SIM - Ver. 1.13

What is SIM?

SIM is a slotted-time simulator for ATM switches, written in ANSI C. It was designed to simulate an ATM switch, as configured by the user.

Features of SIM

- **Slotted Time**: Instead of using discrete-event simulation (i.e. pulling out events from the 'event list' and thus advancing the simulation time to the point when the event occurs), SIM operates in time-slotted manner. That is, the whole system progresses one slot-time(ATM cell) at a time. So, the main loop of the simulation looks like this:

```
/**... some initialization steps... */
for(now = 0; now < simulationLength; now++){
   /* 1. Check for new arrivals to the switch */
   /* 2. Schedule transfer of cells from input of switch to output of switch */
   /* 3. Schedule departure of cells from switch. */


}
/* Print Results */
```

  Although this can be less efficient for low simulated loads, the use of slotted time enables the simulator to run much faster at high simulated loads
- **Modular architecture**: SIM has separate modules for different components of the switch: the switching fabric, queueing policies, scheduling algorithms, as well as traffic models. So, many different combination of these can be easily simulated, and it is also easy to modify one aspect of a switch (see Modifying SIM).
- **User-defined statistics:** SIM is capable of measuring numerous statistics about queues, but users can define which among all the possible statistics to be collected (more on this in Using SIM). This feature enables users to accelerate simulation considerably by 'turning off' some statistics.

Architecture of a Switch simulated by SIM

Figure 1 shows the generic architecture of an ATM switch and its input(Traffic) simulated by SIM. Each of the elements -- Traffic, InputAction, Fabric, Scheduling Algorithm and OutputAction -- may be selected by the user in a run-time configuration file.

Traffic

A Traffic source generates the input to the switch. It generates a stream of cells which satisfies a certain property (e.g. i.i.d. Bernoulli traffic with destinations uniformly distributed across all output ports), or provides the cell stream by reading 'trace files'. At each cell time, Traffic source checks if it is the right time to feed a cell to the switch, creates a cell accordingly, and the result is passed to the InputAction of the switch, which is then informed whether it has an arrival at that cell time or not. Several types of traffic source have been implemented:

- *bernoulli_iid_uniform*: Bernoulli arrivals, i.i.d., destinations uniformly distributed over all outputs, Unicast or Multicast,
- *bernoulli_iid_nonuniform*: Bernoulli arrivals, i.i.d., destinations defined by array of utilizations,
- *bursty*: Bursts of cells in busy-idle periods, destinations uniformly distributed cell by cell or burst-by-burst over all outputs
- *keepfull:* Always keep all input VOQs (Virtual Output Queues) full
- *trace*: Trace driven from a file of <time,vci> cells
- *tracePacket*: Trace driven from a file of <time,destination,length> packets
- *periodicTrace*: Trace driven from a file: one file for whole switch, connect to any one input

InputAction

When a cell arrives at the switch, it is accepted by the InputAction, which determines whether to accept the cell, which input queue to place the cell in and, if applicable, whether to send flow control cells back to the previous stage. The InputAction also examines the VCI to determine the output port and, if necessary, attaches a switch-specific header to the cell. This header is removed when the cell leaves the switch.

