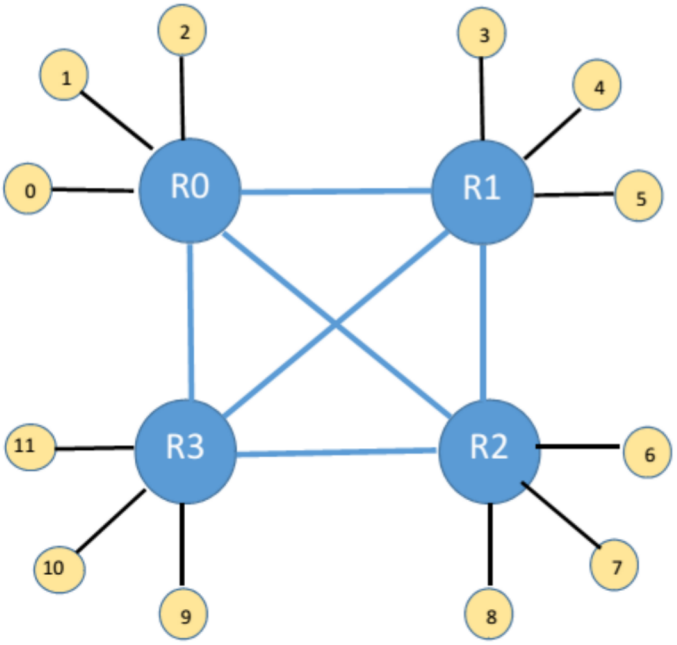
Incorporating a new topology in Booksim

Let’s implement this one:



Let’s call it *testnet.*

Files we need to work on to incorporate *testnet*:

- Create the topology files:
  - *testnet.cpp*
  - *testnet.hpp*
- Create a configuration file:
  - *Testnetconfig*
- Modify the existing network file:
  - *network.cpp*

testnet.hpp

```
#ifndef _TestNet_HPP_
#define _TestNet_HPP_

#include "network.hpp"
#include "routefunc.hpp"

class TestNet : public Network{

    int _a; //total # of routers
    int _p; //total # of processing nodes per router
    int _k; //total # of ports per router

    void _ComputeSize(const Configuration &config);
    void _BuildNet(const Configuration &config);

public:
    TestNet(const Configuration &config, const string &name);
    static void RegisterRoutingFunctions();
};


int testnet_port(int rID, int src, int dest);
void min_testnet(const Router *r, const Flit *f, int in_channel, OutputSet *outputs, bool inject);

#endif
```

the Constructor

```
TestNet :: TestNet(const Configuration &config, const string &name) : Network(config, name)
{
    //cout << "testnet constructor starts ..." << endl;

    _ComputeSize(config);
    _Alloc();
    _BuildNet(config);

    //cout << "testnet constructor ends ..." << endl;
}
```

_ComputeSize()

```
void TestNet :: _ComputeSize(const Configuration &config)
{
    //cout << "_ComputeSize starts ..." << endl;

    //hard-coding now, but can (and should) be fetched from the config file
    _a = 4; //total # of routers
    _p = 3; //total # of processing nodes per router

    _k = (_a - 1) + _p; //total # of ports per router


    //these three are variables declared in network.hpp and used in network.cpp
    _nodes = _a * _p;           //# of processing nodes
    _channels = _a * (_a-1);    //# of uni-directional link between the routers only
                                    //(not processing nodes)
    _size = _a + _a*_p;         //# of nodes, including routers and processing nodes

    gP_testnet = _p;            //global variables, needed for the routing functions
    gA_testnet = _a;

    //cout << "_ComputeSize ends ..." << endl;
}
```

_Alloc()

- Defined in *network.cpp*
- uses data structures declared in *network.hpp*

```
class Network : public TimedModule {
protected:

  int _size;
  int _nodes;
  int _channels;
  int _classes;

  vector<Router *> _routers;

  vector<FlitChannel *> _inject;
  vector<CreditChannel *> _inject_cred;

  vector<FlitChannel *> _eject;
  vector<CreditChannel *> _eject_cred;

  vector<FlitChannel *> _chan;
  vector<CreditChannel *> _chan_cred;
```

