Seeding a Community Cloud Through End User Participation

Justin Cappos
NYU Poly
Computer Science and Engineering
Vibrant Ecosystem ...
Or Squandered Resources?
Our Vision

Commoditize Resources on End Hosts
Why Are End Hosts Useful?

- 3-4 orders of magnitude more end hosts
- P2P responsible for about 38% of Internet traffic [Cisco 2009]
- BitTorrent has 100x more distribution capacity than YouTube
- 15-30% of web traffic comes from the edge of the network [Akamai]
- Lots of growth and focus at the edges
- Proximity necessary to minimize latency
  - The speed of light isn't improving!
Seattle Model

• Scales with demand for free
• Developers and end users have power
• No vendor API lock-in

[Cappos SIGCSE 09, Seattle website]
Our First Step

• **Build a testbed from end hosts!**
Seattle Testbed

Open peer-to-peer application hosting

• Unknown users donate resources (VMs)
• Unknown developers push code
• Tit-for-tat like model for resource sharing
• Commonly used like a P2P PlanetLab

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

• Classroom experience
  • Released in Spring 2009
  • Used in almost two dozen classes (more in progress)
  • 3 tutorials, 3 library references, etc.
  • 8 battle tested assignments
    • Overlay routing, flow control, NAT / Non-transitive connectivity, Chord (DHT), reference monitors, etc.
    • Security, OS classes are coming
  • Advanced projects
    • MapReduce, Distributed Web Servers, etc.

• Community support
  • Supported by educational groups
  • SIGCSE paper, several workshops, etc.
  • Top ranked SIGCOMM Educational Resour
  • Coming in Computer Networking by Kurose & Ross
    • Most popular networking book!
Research use

- **Projects**
  - YouTube CDN mapping
  - Wireless mobility patterns
  - Network heterogeneity
  - Overlay routing across P2P networks
  - P2P resource allocation fairness
  - Multiple top conference paper submissions

- **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
Commercial use

- **Projects**
  - CDN
  - P2P Backup
  - OpenFlow cloud control
  - Single Cloud Image
  - Zenodotus service
  - End user web hosting

- **Community support**
  - Network troubleshooting by U Victoria / HP Labs
  - Extra traffic relays for Tor (pending)
  - CDN for major content provider (pending)
  - AppStore for networks (pending)

Commercial use is very early stage!
## Current Node Composition

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Quantity*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testbed</td>
<td>791</td>
</tr>
<tr>
<td>University nodes</td>
<td>1720</td>
</tr>
<tr>
<td>Home machines</td>
<td>2849</td>
</tr>
<tr>
<td>Phone in name</td>
<td>67</td>
</tr>
<tr>
<td>Unknown nodes</td>
<td>3370</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8797</strong></td>
</tr>
</tbody>
</table>

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.
Demonstration

Typical Seattle Workflow
• Registration
• Download installer
• Demo HuXiang
• Acquire resources
  • Use SeattleGENI website
• Deploy all pairs ping
  • Use shell to locate and control resources

Demo HuXiang!
Seattle Research Challenges

• Network Heterogeneity
  • This talk!

• Need to execute untrusted code on end hosts
  • RepyV2 Security Layers [Cappos CCS 10]

• Performance isolation
  • Repy VM

• Heterogeneous OSes → uniform VM
  • CheckAPI

• Legacy code → sandbox isolation
  • Lind

• Need support for fault-tolerance and scalability, resource allocation, incentives, usable policies, ...
A Long Time Ago...

- The Network was simple
  - Nodes had fixed locations
- Every node had a public IP
- Relatively little heterogeneity in device types and networks
- Applications used a (semi-)standard API
  - Berkeley / BSD Socket API
Today...

- The Network is complex
  - Nodes move around

- Many nodes behind NATs / Firewalls / IPSes / proxies

- Diversity evident throughout devices and networks

- Tomorrow will be worse!

- Applications still use the (semi-)standard API
  - Berkeley / BSD Socket API
How Do Programmers Deal With Network Diversity?

- Network libraries
  - Usually target a single issue
    - Rarely a few related items
  - Change API semantics
    - Require porting the application
      - Hence usually only done for 'popular' configurations
      - Focus on availability, not optimality
    - Complicate debugging

Multiple problems often co-exist!
Can We Use Multiple Libraries?

- Network libraries
  - Semantics change subtly
    - Each library has slightly different semantics / behaviors
  - Combining means porting one to the other
    - The order matters
    - You may have to port both directions
Are Multiple Libraries Needed in Practice?

Skype

- Forward Error Correction.
- NAT traversal.
- Mobility support.
- Checks rate limiting
- Variable bit-rate compression
- Encryption
- Disconnection / reconnection (New!)
Why Hasn't The API Evolved?

- Applications need to be backward compatible
  - Networks too!
What Are Our Goals?

- Add functionality
  - Must add value
    - Composability
    - No ordering constraints
- Transparency
  - No application changes
  - No network changes
- Correctness
  - Semantic bugs must be easy to pinpoint
Shims

- Adds functionality while enforcing semantics.
  - Verification using novel modeling techniques
  - Ensures application need not be modified
- Instantiated per connection / datagram.
  - Inserted between app and net API.
- Compatible with legacy everything
  - Even when one sided
    - ... early adopters still can benefit!
Networks are complex

- We want to use multiple shims together!
- Build a shim-stack
  - Push, pop, etc.
- Any shim stack is transparent because of semantic consistency.
Balancing a Shim Stack

- Application
- Compression Shim
- Encryption Shim
- Network API (OS)

- Application
- Compression Shim
- Decryption Shim
- Network API (OS)

Seattle
Internet
Williamstown
What Have We Done?

