REAL Users Manual

This manual provides a tutorial introduction to setting up a simulation in REAL. If you want to modify REAL you should refer to the REAL Programmers Manual, but read this first.

1. General Orientation

The simulator takes as input a *scenario,* which is a description of network topology, protocols, workload and control parameters. You must describe a scenario using using NetLanguage[1]. The network is modeled as a graph, where nodes (vertices) represent either sources or sinks of data, or gateways[2]. The interconnection between the nodes is the topology. You have to specify the protocol and the workload at each source. Finally, you must specify control parameters such as the latency and bandwidth of each communication line, the size of trunk board buffers, packet sizes etc.

1.1. File organization

The simulator files are in a directory called sim. This has five subdirectories - docs, sim, src, lang and results. sim/docs contains online versions of the manuals. sim/sim has source code for REAL. sim/src has source code for NEST, the simulation package underlying REAL. sim/lang has code for the NetLanguage parser. sim/results stores the results of simulation.

2. Source and Gateway types

Sources and gateways are of different types. A source type corresponds to a particular transport protocol and workload. A gateway type corresponds to a queue scheduling discipline. Each source or gateway type is implemented as a C function. The function is started off as a execution thread by REAL, and can be thought of as being an independently scheduled and preemptable entity (see NEST manuals for more details). I will now describe the protocols implemented by each source and gateway type.

2.1. Source types

There are 20 source types, corresponding to 20 transport protocol and workload types. Most of them have been extensively tested, but I will not rule out any bugs: caveat emptor!

The source types can be grouped into 4 suites, and then some miscellaneous types. The basic suite of protocols is the ‘vanilla’ suite. Elements of this suite implement basic transport level functionality, that is, windowing, timeouts, and go-back-n retransmission. The sources are:

vanilla_ftp: This implements a file transfer protocol like workload, i.e. the source always has data ready to send, and is limited only by the window flow control.

vanilla_telnet: This implements a ‘telnet’ workload. A telnet workload differs from a Poisson workload in two ways a) the net number of outstanding packets cannot exceed the specified window size. b) the maximum possible output rate (inverse of the interpacket spacing) is limited by the bandwidth of the output line from the source (strictly speaking, a Poisson source can send infinitely fast).

vanilla_pulse: This is a ‘pulsed’ or burst workload. It will send a burst of packets, subject to flow control, and then idle for a while. The burst size and the size of the idle period are specified by the ‘pulse_size’ parameter in NetLanguage.

The next suite is the JK suite, that implements TCP with the window adjustment and other modifications described by V. Jacobson[3], M. Karels, P. Karn and C. Partridge. **jk_ftp, jk_telnet** and **jk_pulse** are similar to the protocols described above. There are two more protocols in this suite:

---

[1] The mouse-based interactive display of version 2.0 is no longer supported.

[2] Gateways are used synonymously with routers, bridges and switches to mean elements that route, buffer and schedule packets.

[3] [JAC 88], "Congestion Avoidance and Control", Proc. ACM SigComm ’88.

jk_mftp: This incorporate further modifications made to TCP by V. Jacobson (as of Aug '90)[4].

jk_rate: This is an experimental flow control protocol that uses the window and rtt estimator of TCP to compute a sending rate = window/rtt. It is thus a hybrid window/rate flow control protocol (rate schemes are joint work with S. Shenker at Xerox PARC).

The third suite is the DEC suite, that implements the DEC congestion bit based window adjusting scheme [RJ 90][5]. The four protocols in this suite, **dec_ftp, dec_pulse, dec_rate** and **dec_telnet** are as described above.

The 'malicious' protocol suite, so named because it does not do any flow control, has two protocols.

malic_ftp: sends out a stream of FTP packets as fast as it can, limited only by the bandwidth of the output line.

malic_pulse: Implements a pulse workload with no flow control.