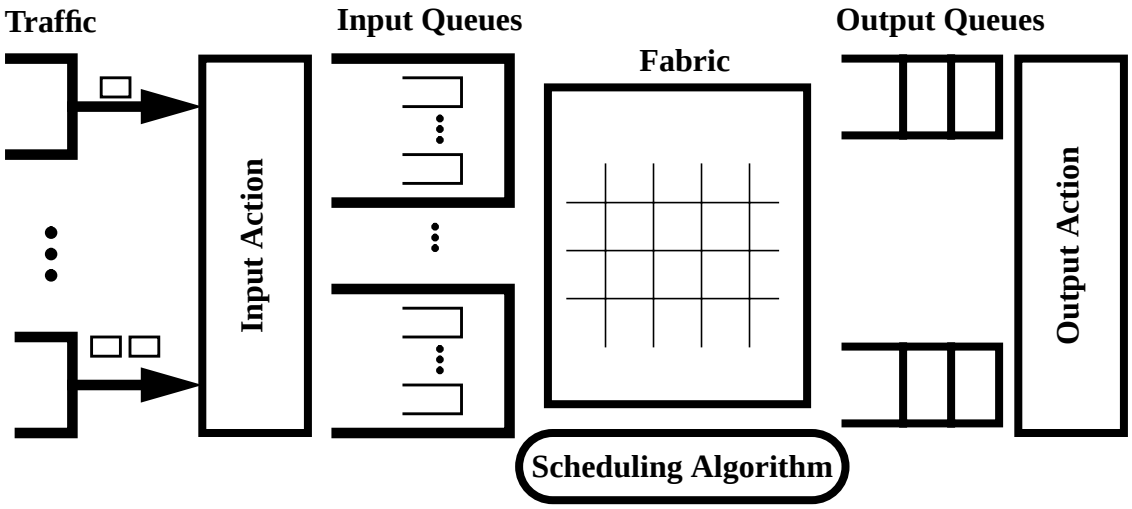

Figure 1: Architecture of a switch assumed by SIM

InputQueues

Each input of the switch maintains a separate FIFO for each output port, i.e. supports virtual output queueing, as well as a single multicast FIFO. For example, the default Input Action (defaultInputAction.c) will place a cell into its correct VOQ upon arrival. However, other Input Actions may be written that do other (perhaps crazy!) things. e.g. only place a cell in its VOQ if there is room, or if some other condition is met. The directory "INPUT_ACTIONS" contains some other examples of InputActions.

Fabric

The Fabric is the interconnection mechanism between the InputQueues and the OutputQueues. Examples include simple crossbar switches, fast busses running at multiples of the line rate and buffered Batcher-Banyan networks.

Scheduling Algorithm

The switch fabric may use a switch-specific scheduling algorithm to decide which cells to transfer in the next cell time. This is particularly important for input-buffered switches. Many scheduling algorithms have been implemented, including FIFO, Parallel Iterative Matching, iSLIP (which is called slip_prime.c in the source code), as well as a number of variations.

OutputQueues

The cells are transferred from the Fabric to the OutputQueue, which, by default, is a simple single FIFO.

OutputAction

The OutputAction removes cells from the output queue and delivers them onto the outgoing line. Currently, this actually leads to the destruction of the cell.

How to Use SIM

Where to get the source code

The source code is available on the SIM web page.

What to copy: Structure of Source Tree

The SIM source tree contains a top directory(main) and 5 subdirectories: ALGORITHMS, FABRIS, INPUTACTIONS, OUTPUTACTIONS, and TRAFFIC. The top directory also contains miscellaneous libraries, like 'lists.c'.
Each subdirectory corresponds to a block in Figure2 and contains one or more block types. Each block parses its own options from the command line or configuration file.

Modifying SIM

Although SIM has a variety of modules already in place, you might want to experiment with creating something new. Suppose that, inspired by PIM (Parallel Iterative Matching) by Anderson et. al., you've come up with a new scheduling algorithm, called PAM (Parallel *Arbitrary Matching*). SIM doesn't already have this (although some of the algorithms might perform in seemingly arbitary ways!), and here's what to do, in order to simulate an ATM switch with PAM scheduling

algorithm:

- Write a new.{c,h} file for PAM, i.e. main/ALGORITHMS/pam.{c.,h}
- Add pam.{c,h} to the ALGORITHMS/Makefile
- Add “pam” as well as some required descriptions, to the ‘table file’, namely, ALGORITHMS/algorithmTable.h.
- Go to main/ directory and re-compile, by ‘make all”.
- Now debug your code:-) (Note that there are function-specific debug flags defined in each sub-directory. e.g. take a look at the first line in ALGORITHMS/algorithm.h)

Running SIM

In order to run SIM, two things should be specified:

- Architecture of the switch
- The nature of the traffic sources

These can be specified either as command-line arguments or configuration file.

