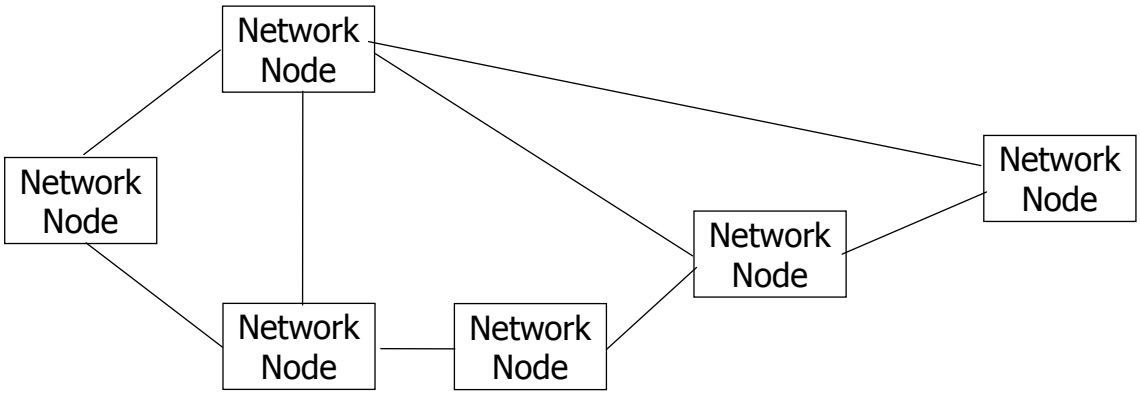
Simulative Evaluation of Internet Protocol Functions

Chapter 4
Overview of the `protsim` Framework

Introduction

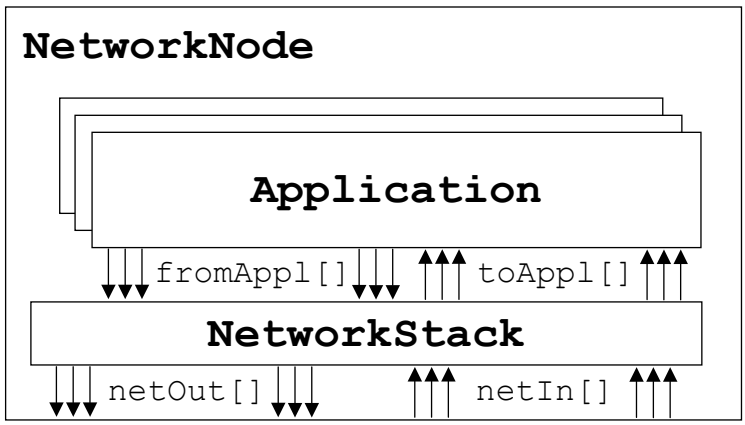
- A semester may look like a bunch of time at first, but you will soon see that it is not long enough to write the complete simulation, especially since you
  - may have never been in touch with networking
  - may have never programmed in C++
  - probably have never written a simulation
- Therefore, we developed the `protsim` framework for you, that already offers you the overall architecture of the simulation programm, as well as some already written support functions.
- Thus, you can concentrate on the interesting stuff by extending the `protsim` framework
- The following slides will give you a short overview over `protsim`
- Detailed information on the available classes, their attributes and methods can be found in the online API documentation that can be generated from the sources
- You are encourage to read the sources to learn from them

Network



- A network is set of connected `NetworkNodes`
- Although OMNeT++ connections are uni-directional we usually use them in pairs to imitate bi-directional links
- The connections may be attached propagation delay, throughput and errors
- A network is defined in a ned file

Network Node



- Each `NetworkNode` is identified by a node address (type `NodeAddressT` which is defined as a `long`)
- A `NetworkNode` consists of a `NetworkStack` and multiple instances of subclasses of `Application`
- Network interfaces of the node are represented by the gate arrays `netOut[]` and `netIn[]` which are connected to the `NetworkStack`

The Network Stack

- The network stack is responsible for forwarding packets between network interfaces and applications
- It contains
  - a table of local applications that is automatically set up
  - a table of neighbor nodes and links to these nodes (automatically set up)
  - a forwarding table associating an outgoing link to each destination (empty by default, must be maintained by a routing daemon)
  - methods for maintaining the above tables
  - some support methods
  - stubs for the methods implementing the forwarding engine (it will be your task to fill them with life)
- For a detailed reference on the available methods and fields, please look into the online API documentation