There are five miscellaneous protocols. These are

"poisson:" This is a poisson source that sends packets no faster than the bandwidth of the output line. Thus, the source is *clamped* to that speed. This source is useful in simulating cross traffic, and in validating queueing models.

netblt: implements a NETBLT like flow control scheme [Clark 87][6].

background: is used to create and remove bottlenecks. The background source sends data as fast as it can for some period of time, then idles for some time, and repeats this cycle. The parameters for this source are set from the file background.c.

controlled_rate: This is an on_off source that requests resources using a SETUP packet. The parameters for the source are on_time, off_time and num_pkts, described below in the 'REAL parameters' section.

random_rate: This source sends packets so that the sending rate is no faster than some peak bandwidth, and the expected sending rate is some specified average over some specified interval of time. These parameters are described through NetLanguage. Random_rate sources also do call setup.

2.2. Gateways

There are six routers that differ mainly in their queue service discipline.

fcfs_router: implements FCFS queueing. Does not support reservations.

fq_router: implements Fair Queueing [DKS 89][7]. If RESERVATIONS is #defined in types.h, then weights will be computed, and buffers computed, so that a source can be assured bandwidth and maximum packet loss rate. Threshold parameters allow overbooking of resources.

fqbit_router: is identical to fq_router, except that it does bit-setting on packets.

hrr_router: Implements the HRR discipline [KKK 90][8]. RESERVATIONS must be defined.

dec_router: implements DEC bit-setting[RJ 90], and FCFS queueing.

router: combines all of the above, and can be made to emulate any of the routers using the 'policy' parameter in NetLanguage.

---

[4] Thanks to S. McCanne and S. Floyd at Lawrence Berkeley Lab for this implementation.

[5] [RJ 90] Ramakrishnan, and Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks", ACM Trans. Comp. Sys. May 1990, V8 N2:158-181.

[6] [ClLa87] D. D. Clark, M. L. Lambert and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol" NIC RFC-998, March 1987.

[7] [DKS 89] Demers, Keshav, Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", Proc. SigComm 89 pp1-12, Sep. 1989.

[8] [KKK 90] Kalmanek, Kanakia and Keshav, "Rate Controlled Servers for Very High Speed Networks", Proc. GlobeCom 1990", Dec. 1990.

2.3. Sinks

There are two sinks. Sink.c implements a sink that receives data from any source. If the source is a guaranteed service host, the sink only collects statistics, and does not send an acknowledgment. Otherwise, it keeps track of the highest in-sequence packet received thus far, and sends an acknowledgment with that sequence number.

The other sink, gsink.c, implements a leaky bucket algorithm per source. It uses the SETUP packet to determine a packet removal rate (which is the same as the requested jitter interval), and every time it visits the queue for that source, removes (average_rate * jitter_interval) packets from the queue, keeping track of the maximum queue size attained. This is useful in computing the jitter received by the source.

3. NetLanguage 3.0

I now describe NetLanguage, a user-friendly way of describing network scenarios. Refer to sim/lang/lang.inp or to the Appendix for an example.

The header contains identification information for your own use. It does not affect simulation in any way. This is followed by Nest parameters.

passtime: How often (in virtual time) the monitor is called. I’d recommend 1 second, but you can make it larger if you want a faster simulator. An entry of the form a,b is a time value that represents ‘a’ seconds and ‘b’ microseconds.

maxnodes: This is the most number of nodes in the simulation. Set this to the number of nodes in the simulation. Note that memory is allocated in proportion to maxnodes, so you should not choose something too large. Note also that REAL does not support simulations with more than ABSOLUTE_MAX_NODES nodes. ABSOLUTE_MAX_NODES is defined in config.h.

monitor: Selects the monitor to use. You should use ‘custom_monitor’ unless you write your own.