- ~20 shims (needed for Seattle)
  - NAT Traversal, Mobility, FEC, Encryption, One-hop detour, compression, ...
- Multiple ways to connect applications
  - Proxy, LD_PRELOAD
- Legacy hooks
  - Map DNS names to shimstacks via DHT
    - Layers nicely into the Internet
- Taxonomy: Split, Branch, Decider
Real-World Scenario: FEC Shim

- Works well on a network with excess bandwidth and high loss rate.
- Tested on WiMAX testbed which showed lossy network behavior.

<table>
<thead>
<tr>
<th>Packet Length (Bytes)</th>
<th>FECShim,2</th>
<th>FECShim,3</th>
<th>NoopShim</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-799</td>
<td>1.042</td>
<td>1.143</td>
<td>2.623</td>
</tr>
<tr>
<td>800-1199</td>
<td>1.602</td>
<td>1.773</td>
<td>3.279</td>
</tr>
<tr>
<td>1200-1600</td>
<td>1.659</td>
<td>1.935</td>
<td>5.467</td>
</tr>
</tbody>
</table>

Observed packet loss was reduced by more than half when using FEC shim.
Shims with Legacy Application

- Added FEC to VLC using shims
  - Resolved a 5 year old feature request
  - Lines of code changed: 0 (proxy)

Without FEC  |  With FEC shim
Real-World Scenario: Compression Shim

Experimental setup:

- Apache file server hosting files of various size and type.
- Client downloads files multiple times using wget.
- Redirect network traffic through a shim capable proxy.
- Download files using shim capable proxy vs direct connection.
Real-World Scenario: Compression Shim

Dashed lines represent text files and solid lines represent PDF files.
Figure shows average time to transfer a 4MB file. The Compression shim and the OneHopDetour shim individually show improvement over direct connection and Noop shim. However combined together Compression shim + OneHopDetour shim reduces transfer time more than each individual shim!
Conclusion

- Network is complex
  - Need to add functionality to apps
- Shims preserve semantics
  - Applications work!
  - Shim stack combines functionality.
- Shims help applications in real environments!
- Backwards compatible with the Internet
Software Provisioning Work

• Application monitoring [NSDI 08] and redeployment [MIDDLEWARE 2009]
• Package management for the cloud [USENIX 2005, LISA 2007]
  • Package manager / software updater security:
    • mirrors [CCS 08]
    • key compromise [CCS 10]

Future / On-going work
• TUF https://www.updateframework.com
• Automatic detection of mirror misbehavior
• upPIR -- Privately retrieving content
Seattle testbed

- Real system deployed around the world
  - Geographic diversity, network diversity, device diversity...
  - Lots of devices
- Battle tested educational platform!
- Good research platform
  - (6 papers published, many more on the way / under submission)
- Interesting research problems
  - Portability
  - Network Heterogeneity
  - Legacy cloud containers (SFI meets Seattle VM)
- Etc.

https://seattle.cs.washington.edu/
Verifying Semantic Consistency

- Use CheckApi Shim – a model-based testing.
  - Ensures that a shims response to a network call is consistent with the expected behavior of the network API.
- Ensure shims can be stacked in any permutation and still be transparent to the application.

\[
\begin{array}{cccc}
C & C & E & R \\
E & R & C & E \\
R & E & C & E \\
\end{array}
\]

C = Compression-zlib
E = Encryption-rot
R = RateLimit
# Example Shims

<table>
<thead>
<tr>
<th>Shim</th>
<th>Description</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckApi</td>
<td>Validates network API semantics</td>
<td>725</td>
</tr>
<tr>
<td>Coordination</td>
<td>Builds a balanced shim stack</td>
<td>128</td>
</tr>
<tr>
<td>Compression</td>
<td>Compresses data using zlib or snappy</td>
<td>201</td>
</tr>
<tr>
<td>DataLimit</td>
<td>Restricts volume of traffic over a period</td>
<td>78</td>
</tr>
<tr>
<td>RateLimit</td>
<td>Restricts bandwidth utilized</td>
<td>112</td>
</tr>
<tr>
<td>ForwardErrorCorrection</td>
<td>Writes error recovery UDP datagram</td>
<td>129</td>
</tr>
<tr>
<td>Logall</td>
<td>Logs all network calls and traffics</td>
<td>162</td>
</tr>
<tr>
<td>Duplicate-UDP</td>
<td>Sends data over multiple shimstacks</td>
<td>139</td>
</tr>
<tr>
<td>Splitter-TCP</td>
<td>Splits data, favors fast shimstacks</td>
<td>274</td>
</tr>
<tr>
<td>NAT-TURN</td>
<td>TURN-like protocol for NAT traversal</td>
<td>139</td>
</tr>
<tr>
<td>One-Hop-Detour</td>
<td>Routes traffic through a relay</td>
<td>98</td>
</tr>
<tr>
<td>Encrypt-rsa</td>
<td>Encrypts traffic using RSA</td>
<td>206</td>
</tr>
<tr>
<td>Encrypt-rot</td>
<td>Encrypts with a Caesarean cipher</td>
<td>16</td>
</tr>
</tbody>
</table>
Coordinating Shim Stacks

- Subproblem: Describing a shim stack
  - Serialize and transmit the string.
- Global naming service allows coordination.
  - Essential for nodes that can't talk directly.

NodeA: “(ShimA)(ShimB,ShimB_args)”
NodeB: ”(ShimD)”
NodeC: “(ShimD)(ShimA)”

DHT server with key value pair.

Diagram showing the flow of network traffic between server and client applications through shimmed layers at Nodes 1, 2, and 3.