Huawei Project Experiment Platform Setup

1. Introduction

This document is targeted to explain the overall architecture and all possible operations of the experiment environment.

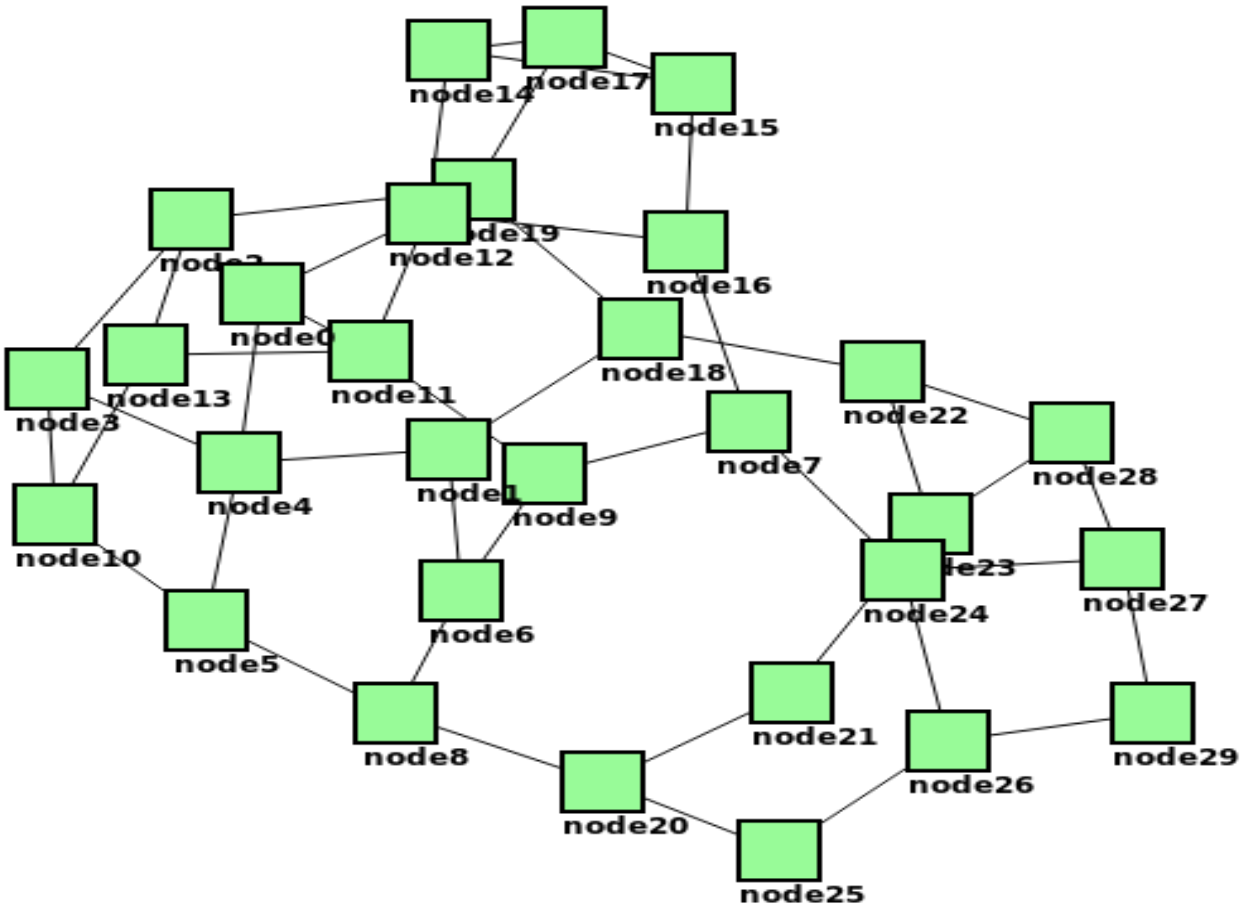
For convenience, we use Emulab as the platform of the experiment. *Emulab* is a network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. Emulab is a **public facility**, available without charge to most researchers worldwide.

Emulab uses a NS-like format to construct customized network topologies which makes doing experiments on Emulab easy and straight-forward. It supports most NS script grammars to configure network.

In our experiment, we need to setup a network topology, and measure the round-trip delay among overlay nodes. Given the probability of congestion and routing table among all overlay nodes, we generate congestion among physical nodes. By measuring a certain number of pairs of overlay nodes, the erratic connections will be figured out.

