

Seattle: Building a Million Node GENI by End User Opt-In

[Justin Cappos](#), Ivan Beschastnikh,
Arvind Krishnamurthy, Tom Anderson
Computer Science and Engineering
University of Washington

What is Seattle?

- Incentivized Platform for Resource Donation
 - End User Opt-In
 - Programming Language VM
 - Goals
 - Minimize Adoption Hurdles
 - Broad Set of Potential Adopters
 - Attractive to Multiple Communities
- Currently focused on Education

What is Seattle?

- **Incentivized Platform for Resource Donation**
 - End User Opt-In
 - Programming Language VM
 - **Goals**
 - Minimize Adoption Hurdles
 - Broad Set of Potential Adopters
 - Attractive to Multiple Communities
- **Currently focused on Education**

Donation

- Donation has worked well in other contexts

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

- SETI@Home, Protein Folding, Great Mersenne Prime Search, etc. (> 500K)

- Donation for Education

- Incentives for donating (10 for 1)

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

How Seattle Works

- An installer is downloaded
 - Requiring root / admin access would limit adoption
 - Can be installed in a restricted account
 - Multiple installs may happen on the same machine
 - Only one copy will run at a time
 - Credit for installation can be transferred
 - Restricting access to local users would complicate sharing
 - Node Manager allows remote users to execute programs
 - Limited platform support would prevent donation
 - Many platforms supported (Windows, Linux, Mac, BSD, mobile devices)

How Seattle Works (cont)

- Programs are run in a virtualized environment
 - Supporting diverse platforms raises portability concerns
 - VM ensures programs are portable
 - Students may have difficulty using the environment
 - Programming language VM based upon Python
 - Malicious programs could pose a security risk
 - Programs are checked for security
 - API calls are tightly controlled
 - Buggy programs could use too many resources
 - Resources are restricted by VM (similar to Xen, VMWare)

Common use

- Instructor sets up donations on computers
 - Credits can be shared with students
 - Students may also install
- Students
 - Request resources via website
 - Use shell to start / deploy program
 - View program output / tracebacks
- Students can also test on a local install

Demonstration

- Registration
- Download Installer
- Acquire resources
 - Use seattlegenit website
- Deploy all pairs ping
 - Use shell to locate and control resources

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'+encode_row(getmyip(), mycontext["neighborlist"]
), mycontext['latency'].copy()))
    # sleep for a while as from getruntime() we know of
    # the time when we sent the message
    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.startswith('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'):]
```

Handle incoming UDP pings

7 LOC

```
def encode_row(rowip, neighborlist, latencylist):
    retstring = "<tr><td>"+rowip+"</td>"
    for neighborip in neighborlist:
        retstring = retstring + "<td>"+str(latencylist[neighborip][:4])+"</td>"
    else:
        retstring = retstring + "<td>no ping</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><table border='1'>
    y information from "+getmyip()+"</td><td>"
    webpage = webpage + "<tr><td><td>"+ "</td><td>".join(mycontext['neighborlist']
t')+ "</td></tr>"

    for nodeip in mycontext['neighborlist']:
        if nodeip in mycontext['latency']:
            webpage = webpage + "<tr><td>"+str(nodeip)+"</td><td>"+str(mycontext['latency'][nodeip])+"</td></tr>"
        else:
            webpage = webpage + "<tr><td>"+str(nodeip)+"</td><td>No Data Reported</td></tr>"

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

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

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

Return a webpage

11 LOC

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

    # get the nodes to probe
    mycontext['neighborlist'] = []
    for line in file("neighbors.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

