IRFlexSim0.5 User Guide

This user guide describes the capability of the IRFlexSim0.5 flit-level network simulator, how to use the simulator, and how to interpret the outputs.

1 Introduction

IRFlexSim0.5 is a simulator for flit-level simulation of networks with arbitrary topologies. IRFlexSim0.5 is based upon FlexSim1.2 developed by SMART Interconnects Group at University of Southern California. FlexSim1.2 is an extensively enhance version of FlitSim originally developed at Georgia Tech. The main purpose of IRFlexSim0.5 is for studying the performance of various routing algorithms over networks with arbitrary topologies and different parameters such as number of virtual channels, different traffic patterns, and different switch characteristics (delays and buffer sizes). A user specifies the various options with input parameters. IRFlexSim0.5 simulates the specified network and collects the statistics on performance such as latency and throughput and reports it to standard output.

IRFlexSim0.5 has a built-in network generator that constructs arbitrary topologies with a specified number of switches and links. It also supports automatic generation of the routing table based on a number of different routing algorithms. The topology and routing table is written into files and can be used as input in later runs for producing the same network and routing table. The user can also supply their network and routing table via input files.

IRFlexSim0.5 is also installed with a deadlock detection module which can detect true deadlocks based on cyclic dependencies between virtual channels. The virtual channel dependencies are collected periodically during the simulation. The deadlock detection module builds a dependency graph and locates the knotted cycles, which indicate true deadlocks. The deadlock detection module is intended for characterization of deadlocks (deadlock frequency, size, etc.).

The rest of the User Guide is organized as follows. Section 2 describes the basic capability of the simulator, including the network model, the switch model, the routing functions and the message generations. Then, each input parameter is explained with examples in Section 3, followed by sample outputs and the description of their format in Section 4. Finally, limitations of the simulator are given in Section 5.

2 Capability of IRFlexSim0.5

IRFlexSim0.5 is a flit-level network simulator capable of simulating networks with arbitrary topologies with choices in routing function and traffic pattern. There is also a deadlock detection module which can detect true deadlocks.

2.1 Network

The network model of IRFlexSim0.5 consists of switches connected in an arbitrary topology. Each switch is attached to one end-node which has one or more injection channels to the switch. Each end-node is associated with message queues which receive generated messages either from a network trace or from a network traffic synthesizer.

IRFlexSim0.5 has a built-in network generator that generates an arbitrary topology with a specified number of switches (IRNUMOFNODES) and number of links (IRNUMOFLINKS). You may also control the minimum and maximum node degree of the switches (IRMINDEGREE and IRMAXDEGREE) and whether multiple links are allowed between switches (IRDUPLINKS). Topology can also be specified manually using appropriate input files. This is further discussed in Section 3.

Static faulty physical or virtual links are supported. The number of faults is specified by the parameters FAULTS and PFAULTS for virtual channel and physical channel, respectively. The faults are chosen randomly at the beginning of the simulation.

2.2 Switch Architecture

Switches are connected to each other with physical links, which can be half-duplex or full-duplex. The physical channels may be partitioned into virtual channels (see VIRTS and USEVIRTS), each associated with its own message buffer. The switch architecture assumes input buffering, which means the message buffers are associated only with input virtual channels. The size of message buffers is set with the parameter BUFFERS.

IRFlexSim0.5 supports wormhole switching only.

Various delays regarding links and routers can be set. This includes the routing delay (RDELAY), cut-through delay (DDELAY), switch-to-switch delay (LDELAY), and hand-shaking (control signal) delay (ADELAY). The link arbitration delay for half-duplex channels (LARB) can also be set.

2.3 Routing and Channel Selection Function

IRFlexSim0.5 implements four routing functions: deterministic, minimal true fully adaptive, Up*/Down* and Duato's enhancement to Up*/Down* routing. The first two routing functions support deadlock detection and deadlock recovery. The latter two are based on deadlock avoidance. The routing function is set with the parameter PROTO.

IRFlexSim0.5 only supports random channel selection.