2. Topology

The topology we use in our experiment is shown in Figure~1, the relative NS script is also presented here.



Figure~1. Experiment Topology

The following is the NS script of the topology. We can see that totally there are 30 nodes with 45 links. Each link has a bandwidth of 100Mbps and one-way delay of 5ms.

```
#Topology.ns
set ns [new Simulator]
source tb_compat.tcl

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
```

```
set node6 [$ns node]
set node7 [$ns node]
set node8 [$ns node]
set node9 [$ns node]
set node10 [$ns node]
set node11 [$ns node]
set node12 [$ns node]
set node13 [$ns node]
set node14 [$ns node]
set node15 [$ns node]
set node16 [$ns node]
set node17 [$ns node]
set node18 [$ns node]
set node19 [$ns node]
set node20 [$ns node]
set node21 [$ns node]
set node22 [$ns node]
set node23 [$ns node]
set node24 [$ns node]
set node25 [$ns node]
set node26 [$ns node]
set node27 [$ns node]
set node28 [$ns node]
set node29 [$ns node]

set link1 [$ns duplex-link $node0 $node11 100Mb 5ms DropTail]
set link2 [$ns duplex-link $node0 $node4 100Mb 5ms DropTail]
set link3 [$ns duplex-link $node0 $node12 100Mb 5ms DropTail]
set link4 [$ns duplex-link $node1 $node6 100Mb 5ms DropTail]
set link5 [$ns duplex-link $node1 $node4 100Mb 5ms DropTail]
set link6 [$ns duplex-link $node1 $node18 100Mb 5ms DropTail]
set link7 [$ns duplex-link $node2 $node13 100Mb 5ms DropTail]
set link8 [$ns duplex-link $node2 $node3 100Mb 5ms DropTail]
set link9 [$ns duplex-link $node2 $node19 100Mb 5ms DropTail]
set link10 [$ns duplex-link $node3 $node10 100Mb 5ms DropTail]
set link11 [$ns duplex-link $node3 $node4 100Mb 5ms DropTail]
set link12 [$ns duplex-link $node4 $node5 100Mb 5ms DropTail]
set link13 [$ns duplex-link $node5 $node10 100Mb 5ms DropTail]
set link14 [$ns duplex-link $node5 $node8 100Mb 5ms DropTail]
```

```
set link15 [$ns duplex-link $node6 $node9 100Mb 5ms DropTail]
set link16 [$ns duplex-link $node6 $node8 100Mb 5ms DropTail]
set link17 [$ns duplex-link $node7 $node9 100Mb 5ms DropTail]
set link18 [$ns duplex-link $node7 $node16 100Mb 5ms DropTail]
set link19 [$ns duplex-link $node7 $node24 100Mb 5ms DropTail]
set link20 [$ns duplex-link $node8 $node20 100Mb 5ms DropTail]
set link21 [$ns duplex-link $node9 $node11 100Mb 5ms DropTail]
set link22 [$ns duplex-link $node10 $node13 100Mb 5ms DropTail]
set link23 [$ns duplex-link $node11 $node13 100Mb 5ms DropTail]
set link24 [$ns duplex-link $node11 $node12 100Mb 5ms DropTail]
set link25 [$ns duplex-link $node12 $node14 100Mb 5ms DropTail]
set link26 [$ns duplex-link $node12 $node16 100Mb 5ms DropTail]
set link27 [$ns duplex-link $node14 $node17 100Mb 5ms DropTail]
set link28 [$ns duplex-link $node14 $node15 100Mb 5ms DropTail]
set link29 [$ns duplex-link $node15 $node16 100Mb 5ms DropTail]
set link30 [$ns duplex-link $node15 $node17 100Mb 5ms DropTail]
set link31 [$ns duplex-link $node17 $node19 100Mb 5ms DropTail]
set link32 [$ns duplex-link $node18 $node19 100Mb 5ms DropTail]
set link33 [$ns duplex-link $node18 $node22 100Mb 5ms DropTail]
set link34 [$ns duplex-link $node20 $node21 100Mb 5ms DropTail]
set link35 [$ns duplex-link $node20 $node25 100Mb 5ms DropTail]
set link36 [$ns duplex-link $node21 $node23 100Mb 5ms DropTail]
set link37 [$ns duplex-link $node22 $node23 100Mb 5ms DropTail]
set link38 [$ns duplex-link $node22 $node28 100Mb 5ms DropTail]
set link39 [$ns duplex-link $node23 $node28 100Mb 5ms DropTail]
set link40 [$ns duplex-link $node24 $node26 100Mb 5ms DropTail]
set link41 [$ns duplex-link $node24 $node27 100Mb 5ms DropTail]
set link42 [$ns duplex-link $node25 $node26 100Mb 5ms DropTail]
set link43 [$ns duplex-link $node26 $node29 100Mb 5ms DropTail]
set link44 [$ns duplex-link $node27 $node28 100Mb 5ms DropTail]
set link45 [$ns duplex-link $node27 $node29 100Mb 5ms DropTail]

$ns rtproto Static
$ns run
```

