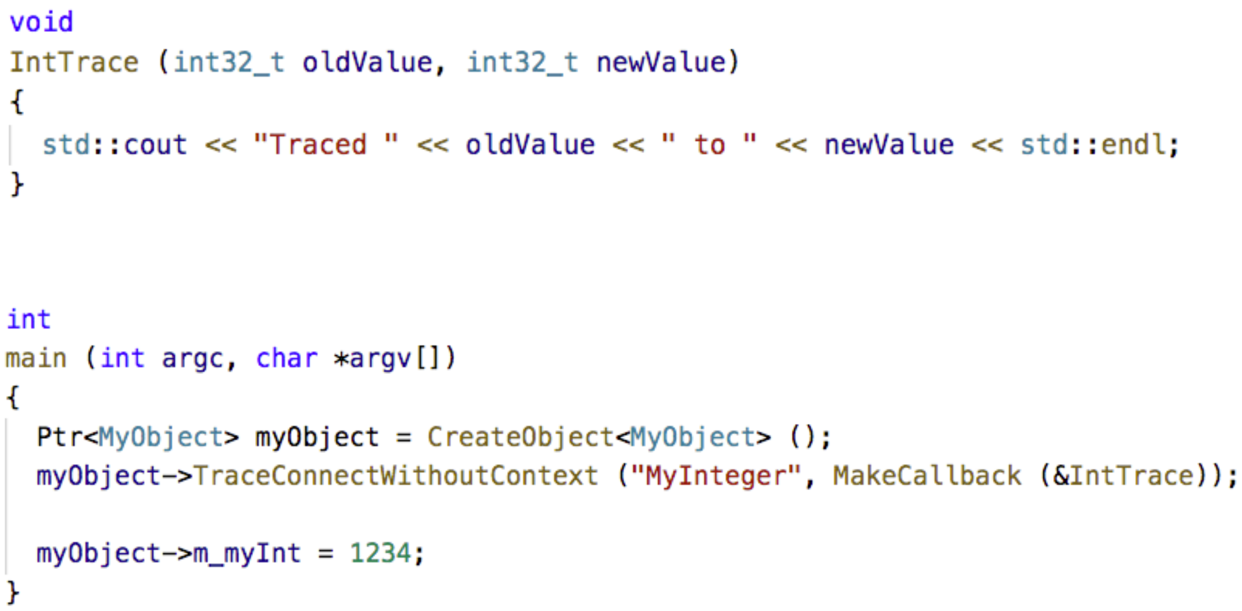
- Connecting trace source and trace sink:
  - Connect with TraceConnectWithoutContext
    - Example: fourth.cc
      - Data to be collected: MyObject::m_myInt
      - Add trace source: make the source visible in Config system
      - Trace value: the value that will be input to trace sink
      - Trace sink: function IntTrace print out the traced value
      - TraceConnectWithoutContext: connect source to sink using callback. Whenever m_myInt changed, trace sink function will be called.

Tracing system (cont)



```
class MyObject : public Object
{
public:
  /**
   * Register this type.
   * \return The TypeId.
   */
  static TypeId GetTypeId (void)
  {
    static TypeId tid = TypeId ("MyObject")
      .SetParent<Object> ()
      .SetGroupName ("Tutorial")
      .AddConstructor<MyObject> ()
      .AddTraceSource ("MyInteger",
                       "An integer value to trace.",
                       MakeTraceSourceAccessor (&MyObject::m_myInt),
                       "ns3::TracedValueCallback::Int32")
    ;
    return tid;
  }

  MyObject () {}
  TracedValue<int32_t> m_myInt;
};
```

Tracing system (cont)



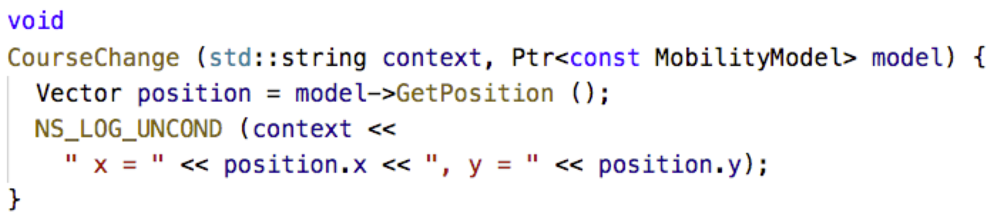
```
void
IntTrace (int32_t oldValue, int32_t newValue)
{
  std::cout << "Traced " << oldValue << " to " << newValue << std::endl;
}


int
main (int argc, char *argv[])
{
  Ptr<MyObject> myObject = CreateObject<MyObject> ();
  myObject->TraceConnectWithoutContext ("MyInteger", MakeCallback (&IntTrace));

  myObject->m_myInt = 1234;
}
```