A. Running SIM from Command-line

This is useful when you simply want to try out SIM, or to debug the code, in case you’ve modified SIM. First, to see the usage information, try “sim -h”. (Note that each module reports its own usage information. Remember this when you write your own module). Below is the resulting output.

```
klamath:~/SIM/SIM/main>   sim -h
################################################
./sim -h
Date Wed Apr 29 14:18:28 1998
Machine klamath
################################################
usage: ./sim
GENERAL options:
    -h
    -n numswitches. Default: 1
    -i numInputs.   Default: 8
    -o numOutputs.  Default: 8
    -l simulationlength. Default: 100
    -r resetStatsTime.Default: simLength/10
    -p Indicate progress
    -f configFilename.Default: none

Traffic Models:
    -T trafficModel.  Default: bernoulli_iid_uniform
---------------------------------------
 ...... Options for each traffic model ...

Input Actions:
    -I inputAction. Default: defaultInputAction
---------------------------------------
... Options for each input action...

Switch scheduling algorithms:    -A switchAlgorithm.  Default: none
---------------------------------------
... Options for each scheduling algorithm ...:
```

```
Output Actions:
    -O outputAction.Default: defaultOutputAction
---------------------------------------
... Options for each output action...

Switch Fabrics:
    -F switchFabric.Default: crossbar
---------------------------------------
... Options for each fabric...
```

However, this doesn't mean that you have to specify every single parameter listed above. Since SIM is equipped with default settings, you could leave some of them unspecified. For instance, you could do:

```
klamath:~/SIM/SIM/main> sim -i 4 -o 4
```

From the output printed below, you can see what the default values were used, other than the number of input and output ports.

```
##################################################
sim -i 4 -o 4
Date Wed Apr 29 14:27:15 1998
Machine klamath
##################################################
PARAMETERS
----------
Numswitches     1
Simlength      100
Switch 0
    Numinputs    4
    Numoutputs   4
    InputAction  defaultInputAction
    Algorithm
    Fabric       crossbar
    OutputAction defaultOutputAction
    Max cells per input buffer: INFINITE
    Max cells per input FIFO: INFINITE
Switch algorithm "" set to null.
    Input 0 traffic bernoulli_iid_uniform
    Using default utilization
    Utilization 0.500000
    Input 1 traffic bernoulli_iid_uniform
    Using default utilization
    Utilization 0.500000
    Input 2 traffic bernoulli_iid_uniform
    Using default utilization
    Utilization 0.500000
    Input 3 traffic bernoulli_iid_uniform
    Using default utilization
    Utilization 0.500000
ResetStatsTime 50
Resetting stats for switch: 0 at time 0
Resetting stats for switch: 0 at time 50
```

```
================================================================
======================== RESULTS ===============================
================================================================
Simulation stopped at time: 100
Maximum memory used: 196608 bytes
Simulation runtime: 0 secs
 ......
```

B. Running SIM with a Configuration File.

The configuration files specifies all the three aspects listed above, in an ASCII text file, and it is required that a configuration file follow a specific format. Below is an example of configuration file for a single 8x8 input-buffered crossbar switch with the PIM scheduling algorithm. (There are more example configuration files in the TEST_INPS subdirectory). Inputs 4 through 7 have trace sources that read from an input file. Statistics are gathered: mean/S.D. for time average occupancy and cell latency for fifos 4 and 3 of InputQueue 0 and OutputQueues 0 and 3. Histograms are collected for the occupancy seen by departing cells from fifo 2 of InputQueue 3.

```
Numswitches 1
    Switch 0
    Numinputs    8
    Numoutputs   8
    InputAction defaultInputAction
    OutputAction defaultOutputAction
    Fabric       crossbar
    Algorithm pim -n4
    0 bernoulli_iid_uniform  -u 0.3
    1 bernoulli_iid_uniform  -u 0.6
    2 bernoulli_iid_uniform  -u 0.2
    3 bernoulli_iid_uniform  -u 0.4
    4 trace -f inputFile4
    5 trace -f inputFile5
    6 trace -f inputFile6
    7 trace -f inputFile7
    Stats
       Arrivals          (0,3) (0,4) 0 3
       Departures
       Latency          (0,3) (0,4) 0 3
       Occupancy
    Histograms
       Arrivals
       Departures    (3,2)
       Latency
       Occupancy
```

Frequently Asked Questions:

Q1. How do you generate a load-delay curve?
A1. To generate a load-delay curve you need to run SIM a number of times for each load, recording the latency over all cells as you go. For example if you want to cover the load range from 5% to 50% with 5% increments, then you should run SIM once for each of these loads (i.e. 10 different simulation runs). You can do this by controlling your traffic source to use one of these loads during each run (e.g. "bernoulli_iid_uniform -u 0.05" for 5%). After each run ends you should look at the end of the output stats file and record the "Total Latency over all cells". By doing this a number of times you will get a number of (load, latency) pairs which you can plot using gnuplot, or any other plotting tool, to get your load-delay curve. You can also write a simple script to do this for you.