2.4 Message Generation

IRFlexSim0.5 supports synthesized traffic patterns or trace-driven traffic. The synthesized traffic patterns controlled by parameter DIST include uniform random, bit-reversal, matrix transpose, perfect shuffle, flipped bits, and hotspots. For hotspot traffic, the number of hotspots (HSPOTS), placement of hotspots (HSPLACE) and the percentage of messages that are directed to the hotspots (HSPERCENT) can be specified. Messages can be of uniform length (MSGL) or hybrid length (see HYBRID, LMSGL and LMSGPCT). The load rate is specified by the average injection period (PER) and the injection multiplier (INJECT). How to determine suitable PER values is further explained in Section 0.

Details of trace-driven traffic will be specified later.

2.5 Deadlock Detection

IRFlexSim0.5 supports deadlock detection by constructing the channel dependency graph. The cycle and deadlock detection module is turned on with the parameters (CD and DD).

Since the cycle and deadlock detection is computation intensive, the frequency of which the detection modules are run can be controlled (CDFREQ and DDFREQ) to speed up the simulation.

3 Input Parameters

The command for IRFlexSim0.5 is `irsim`. Input parameters are specified at the command line. The options are all in the form:

<keyword>=<value>

Options are separated by space. The general command line to run the simulator is:

irsim <keyword1>=<value1> <keyword2>=<value2> …

The following are explanations and examples of the input parameters.

3.1 Network

The network topology can either be automatically generated by the simulator or manually configured. Each switch is assumed to connect one end-node.

3.1.1 Manually Defining a Topology

By setting the option IRAUTOGEN=0, the topology of the network can be specified by an input file. The file must be named "top.dat" and it should have the following format:

```
IRNUMOFNODES=5
IRNUMOFLINKS=4
0-1
1-2
1-3
1-4
```

The first two lines of the file specify the number of nodes and the number of links in the network, respectively. The subsequent lines specify the physical link connections. Each line is of the form <node>-<node>. Links are bidirectional, therefore, the order in specifying the nodes are unimportant. For example, 0-1 is the same as 1-0. The topology above is shown in Figure 1.

The links are locally labeled by the order in which they appear in the topology file. This number assigned for each switch is called the connection number of the link. For example, the link 0-1, 1-2, 1-3, 1-4 are assigned connection numbers of 0, 1, 2 and 3 at Switch 1, respectively. This connection number is used for specifying the routing table. See also Section 3.3.

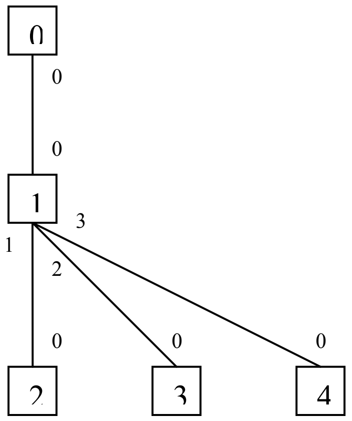

Figure 1 A sample topology. The small numbers are the connection numbers.

3.1.2 Automatic Generation of Topology

The built-in network generator is turned on by the option IRAUTOGEN=1. The following options are used for configuring the network generator:

• IRNUMOFNODES=n	(default=32) Number of switches in the network.
• IRNUMOFLINKS=n	(default=64) Number of bidirectional links in the network.
• IRMINDEGREE=n	(default=1) Minimum degree of a switch.
• IRMAXDEGREE=n	(default=8) Maximum degree of a switch.
• IRDUPLINKS=n	(default=0) Allow (1) or disallow (0) multiple links between node pairs.

IRFlexSim0.5 will automatically generate a connect network topology consisting of IRNUMOFNODES switches and IRNUMOFLINKS bidirectional physical links, which satisfies the minimum and maximum degree constraints. Multiple links between switches may be disabled by the IRDUPLINKS option. IRFlexSim0.5 writes the generated topology to a file named "top.out", which has the same format as the input file described. This file can be used as the input file to simulate the same topology (probably with other parameters) by renaming it to "top.dat".

Other options for changing the network:

• FAULTS=n	(default=0) Number of static virtual channel faults to be placed randomly in the network.

• PFAULTS=0	(default=0) Number of static physical channel faults to be placed randomly in the network.

Note that there is no option for setting whether there are wraparound channels or not. The wraparound channels are always present. It is up to the routing function to restrict the use of wraparound channels to simulate a mesh network with no wraparounds.

3.2 Switch Architecture

These options set the switch architecture. Usually the default values are used for all the options except VIRTS and USEVIRTS.