You now have to set REAL parameters. Here is a brief explanation of each field.

inter_pkt: is the mean of the exponential distribution of the inter-packet delay for the Poisson or telnet workload. Specified in seconds.

ack_size: is the size of the acknowledgment packet in bytes.

random_seed: This is the seed used to initialize the random number generator. If this is 0, the seed is chosen at random.

buffer_size: This is the number of buffers that are available per output line (trunk board). One buffer can store a packet of arbitrary size (this is the Xerox XNS convention).

telnet_pkt_size: is the size of a telnet packet in bytes.

ftp_pkt_size: is the size of a FTP packet in bytes.

ftp_window, telnet_window: are the maximum sizes of ftp and telnet windows. (These are not the same as the congestion windows, but are their upper bounds.) The window size should not exceed the MAX_WINDOW_SIZE parameter in sim/sim/config.h.

decongestion_mechanism: This selects which packet to drop when a conversation’s queue is full. 0 selects the packet at the head of the queue and 1 selects the tail. When this parameter is set to 2, we select a packet at random amongst all the packets queued at the trunk and drop it.

policy: This is used to select the scheduling policy at the gateways. FCFS is 1, Fair Queueing [DKS 89] is 2, 3 is FCFS + bit setting (DEC bit)[RJ 90], 4 is Fair Queueing + bit setting, and 5 is Hierarchical Round Robin(HRR [KKK 90]).

router, real_number: Used in distributed simulation.

pulse_size: An entry of the form X.(Y+1) will generate a pulse of X packets every Y seconds. Y has to be 3 digits long. So, for example, an entry of 3.008 will generate a pulse of 3 packets every 7 seconds. (Do not use 3.8 - that will be interpreted as 3 packets every 799 seconds.) This format is ugly and needs to be changed. Soon, perhaps.

end_simulation: This is the time at which to end the simulation run.

print_interval: Number of seconds after which to print a report. All statistics, except min and max values, are refreshed every print_interval seconds.

bucket_size: This is the size of a bucket in seconds used when printing out the end to end delay distribution. The net number of buckets, NUM_BUCKETS is #defined in config.h.

hrr_levels: All HRR servers use the same value for nb and b, as defined in [KKK 90].

These parameters are followed by a list of node functions. If you add a new function, it should be added to the list. Next, node and edge parameters are defined. To make it easier to declare the nodes, the first declaration in the block is the default declaration. The default defines the parameters of each node unless they are overridden by an explicit redeclaration inside some node. You are not allowed to skip any entries in the defaults declaration, but you may skip as many entries in the node declaration as you wish.

The function field is the name of the C function that should run on that node. ‘dest’ is the ID of the sink to which data should be sent. ‘start_time’ is the simulation time at which the node becomes active. You should set plot to true if you want the node function at that node to generate plots (described later). The other values are self explanatory, and are commented in sim/lang/lang.inp.

Edges are defined by source, destination, bandwidth and the latency in communication. The endpoints of an edge are specified as source->destination, and all edges are bidirectional. The bandwidth is specified in bits per second, and the delay in microseconds.

3.1. A note on usage

Usually, you will not need to change the nest parameter and function declaration sections of the language file. In most cases, you should just copy these from the file sim/lang/lang.inp. It is conventional to give a .l extension to a language file.

It is often useful to declare default values in a file, and to include them in the language file, particularly when several simulation runs are batched. One trick is to create a .s (for small) file that #includes some default files. The ‘asm’ script in sim/sim/scenarios will assemble a .l file from a .s file, and may be of use.

4. Running the simulation

Take a deep breath, and type

```
simulate < <language_file.l>
```

5. Simulation output

The simulator produces three kinds of outputs a) announcements of special events b) time ticks and c) a simulation report.

By default, at the start of the simulation, the current control parameters will be printed out. Special events refers to errors in simulation, or events that you have coded into the node functions. Some error messages are not quite intuitive. If you ever get a error ‘... got an unknown packet’, you should probably send me mail. There is something seriously wrong in the simulator or in your set up.

Time ticks are printed out every ‘print_interval’ seconds of simulated time. They are just to show you that something is going on.