_Alloc()

```
void Network::_Alloc( )
{
  ...

  _routers.resize(_size);

  ...

  _inject.resize(_nodes);
  _inject_cred.resize(_nodes);

  for ( int s = 0; s < _nodes; ++s ) {

    ...
    _inject[s] = new FlitChannel(this, name.str(), _classes);
    _inject[s]->SetSource(NULL, s);

    ...
    _inject_cred[s] = new CreditChannel(this, name.str());

    ...
  }

  ...

  _chan.resize(_channels);
  _chan_cred.resize(_channels);
  for ( int c = 0; c < _channels; ++c ) {

    ...
    _chan[c] = new FlitChannel(this, name.str(), _classes);
    ...

    }
}
```

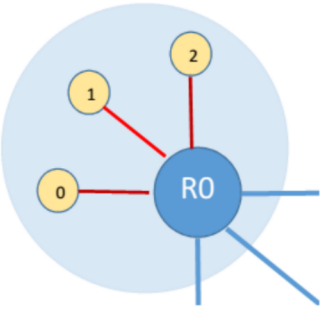
_BuildNet()

- Initialize router objects

```
void TestNet :: _BuildNet (const Configuration &config)
{
    //for every router
        //build the router object
        //add the links to the processing nodes
        //add the links to the other routers

    ostringstream router_name;
    int node;
    int c, cnt;
    int port_to_routers = _a - 1;

    for(node = 0; node < _a; node++){
        //create router
        router_name << "router";
        router_name << "_" << node;
        _routers[node] = Router::NewRouter( config, this, router_name.str( ),
                    node, _k, _k );     //_k's are the # of input and output ports, respectively
        _timed_modules.push_back(_routers[node]);
        router_name.str("");
```

_BuildNet()

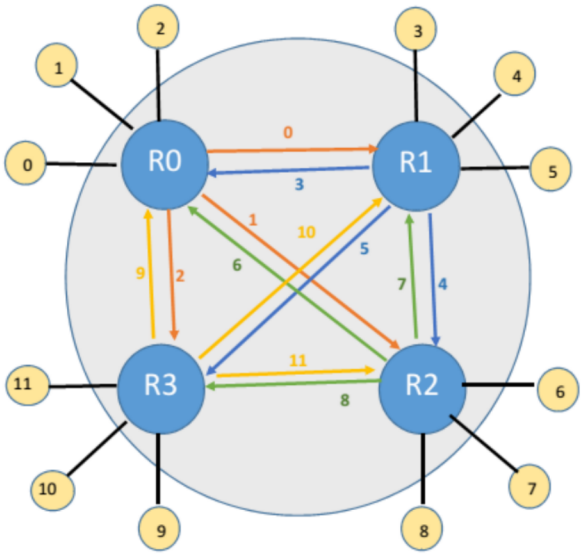
- Add links to the processing nodes




```
//add input and output channels to processing nodes
for (cnt = 0; cnt < _p; cnt++ ) {
    c = _p * node +  cnt;   //for router 0, c is 0,1,2
                            //for router 1, c is 3,4,5
                            // and so on.
    _routers[node]->AddInputChannel( _inject[c], _inject_cred[c] );
                            // _inject and _inject_cred arrays are initialized in _alloc()

}


for (cnt = 0; cnt < _p; cnt++ ) {
    c = _p * node +  cnt;
    _routers[node]->AddOutputChannel( _eject[c], _eject_cred[c] );

}
```

_BuildNet()

- Add links to other routers




```
//add output and input channels to other routers
for (cnt = 0; cnt < _a-1 ; cnt++){
    c = node * port_to_routers + cnt;
    //cout << "c: " << c << endl;
    _routers[node]->AddOutputChannel(_chan[c],_chan_cred[c]);
}
                    /* So,  0 -> 0,1,2
                            1 -> 3,4,5
                            2 -> 6,7,8
                            3 -> 9,10,11 */


for(cnt = 0; cnt < _a; cnt++){
    if (cnt == node){
        continue;
        //do nothing
    }
    else if (cnt < node){
        c = cnt * port_to_routers -1 + node;
    }else if (cnt > node){
        c = cnt * port_to_routers + node;
    }
    //cout << "c: " << c << endl;
    _routers[node] -> AddInputChannel(_chan[c],_chan_cred[c]);

}
                    /* So,  0 ->   3,6,9
                            1 -> 0,  7,10
                            2 -> 1,4  11
                            3 -> 2,5,8    */
}
```

RegisterRoutingFunctions()

- As the name suggests

```
void TestNet::RegisterRoutingFunctions(){

    gRoutingFunctionMap["min_testnet"] = &min_testnet;
}
```

- gRouingFunctionMap is declared in *routefunc.cpp:*

map<string, tRoutingFunction> gRoutingFunctionMap;