3. Routing Table

In the NS script above, we use static routing policy which means Emulab software will generate the routing table for the network. But we need to figure out the routing between

each two nodes ourselves. Traceroute is used to do this job. An open source python traceroute program is used to detect the routing. It is shown below. We modified the script to record the result into a routing file.

```
#traceroute.py
#!/usr/bin/python
#import optparse
import socket
import sys

icmp = socket.getprotobyname('icmp')
udp = socket.getprotobyname('udp')
long_local_node_name = socket.gethostname()
index = long_local_node_name.find('.')
if index > 0:
        local_node_name = long_local_node_name[:index]
else:
        local_node_name = long_local_node_name


def create_sockets(ttl):
    """
    Sets up sockets necessary for the traceroute.  We need a receiving
    socket and a sending socket.
    """
    recv_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
    send_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, udp)
    send_socket.setsockopt(socket.SOL_IP, socket.IP_TTL, ttl)
    return recv_socket, send_socket

def main(dest_name, port, max_hops):
    routing = local_node_name + ' ' + dest_name + '\n' + local_node_name + ' '
    dest_addr = socket.gethostbyname(dest_name)
    ttl = 1
    while True:
        recv_socket, send_socket = create_sockets(ttl)
        recv_socket.bind(("", port))
        send_socket.sendto("", (dest_name, port))
        curr_addr = None
```

```
    curr_name = None
    try:
        # socket.recvfrom() gives back (data, address), but we
        # only care about the latter.
        _, curr_addr = recv_socket.recvfrom(512)
        curr_addr = curr_addr[0]  # address is given as tuple
        try:
            curr_name = socket.gethostbyaddr(curr_addr)[0]
        except socket.error:
            curr_name = curr_addr
    except socket.error:
        pass
    finally:
        send_socket.close()
        recv_socket.close()

    if curr_addr is not None:
        curr_host = "%s (%s)" % (curr_name, curr_addr)
          i1 = curr_name.find('-')
          if i1 > 0:
                routing = routing +  curr_name[:i1] + ' '
          else:
                routing = routing + curr_name + ' '

    else:
        curr_host = "*"
    print "%d\t%s" % (ttl, curr_host)

    ttl += 1
    if curr_addr == dest_addr or ttl > max_hops:
        break

routing = routing + '\n'

f = open('routing', 'a')
f.write(routing)
f.close()

return 0
```

```
if __name__ == "__main__":
    parser = optparse.OptionParser(usage="%prog [options] hostname")
    parser.add_option("-p", "--port", dest="port",
                      help="Port to use for socket connection [default: %default]",
                      default=33434, metavar="PORT")
    parser.add_option("-m", "--max-hops", dest="max_hops",
                      help="Max hops before giving up [default: %default]",
                      default=30, metavar="MAXHOPS")
    options, args = parser.parse_args()
    if len(args) != 1:
        parser.error()
    else:

    dest_name = sys.argv[1]
    sys.exit(main(dest_name=dest_name,
                  port=33434,
                  max_hops=64))
```

On each node, we run another python script below to get the routing from the source node to all other nodes. Because the result from all nodes will be written into the same file, we run the traceroute script serially on each node, that is what the line *time.sleep(index * 200)* does.

```
#routing.py
#!/usr/bin/env python
import os
import time
import random
import socket
import sys;

node_table = ['node0', 'node1', 'node2', 'node3',
              'node4', 'node5', 'node6', 'node7',
              'node8', 'node9', 'node10', 'node11',
              'node12', 'node13', 'node14', 'node15',
              'node16', 'node17', 'node18', 'node19',
              'node20', 'node21', 'node22', 'node23',
              'node24', 'node25', 'node26', 'node27',
```

```
            'node28', 'node29']

local = socket.gethostname()
index = 0
for value in node_table:
        if local.find(value + '.') >= 0:
                index = node_table.index(value)
                break

time.sleep(index * 200)

for value in node_table:
        os.system('sudo python traceroute.py ' + value)
        time.sleep(2)
```