Reports are generated every ‘print_interval’ seconds and are appended to a file called sim/results/dump. The report is in the form of table and will look like this:

The header displays current control parameters used in the simulation, as explained above. This is followed by a description of sink assignments, and edge data. Every print_interval, a report of simulation statistics is printed. The first entry is the time (in this example, 100 seconds). For each source, the following data is printed:

Src: This is the node ID that of the node.

```
---------------------------------

Current control parameters :

inter_pkt   : 0.20 s               ack_size      : 25.00 bytes
random_seed : 651976628            buf_size      : 500 buffers
ftp_window  : 5 pkts               telnet_window : 80 pkts
policy      : 5                    decong_mech   : 1
telnet_size : 25 bytes             ftp_size      : 25 bytes
cc_router   : 0                    realnum       : 0
pulse_size  : 0.000                end_time      : 100 s
bucket_size : 0.250 s              print_interval : 10 s

sink assignments:
node 8 -> sink 7 ;
node 7 -> sink 7 ; node 6 -> sink 7 ; node 5 -> sink 7 ; node 4 -> sink 7 ;
node 3 -> sink 7 ; node 1 -> sink 7 ;
Edges:
4 -> 8, speed: 10000.00, latency 0
7 -> 6, speed: 10000.00, latency 0
6 -> 5, speed: 10000.00, latency 0
5 -> 4, speed: 10000.00, latency 0
4 -> 3, speed: 10000.00, latency 0
3 -> 1, speed: 10000.00, latency 0

---------------------------------------------------------------------------
Time   Src.   Type        G/w    Q_delay(n,min,ave,max)     inter_pkt
 ..........................................................................


    /*     ....time intervals 10..90 skipped .....   */

(100)  1     Cont. Rate    3(RO)   190,0.018,0.085,0.126[2]
                                   4(RO)   190,0,0.064,0.140[2]
                                   5(RO)   190,0,0.073,0.180[2]
                                   6(RO)   190,0.020,0.069,0.180[2]   190,0.020,0.053,0.160 [5]
     drops 0, e2e delay(0.250):| 0 190 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0
     8     Poisson                 4(RO)   51,0.000,0.035,0.132[0]
                                   5(RO)   51,0,0.022,0.140[0]
                                   6(RO)   50,0,0.049,0.120[1]   0, -
     drops 0, e2e delay(0.250):| 42 8 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0
          ---------------- Summary -------------

1          3(RO)   c: 190.000, 0.000;v: 0.085, 0.000
          4(RO)   c: 190.000, 0.000;v: 0.065, 0.001
          5(RO)   c: 190.000, 0.000;v: 0.076, 0.002
          6(RO)   c: 190.000, 0.000;v: 0.064, 0.003      v: 0.053, 0.000
8
          4(RO)   c: 46.667, 8.919;v: 0.031, 0.003
          5(RO)   c: 46.667, 8.819;v: 0.026, 0.003
          6(RO)   c: 46.667, 8.794;v: 0.042, 0.008       v: 0.000, 0.000


---------------------------------
```

Type: This is the name of the node function running on the node.

G/w: This is gateway number for which the delays are reported. If the source sends packets through multiple gateways, then the report will print out data at each of the intermediate gateways. The gateway number is followed by the type of the gateway. RO is the old router, with all the scheduling disciplines folded in. FC is for a router obeying FCFS, FQ, Fair Queueing, FB, Fair Queueing with Bit Seting, DB, DEC Bit setting, HR Hierarchical Round Robin.

Q_delay: The data is in the format: number of packets transmitted for that source from that gateway in the previous print_interval seconds of simulation time, the minimum, average and maximum queueing delays, and finally, in square brackets, a snapshot of the number of packets queued in the per-conversation queue for that source at the time the report was printed. The queueing delay is computed as the time a packet had to wait in the queue from the time that its last bit arrived at the gateway to the time that the first bit in that packet was placed on the output trunk.

If RESERVATIONS has been #defined in config.h, the other fields are:

inter_pkt: This is the minimum, average and maximum interpacket spacing at the receiver for that source. If gsink.c is used, the three fields are followed, in square brackets, by the maximum excess packets at the sink when packets are removed in leaky bucket fashion at the average rate, once every jitter interval.