Routing:

- For a particular router, flit and input channel, need to provide:
  - A output port
  - A output VC
- We wrote two functions:
  - *int testnet_port(int rID, int src, int dest)*
  - *void min_testnet(const Router *r, const Flit *f, int in_channel, OutputSet *outputs, bool inject)*

Routing:

- To find the right port:

```
int testnet_port(int rID, int src, int dest){
    int dst_router;
    int out_port;

    dst_router = dest / gP_testnet;

    //check if dest is under the same router as the current one
    if(rID == dst_router){
        //find the port that goes to the destination processing node
        out_port = dest % gP_testnet;
    }

    //else, dest in a different router that the current one
    else{
        if(dst_router < rID){
            out_port = gP_testnet + dst_router;
                //example: current node: 2, dest 1. Then out_port = gP_testnet + 1 = 4
                //[port 0,1,2 goes to processing nodes, port 3, 4 to routers 0,1. Port 5 to router
3.];
        }else{
            out_port = gP_testnet + dst_router - 1;
        }
    }

    return out_port;
}
```

Routing:

- Minimal routing function

```
void min_testnet( const Router *r, const Flit *f, int in_channel,
                  OutputSet *outputs, bool inject )
{
    int debug = f-> watch;

    outputs -> Clear();

    if(inject){
        int inject_vc = RandomInt(gNumVCs - 1);
                    //int gNumVCs; declared in routefunc.cpp
                    //value provided by the user in the config file, through
                        "num_vcs"
        outputs->AddRange(-1,inject_vc,inject_vc);
        return;
    }

    int rID = r->GetID();
```

Routing

- Minimal routing function (contd.)

```
    int rID = r->GetID();

    int out_port = -1;
    int out_vc = 0;

    //cases:
        //1. It can be source node, assign it to VC 0
        //2. It can be dest node, assign it to VC 1

    if (in_channel < gP_testnet){
        out_vc = 0;
    }else{
        out_vc = 1;
    }

    out_port = testnet_port(rID,f->src,f->dest);

    outputs->AddRange( out_port, out_vc, out_vc );

    if (debug)
    *gWatchOut << GetSimTime() << " | " << r->FullName() << " | "
           << " through output port : " << out_port
           << " out vc: " << out_vc << endl;
}
```

Integrating testnet with existing topologies:

- In *network.cpp*

```
#include "kncube.hpp"
#include "fly.hpp"
#include "cmesh.hpp"
#include "flatfly_onchip.hpp"
#include "qtree.hpp"
#include "tree4.hpp"
#include "fattree.hpp"
#include "anynet.hpp"
#include "dragonfly.hpp"
#include "testNet.hpp"


Network::Network( const Configur
  TimedModule( 0, name )
{
```

Integrating testnet with existing topologies:

- In *network.cpp*

```
Network * Network::New(const Configuration & config, const string & name)
{
  const string topo = config.GetStr( "topology" );
  Network * n = NULL;
  if ( topo == "torus" ) {
    KNCube::RegisterRoutingFunctions() ;
    n = new KNCube( config, name, false );
  } else if ( topo == "mesh" ) {
    KNCube::RegisterRoutingFunctions() ;
    n = new KNCube( config, name, true );
  }

  ...

  } else if (topo == "testnet"){
    TestNet::RegisterRoutingFunctions();
    n = new TestNet(config,name);
  }
  else {
    cerr << "Unknown topology: " << topo << endl;
  }

  /*legacy code that insert random faults in the networks
   *not sure how to use this
   */
  if ( n && ( config.GetInt( "link_failures" ) > 0 ) ) {
    n->InsertRandomFaults( config );
  }
  return n;
}
```

Configuration file:

- Used to provide simulation parameters by the user (including the topology and routing function to be used.
- We created one called *testnetconfig*

```
// Topology
topology = testnet;

//a = 4;
//p = 3;

// Routing
routing_function = min;
// Flow control
num_vcs = 2;
// Traffic
traffic = uniform;
injection_rate = 0.25;
```

Build and Run:

- *make* (and pray)
- *./booksim examples/testnetconfig*

One last thing …

To customize network parameters:

- Pass the parameters in the config file

```
a  = 8;
p  = 4;
```

- Update the *booksim_config.cpp* to get those values

```
_int_map["a"] = 0;
_int_map["p"] = 0;
```

- Use the extracted values in the topology code

```
_a = config.GetInt( "a" );
_p = config.GetInt( "p" );
```