• PLINK=n	(default=1) Number of physical links that connects the end-node to the switch.
• SRCQ=n	(default=8) Number of message buffers in the inject queue. When the queue is full, newly generated messages are discarded.
• HALF=<option>	(default=0) 0=full duplex, 1=half duplex; this option controls whether the links between the switches are in full duplex mode or half duplex mode. In half duplex mode, link arbitration (see LARB) takes time to switch the link's direction.
• BUFFERS=n	(default=8) Buffer depth of each input virtual channel in number of flits. This value must be greater than 1.
• RDELAY=n	(default=1) Routing header intra-node delay, i.e., number of cycles for the header to be routed and arbitrated to the output channel.
• ADELAY=n	(default=1) Acknowledgement flit intra-node delay.
• DDELAY=n	(default=1) Data flit intra-node delay, i.e., number of cycles for a data flit to be transferred from the input channel to output channel.
• LARB=n	(default=0) Link arbitration time for a half-duplex physical link if the direction has to be reversed.
• VIRTS=n	(default=2) Number of virtual channels per physical channel. The value must be a power of 2. To simulate networks with a number of virtual channels not equal to a power of 2, VIRTS should be set to the next power of 2 that is greater than the desired number of virtual channels, and set the

USEVIRTS parameter to use the desired number of virtual channels. Example: VIRTS=4 USEVIRTS=3, if 3 virtual channels are used.

- USEVIRTS (default=2) Number of virtual channels that is used by the routing algorithm. Specified only when number of virtual channels is not a power of two. USEVIRTS is set to VIRTS if VIRTS is specified at the command line. Therefore, VIRTS must precede USEVIRTS. See VIRTS and PROTO.

3.3 Routing and Selection Function

Routing options:

Only wormhole switching and random channel selection is supported. All routing protocols are implemented by a destination-based routing table. Currently, the simulator supports manual definition of a deterministic routing table.

- IRAUTOGENTABLE=n (default=1) Turn ON (1) or OFF (0) automatic generation of the routing table

- PROTO=<routing protocol> (default = j)

  i = Static/deterministic routing based on deadlock-recovery
  j = Minimal True Fully-Adaptive Routing (MTFA) based on deadlock-recovery
  u = Up*/Down* routing
  d = Up*/Down* routing with Duato's enhancement

  For static/deterministic routing based on deadlock-recovery, a route is chosen when the network is created between each pair of nodes. The route remains unchanged throughout the simulation. Deadlock freedom is not guaranteed in choosing the routes. Deadlocks are detected either by the built-in deadlock mechanism or a timeout mechanism. Messages suspected to be deadlocked are removed.

  For minimal true fully-adaptive routing based on deadlock-recovery uses the same deadlock handling mechanism as described above.

  For Up*/Down* routing, since only destination-based routing table is supported, each message is tagged with the up/down direction at which the message is traveling and only the routes with the allowed direction is chosen for the message.