The next line prints out the number of packets lost in the last print_interval seconds, and the end to end delay distribution histogram in buckets of size printed in parentheses.

If RESERVATIONS is not #defined, you will see

Num dropped: This is the number of packets from that source that were dropped by intermediate gateways.

Num retx.: This is number of packets that the source retransmitted.

Tao: This is the min, average and max round trip time that packets from that source took from the time that the first bit of the packet was placed on the line to the time that the last bit of the acknowledgment arrived. In case of JK sources, it is computed only for packets that have never been retransmitted.

In order to avoid corrupting simulation results with data from transients that happen at start up, simulation statistics are flushed after each dump. In other words, the report presents statistics for the last print_interval seconds of simulation only.

Note that to compute effective throughput you must subtract the number of retransmissions from the number of packets sent by the gateway. The effective load is the number of packets sent by the gateway plus the number of packets dropped by the network.

Each mean value printed out at the end of a print interval can be thought of as a batch mean. If the print_interval is long enough, then the batch mean will be normally distributed. REAL computes both the mean of batch means, and the standard deviation of the means. Using the normal assumption, 99% confidence will be achieved at 3 SD on either side of the mean of means. At the end of the simulation, the mean of batch means and SD of batch means is printed. This is headed by a line marked ‘Summary’. Then, for each throughput (marked c: for count), and delay value (marked v: for value) at each gateway, for each source, the mean of means and SD of means is printed. This is repeated for the interpacket spacing. To avoid start-up transients from affecting the variance value, the parameter NSKIP #defined in table.c selects how many of the initial time periods to skip. The default value is 1.

6. Plotting simulation variables

The node functions in REAL have been instrumented to produce traces of certain variables as the simulation proceeds. I’ll describe the nature of these traces, and then how you can plot them out on the screen or on a printer. Finally, I’ll describe how you can trace additional variables.

The plotting function is implemented very simply. Node functions are asked to print out the simulation variable whenever it could change, along with the current simulation time. For example, TCP sources print out the size of the congestion control window whenever the congestion window could change. These (time, value) tuples can then be post processed to derive other statistics, such as average values, or can be plotted out.

To plot the time vs. value graphs for these variables, you must first start up a graphical display tool called tektool[9].

You can do this by typing:

tektool &

Now, select the tektool window, and type in ‘tty’. This will tell you what the tty for the tektool window is - it will be of the form ttypX, where X is an integer. To display a graph on this tty, type

P filename X

where filename is a plot file. The semantics of the plot files are described below.

To get a hard copy of the graph, you will need to talk to your local laser printer. Something of the form

uniq filename | graph | lpr

will do the job, you may need to consult your local guru for details.

The files that plot creates are of the form CCCxx, where CCC is a three character identification tag and xx is the node ID of the node which is producing the plot. The currently available files are

rto: Contains the sequence of retransmission timeout values selected by the node.

rtt: Contains the estimated round trip times.

tao: Contains the *actual* round trip times.

win: Contains the congestion control window size.

seq: The sequence number of the packet transmitted at that time.

In each case, the X axis is time in seconds, and the Y axis is the data value.

To add your own instrumentation, you have to edit the node functions in sim/sim. Choose the variable that you want to plot, and at the appropriate point(s), insert the function call ‘make_plot(filename, variable);’. For example, to plot the window size, the command I use is ‘make_plot("win", cur_window);’. To plot real numbers, use ‘make_flt_plot()’. Recompile the simulator (‘make final’), and the plot files will be produced automatically for each node that has the plot option set. Use ‘make clean’ to remove old plot files.

7. A Note on REAL Configuration

Note that some hard limits on simulation size are defined in the file config.h. If you plan to do large simulations, you should change this file and recompile the binary. In particular, if you want to simulate more than 99 nodes, you should reset ABSOLUTE_MAX_NODES to 1000 (or some higher power of 10).

