Chris Forlano

CS 370

Chris Edmondson-Yurkanan

May 17, 1995

Simulating a Mobile Internet with REAL

Simulating a Mobile Internet with REAL

Abstract

Research in the field of mobile computing has turned towards the goal of enabling portable computers to access facilities on the Internet. Current reliable transport protocols such as TCP/IP are ill-equipped to discern packet losses due to motion and corruption on wireless lines from those of network congestion, for which such protocols were initial designed. To examine the effects of motion and corruption on reliable protocols, REAL, a network simulation tool, has been modified to quantify these losses. This paper is a quick overview of these initial findings

Background

In the near future, computers will be able to access the Internet without being physically connected to it. Therefore users will be able access information services such as the World Wide Web and communicate through electronic mail while using laptops, palmtops, and dumb terminals. This use of portable computers to access the Internet is called mobile or nomadic computing.

Unfortunately, this change is not as simple as creating a wireless connection to the Internet from your host machine. Early research into the possibilities of nomadic computing have shown that the high corruption rate of the wireless link between the mobile host and the Internet and the motion of these portable workstations has severe effects on data loss. The problem lies in reliable transport protocols discerning where packet losses occur. Reliable transport protocols like TCP/IP have been perfected to get maximum throughput on low error, high bandwidth fiber optic cable. When data is lost, TCP assumes the loss was due to congestion on the network, and subsequently slows down its rate of transmission, thus, allowing the network time to recover. This action is inappropriate if a mobile host temporarily was out of range for a second, since the problem was not congestion, but instead temporary disconnection from the network. The ideal solution would be to resend the lost packets as soon as the host reconnected to the network. Similar problems can occur due to the highly corruptive nature of the wireless link such as interference.

To identify where these losses occur, researchers have explored a number of possibilities. One common theme is the use of an end of handoff signal being sent from the mobile host to the fixed host on the Internet, informing it of the fact that it had moved. This notifies the fixed host that any recent packet losses were most likely lost due to the motion. Another solution is to create two connections at the mobile service router, which bears the responsibility of insuring the packets safe delivery from source to destination. Other have explored improving the properties of the wireless link to limit the losses due to data corruption.

In this paper, we will demonstrate the effects of motion and corruption on reliable protocols and compare some proposed solutions to the problems that it causes using REAL, a network simulation package.

REAL: A Network Simulator

REAL [4] is a simulator used to observe the behavior of packet switching networks. Users can submit REAL scenario files containing a network topology, congestion control policies, workload, and other control parameters. REAL produces as output a summary report of the data flow through each router and a series of files containing plots for graphing factors such as transmission window size, TCP sequence number, and timer values.

Since REAL was not originally designed to handle situations containing mobile hosts, modifications to the source code were made so it could. The following fields can be added to a scenario file in the *nodes* field:

mobile Type: Boolean Default: True

If set to true in a sink, the sink is considered a mobile host. If set to true in a router, the router becomes a mobile service router.

corruption_rate Type: Integer Default: 0

This is used to indicate the percentage of packet losses on that mobile host's wireless connection.

handoff_freq Type: Integer, Integer Default: Never

Indicates in seconds and microseconds how often a mobile host moves out of its cell.

handoff_period Type: Integer, Integer Default: Never

Indicates in seconds and microseconds how long a mobile host remains out of range of its cell.

Also added is the ability to select a packet loss retransmission policy for the entire network by setting the *rexmt_policy* variable in the *real_params* section of the scenario file.

rexmt_policy Type: range [0..3] Default: 0

The following retransmission policies are supported: (0) Standard TCP congestion control, (1) Activation of Fast Retransmission after cell handoff, (2) Extension of Slow Start period after cell handoff, and (3) Last hop acknowledgment.

More on the *rexmt_policy* variable will be discussed in Section X. An example scenario file is included in appendix A.

Simulation of Motion and Corruption

Data losses in a mobile networking environment can be attributed to two factors: The high corruption rate of the wireless link and the unexpected motion of one of the hosts. By setting the corruption rate in our scenario to 3%, we can see the results on the window size and throughput in Figures 1 and 2. Throughput drops because the smaller transmission window limits the amount of packets that can be sent without an acknowledgment. We can also see similar problems when the destination host moves in and out of its cell roughly every 8 seconds for 0.5 seconds at a time in Figures 3 and 4.

