On the Accurate Identification of Network Service Dependencies in Distributed Systems

Barry Peddycord III  
NC State University  
bwpeddyc@ncsu.edu  
@isharacomix

Dr. Peng Ning  
NC State University  
pning@ncsu.edu

Dr. Sushil Jajodia  
George Mason University  
jajodia@gmu.edu

Motivation  
NSDMiner  
Evaluation  
Deployment  
Conclusions

NSDMiner
- Non-invasive and fairly accurate
- Open Source Python Module
  - http://nsf.net/nsdm
- Future work includes
  - Making it work in real time
  - Identifying remote-remote dependencies

USENIX LISA '12  
#NSDMiner

This work is supported by the U.S. Army Research Office (ARO) under MURI grant W911NF-09-1-0525
Motivation

Example

Problem

Network Service Dependencies
- Defined by configuration parameters and source code
- Each service does it differently!
- Often very intricate and subtle
- Hard to keep track
- How good are YOUR docs?
- Want to identify them automatically

Why bother?

Know thyself
- If dependencies are discovered after a failure occurs, it's too late
- Knowing in advance
  - Improves response time
  - Allows pro-active action to be taken on mission-critical services
- Networks are dynamic

Prior Work

Two Paradigms
- Patterns in the behavior of the network can model its structure
- Previous approaches fall into two categories:
Network Service Dependencies

- Defined by configuration parameters and source code
  - Each service does it differently!
- Often very intricate and subtle
- Hard to keep track
  - How good are YOUR docs?
- Want to identify them automatically
A network service is a software application that runs on a server and listens on a port for connections from other applications.
A dependency is a relationship between two services A and B such that A (the depending service) contacts B (the depended service) to complete a task.
Problem

Network Service Dependencies

- Defined by configuration parameters and source code
  - Each service does it differently!
  - Often very intricate and subtle
  - Hard to keep track
  - How good are YOUR docs?
- Want to identify them automatically
Why bother?

Know thyself

- If dependencies are discovered after a failure occurs, it's too late
- Knowing in advance
  - Improves response time
  - Allows pro-active action to be taken on mission-critical services
- Networks are dynamic
Prior work

Two Paradigms

- Patterns in the behavior of the network can model its structure
- Previous approaches fall into two categories:

**Host-Based**

*Accurate, but intrusive*
- Install an agent (i.e. a kernel module) to track socket/application behavior
  - Magpie [OSDI 2004]
  - Pinpoint [NSDI 2004]
  - MacroScope [CoNEXT 2009]
- Intrusiveness makes them unattractive
  - Security risks
  - Resource contention

**Network-Based**

*Treat hosts as black boxes*
- Data-mine on-the-wire network traffic to extract relationships
  - Sherlock [SIGCOMM 2007]
  - eXpose [SIGCOMM 2008]
  - Orion [OSDI 2008]
  - NSDMiner [INFOCOM 2011]
- High-false positive/false negative rates
Host-Based

Accurate, but intrusive

- Install an agent (i.e. a kernel module) to track socket/application behavior
  - Magpie [OSDI 2004]
  - Pinpoint [NSDI 2004]
  - Macroscope [CoNEXT 2009]
- Intrusiveness makes them unattractive
  - Security risks
  - Resource contention
Network-Based

Treat hosts as black boxes

- Data-mine on-the-wire network traffic to extract relationships
  - Sherlock [SIGCOMM 2007]
  - eXpose [SIGCOMM 2008]
  - Orion [OSDI 2008]
  - NSDMiner [INFOCOM 2011]
- High-false positive/false negative rates
Intuition

Why logs?
Two Important Properties
- Not all nested flows are equal
- Glue conditions with more evidence more weight
  - "Every other flow" means more when it’s 10000 than 100
  - Earlier flows are worth less
  - Is 90% less convincing than 90%2

Ranking

Confidence equals:
\[
\frac{\log_2 \text{number of times } A \text{ is accessed}}{\log_2 \text{number of nested } A \rightarrow B \text{ flows}}.
\]

Post-processing

Given a Communication Graph
- Less-used services are vulnerable to false positive, false negative
- Post-processing uses overall structure to fine-tune results

The Output

List of Dependency Candidates
- Returns each network service and all of its dependency candidates
- Dependencies ordered by most-likely to least-likely
- Should be verified by hand, so a few false positives are acceptable
Intuition

C

Client

A

Fileservlet

B

Kerberos
**Ranking**

**Confidence equals:**

\[
\frac{\log(\text{number of times A is accessed})}{\log(\text{number of nested A->B flows})}
\]

weight(A) = 1000

weight(B) = 300

weight(C) = 1200

weight(D) = 1500
Post-processing

Given a Communication Graph

- Less-used services are vulnerable to false positive, false negative
- Post-processing uses overall structure to fine-tune results
Inference

**Intuition**

*Consider a Web Host*
- Many servers are configured the same way (HTTPD) with the same dependencies (MySQL, SMTP, etc)
- Some are more popular than others, having more traffic
- Identify dependencies of less used servers by identifying ‘similar’ services

**Example**

![Diagram of network with nodes and edges indicating similarity and agreement]