The routing table for the deterministic routing algorithm can be specified by an input file "route.dat". The format is as follows:

```
TABLESIZE=5
0 0 0 0 0
0 0 1 2 3
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

The first line specifies the number of nodes in the network. It must match the network definition. The subsequent lines are routing table entries for each switch in node order, i.e., the first row is for Switch 0, second row is for Switch 1, etc. Each row contains the routing table entries for destinations from Switch 0 to Switch N, respectively. Each entry is specified by its connection number (see Section 3.1) which is a locally assigned number for the links incident to that switch.

3.4 Message Generation

• MSGL=n	Message length in number of flits.
• INJECT=n	Injection multiplier (rarely used)
• PER=n	Average injection period. For all message patterns, messages are generated by the Poisson distribution with average injection period of PER.
• TRACE=<trace file>	Turns on trace driven simulation and specifies the trace file.

- DIST=<communication pattern>

  (default = 0)
  0 = destinations are selected uniformly and randomly from entire network
  -100 = bit reversal
  -900 = matrix transpose
  -999 = perfect shuffle
  -1000 = flip bits
  DIST>0 random destinations with distance less than or equal to DIST
  DIST<0 random destinations with distance equal to |DIST|

3.4.1 Specifying the load rate

The load rate is specified in the simulator using the PER parameter. PER is the average injection period for each node. A load rate that is specified as a fraction of the saturation load rate should be converted to a corresponding average injection period. To do so, the saturation load rate must be clearly defined. Unfortunately, this is not true for most of the non-uniform traffic patterns. For non-uniform traffic patterns, there may be links that are more heavily loaded and, therefore, saturate far earlier than other links in the network. Therefore, it is not possible to saturate the entire network uniformly. Similarly, mesh network without wraparound links suffers a similar problem where the mesh center seems to saturate far earlier than links far away from the center. Furthermore, there are routing algorithms that support non-minimal routes, making the estimation of load generated difficult. In such cases, the saturation load is not well-defined and we can only estimate the appropriate PER that should be used for a simulation run. In the ideal case of uniform random traffic on a network, the PER value can be calculated by:

$$\begin{aligned} &(\#\text{ of nodes}) \times (\text{message length} / \text{PER}) \times (\text{average routing distance}) \\ = \ &(\text{load rate}) \times (\#\text{ of channels}) \end{aligned}$$

The idea is to utilize all the links fully. Every node is occupying the injection channel for a (message length / PER) fraction of the time, assuming each flit takes one cycle to transfer from the end-node to the switch. Each message will occupy a number of virtual channels in the network. On average, that number will be equal to the average number of hops from the source to the destination, i.e., the average routing distance. Therefore, the left-hand-side of the equation represents the average number of channels that are utilized and that should equal the load rate multiplied by the total number of channels in the system.

A common estimation of PER value is to use diameter / 2 as an estimate of the average routing distance, which gives a relative load rate. The absolute load rate is not that important in the analysis of the performance because the final measure of performance is throughput, which would be in absolute value measured in terms of the number of flits per node per cycle. Also, one can always find out the exact load rate from the relative load rate by multiplying it by a factor (i.e., the average routing distance / PER).

Another way to estimate the load rate is to consider the amount of load passing through the bisection of the network, assuming the most heavily loaded links are located at the bisection. Approximately half of the loads with go through the bisection, therefore, the saturation load rate for uniform random traffic can be defined by 2B/N, where B is the bisection bandwidth, and N is the number of nodes (see the book Interconnection Networks – an Engineering Approach by José Duato, Sudhakar Yalamanchili and Lionel Ni). The actual loads that go across the bisection are slightly more than half in reality, since an end-node never sends a message to itself.

For the cases where the load rate cannot be calculated directly, it can be estimated using the above methods. But the simulation results should be interpreted carefully.

3.5 Deadlock Detection

• CD=n	(default=0) Turn ON (n=1) or Turn OFF (n=0) cycle detection.
• CDFREQ=n	(default=25) Number of cycles over which cycle detection is run.
• DD=n	(default=0) Turn ON (n=1) or Turn OFF (n=0) deadlock detection.
• DDFREQ=n	(default=100) Number of cycles over which deadlock detection is run.

3.6 Simulation

These are essential parameters to control the simulator such as duration of the simulation runs, statististics, etc.

• DUR=n	(default=100) Number of cycles over which deadlock detection is run.
• TRANS=n	(default=1) Turn ON (n=1) or OFF (n=0) transient statistics.
• SLOW=n	(default=0) Turn ON (n=1) or OFF (n=0) a one-second delay between flit movement stages, which may help debugging.
• DEBUG=n	(default=0) Set debug level. (Details later)

4 Output

The following is an excerpt of a sample output. The command line for the simulation is:

```
irsim IRAUTOGEN=0 IRAUTOGENTABLE=1 PROTO=j VIRTS=2 MSGL=32 PER=256 DUR=20000
```

The first part is the network configuration which prints the options being specified or the default values being used by the simulator. The options are self-explanatory.