Q2. How you can estimate the maximum throughput of a switch?
A2. Maximum throughput is defined as the maximum input load after which the switch becomes unstable. Instability means that the input load is higher than the throughput of the switch, hence queues will keep growing indefinitely. The trick here is detecting the input load at which the switch becomes unstable. The other important fact to notice is that as you get closer and closer to unstability, the simulation will need longer and longer running time to give correct results. So to get smoother load-delay curves, you need to run the simulation for a very long time (using the sim -l option). To get the maximum throughput you can plot the load-delay curve and look for the asymptote. But note that you might not be able to easily spot the asymptote in the simulation, this is because the queues are growing without bound, and an ever increasing number of cells may be missing from the average latency. Hence the latency becomes skewed downwards; this can be so severe as to mask the asymptote.

The best way to get the maximum throughput is as follows: for the load which you suspect might be the maximum throughout you can plot the average queue occupancy against time; If the queue is unstable, the average occupancy will grow linearly with time (note that if the average occupancy is growing more than linearly with time then this means that you are using an input load which is much higher than the maximum throughput). To watch this happen, you can modify SIM to print out the current average as it runs, and plot this. This way you can watch it make progress, and will see if the simulation is converging to the expected value. You may also use the optional SIMGRAPH animation tool to follow the progress of the simulation.

You can also tell that the simulation has gone unstable if the amount of allocated memory grows beyond the allowable amount. The allowable amount is set in sim.c to be 20MB (MAX_ALLOWED_MEMORY_USAGE).

Q3. Can you provide us with a configuration file that demonstrates the 58.6% maximum utilization of FIFO input-queued switch (with no VOQs).
A3. First note that the 58.6% result is asymptotic with N, the number of ports. For small N, the asymptote is above 60%. The following example has only 8 ports (N=8), so system will be stable to above 58.6%.

```
Numswitches 1
Switch 0
        Numinputs    8
        Numoutputs   8
        InputAction  defaultInputAction
        OutputAction defaultOutputAction
        Fabric       crossbar
        Algorithm       fifo
        0       bernoulli_iid_uniform  -u 0.586
        1       bernoulli_iid_uniform  -u 0.586
        2       bernoulli_iid_uniform  -u 0.586
        3       bernoulli_iid_uniform  -u 0.586
        4       bernoulli_iid_uniform  -u 0.586
        5       bernoulli_iid_uniform  -u 0.586
        6       bernoulli_iid_uniform  -u 0.586
        7       bernoulli_iid_uniform  -u 0.586
        Stats
                Arrivals
                Departures
                Latency
                Occupancy
        Histograms
                Arrivals
                Departures
                Latency
                Occupancy
```

Q4. What are the three numbers given at the end of the output stats file for the "Total Latency over all cells"?
A4. These three numbers are the (1) Estimate for the latency averaged over all cells, (2) Standard Deviation of this estimate, and (3) The total number of cells that passed through the switch (in other words the number of samples taken to make the estimate).

Q5. How long should I run SIM?
A5. Well, this depends on whether the input traffic load you are using is close to the maximum throughput of the switch or not. In general running sim for 200,000 cells should be enough (you do this by using the -l option as follows "sim -l 200000"). But if you are close to saturation then you may need to run it for longer (e.g. for 1,000,000 cell times). Note that SIM has an auto-detect feature which allows it to terminate before the given run length, if it determines that it reached accurate values. So for example if you run sim for 100,000 cells, but around 50,000 cells sim observed that there is no change in the estimate for cell latency then sim will just quit at 50,000.

Q6. What is the trace file formats?
A6. There are two types of trace file traffic sources: (1) trace: trace driven from a file of <time,vci> cells, and (2) tracePacket: trace driven from a file of <time,destination,length> packets. In the tracePacket case the packets will be broken down into equal size cells before being fed

to the inputAction. You can take a look at “TRAFFIC/trace.c” and “TRAFFIC/tracePacket.c” for more information on how they interpret the trace files.