Tracing system (cont)

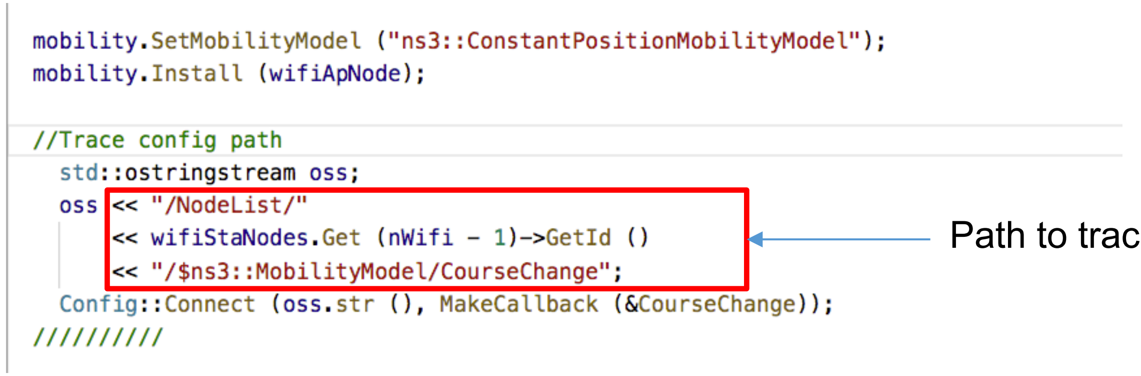
- Connecting trace source and trace sink:
  - Connect with Config
    - Using config path to connect trace source to trace sink
    - Example: third.cc
      - Pre-defined trace source at **Config path**: “/NodeList/” nodeid “/$ns3::MobilityModel/CourseChange ”
      - Trace sink: function CourceChange
      - Connect source and sink using Config::Connect ()
      - **Callback template** is defined at the trace source
    - Finding Source:
      - Available sources: NS-3 API Documentation/all trace sources

      https://www.nsnam.org/docs/release/3.33/doxygen/_trace_source_list.html

Tracing system (cont)



```
void
CourseChange (std::string context, Ptr<const MobilityModel> model) {
  Vector position = model->GetPosition ();
  NS_LOG_UNCOND (context <<
    " x = " << position.x << ", y = " << position.y);
}
```



```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);

//Trace config path
  std::ostringstream oss;
  oss << "/NodeList/"
      << wifiStaNodes.Get (nWifi - 1)->GetId ()
      << "/$ns3::MobilityModel/CourseChange";
  Config::Connect (oss.str (), MakeCallback (&CourseChange));
//////////
```

Path to trace source

Analyze collected data

- Data Processing
  - Raw data → Metrics
  - Raw data → Expected format data set

Analyze collected data

- Analyzing:
  - Embedded script:

  Log the raw data and compute directly with C++
  - External tools:
    - Log the raw data in text format
    - Tools: grep, awk … to extract wanted data with selected format
    - Using Matlab, python … to compute metrics from raw data
- Representing results:
  - Draw graphs, tables
  - Comparing results when varying simulation parameters

Examples

- Labwork 2: first.cc
  - Application: client sends 100 packets with interval of 0.1s to echo server
  - Raw data: packet traces at client and server for a simulation duration of 10s
  - Metrics:
    - packet delivery ratio
    - Delay

Examples

- Labwork 2: first.cc
  - Data Processing: raw data→ Metrics?
    - Save output from screen to a file:

      2>&1 | tee file.txt
    - Extract informative data from the file

      awk ‘commands’ file.txt
    - Compute metrics from extracted data

      Matlab

Examples

- Labwork 5:

Consider CSMA/CA protocol in Wifi networks that working in adhoc mode. Evaluate performance of the protocol without RTS/CTS scheme when the number of nodes within a communication range increases from 2 to 30.