Use of Fast Retransmission to Ease the Effects of Motion

One way to soften the blow of multiple timeouts during cell handoffs is to use TCP's fast retransmission capability. Fast retransmission is activated when the sender receives triplicate acknowledgments, indicating that data was lost. Once activated, the sender, without waiting for the next timeout, drops the transmission window, initializes the slow start algorithm, and resends the missing data. This facility could just as easily be activated by an end of handoff signal, sent by the receiver [1]. This can increase the data transmission turnaround by up to a full timeout period, thus, increasing the throughput. The critical element in this methodology is the time the handoff ends. The sooner the mobile host returns to its cell before a timeout, the more effective this policy is. As the end of the handoff time approaches the next timeout, the more this policy grows towards standard TCP performance. Although this method does slightly increase throughput and require little change in current TCP policy, it still performs a wasteful window drop and timer backoff period.

Extension of the Slow Start Period

Another way to use the knowledge of host movement via an end of handoff signal is by extending the slow start period to allow quick growth of the retransmission window (see Figure 6) [2]. This method, originally proposed by Manzoni (et. al.), required complicated restructuring of the network layer protocol to allow for a series of new messages identifying host movement. In this experiment, we isolate the central idea of extension of slow-start period by activation in the form of a handoff signal. This approach allows for a quicker recovery from handoffs due to motion, as displayed in Figure 7.

Use of this methodology could become a problem in the event of both congestion and mobility. For example, a fixed host is transmitting data over a fairly busy network and simultaneously the mobile host moves briefly out of its cell. This new algorithm would shoot the transmission window up to its maximum size, a size possibly too large for the congested area of the network. This would not only flood the fixed network with an excess of data, but the transmission window would shift rapidly up and down, causing drops in throughput. Although more research in this area needs to be done to confirm this.

Last Hop Acknowledgment

This next method attacks the heart of the problem: Identifying where the data loss occurs. For each data packet destined for the receiver, the associated mobile service router sends a last hop acknowledgment [3] to the sender. When the packet arrives at its final destination a destination acknowledgment packet is also sent. All the sender needs to do is keep track of the acknowledgments received. When packet loss occurs, it checks to see if a last hop acknowledgment was received. If so, the loss occurred on the wireless end. If not, the loss occurred due to congestion on the fixed network. In addition, data lost due to corruption can also be effectively remedied by this method.

Not only does last hop acknowledgment ensure that the location of the data loss is determined, but it minimizes the effects on the congestion window (see Figure 8) and associated timers. When losses occur over the wireless link, the transmission window is not reduced, nor are its retransmission timers backed off. Results of this policy clearly show a significant increase in throughput (Figure 9).

Conclusion and Future Work

We have shown how we have modified REAL to simulate a mobile Internet. Through use of this tool, the severe losses in throughput due to host motion and high corruption rates of wireless data lines can be clearly seen. Three methods have been explored. The use of handoff triggered fast retransmission, though having a small effect in improving the throughput, is dependent on the time the mobile host enters a new cell. The extension of the TCP slow start period to quickly increase the transmission window to the maximum size had more profound effects on throughput. However, there may be an issue with increasing the window size past the size it was before the handoff occurred. More research in this area is needed to confirm this. While use of these two methods increase the overall throughput during host motion, they fail to address the issue of the lossy wireless links due to high interference and other factors. The use of last hop acknowledgments from the final mobile router to the source aid protocols like TCP in identifying where packets were lost, either on the wired or wireless end. This capability enables it to effectively handle losses due to both data corruption and host motion.

This coupled with the policy of not dropping the transmission window for wireless losses make it the better of the three methods in increasing overall throughput (see Figures 10 and 11).

Simulating realistic mobile network topologies using REAL needs to be further explored, particularly in high congestion, frequent motion scenarios. Further modifications to REAL such as source motion and implementations of TCP Tahoe using fast retransmit, extended slow start, and last hop acknowledgment should also be explored. It may also be beneficial to explore the means by which REAL performs fast retransmission. There may be ways to improve upon its implementation.

References

[1] R. Caceres and L. Iftode. The Effects of Mobility on Reliable Transport Protocols. In Proc. of the 14th Intl. Conf. on Distributed Computing Systems, June 1994.

[2] P. Manzoni, D.Ghosal, G. Serazzi. A Simulation Study of the Impact of Mobility on TCP/IP. IEEE International Conference on Network Protocols, 1994.

