LAB4: ns-3 Tracing System

CS169: Mobile Wireless Networks - Winter 2018

Xukan and Thomas

Department of Computer Science and Engineering
University of California, Riverside

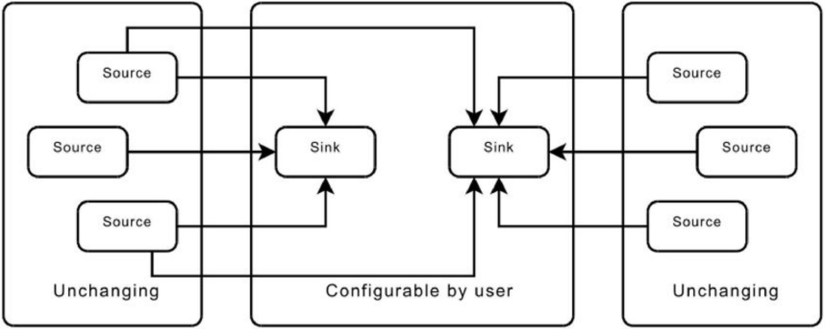
February 2, 2018

Tracing Revisited

- NS_LOG and std::cout are quick and dirty, so they may not be effective for serious work
- What if we want to look at specific data or state changes ?
- Trace source (generators of trace data) → trace sink (consumer)
- Ex. we are interested in Congestion Window size or Mobility Tracking locations.

ns-3 Tracing Model

- Decouple trace sources from trace sinks:

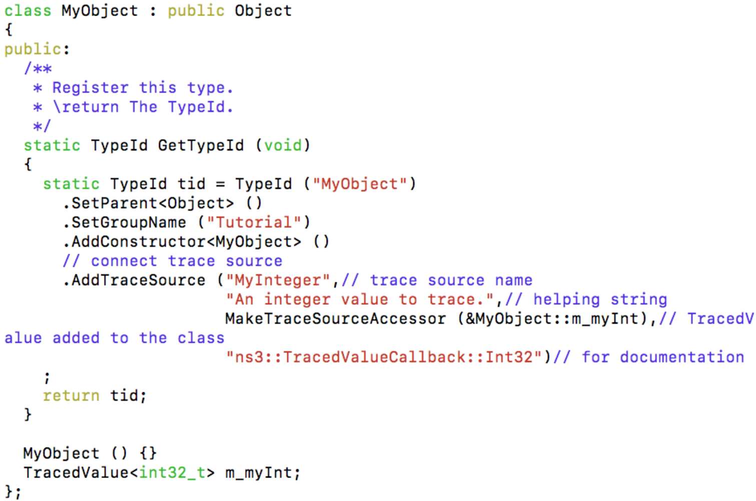


Callbacks

- pointer-to-function-returning-integer (PFI)
- *int (*pfi)(int arg) = 0 ;*
- Creating MyFunction returning int
- *int MyFunction (int arg)* {}
- Initialize pfi to point to MyFunction
- *pfi = MyFunction;*
- Then we can call MyFunction indirectly by
- *int result = (*pfi) (1234)*
- or
- *int result = pfi (1234)*
- The system maintains a list of callbacks triggered by events of interest, whose data are passed along from trace source to the target function (sink)

fourth.cc

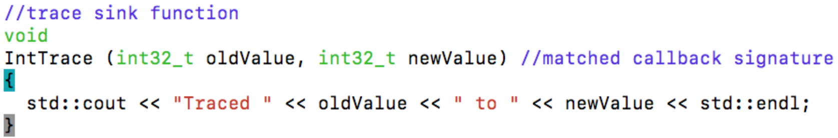
- $ cp examples/tutorial/fourth.cc scratch/myfourth.cc
- $ vim scratch/myfourth.cc



```
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> ()
      // connect trace source
      .AddTraceSource ("MyInteger",// trace source name
                       "An integer value to trace.",// helping string
                       MakeTraceSourceAccessor (&MyObject::m_myInt),// TracedV
alue added to the class
                       "ns3::TracedValueCallback::Int32")// for documentation

    ;
    return tid;
  }

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

fourth.cc

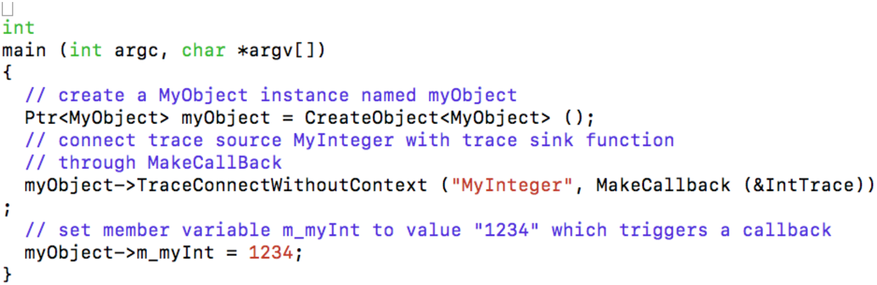
- Trace sink function