Applications

- Each application is identified within a `NetworkNode` by its application address (type `ApplicationAddressT` which is defined as a `long`)
- Note: The application address is unique only within a node. To identify an application network-wide (e.g. as a destination of a packet) use a combination of node and application address.
- `Application` is only an abstract base class. Real applications must be derived from it.
- Each application possesses an `in` and an `out` gate connected to the local `NetworkStack`
- Routing daemons are also just applications
  - identified by the special address `ROUTING_DAEMON`
  - calls the appropriate methods of `NetworkStack` to configure the forwarding table
  - gets local messages if a link changes
  - at most one routing daemon per node possible

Messages

- Messages are used to carry application data or application-specific signaling (e.g. used by routing daemons)
- All messages in protsim are subclassed from `NetworkPacket`
- `NetworkPacket` contains the following informations:
  - Packet ID (can be used by the application)
  - Source address (node and application address)
  - Destination address (node and application)
  - An `alert` flag that tells the `NetworkStack` to send the message to the application on each hop, not only at the destination
  - The node address of the last hop
  - A time-to-live field counting the number of hops
- Application messages additionally contain application-specific fields

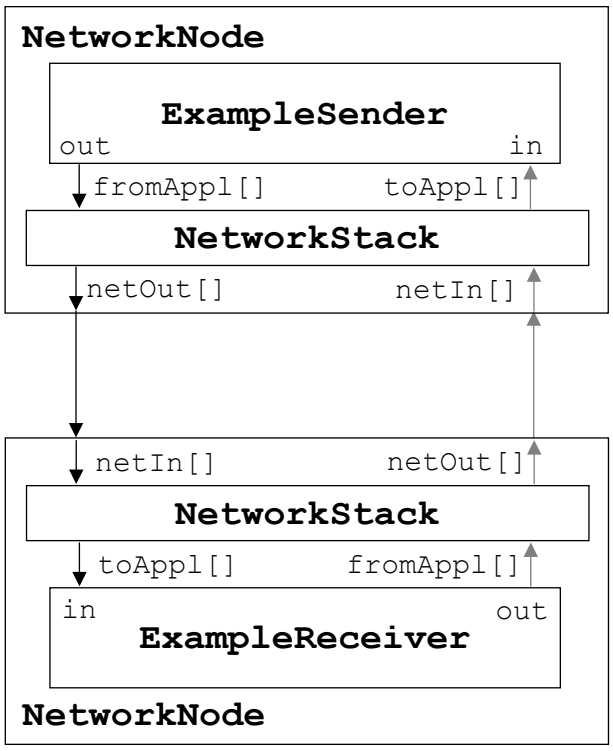
Topology

- Represents a network graph
- Consists of nodes and links connecting the nodes
- You can construct a topology in two ways:
  - Copy it from an OMNeT++ `cTopology` instance
  - Create it from scratch
  - The latter is not possible with `cTopology` (you will need this for the implementation of some routing algorithms, that is why `Topology` was developed)
- `Topology`, `TopologyNode` and `TopologyLink` offer methods to
  - iterate over nodes or get nodes by node address
  - add a node for a node address
  - iterate over links starting at a certain node or get links by destination node address
  - add a link between two nodes

Directory Hierarchy of protsim Framework

- `doc/api`
  - By pointing a browser (e.g. konqueror) to the `index.html` in this directory, you get the online API documentation of the framework.
  - **Please use it frequently!**
- `common`
  - Here you find the files for the `Application` base class and for the `NetworkStack`
- `messages`
  - Contains the `msg`-files for network packets
- `nodes`
  - Contains the `ned`-files for the compound modules of nodes (e.g. `NetworkNode`)
- `networks`
  - Contains the `ned`-files for network topologies
- `routing`
  - Files describing and implementing routing daemons
- `userapps`
  - Files describing and implementing user applications
- `support`
  - Support classes, such as `Topology`, `Parser`, `ForwardingTable` and files defining metrics

Example – Overview



- This picture shows a simple example consisting of
  - two directly connected network nodes
  - a simple sender application
  - a simple receiver application
  - two network stacks
- The direction from receiver to sender is not used
- The example is a bit simplified as compared to the real framework