[3] J. Cobb and P. Agrawal. Congestion or Corruption? A Strategy for Efficient Wireless TCP Sessions. ISCC, 1995.

[4] S. Keshav,” The REAL Simulation Package”, available from keshav@research.att.com

Appendix A: An Example Scenario File

```
# This is a sample input file for the simulator
# it can also be found as lang/example.l
# To see if your simulator is OK, run it on this example as
# simulate example.l
# Remember that your REALDIR environment variable should be set
# to something reasonable

# the header section is optional

header
{
network: example;
version: 0;
}

# nest_params section is optional. Default passtime is 1 second
# default maxnodes is MAXNODES in config.h and default monitor is
# custom_monitor in kernel/monitor.c

nest_params {
passtime = 1,0;
maxnodes = 30;
monitor = custom_monitor;
}

# real_params section is also optional. Default definitions are in
# lang/lang.yacc

real_params {
inter_pkt_delay = 5.0;
#                                       seconds (optional - not used)
random_seed = 23;
buffer_size = 440000;
#                       bytes       (different from REAL3.0)
ftp_pkt_size = 500;
#                                       bytes
telnet_pkt_size = 40;
#                       bytes
ack_size = 40;
#                                       bytes
ftp_window = 600;
#                                       packets
telnet_window = 400;
#                                       packets
decongestion_mechanism = 1;
#                       1 = drop from tail
#                       2 = drop from head
#                       3 = drop from random position
real_number  = 0;
#                       used for distributed REAL
```

```
router_node = 0;
#                       used for distributed REAL
util_time = 1.0;
#                       time interval over which to
#                       measure router utilization (secs)
end_simulation = 100;
#                       simulation end time in seconds
print_interval = 10;
#                       print and flush statistics every
#                       print_interval seconds
scale_factor = 1;
#                       time scale factor; if scale factor =
#                       1000, 1 simulated second = 1ms realtime
      rexmt_policy = 0;
#                       0 = standard TCP congestion control
#                       1 = Fast Retransmission
#                       2 = Extended Slow Start
#                       3 = Last Hop Acknowledgment
}

# this section is only for backwards compatibility with REAL 3.0
# it is optional in REAL 4.0

node_functions {
#                       This section is for documentation only:
#                       and you can skip it if you want.
#                       if you do use it, the name must match
#                       the function name in the source file.
poisson;
router;
sink;
}

nodes{
#                       first define the default node type
#                       all fields are optional
default{
      function = poisson;
      dest = 3;
#                       destination node to send to
      start_time = 0,10000;
#                       start node at seconds, microseconds
      plot = false ;
#                       set to true if you want to plot data
      num_pkts = 1000;
#                       number of packets to send
      on_time = 0;
#                       for on-off style sources:
#                       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 = 40000;
```

```
#                        off_time is time between start
#                        of last pkt. in prev burst
#                        to start of first pkt in next burst
      peak = 20000.0;
#                        for controlled sources: this is the
#                        declared peak rate in bits/sec
      average = 5000.0;
#                        declared average rate in bits/sec
      interval = 1000000;
#                        interval over which average is computed
#                        (in microseconds)
      jitter = 20000000;
#                        desired max delay jitter from the
#                        network (in microseconds)
      hrr_levels = { 10:0 ; } ;
#                        for a hrr router, the b and nb values
#                        at each level. Ignored for a source.
      sch_policy = 2;
#                        for router nodes
#                        1 = fcfs
#                        2 = fair queueing
#                        3 = decbit
#                        4 = fqbit
#                        5 = hrr
      handoff_freq = 8,0;
#                        handoff frequency in seconds and microseconds
      handoff_period = 0,50000;
#                        handoff period in seconds and microseconds
      }

#                        for each source, just need to specify
#                        what is different compared to the default
node 1 { function = poisson; plot = true; num_pkts = 4000; dest = 3;}
node 2 { function = router; plot = true; mobile = true;}
node 3 { function = sink; plot=true; mobile = true;corruption_rate = 3;}


}

edges{
default {
      bandwidth = 2000000;
#                  in bits/sec
      latency = 10000;
#                  in microseconds
      }
{ 1 -> 2;}
{ 2 -> 3; bandwidth = 400000; latency = 20000;}
}
```