Building a Million-Node GENI Through End User Opt-In

Ivan Beschastnikh, Justin Samuel, Jennifer Hanson,
Eric Kimbrel, Monzur Muhammad,
Arvind Krishnamurthy, Justin Cappos, Tom Anderson

Computer Science and Engineering

University of Washington

Testbeds: Limited Scale and Diversity

Common testbed characteristics

- Homogeneous platforms and connectivity
- Nodes on specific types of networks (e.g. Universities)
- Small size

Real-world characteristics

- Heterogeneous platforms and connectivity
- Nodes in all types of networks (residential, mobile, etc.)
- Large size

Bringing the Real World to Testbeds

Make testbeds Internet-scale

- Run on a subset of the Internet

Cannot be done with only private resources

- Logistically and financially impossible

Design for public participation

- Provide incentives for participation
- Address security and performance impact

Seattle: Internet Testbed

Open network of sandboxed VMs

- Grow testbed network from donated public resources
- VMs provide isolation and resource limitation

Low impact

- Users donate 10% of each system's resources

Incentives for donations

- Example: researcher gets use of 10 remote VMs per donation
- or own your own private testbed

Seattle: Architecture

Components

- Vessel: An isolated programming language VM
- Node Manager: Handles multiple vessels per machine
- Experiment Manager: Locate and control vessels

Security and Restrictions

- Restricted programming language designed for security
- Hard limits on RAM, CPU, bandwidth, hard drive space
- Egress traffic restrictions in development

Demonstration: All Pairs Ping

Obtain resources

- https://seattlegeni.cs.washington.edu/
- Register account and download toolkit
- Request remote VMs (can use XML-RPC to do this)

Deploy and run All Pairs Ping

- Deployed and run using *seash*, the Experiment Manager

View the results

- Our experiment code has the VMs serve a results webpage

Educational Use

Why?

- LAN !~ Internet
- Seattle ~ Internet

Tutorials for students

Example assignments for educators

Community support (NW-DCSD / CCSC)

Already used in 6 classes

Coming Soon: Autograder

Developer Use

Why?

- Portable PL != portable programs
- What about cloud computing?
- Build your own testbed!
  - Virtualization protects user machines
  - Allows developers to test with real inputs from real machines on real networks

Easy to develop large applications

- but standard libraries are not as rich…

Researcher Use

Why?

Limited network heterogeneity (wired, high bw, university, no middle boxes)

Small scale (<1K)

No real world use / mobility patterns

Send TCP / UDP traffic amongst testbed machines

No ability to run binaries, gain root, etc.

What do you want your testbed to do?

Node Composition

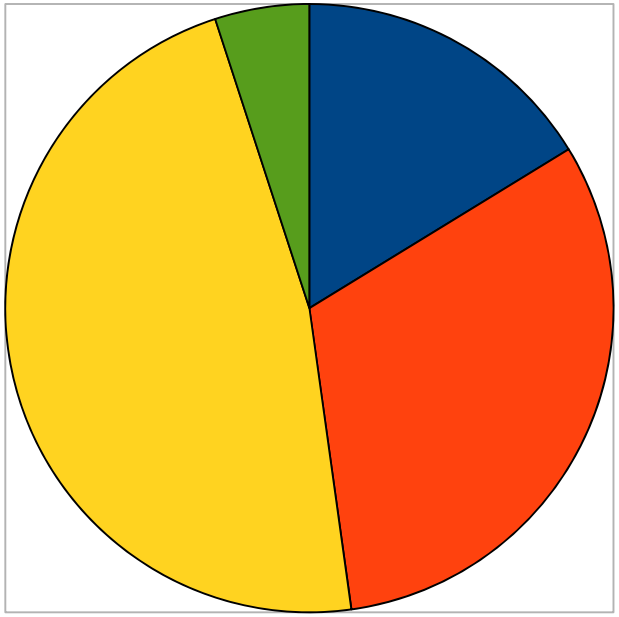
% Testbed by donation type

Planet Lab Nodes: **32%**

University Donations: **16%**

Individual Donations: **47%**

Uncategorized: **5%**



~2000 unique IPs

Nodes by donation type that have been logged accessing the Seattle software updater.