Example – Simple Modules – NED Files

```
[ExampleSender.ned]
simple ExampleSender
parameters:
  applAddr: numeric;
gates:
  in: in;
  out: out;
endsimple;
```

```
[ExampleReceiver.ned]
simple ExampleReceiver
parameters:
  applAddr: numeric;
gates:
  in: in;
  out: out;
endsimple;
```

```
[NetworkStack.ned]
simple NetworkStack
    gates:
        in: netIn[], fromAppl[];
        out: netOut[], toAppl[];
endsimple;
```


Example – Sender Implementation

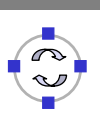
```
[ExampleSender.cc (includes omitted)]

class ExampleSender : public Application {
public:
  Module_Class_Members(ExampleSender,Application,16384);
protected:
  virtual void activity();
};

Define_Module(ExampleSender);

void ExampleSender::activity() {
  while (true) {
    NetworkPacket * packet = new NetworkPacket("Ping",NETWORK_PACKET);
    packet->setLength(8000);
    send(packet,"out");
    wait(1.0);
  }
}
```


Example – Receiver Implementation

```
[ExampleReceiver.cc]
#include <omnetpp.h>
#include "protsim_defines.h"
#include "Application.h"

class ExampleReceiver : public Application {
public:
  Module_Class_Members(ExampleReceiver,Application,0);
protected:
  virtual void handleMessage(cMessage * msg);
};

Define_Module(ExampleReceiver);

void ExampleReceiver::handleMessage(cMessage * msg) {
  ev << "Message " << msg->name() << " received\n";
  delete msg;
}
```


Example – NetworkStack Implementation

```
[NetworkStack.h / NetworkStack.cc]
class NetworkStack : public cSimpleModule {
public:
  NetworkStack(const char *name, cModule *parentmod)
    : cSimpleModule(name, parentmod, 0) {}
protected:
  virtual void handleMessage(cMessage * msg);
};

Define_Module(NetworkStack);

void NetworkStack::handleMessage(cMessage * msg) {
  if (msg->arrivalGate() &&
      strcmp(msg->arrivalGate()->name(),"fromAppl")==0) {
    send(msg,"netOut",0);
  }
  else {
    send(msg,"toAppl",0);
  }
}
```

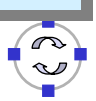



Example – ExampleNode

```
[ExampleNode.ned (imports and ExampleReceiverNode omitted)]

module ExampleSenderNode
  parameters:
    nodeAddr: numeric;
  gates:
    in: netIn[];
    out: netOut[];
  submodules:
    networkStack: NetworkStack
      gatesizes:
        netIn[sizeof(netIn)],
        netOut[sizeof(netOut)],
        fromAppl[1],
        toAppl[1];
    app: ExampleSender
      parameters:
        applAddr = 1;
  connections:
    netIn[0] --> networkStack.netIn[0];
    netOut[0] <-- networkStack.netOut[0];
    networkStack.toAppl[0] --> app.in;
    networkStack.fromAppl[0] <-- app.out;
endmodule ExampleSenderNode;
```


Example – ExampleNetwork

```
[ExampleNetwork.ned]
import "../nodes/ExampleNode.ned";

module ExampleNetworkModule
  submodules:
    Sender: ExampleSenderNode;
      parameters:
        nodeAddr = 1;
      gatesizes:
        netIn[1],
        netOut[1];
    Receiver: ExampleReceiverNode;
      parameters:
        nodeAddr = 2;
      gatesizes:
        netIn[1],
        netOut[1];
  connections:
    Sender.netOut[0] --> delay 0.001 datarate 1E6 --> Receiver.netIn[0];
    Sender.netIn[0] <-- delay 0.001 datarate 1E6 <-- Receiver.netOut[0];
endmodule;

network ExampleNetwork : ExampleNetworkModule
endnetwork;
```

Example – Initialization File (omnetpp.ini)

```
include general.ini
include seeds.ini

[General]
network = ExampleNetwork

[DisplayStrings]

[Parameters]
*.applAddr = -1;

[Run 1]
snapshot-file = protsim.sna
output-vector-file = protsim.vec
output-scalar-file = protsim.sca
```