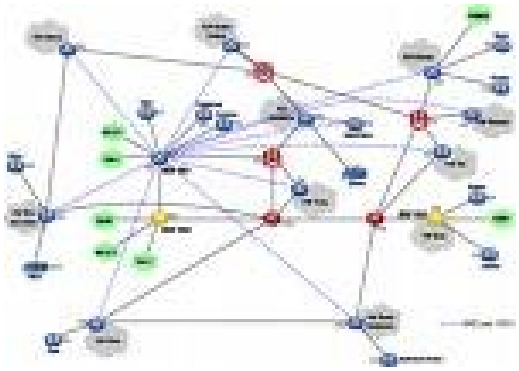
Seattle: The Internet as an Educational Testbed

Justin Cappos

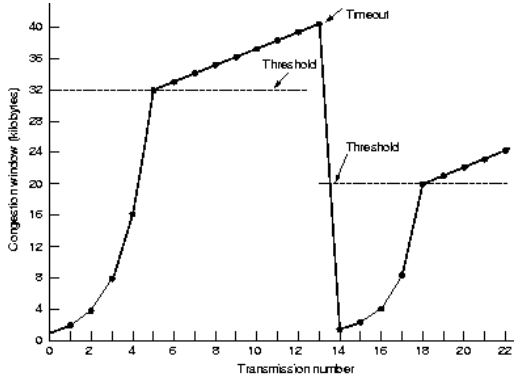
NYU Poly

Computer Science and Engineering

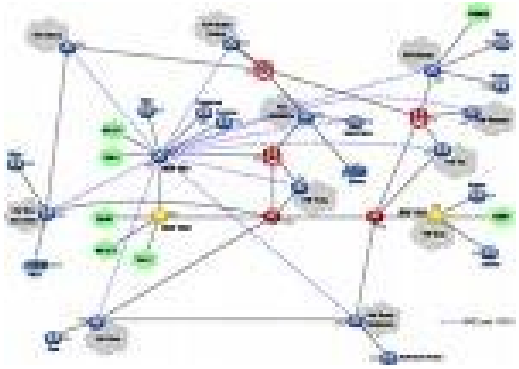
The “Dark Ages”

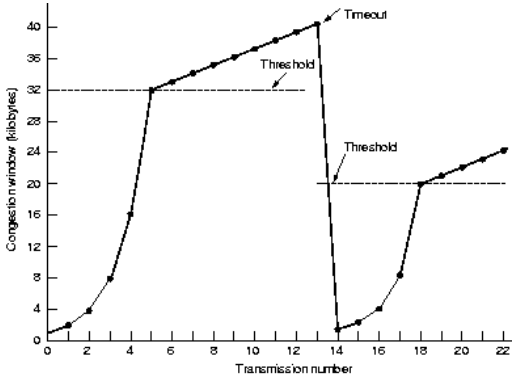


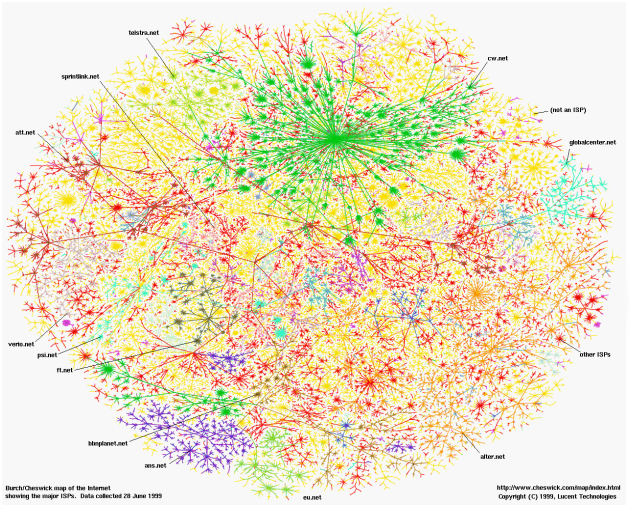
Simulation LAN



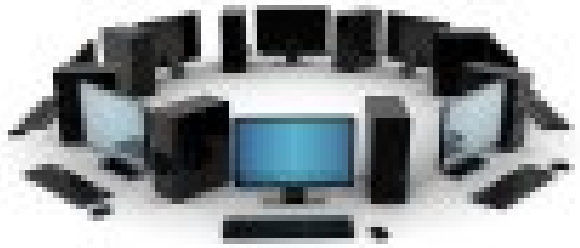
The “Golden Age”



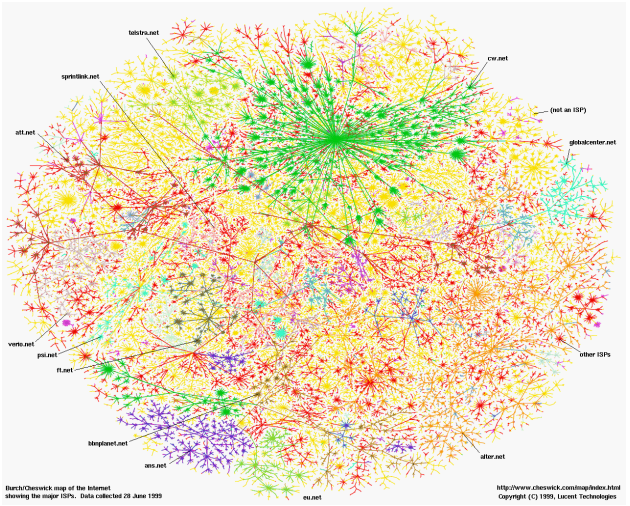




The Path to Enlightenment



Seattle



Common use

- Students
  - Request resources via website
  - Use shell to start programs on computers all around the world
    - Not just the machines at your university!
  - View program output / tracebacks
- Students can also test on a local install
  - Easy way to test quickly or when disconnected
- Instructor or students optionally install Seattle
  - Each install allows the use of 10 more VMs