```
//trace sink function
void
IntTrace (int32_t oldValue, int32_t newValue) //matched callback signature
{
  std::cout << "Traced " << oldValue << " to " << newValue << std::endl;
}
```

fourth.cc

- main

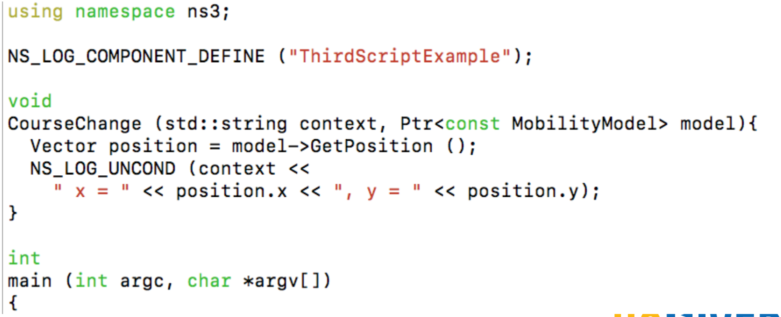


```
int
main (int argc, char *argv[])
{
  // create a MyObject instance named myObject
  Ptr<MyObject> myObject = CreateObject<MyObject> ();
  // connect trace source MyInteger with trace sink function
  // through MakeCallBack
  myObject->TraceConnectWithoutContext ("MyInteger", MakeCallback (&IntTrace))
;
  // set member variable m_myInt to value "1234" which triggers a callback
  myObject->m_myInt = 1234;
}
```

Let's make some change to *myObject* → *m_myInt* and see what will happen.

Connect with Config Subsystem

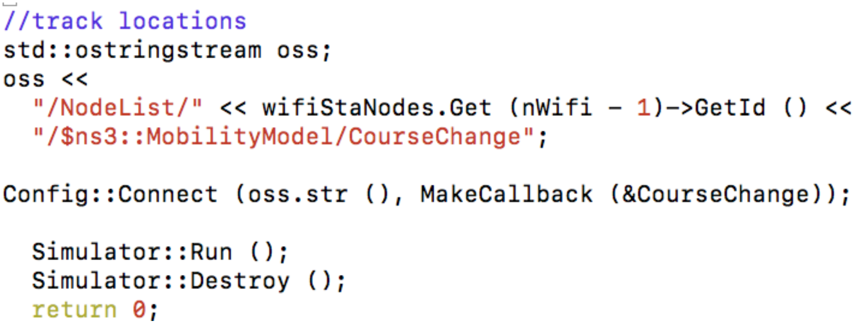
- Let's re-visite mythird
- `$ vim scratch/mythird.cc`
- and insert this code before *int main* and do you think this is trace source or trace sink?



```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

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

int
main (int argc, char *argv[])
{
```

Connect with Config Subsystem

- Then we use a config path as a trace source by inserting this before *Simulator::Run ( );*
- Then try running mythird



```
//track locations
std::ostringstream oss;
oss <<
  "/NodeList/" << wifiStaNodes.Get (nWifi - 1)->GetId () <<
  "/$ns3::MobilityModel/CourseChange";

Config::Connect (oss.str (), MakeCallback (&CourseChange));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
```

Connect with Config Subsystem

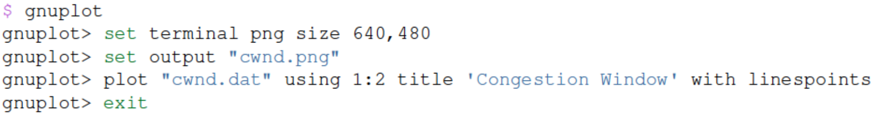
- Actually, the config path */NodeList/7/$ns3::MobilityModel/CourseChange* is broken down to
- */NodeList/7/* is a node object and *$ns3::MobilityModel* is another object aggregated with the node object and finally CourseChange is the attribute we want to take a look

fifth.cc

- `$ cp examples/tutorial/fifth.cc scratch/myfifth.cc`
- `$ vim scratch/myfifth.cc`
- `$ ./waf --run scratch/myfifth.cc > cwnd.dat 2>&1`
- Take a look at the result. Do you see congestion window changes? Do you see packet drop? How many times per second?

gnuplot

- Edit up “cwnd.dat”: Get rid of all of unrelated traces and keep only congestion window trace
- Plot congestion window with time using the following commands



```
$ gnuplot
gnuplot> set terminal png size 640,480
gnuplot> set output "cwnd.png"
gnuplot> plot "cwnd.dat" using 1:2 title 'Congestion Window' with linespoints
gnuplot> exit
```

Exercise

- Set receive error rate to 10^{-4}, 10^{-3}, 10^{-2} and compare congestion window and Rx drop rate.
- From mythird, plot the CourseChange of the last WiFi node into x-y axes to show where it goes.

Questions?