Interconnection Networks
ECE 8823 A / CS 8803 – ICN
Spring 2017
Lab 3: Deadlock Avoidance

George P. Burdell attended the Routing lecture of Interconnection Networks and felt that a minimal deterministic XY routing is inefficient as the Mesh topology loses its path diversity. He implemented a new minimal *oblivious* routing algorithm that randomly forwards a flit out of one of the ports along its minimal path at each router. Recall that an oblivious routing algorithm randomly chooses one out of available minimal output links (without taking any congestion into account).

But once he started running the network, George noticed that he is unable to inject any new messages into the network after a while and some of the old messages have not been received despite waiting for thousands of cycles. George missed the next couple of lectures of Interconnection Networks where techniques to solve this problem were discussed, because he was cross-registered for multiple classes at the same time[1]!. Your goal is to fix George's network.

Step 0:
Update your copy of gem5 which contains George's network.
```
hg pull -u
```

[You can also clone a fresh copy if you wish]
```
hg clone /nethome/tkrishna3/teaching/simulators/gem5/repo/gem5
```

Now build the simulator.
THIS NEEDS TO BE DONE EVERYTIME YOU CHANGE CODE.
```
./my_scripts/build_Garnet_standalone.sh
```

On a fresh login to the machines, don't forget to source the environment file before running
```
source <path_to_gem5>/my_scripts/set_env.sh
```

Step 1:
Run Command:
```
./build/Garnet_standalone/gem5.debug configs/example/garnet_synth_traffic.py \
--network=garnet2.0 \
--num-cpus=64 \
--num-dirs=64 \
--topology=Mesh \
--mesh-rows=8 \
--sim-cycles=20000 \
--synthetic=uniform_random \
--vcs-per-vnet=4 \
--inj-vnet=0 \
--injectionrate=0.02 \
--routing-algorithm=random
```

The parameters in blue are what you will be varying at various points in this lab.

[1] https://en.wikipedia.org/wiki/George_P._Burdell

Note:
- You will be running experiments on a 64-core system in this lab, with 4 VCs (per VNet).
- Instead of plotting latency vs. injection rate, you will plot **reception rate vs. injection rate**.
  The reception rate (packets/node/cycle) is **total_packets_received/num-cpus/sim-cycles**

*Beyond a certain injection rate, the reception rate will saturate. This is **peak throughput**.*

Add the following to a Report.doc/pdf:

Part I: Deadlock Detection (1 point)

Plot the reception rate vs. injection rate for *uniform random* traffic for `XY` and `random` routing on the same graph.
(`--routing-algorithm=xy` / `--routing-algorithm=random`)

For random routing, once the network deadlocks, you will see a deadlock assertion failure.
Plot all data points up to this point.

Write down the injection rate at which the network deadlocks.

Part II. Deadlock Avoidance using Turn Model (2 points)

Avoid the deadlock by **implementing *any deadlock-free turn model routing scheme of your choice*** in Garnet.
The routing code is here: `src/mem/ruby/network/garnet2.0/RoutingUnit.cc`
Implement the function `outportComputeTurnModel()`
See how `outportComputeXY()` and `outportComputeRandom()` are implemented in that file for reference.
You can invoke this by setting `--routing-algorithm=turn_model`

Write a few lines about which routing scheme you implemented, and copy-paste your `outportComputeTurnModel()` function implementation into the report.

You will only get points if running the network with `--routing-algorithm=turn_model` does not deadlock for any traffic pattern.
Run your design with different traffic patterns to make sure it is robust.

III. Deadlock Avoidance using Escape VC (5 points)

Avoid the deadlock by **implementing an escape-*VC based deadlock avoidance scheme of your choice*** from the ones discussed in class (or any other) in Garnet. In other words, you will run the simulation with George's random routing algorithm (`--routing-algorithm=random`), but avoid deadlocks by controlling which output VC a flit can go into.
Hint: Look at the `has_free_vc()` and `select_free_vc()` functions in `OutputUnit.cc`.
These functions are called from inside `SwitchAllocator.cc`

There are no guidelines on where to add code. In addition to `has_free_vc()` and `select_free_vc()`, depending on your design, you may restrict the routing based on which VC the flit arrived on. You are free to add any code/functions/header files. The only requirement is that the routing algorithm code inside the outportComputeRandom() function should not be changed.

Write a few lines about which VC scheme you implemented, and copy-paste the relevant functions you changed/added in the code into the report.

You will only get points if running the network with `--routing-algorithm=random` does not deadlock for any traffic pattern.
Run your design with different traffic patterns to make sure it is robust.

Part IV. Analysis (2 points)

Plot the reception rate graph for the following configurations:
Graph 1
Traffic: `uniform_random`
Routing Algorithms: XY, Turn Model, Random+EscapeVC.

Graph 2
Traffic: `transpose`
Routing Algorithms: XY, Turn Model, Random+EscapeVC.

In each case, report which traffic pattern has the highest throughput.

Extra Credit (1 point)
Does deterministic XY always perform worse than the oblivious routing algorithms as George believed? If no, can you construct a toy example showing a case where XY might lead to lower congestion than a deadlock-free random routing algorithm?
Add this to your report.

What to Submit:

Report.pdf
garnet2.0.tar.gz (a copy of your code)