Q7. How can I change the load if I am using trace files?
A7. First you need to know what is the original line rate that this trace file is using. You can do this by either knowing the original source of this trace file (e.g. the Internet Traffic Archive) and check from at what line-rate was it captured, or you can run through the trace looking for the lowest rate such that no consecutive packet times are too close together. You can deduce this easily from the start+length information. Take this largest rate to be the line rate.
Once you have picked a nominal rate for the trace you can do the following to vary the line rate (load): a) making the trace a lower rate: Just randomly throw away a fraction of the packets. This is best done if you throw away all the packets in a flow. This way, you can create almost arbitrary (slower) rates. b) making the trace a higher rate: First, write a simple script that multiplexes (with a buffer of course) two traces together. Then: Method (i): extract some packet flows from the original trace. Multiplex them (with a different offset) with the original trace to give the rate you are looking for, or Method (ii): multiplex multiple copies of the original trace (with different offsets) to create the desired rate.

Q8. How can I do multicast runs?
A8. A multicast run is similar to a unicast run except that you need to indicate to the traffic source that it should generate multicast traffic (using the -m option), and you should use a multicast scheduling algorithm (e.g. mcast_slip). Note that bernoulli_iid_uniform will uniformly distribute traffic over all output ports, while bernoulli_iid_nonuniform allows you to specify an array of utilizations for each output port. Also for both of these traffic sources you can use the -f option to change the fanout (default fanout is the number of output ports). Here is an example of a configuration file for an 8x16 switch with uniform multicast traffic:

```
Numswitches 1
Switch 0
        Numinputs    8
        Numoutputs   16
        InputAction  defaultInputAction
        OutputAction defaultOutputAction
        Fabric       crossbar
        Algorithm       mcast_slip
        0       bernoulli_iid_uniform -m -u 0.22
        1       bernoulli_iid_uniform -m -u 0.22
        2       bernoulli_iid_uniform -m -u 0.22
        3       bernoulli_iid_uniform -m -u 0.22
        4       bernoulli_iid_uniform -m -u 0.22
        5       bernoulli_iid_uniform -m -u 0.22
        6       bernoulli_iid_uniform -m -u 0.22
        7       bernoulli_iid_uniform -m -u 0.22
        Stats
```

```
        Arrivals
        Departures
        Latency (*, m)
        Occupancy (*, m)
Histograms
        Arrivals
        Departures
        Latency (*, m)
        Occupancy (*, m)
```

Q9. Is fanout-splitting implemented for multicast traffic?
A9. By default fanout-splitting is enabled for all mutlicast algorithms, since otherwise the switch would have very low utilization. If you would like to experiment with an algorithm that does not implement fanout splitting then take a look at "ALGORITHMS/mcast_wt_fanout.c" which does not implement fanout splitting.

Q10. How can I change the fabric speed up?
A10. Speedup is a top-level function. To have a speedup of X, you use the following: "sim -f configFile -tX -s1 -eX". It is a bit bizarre, but if you take a look at the main sim.c loop, you will see that it works quite simply.

Q11. Some of the algorithms in "ALGORITHMS/algorithmTable.h" are commented out, do these algorithms work?
A11. Commented algorithms in algorithmTable.h either are not fully debugged or are not implemented at the first place (just place holders to reserve their name for future implementations). You can uncomment and use them, but at your own risk. If you fix the bugs in some of them, or if you implement one of them from scratch, then please let us know so that we can incorporate your change into our source tree.

Q12. How does debugging work?
A12. Works just like for any other program you wrote :). You can use gdb (to catch bugs!) and purify (to catch dangling pointers and memory leaks). In case you are going to use gdb then you should compile the sim source tree with debug info enabled. To do this you should "make cleanall" then "make debugall". Note that there are also function-specific debug flags defined in each sub-directory. e.g. take a look at the first line in ALGORITHMS/algorithm.h. When these debug flags are set to 1 (i.e. TRUE) a lot of debugging information will be printed out while the simulator is running. If you are adding your own new module it is your responsibility to add debug printf statements in the critical points of your code to facilitate your job in debugging that code later. You should always use a debug flag when printing debugging information, so that you can switch off debugging information later when everything is debugged. This allows the simulator to run much faster since calling printf while running may cause huge simulation run-time speed degradation.

Q13. Can I specify the simulation runtime length in the input configuration file?
A13. Unfortunately, you can not do that. The only way you can specify the simulation runtime is through the command line as follows "sim -l 100000 -f configFile". There is a number of other

seldomly used parameters which you can only specify through the command line. There is also some parameters which you can only specify in config files, for example when you want to run the simulator with each input having a different traffic source.