Summary

Existing testbeds don't model the Internet

Seattle: build a testbed from the Internet

- Encourage resource donations from the public

We have Seattle users:

- Education (growing classroom use)
- Research (alpha – only adventurous users)
- Development (alpha – only adventurous users)

Code Example: All Pairs Ping

```
# send a probe message to each neighbor
def probe_neighbors(port):

  for neighborip in mycontext["neighborlist"]:
    mycontext['sendtime'][neighborip] = getruntime()
    sendmess(neighborip, port, 'ping',getmyip(),port)

    sendmess(neighborip, port, 'share',getmyip(),mycontext['row'][getmyip()],mycontext["neighborlist"
], mycontext['latency'].copy())
    # sleep in between messages to prevent me from getting a huge number of
    # responses all at once...
    sleep(.5)

  # Call me again in 10 seconds
  while True:
    try:
      settimer(10,probe_neighbors,(port,))
      return
    except Exception, e:
      if "Resource 'events'" in str(e):
        # there are too many events scheduled, I should wait and try again
        sleep(.5)
        continue
      raise
```

Send periodic UDP pings

15 LOC

```
# Handle an incoming message
def got_message(srcip,srcport,mess,ch):
  if mess == 'ping':
    sendmess(srcip,srcport,'pong',getmyip())
  elif mess == 'pong':
    # elapsed time is now - time when I sent the ping
    mycontext['latency'][srcip] = getruntime() - mycontext['sendtime'][srcip]

  elif mess.startswith('share'):
    mycontext['row'][srcip] = mess[len('share'):]
```

Handle incoming UDP pings

7 LOC

```
def encode_row(rowip, neighborlist, latencylist):

  retstring = "<tr><td>"+rowip+"</td>"
  for neighborip in neighborlist:
    if neighborip in latencylist:
      retstring = retstring + "<td>"+str(latencylist[neighborip])[:4]+"</td>"
    else:
      retstring = retstring + "<td>unknown</td>"

  retstring = retstring + "</tr>"
  return retstring
```

Format latency data into HTML

9 LOC

```
def show_status(srcip,srcport,connobj, ch, mainch):

  webpage = "<html><head><title>Latency Information</title></head><body><h1>Latenc
y information from "+getmyip()+" </h1><table border="1">"

  webpage = webpage + "<tr><td></td><td>"+ "</td><td>".join(mycontext['neighborlis
t'])+"</td></tr>"

  for nodeip in mycontext['neighborlist']:
    if nodeip in mycontext['row']:
      webpage = webpage + encode_row(nodeip, mycontext['neighborlist'],
    else:
      webpage = webpage + '<tr><td>'+nodeip+'</td><td>No Data Reported</td></tr>\n
'

  # now the footer...
  webpage = webpage + '</table></html>'

  # send the header and page
  connobj.send("HTTP/1.0 200 OK\nContent-Length: "+str(len(webpage))+"\nDate: Fri,
31 Dec 1999 23:59:59 GMT\nContent-Type: text/html\n\n"+webpage)

  # and we're done, so let's close this connection...
  connobj.close()
```

Return a webpage

11 LOC

```
if callfunc == 'initialize':

  # this holds the response information (i.e. when nodes responded)
  mycontext['latency'] = {}
  # this remembers when we sent a probe
  mycontext['sendtime'] = {}
  # this remembers row data from the other nodes
  mycontext['row'] = {}

  # get the nodes to probe
  mycontext['neighborlist'] = []
  for line in file("neighborlist.txt"):
    mycontext['neighborlist'].append(line.strip())

  ip = getmyip()
  if len(callargs) != 1:
    raise Exception, "Must specify the port to use"
  pingport = int(callargs[0])

  # call gotmessage whenever receiving a message
  recvmess(ip,pingport,got_message)

  probe_neighbors(pingport)

  # we want to register a function to show a status webpage (TCP port)
  pageport = int(callargs[0])
  waitforconn(ip,pageport,show_status)
```

Initialization

15 LOC