4. Delay Measurement

In our experiment, delay is the only metric to be measured between two given overlay nodes. We use a modified version of Ping program. The program does regular ping operation from one node to the other, meantime, it records the delay value to a designated file, the filename is the source node's node name.

During our measurement, we consider all nodes as overlay nodes. One python script is running at each node to measure the round-trip delay from the source node to all other nodes. The python script is shown below. This script runs at all nodes. It pings all the nodes including itself. All the ping programs are ran as background tasks and they are running simultaneously. At each node, one file will be recorded.

```
#ping.py
#!/usr/bin/env python
import os

os.system('sudo ping node0 &')
os.system('sudo ping node1 &')
os.system('sudo ping node2 &')
os.system('sudo ping node3 &')
os.system('sudo ping node4 &')
os.system('sudo ping node5 &')
os.system('sudo ping node6 &')
```

```
os.system('sudo ping node7 &')
os.system('sudo ping node8 &')
os.system('sudo ping node9 &')
os.system('sudo ping node10 &')
os.system('sudo ping node11 &')
os.system('sudo ping node12 &')
os.system('sudo ping node13 &')
os.system('sudo ping node14 &')
os.system('sudo ping node15 &')
os.system('sudo ping node16 &')
os.system('sudo ping node17 &')
os.system('sudo ping node18 &')
os.system('sudo ping node19 &')
os.system('sudo ping node20 &')
os.system('sudo ping node21 &')
os.system('sudo ping node22 &')
os.system('sudo ping node23 &')
os.system('sudo ping node24 &')
os.system('sudo ping node25 &')
os.system('sudo ping node26 &')
os.system('sudo ping node27 &')
os.system('sudo ping node28 &')
os.system('sudo ping node29 &')
```

5. Congestion Configuration

Normally, there will be no congestion on each link, but each link does have a preset value of congestion probability. The background traffic and congestion is controlled by the python script below.

From the script, we can see that all background traffic is generated by iperf UDP. When no congestion is added to one link, the background traffic on that link will be 20Mbps. If congestion is added, the background traffic will increase to 120Mbps.

Each link has a probability of congestion. We divide the value into 5 levels. Since there are totally 45 links, each 9 links have the same congestion probability. As designed, the whole experiment lasts for 600 seconds. In the first 200 seconds, the background traffic on each link is 20Mbps which doesn't cause congestion at all. After that, each link chooses having congestion or not by its congestion probability. This is done by random numbers. If

