RapidNet - A small Tutorial

RapidNet allows you to write network applications using DataLog rules. The project is based on Network DataLog, that associates rules with an IPAddress stating where the rule should be executed.

Overview of how to use RapidNet:

RapidNet uses waf to build the project. It recursively builds sub directories to build the final project. The build process can be configured by the wscript present in each directory that is being built. For example when creating a new application, one can add the application to the src/wscript file to ensure that it will be included in the build next time.

The main implementation of rapidnet is in its src directory. A brief overview of the directories present inside it is given below :

1. applications/ : Implementation of various network applications are added here.
2. common/ : The Packet class is defined here - this is used to transmit DataLog tuples to different nodes on the network
3. contrib/ : Contains additional utilities to RapidNet like garbage collector or classes to plot graphs using GNUplot
4. core/ : Contains definitions of data types used in the project
5. devices/ : Contains definitions of types of devices in the network like CSMA
6. helper/ : Contains definitions of classes that help set up the application - for example node-container is used to initialize the network for simulation.
7. internet-stack/: Contains important definition of classes that implement network protocols
8. l4-platform/: Contains definition of classes used to run RapidNet over a real network.
9. node/: Contains definition of classes used for networking - like sockets, IPAddress etc.
10. rapidnet/: Contains important classes like rapidnet-application-base and rapidnet-application-helper that help start stop the RapidNet program and execute the actual Datalog rules
11. rapidnet-compiler/: Contains the implementation of the RapidNet compiler

When writing the NDLog rules, the attributes should start with a capital letter. The rapidnet compiler will generate a cc file for the corresponding NDLog rules. The file declares a helper class that is an instance of the Rapidnet-application-helper class and contains functions corresponding to the rules of rapidnet. Note the compiler does not detect some errors (for example if it cannot find a variable in a rule). These errors however are uncovered when building the project. Once the project builds correctly, a test file can be written in the examples/ directory and that test can be run using ./waf --run command.

There are two ways to use RapidNet. Either in simulation mode or over a real network using L4 devices.

Simulation Mode :

Simulation mode uses ns-3 to simulate a network. An example can be found here http://netdb.cis.upenn.edu/rapidnet/doxygen/html/rapidnet-ndlog-application.html .

L4 Devices :

Using L4 Devices, RapidNet can be used over a real network. The following example gives a brief tutorial about how to use it. The example we use is ping-pong example that can be found in src/applications/pingpong-l4.

The Ping-Pong example contains two nodes. Each node initially sends an ePing tuple with an associated id to the other node. On receiving the ePing tuple, a node replies with an ePong tuple acknowledging the fact that it received the ePing. After receiving an ePong tuple, the node locally generates an ePingPong tuple and terminates.

For our experiment, we will use the examples/ping-pong-l4.cpp test file with it’s entry in examples/wscript called ping-pong-l4-test.

First, we create the olg file containing the NDlog rules for the ping-pong example.The NDlog rules are given below :

```
materialize(tLink, infinity, infinity, keys(1,2)).

// Every 1 second, node "Src" will send out a ePing event to its neighbors, i.e. node "Next"
r1  ePing(@Next, Src):-
        //periodic(@Src, E, 5),
        tLink(@Src, Next).

// When a node receive a ePing event from its neighbor, it sent back a ePong message
r2  ePong(@Next, Src):-
        ePing(@Src, Next).

// When a node receive a ePong event from its neighbor, this Ping-Pong procedure is indicated
as finished
r3  ePingPongFinish(@Src):-
        ePong(@Src, Next).
```

The rules basically do the following- It materializes the tLink table - this makes it persistent over the course of the experiment (the infinity value in the second and third parameter of materialize make the table persistent). The keys specify the primary keys of the table.

The rules are evaluated right to left so for every tLink entry at IPAddress Src, a ePing tuple is generated at IPAddress Next. When the ePing tuple is generated at Next it triggers the second rule which generates an ePong at the source. Once the ePong is generated, an ePingPong tuple is generated which acts as the base case and the application finishes.

Next, we create the test file (ping-pong-l4.cc) for running our experiment.