If you would like to do distributed simulation, something that is explained in the REAL technical report and in the programmer’s manual, you should #define DISTRIBUTE. (The distributed simulation facility is currently unsupported, but you are welcome to try it at your own risk.) If you want virtual circuits established using the Controlled_Rate source, you should #define RESERVATIONS in sim/types.h.

8. Acknowledgements

S. McCanne at Lawrence Berkeley Lab contributed the code for jk_mftp.c and plotting.c, and reorganized the code in jk_ftp.c. I have reimplemented some features based on his additions to REAL. K.K.

[9] tektool is distributed by Sun Microsystems. If you are not on a Sun workstation, you should use an equivalent display in which the ‘plot’ utility can display graphics.

Ramakrishnan helped with the testing of DEC sources. R. Sethi suggested the use of defaults in Net-Language. D. Ferrari provided advise on statistics and other features. Last but not least, S. Shenker was the main instigator of REAL, and has provided invaluable advice over the years.

APPENDIX : NetLanguage sample

This is a sample description in NetLanguage that is in sim/lang/lang.inp.

```
# comments should start with a '#'
header
{
network:test;
version:1;
}

nest_params {
        passtime = 1,0;
        maxnodes = 30;
        monitor = custom_monitor;
}
real_params {
        inter_pkt_delay    = 1.0 ;
#                                       seconds
        ack_size                = 50;
#                                       bytes
        random_seed             = 0;
        buffer_size             = 500;
        telnet_pkt_size         = 50 ;
        ftp_pkt_size            = 50;
        ftp_window              = 5;
        telnet_window           = 5;
        decongestion_mechanism  = 1;
        sch_policy              = 5;
        router_node             = 0;
        real_number             = 0;
        pulse_size              = 0.0;
        end_simulation          = 200;
        print_interval          = 100;
#                                       in seconds
        bucket_size             = 0.01;
#                                       in seconds
        hrr_levels = {
#                 nb: b

                   9 : 1 ;
                  10 : 0 ;
                   0 : 0 ; } ;

# nb = 0 is used as a delimiter and is necessary
}

node_functions {
        jk_ftp;
        jk_mftp;
        jk_telnet;
        jk_pulse;
        jk_rate;
        dec_ftp_source;
        dec_telnet_source ;
        dec_pulse;
        dec_rate ;
        vanilla_ftp_source;
```

```
            vanilla_telnet_source;
            vanilla_pulse;
            malicious_ftp_source;
            netblt;
            background;
            malicious_pulse;
            controlled_rate;
            random_rate;
            poisson;
            router;
            dec_router;
            hrr_router;
            fq_router;
            fqbit_router;
            fcfs_router;
            sink;
}

nodes {
            default {
                        function = controlled_rate;
                        dest  = 4;
                        start_time = 0,10000;
                        plot = false;
                        num_pkts = 3;
# number of pkts in burst
                        on_time = 200000;
#                                               microseconds
# on_time is the time from the start of the sending of the first
# packet in the on period to the start of sending of the last pkt.
                        off_time = 800000;
#                                               microseconds
# off_time is time between start of last pkt. in prev burst to
# start of first pkt in next burst
                        peak = 1000.0;
#                                                bits per sec
                        average = 1200.0;
#                                               bits per sec
                        interval = 2000;
#                                               microseconds
                        jitter =  2000000;
#                                               microseconds
             }

            node 1 { }
            node 2 { num_pkts = 1; on_time = 0;
                        off_time = 1000000; average = 400.0;
                        jitter = 20000000;}
            node 5 { function = vanilla_telnet_source; }
            node 3 { function = router; }
            node 4 { function = sink; }
}
edges{
            default {
                        bandwidth = 8000.0;
#                                               bits per second
                        latency = 0;
```

```
#                                   in microseconds
                 }
        { 1 -> 3;  bandwidth = 30000.0;}
        { 2 -> 3; bandwidth = 30000.0;}
        { 5 -> 3;  bandwidth = 30000.0;}
        { 3 -> 4; }
}
```