Phoenix - Survive Internet Catastrophes

Yanhua Mao
Shan Yan
Siddhartha Saha

maoyanhua@cs.ucsd.edu
shyan@ucsd.edu
ssaha@ucsd.edu
Outline

- Motivation
- Phoenix
- Project contribution
- Methodology
- Static simulation
- Dynamic simulation
- Conclusion
Internet is highly vulnerable to worm

- **Code Red (CRv2)**
  - Infect 359,000 hosts in less than 14 hours

- **Slammer**
  - Infect 90% vulnerable hosts in 15 minutes

- **Flash worm**
  - Can probably infect 95% vulnerable host in 30 seconds.
Protect user data from worm attack

- Worm can potentially destroy user data
- Traditional backup method
  - If at most N hosts can fail at the same time, then we backup data in N+1 hosts
  - N is too large for the traditional method to be feasible
- Solution
  - Informed replication
Observation and Idea

- Properties of worms
  - Worms usually attack only single service (vulnerability)
  - When attack multiple vulnerabilities, it aim at one operating system

- Vulnerability as Attributes
  - Operating system & open services (ports)

- Idea
  - Backup my data on hosts that do not share vulnerabilities with me.
  - Need diversity
### Host Diversity Study on UCSD campus

<table>
<thead>
<tr>
<th>OS</th>
<th>Count</th>
<th>Number</th>
<th>Port</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>1604(54.1%)</td>
<td>139(netbios-ssn)</td>
<td>1640(55.3%)</td>
<td></td>
</tr>
<tr>
<td>Solaris</td>
<td>301(10.1%)</td>
<td>135(epmap)</td>
<td>1496(50.4%)</td>
<td></td>
</tr>
<tr>
<td>Max OS X</td>
<td>296(10.0%)</td>
<td>445(microsof-ds)</td>
<td>1157(39.0%)</td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td>296(10.0%)</td>
<td>22(sshdd)</td>
<td>910(30.7%)</td>
<td></td>
</tr>
<tr>
<td>Max OS</td>
<td>204(6.9%)</td>
<td>111(sunrpc)</td>
<td>750(25.3%)</td>
<td></td>
</tr>
<tr>
<td>FreeBSD</td>
<td>66(2.2%)</td>
<td>1025(active directory)</td>
<td>735(24.8%)</td>
<td></td>
</tr>
<tr>
<td>IRIX</td>
<td>60(2.0%)</td>
<td>25(smtp)</td>
<td>575(19.4%)</td>
<td></td>
</tr>
<tr>
<td>HP-UX</td>
<td>32(1.1%)</td>
<td>80(httpd)</td>
<td>534(18.0%)</td>
<td></td>
</tr>
<tr>
<td>BSD/OS</td>
<td>28(0.9%)</td>
<td>21(ftp)</td>
<td>528(17.8%)</td>
<td></td>
</tr>
<tr>
<td>Tru64 Unix</td>
<td>22(0.7%)</td>
<td>515(printer)</td>
<td>462(15.6%)</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Motivation
- **Phoenix**
- Project contribution
- Methodology
- Static simulation
- Dynamic simulation
- Conclusion
Phoenix Recovery System

- Phoenix is a peer-to-peer system in which a host backup its data on its peer

Goal

- Try to find server set (Core) to keep user data from lost against worm attack
Core

- **Core**
  - A minimal set of hosts such that for any attribute, at least one host in the set does not have this attribute.
  - At least one host can survive any worm attack that only exploits one vulnerability at a time.

- **Example**
  - **Attributes**
    - OS={Unix, Windows}
    - Web Server={Apache, IIS}
    - Web Browser={IE, Netscape}
  - **Host:**
    - H1={Unix, Apache, Netscape}, H2={Windows, IIS, IE}
    - H3={Windows, IIS, Netscape}, H4={Windows, Apache, IE}
  - **Core**
    - Cores={{H1,H2}, {H1,H3,H4}}
Finding Core

- Finding optimal minimal core is NP-hard
- Heuristics
  - Random
  - Uniform
  - Weighted
  - DWeighted
- Metrics
  - Average core size (core size)—Effectiveness
    - Core size averaged across all the hosts
  - Average coverage (coverage)—Effectiveness
    - Coverage: percentage of hosts that successfully find their cores
  - Load variance—Fairness
    - Variance of number of other hosts that any host services
Windows, IIS, IE

Linux, Netscape, Apache

“Where to find my core?”