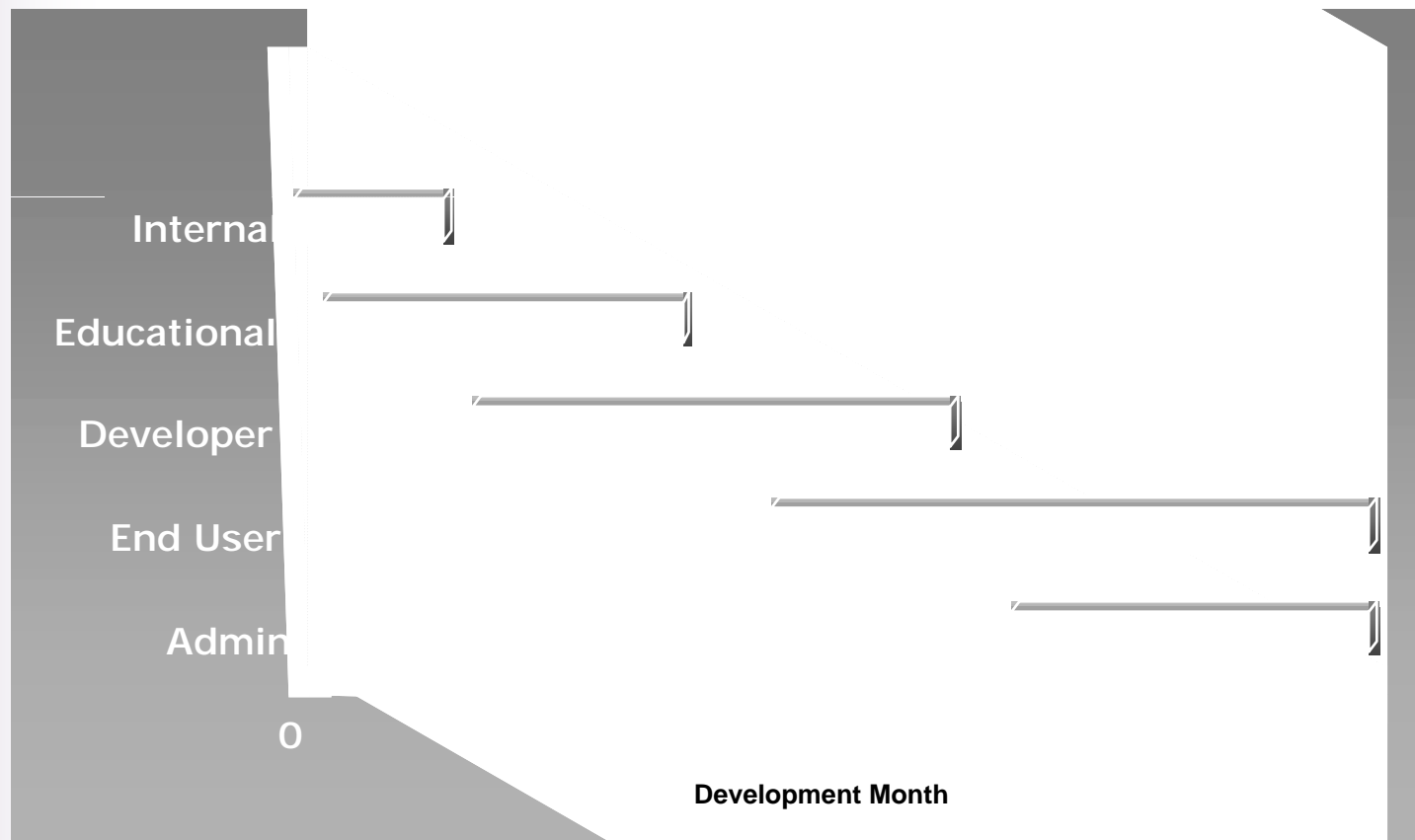
15 LOC

Incentives



QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Development Focus



Opt-In Types

- Wholesale
 - ISPs or universities
 - Network-based interposition
- End User
 - Individuals
 - Anyone who wants to join, can!!!
 - End host software-based interposition

End User Opt-In

- Opt-in mechanism may be specialized
 - Stake-holders must value the mechanism
 - GENI Public License allows forking!
- Stake-holders
 - User
 - Developer

Users' Opt-In

- User
 - Control what types of experiments run
 - Control what developers run experiments
 - Control where traffic can be sent
 - Control which traffic can be sent
 - Dynamically add and remove services

Developers' Opt-In

- Developer
 - Control their experiments
 - Control what their experiment interacts with
 - Control how their experiment is composed with services
 - Choose whether to utilize user resources

Summary

Seattle Testbed

- Safely uses donated resources
- Stays current with the latest technology
- As of Mar 2009, we have resources on over 1200 computers, 100s of universities, 6 continents

End-user opt-in

- Must appeal to developers and users
- Easy, safe, valuable
- Flexible controls with few restrictions

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