****

We have included 2 new header files for running our experiment : ns3/l4-platform-helper.h and ns3/pingpong-l4-module.h

```
#include "ns3/core-module.h"
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/pingpong-l4-module.h"
#include "ns3/rapidnet-module.h"
#include "ns3/values-module.h"
#include "ns3/l4-platform-helper.h"
#include "ns3/uinteger.h"
#include <iostream>
#include <sstream>
```

The following represent macros which are used to insert data into materialized tables. This allows communication between nodes using the NDlog rules using the tLink tuple.

```
#define tlink(src, next) \
  tuple (PingpongL4::TLINK, \
        attr ("tLink_attr1", StrValue, src), \
        attr ("tLink_attr2", StrValue, next))

#define insertlink(from, to) \
  app(from)->Insert (tlink (app(from)->GetLocalLocSpec(), app(to)->GetLocalLocSpec()));
\
  app(to)->Insert (tlink (app(to)->GetLocalLocSpec(), app(from)->GetLocalLocSpec()));
```

```
#define deletelink(from, to) \
  app(from)->Delete (tlink (app(from)->GetLocalLocSpec(),
app(to)->GetLocalLocSpec())); \
  app(to)->Delete (tlink (app(to)->GetLocalLocSpec(), app(from)->GetLocalLocSpec()));

#define tlinkCounter(src,next,id)\
        tuple(PingpongL4::TLINKCOUNTER, \
        attr("tLinkCounter_attr1",StrValue,src) ,\
        attr("tLinkCounter_attr2",StrValue,next),\
        attr("tLinkCounter_attr3",Int32Value,id))

#define insertTlinkCounter(src,next,id)\
  app(1)->Insert (tlinkCounter(app(1)->GetLocalLocSpec(), destLocSpec,id));

using namespace std;
using namespace ns3;
using namespace ns3::rapidnet;
using namespace ns3::rapidnet::pingpongl4;

ApplicationContainer apps;
string destLocSpec = "", localPort = "";
string localIPAddress = "";
```

Unlike simulation (where we specify both the source and destination addresses of the communicating nodes, we can use the ip address of the machine running the current instance of Rapidnet using the GetLocalSpec method.
Note : Here the ip addresses are represented as a string of the form ipaddress : port.

```
void
UpdateLinks1 ()
{
  app(1)->Insert (tlink (app(1)->GetLocalLocSpec(), destLocSpec));
}


void AddTLinkCounter()
{
        // Since there are 10 requests
        for(int i=0; i< 10; i++)
        {
                insertTlinkCounter(app(1)->GetLocalLocSpec(), destLocSpec, i);
        }
```

```
}


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

  std::time_t result = std::time(NULL);
  std::cout << std::asctime(std::localtime(&result)) << result << " -> Starting time\n";

  // Use realtime simulator
  GlobalValue::Bind ("SimulatorImplementationType",
                     StringValue ("ns3::RealtimeSimulatorImpl"));

  LogComponentEnable("PingpongL4", LOG_LEVEL_INFO);
  LogComponentEnable("RapidNetApplicationBase", LOG_LEVEL_INFO);

  uint16_t port;

  CommandLine cmd;

  cmd.AddValue ("dest", "Destination LocSpec IP:Port", destLocSpec);
  cmd.AddValue ("localPort", "Local RapidNet port", localPort);
  cmd.AddValue ("localIPAddress", "Local IPAddress", localIPAddress);
  cmd.Parse (argc, argv);

  if (localPort != "")
  {
          std::istringstream sin (localPort);
          sin >> port;
  }
  else
  {
          port = 11111;
  }
```

Install L4-Platform using the following code snippet :

```
NodeContainer nodeContainer;
  nodeContainer.Create (1);
  L4PlatformHelper platform;
 platform.Install (nodeContainer);
 PingpongL4Helper helper = PingpongL4Helper();
```

This enables L4 mode

```
 helper.SetL4Platform (true);
 if(localIPAddress != "")
           helper.SetLocalAddress(Ipv4Address((const char*)localIPAddress.c_str()));

  helper.SetAttribute ("RapidNetPort", UintegerValue (port));
  // Install ping pong
  apps = helper.Install (nodeContainer);

ofstream* decorator_out = InstallDecorator(apps, ".", 2000, 2000);
```

Set all necessary default values :

```
  if (destLocSpec == "")
           destLocSpec = app (1)->GetLocalLocSpec ();
```

This specifies that our application is going to run for 50 seconds.

```
  apps.Start (Seconds (0.0));
  apps.Stop (Seconds (50.0));

  schedule(0.00001, AddTLinkCounter);
  schedule (20, UpdateLinks1);

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