Heuristics:
• Random
• Uniform
• Weighted
• DWeighted
Average Core Size

![Average Core Size Graph](image-url)
Average Coverage

![Graph showing average coverage vs load limit for different methods: Random, U.OS, W.OS, DW.OS. The graph illustrates the improvement in average coverage as the load limit increases.]
Average Load Variance

![Graph showing variance vs. load limit for different scenarios: Random, U.OS, W.OS, DW.OS.](image)
Existing problem

- What if hosts join and leave the system?
- What if some hosts are malicious?
- Can we get a better load variance?
Our Project Contribution

- *Weighted*, *Uniform* algorithm
  - better load variance
- Dynamic property simulation
  - Hosts joining and leaving the system
- N-resilient core algorithm
  - Better reliability
  - Handle malicious hosts
- Lottery algorithm
  - Better load variance
Outline

- Motivation
- Phoenix
- Project contribution
- Methodology
- Static simulation
- Dynamic simulation
- Conclusion
Simulator implementation

- Timed-event driven
- Simulation world
  - Objects, World timer, Events Queue, ......
- Simulation Objects
  - Host, HostJoiner, Statistician, ......
- Events
  - Request, reply, timeout, ......
Simulation Settings

- System population: 3000 hosts
- Initial population: 20% hosts
- 80% hosts join the system according to Poisson process
- Hosts leave the system according to Poisson process, four times slower than the join rate
- Introduce 5% malicious hosts
  - Behave like normal hosts, except drop user’s data instead of backing it up
- Focus on coverage and load variance
Outline

- Motivation
- Phoenix
- Project contribution
- Methodology
  - Static simulation
  - Dynamic simulation
- Conclusion
Static simulation

- Simulate four heuristics
- Compare our results with the existing paper to test the correctness of our simulator
- Problem of *Uniform* and *Weighted*
  - High load variance
  - Consider both container level and sub-container level when selecting core
- Propose *Uniform* and *Weighted*
  - Consider only container level
  - Low load variance
Windows, IIS, IE

“Where to find my core?”
Static simulation results

**Average Coverage**

- Weighted
- Uniform
- Weighted*
- Uniform*

**Standard Deviation of Core Size**

- Weighted
- Uniform
- Weighted*
- Uniform*

**Average Loads**

- Weighted
- Uniform
- Weighted*
- Uniform*

**Standard Deviation of Loads**

- Weighted
- Uniform
- Weighted*
- Uniform*
Outline

- Motivation
- Phoenix
- Project contribution
- Methodology
- Static simulation
- Dynamic simulation
- Conclusion
Handle host leave

Problem
- One of my server leave, I need to rebuild my core
- One of my client leave, I can discard the data to serve more client.

Solution
- Inform its servers and clients of its leave

What if the message is lost?
- Timeout mechanism
Inform host leave - Coverage

Weighted* algorithm, 1-core, No malicious host, Load Limit=6
Inform host leave - Load variance

Weighted* algorithm, 1-core, No malicious host, Load Limit=6
Handle malicious host

- Malicious Host
  - Behave like normal hosts, except drop user’s data instead of backing it up

- N-resilient core
  - The N-resilient core for a host is a minimal set of hosts such that, for any attributes of any host in this core, there are at least N hosts in the core that do not have such attribute.
  - Any host that successfully finds its N-resilient core can resist any N worms exploiting any N attributes in the attribute space simultaneously.
Introduce 5% malicious hosts

Weighted* algorithm, 1-core, Load Limit=6

Weighted* algorithm, 2-core, Load Limit=6
Handling load variance

- **Problem**
  - Newly joined host has lower load, but is treated the same in previous algorithms

- **Solution - Lottery heuristic**
  - Hosts adjust their tickets when their loads change (hold $2^{L-N}$ tickets, $N=$ current load, $L=$ load limit)
  - The probability a host is chosen in proportional to the number of tickets it holds
Weighted* vs. Lottery

Weighted* algorithm, 2-core, Load Limit=6

Lottery algorithm, 2-core, Load Limit=6
How to implement Lottery algorithm

- Forwarded based
  - Tag request with priority $P$
  - A host will accept a request with probability $2^{P-N}$, $N$ is its current load
  - With probability $1 - 2^{P-N}$, a host will forward the request to another host and increase $P$ by 1.

- ID-based (approximate lottery)
  - ID of a host: \[
  \text{Hash(OS)} \quad \text{Cur Load} \quad \text{random}
  \]
  - Host dynamic adjusts its ID according to its load
  - Host selects $\text{CurLoad}=N$ with probability $2^{-N-1}$
Putting it together

Lottery algorithm, 2-core, 5% malicious host, Load Limit=6
Outline

- Motivation
- Phoenix
- Project contribution
- Methodology
- Static simulation
- Dynamic simulation
- Conclusion
Contribution

- Dynamic simulator implementation
- Inform host leave information
- N-resilient Core
- Weighted* and Lottery algorithm
Future work

- Study the behavior of other kinds of malicious host
- Try to detect malicious host
- Real world experiment
References


Thank You!
Q&A