the link is chosen, then another 100Mbps UDP traffic will be added to the link to generate congestion which will result in delay increment. The 100Mbps background traffic will last for 200 seconds. After that, it will be removed, and the delay among all nodes will be back to normal again for another 200 seconds before the experiment exits.

```
#bg.py
#!/usr/bin/env python
import os
import time
import random
import socket
import sys;

factor = 5
f5 = 5*factor
f4 = 4*factor
f3 = 3*factor
f2 = 2*factor
f1 = 1*factor
link_table = { 1 : ['node0',  'node11',  '20M', '100M', f5],
               2 : ['node0',  'node4',  '20M', '100M', f4],
               3 : ['node0',  'node12',  '20M', '100M', f3],
               4 : ['node1',  'node6',  '20M', '100M', f2],
               5 : ['node1',  'node4',  '20M', '100M', f1],
               6 : ['node1',  'node18',  '20M', '100M', f5],
               7 : ['node2',  'node13',  '20M', '100M', f4],
               8 : ['node2',  'node3',  '20M', '100M', f3],
               9 : ['node2',  'node19',  '20M', '100M', f2],
               10 : ['node3',  'node10',  '20M', '100M', f1],
               11 : ['node3',  'node4',  '20M', '100M', f5],
               12 : ['node4',  'node5',  '20M', '100M', f4],
               13 : ['node5',  'node10',  '20M', '100M', f3],
               14 : ['node5',  'node8',  '20M', '100M', f2],
               15 : ['node6',  'node9',  '20M', '100M', f1],
               16 : ['node6',  'node8',  '20M', '100M', f5],
               17 : ['node7',  'node9',  '20M', '100M', f4],
               18 : ['node7',  'node16',  '20M', '100M', f3],
               19 : ['node7',  'node24',  '20M', '100M', f2],
               20 : ['node8',  'node20',  '20M', '100M', f1],
```

```
                21 : ['node9',  'node11',  '20M', '100M', f5],
                22 : ['node10',  'node13',  '20M', '100M', f4],
                23 : ['node11',  'node13',  '20M', '100M', f3],
                24 : ['node11',  'node12',  '20M', '100M', f2],
                25 : ['node12',  'node14',  '20M', '100M', f1],
                26 : ['node12',  'node16',  '20M', '100M', f5],
                27 : ['node14',  'node17',  '20M', '100M', f4],
                28 : ['node14',  'node15',  '20M', '100M', f3],
                29 : ['node15',  'node16',  '20M', '100M', f2],
                30 : ['node15',  'node17',  '20M', '100M', f1],
                31 : ['node17',  'node19',  '20M', '100M', f5],
                32 : ['node18',  'node19',  '20M', '100M', f4],
                33 : ['node18',  'node22',  '20M', '100M', f3],
                34 : ['node20',  'node21',  '20M', '100M', f2],
                35 : ['node20',  'node25',  '20M', '100M', f1],
                36 : ['node21',  'node23',  '20M', '100M', f5],
                37 : ['node22',  'node23',  '20M', '100M', f4],
                38 : ['node22',  'node28',  '20M', '100M', f3],
                39 : ['node23',  'node28',  '20M', '100M', f2],
                40 : ['node24',  'node26',  '20M', '100M', f1],
                41 : ['node24',  'node27',  '20M', '100M', f5],
                42 : ['node25',  'node26',  '20M', '100M', f4],
                43 : ['node26',  'node29',  '20M', '100M', f3],
                44 : ['node27',  'node28',  '20M', '100M', f2],
                45 : ['node27',  'node29',  '20M', '100M', f1]}


local_node_name = socket.gethostname()

time.sleep(10)

for key, value in link_table.items():
        if local_node_name.find(value[0]) >= 0:
                iperfstr = '/usr/local/etc/emulab/emulab-iperf -u -c ' + value[1] + ' -t 700 -b ' +
value[2] + ' &'
                os.system(iperfstr)

time.sleep(200)
chosenkeys = []
```

```
for key, value in link_table.items():
        if value[4] > 0:
                if local_node_name.find(value[0] + '.') >= 0:
                        r = random.randint(1, 100)
                        if r < value[4]+1:
                                chosenkeys.append(key)

for key in chosenkeys:
        value = link_table[key]
        iperfstr = '/usr/local/etc/emulab/emulab-iperf -u -c ' + value[1] + ' -t 200 -b ' +
value[3] + ' &'
        print 'start:' + iperfstr + '\n'
        os.system('sudo chmod 777 ' + local_node_name)
        f = open(local_node_name, 'a')
        f.write('start:'+iperfstr+'\n')
        f.close()
        os.system(iperfstr)

time.sleep(200)

for key in chosenkeys:
        value = link_table[key]
        iperfstr = '/usr/local/etc/emulab/emulab-iperf -u -c ' + value[1] + ' -t 200 -b ' +
value[3] + ' &'
        print 'end:' + iperfstr + '\n'
        f = open(local_node_name, 'a')
        f.write('end:'+iperfstr+'\n')
        f.close()


time.sleep(200)
os.system('sudo killall python')
os.system('sudo killall ping')
os.system('sudo killall emulab-iperf')
exit()
```

6. Run the Experiment

According to all the items above, when starting the experiment, on each node, several program/script will be started as shown in the script below. The first one is the iperf UDP server because all traffic will be generated through iperf UDP. The second is the ping script to simultaneously measure the real-time delay between each pair. The third one is the background script to generate background traffic and congestion.

Then after 600 seconds, the experiment will exit, we just need to collect all the delay measurement result from each node for future analysis.

```
#!/usr/bin/env python
import os
import time

os.system('/usr/local/etc/emulab/emulab-iperf -s -u &')
os.system('sudo python ping.py >> /dev/null &')
os.system('sudo python bg.py >> /dev/null &')
```