```
seed = -1587346005
Protocol: IRREGULAR Minimal True Fully Adaptive Routing (Deadlock Recovery)
Selection function: Channel Cycling (starting from randomly selected channel)
Comm: Wormhole routing
Buffer Depth: 8
Debug is 0
Number of misroutes allowed : 0
INJECT= 1, PER=256, DUR=20000
MSGL = 32  DIST = 0
Full-duplex
Generating network specified in input file
Nodes: 64, Links: 128
Maximum/Minimum node degree allowed: 1:8
Redundant links allowed: Yes
Virtual channels per network link: 2 (use: 2)
Physical injection/reception links per router node: 4
Virtual channels per injection/reception link: 2
Network created.
```

The second part is transient statistics, which list the network status every 500 cycles.

The information is specified in columns. Time is simulation time. Ave Msg Lat and Ave Msg SU stand for average message latency and average message setup time, respectively. Setup time of a message is the time for which the message header spends in routing from source to destination. Ave Setups is the average number of setups performed per cycle. TotDone is the accumulative number of messages delivered. NewDone is the number of messages delivered in the past 500 cycles. Inject is the accumulated number of messages injected. NewIn is the number of newly injected messages in the past 500 cycles. There are also a few columns dedicated to Disha routing. Disha-Rec is the number of messages delivered by Disha deadlock recovery procedure using deadlock buffers. DB Util is the utilization of the deadlock buffers. Finally, Tok Wait is the waiting time for getting the token before using the deadlock buffer.

The statistics are reset at time=10000 cycles by default. This is to discard the statistics in the initial warmup period. The network is assumed to reach a steady-state after 10000 cycles.

```
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====   500        54.42        20.25         2.30       55       55      60
60         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  1000        55.96        21.27         2.47      120       65     122
62         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  1500        55.86        21.70         2.33      164       44     169
47         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  2000        55.61        21.57         2.36      221       57     229
60         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  2500        55.19        21.09         2.31      282       61     291
62         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  3000        54.91        20.99         2.37      350       68     356
65         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  3500        55.48        21.21         2.39      411       61     414
58         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  4000        55.60        21.46         2.37      462       51     464
50         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  4500        55.47        21.51         2.31      503       41     510
46         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  5000        55.10        21.34         2.31      563       60     569
59         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  5500        55.11        21.38         2.29      615       52     617
48         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  6000        55.07        21.43         2.28      662       47     669
52         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  6500        55.19        21.42         2.28      724       62     726
57         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  7000        55.45        21.52         2.27      769       45     773
47         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
```

```
=====  7500        55.41        21.52         2.21      801       32     810
37         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  8000        55.34        21.48         2.22      856       55     867
57         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  8500        55.40        21.50         2.23      917       61     924
57         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  9000        55.31        21.50         2.21      964       47     972
48         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
=====  9500        55.28        21.44         2.17     1006       42    1007
35         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 10000        55.25        21.43         2.15     1045       39    1051
44         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 10500        56.15        21.07         2.21       52     -993      56
-995         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 11000        56.58        21.19         1.97       92       40     100
44         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 11500        57.28        21.19         1.98      145       53     147
47         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 12000        56.37        20.94         1.95      188       43     197
50         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 12500        55.36        20.81         1.88      232       44     237
40         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 13000        55.77        21.00         1.98      288       56     299
62         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 13500        55.53        21.03         2.04      354       66     357
58         0        0
       Time  Ave Msg Lat   Ave Msg SU   Ave Setups  TotDone  NewDone  Inject
NewIn  Disha-Rec  DB Util    Tok. Wait
===== 14000        55.93        21.04         2.04      405       51     409
52         0        0
```

The final part of the output is the overall statistics. Simulation time is the duration of the simulation in number of cycles. Total Done is the total number of messages sent. Throughput is in units of number of flits per node per cycle. Latency is in number of cycles. Latency std and Setup std are the standard deviation in latency and setup time, respectively.

Other statistics are minor or self-explanatory.

```
Simulation Time:          30000
Total Done:               12915
Throughput:               0.645718
Average message latency:  161.831192
```

```
Latency std:                47.159573
Average message setup:     41.204407
Setup std:                  43.700665
Average message queue time: 0.000000
Queue time std:             0.000000
Average concurrent circs:  -22.914204
Average concurrent setups: 26.357681
Average path length:       2.862101
Average misroutes taken:   0.000000
Total flits delivered:     838876
Total setups:              13111
Total messages done:       12917
Total messages killed:     97

Percentage of retry failures 0.000000
Mean time to inform src of dynamic fault: 0.000000
Control flit collision percentage: 0.000173
DISHA - Number of Token Captured: 0
```

5 Limitations

IRFlexSim0.5 does not support virtual cut-through switching.

IRFlexSim0.5 does not support pipelined switch architecture.

IRFlexSim0.5 only supports one end-node per switch.

IRFlexSim0.5 only supports random channel selection.