How Seattle Works

- **An installer is downloaded**
  - **Can be installed in a restricted account**
    - **Admin access not needed**
  - **Many platforms supported (Windows, Linux, Mac, BSD, some smartphones, tablets, etc.)**
  - **Runs a few processes**
  - **No IT management overhead**

How Seattle Works (cont)

- Programs are run in a virtualized environment
  - Safety / Security
  - Performance isolation (similar to Xen, VMWare)
    - A buggy program can't slow down machines
  - Node Manager allows remote users to control programs
  - Portability
  - Programming language VM based upon Python
    - Students (and instructors!) find it easy to learn
      - Chord implementation (~300 LOC) in 3 weeks!

Demonstration

- Seattle Clearinghouse
  - Register an account
  - Install Seattle
  - Acquire resources
  - Download demokit / shell
- Use shell to control resources
  - Deploy all pairs ping

Educational use

- Classroom experience
  - Released in Spring 2009
  - Used in more than two dozen classes (so far)
  - 3 tutorials, 3 library references, etc.
  - 10 battle tested assignments
    - Overlay routing, flow control, NAT / Non-transitive connectivity, Chord (DHT), web / chat servers, reference monitors, NAT tunneling, etc.
    - OS classes are coming
  - Advanced projects
    - MapReduce, Distributed Web Servers, etc.

- Community support
  - Supported by educational groups
  - SIGCSE paper, 3 CCSC workshops, etc.
  - Top ranked SIGCOMM Educational Resource
  - Coming in Computer Networking by Kurose & Ross
    - Most popular networking book!

Summary

Seattle testbed

- Real system deployed around the world
  - Geographic diversity, network diversity, device diversity...
  - Real networks!
- Battle tested educational platform!
  - Free, simple and safe to use
  - Open participation / open source
  - Broad community
  - Easy to drop into a class

https://seattle.cs.washington.edu/

Current Node Composition

Node Type	Quantity*
Testbed	791
University nodes	1720
Home machines	2849
Phone in name	67
Unknown nodes	3370
Total	8797

About 1% phones, 9% testbed, 20% university, 71% (likely) home nodes

* Nodes by IP address that accessed the Seattle software updater from Nov 2010 to Nov 2011. Location information by pygeoip.

Easy To Code

UDP ping server (4 LOC)

```
def got_message(srcip,srcport,mess,ch):
  sendmess(srcip,srcport,mess)
if callfunc == 'initialize':
  recvmess(getmyip(),54321,got_message)
```

UDP ping client (6 LOC)

```
def got_reply(srcip,srcport,mess,ch):
  print 'received:',mess,"from",srcip,srcport
if callfunc == 'initialize':
  recvmess(getmyip(),43210,got_reply)
  # send the second arg to the first arg's IP
  sendmess(callargs[0],54321,callargs[1],getmyip(), 43210)
  # exit in one second
  settimer(1,exitall,())
```

All Pairs Ping

Send periodic UDP pings
15 LOC

```
# 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'+encode_row(getmyip(), mycontext["neighborlist"
], mycontext['latency'].copy()))
    # sleep in between messages to prevent us 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
```

Handle incoming UDP pings
7 LOC

```
# Handle an incoming message
def got_message(srcip,srcport,mess,ch):
  if mess == 'ping':
    sendmess(srcip,srcport,'pong')
  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'):]
```

Format latency data into HTML
9 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
```

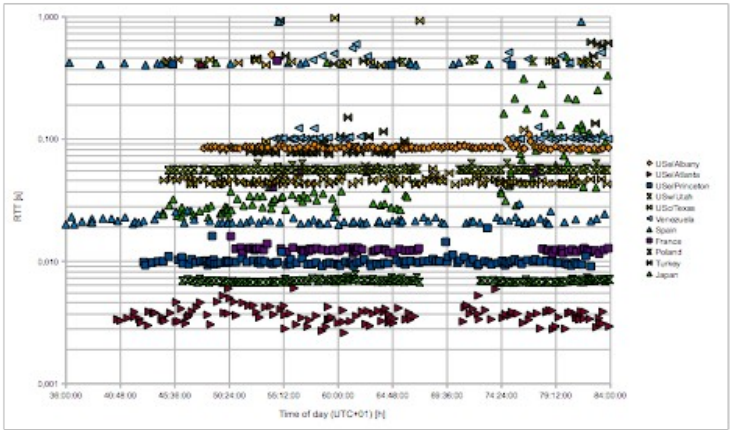
Return a webpage
11 LOC

```
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 + mycontext['row'][nodeip]+'\n'
    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()
```

Initialization
15 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("neighboriplist.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)
```

Research use



- Projects
  - YouTube CDN mapping
  - Wireless mobility patterns
  - Network heterogeneity
  - Overlay routing across P2P networks
  - P2P resource allocation fairness
  - Etc.

- Community support
  - Port to N900 by Nokia researchers
  - Runs on PlanetLab, Emulab, GpENI, DOME, etc.
  - GENI workshops, PyCon, etc.
  - NaCl integration by U Victoria / HP Labs
  - iPad 2 port, tun / tap support, Android, etc. by academics in Europe