**Algorithm**

- Identify all pairs of similar services above a certain similarity threshold
- Combine pairs into similarity groups
- Calculate agreement on dependency candidates
- Infer dependencies from members of similarity group to most agreed-upon candidates
Consider a Web Host

- Many servers are configured the same way (HTTPD) with the same dependencies (MySQL, SMTP, etc)
- Some are more popular than others, having more traffic
- Identify dependencies of less used servers by identifying 'similar' services
Example

Observed traffic:

\[
\text{Similarity}(X,Y) = \frac{\text{shared deps}}{\text{total deps}}
\]

75% Agreement

100% Agreement

Ground truth: All A's depend on D's
Intuition

*Backups and Load-Balancing*

- In a load-balancing cluster, a depending service will eventually utilize all cluster nodes.
- In a backup-cluster, a service will use the primary nodes until they fail, then move to backup nodes.
- In both cases, if a service uses one node in a cluster, it uses them all.

Example

```
A1 -+ A2
    /  \
   /    \
B1   B2
    |    |
    |    |
    |    |
    |    |
    A3   A4
```

Algorithm

- Count the number of times that pairs of services are depended upon by the same service.
- For services that have support above a certain threshold, these services are considered to be in clusters.
- Re-interpret services that depend on services in clusters as depending on the entire cluster itself.
Intuition

Backups and Load-Balancing

• In a load-balancing cluster, a depending service will eventually utilize all cluster nodes
• In a backup-cluster, a service will use the primary nodes until they fail, then move to backup nodes
• In both cases, if a service uses one node in a cluster, it uses them all
Example

A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4

Diagram showing connections between nodes with labels A_1 to A_4 and B_1 to B_4.
The Output

List of Dependency Candidates

- Returns each network service and all of its dependency candidates
- Dependencies ordered by most-likely to least-likely
- Should be verified by hand, so a few false positives are acceptable
<table>
<thead>
<tr>
<th>Services</th>
<th>Instances</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Webservice (80,443)</td>
<td>4</td>
<td>2 DNA, DBMS</td>
</tr>
<tr>
<td>Webservice (80)</td>
<td>1</td>
<td>1 DNA</td>
</tr>
<tr>
<td>SSH (realm-4) (22)</td>
<td>5</td>
<td>2 Kerberos, DNS</td>
</tr>
<tr>
<td>SSH (realm-5) (22)</td>
<td>17</td>
<td>3 Kerberos, DNS, LDAP</td>
</tr>
<tr>
<td>SVN (8443)</td>
<td>1</td>
<td>4 DNS, LDAP, Port Mapper, RPC</td>
</tr>
<tr>
<td>Proxy DHCP (4011)</td>
<td>1</td>
<td>2 DNS, LDAP</td>
</tr>
<tr>
<td>DHCP (68)</td>
<td>1</td>
<td>1 DNA</td>
</tr>
<tr>
<td>Email (25)</td>
<td>1</td>
<td>2 Mail exchange, DNS</td>
</tr>
<tr>
<td>Endpoint Mapper (135)</td>
<td>2</td>
<td>3 DNS, AD, Kerberos</td>
</tr>
<tr>
<td>WDS (RPC)</td>
<td>1</td>
<td>5 DNS, AD (LDAP, port mapper, RPC, kerberos)</td>
</tr>
<tr>
<td>DFS Replication (RPC)</td>
<td>2</td>
<td>5 DNS, AD (LDAP, port mapper, RPC, kerberos)</td>
</tr>
<tr>
<td>SMB (445)</td>
<td>2</td>
<td>5 DNS, AD (LDAP, port mapper, RPC, kerberos)</td>
</tr>
<tr>
<td>TFTP (69)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Database (3306)</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Clustering

![Cluster Graphs]

- [Left Graph]:
  - X-axis: Discovered Clusters
  - Y-axis: Correct Clusters
  - Legend:
    - Support 2
    - Support 4
    - Support 6
    - Support 8

- [Right Graph]:
  - X-axis: Confidence Threshold
  - Y-axis: Number of Candidates
  - Legend:
    - Support 2
    - Support 4
    - Support 6
    - Support 8
    - No Clusters
Deployment

Open Source

- Available on SourceForge
  - Written as a Python Module
  - 'import nsdminer'
  - Comes with a command-line interface for processing data

What's needed

- Collect the Data
  - Collect all network traffic from network switches
  - Export netflows from switches
  - Use packet mirroring: forward and save all pcap headers of packets
  - Usually a week of packets is needed

Using NSDMiner

- Just install and run!
  - Run `nsdminer` to process your data
  - Command line options let you choose various parameters
  - Detailed in the paper and README
  - Output will be a list of services, dependencies, and confidence values

Going Beyond

- Extend and improve NSDMiner using the features of the 'ndminer' Python library.
- Use it in your own networks and let us know how it works for you in the SourceForge forum!
Open Source

Available on SourceForge

- Written as a Python Module
  - 'import nsdminer'
- Comes with a command-line interface for processing data
What's needed

Collect the Data

- Collect all network traffic from network switches
  - Export netflows from switches
  - Use packet mirroring - forward and save all pcap headers of packets
- Usually a week of packets is needed
Using NSDMiner

Just install and run!

• Run 'nsdminer' to process your data
  • Command line options let you choose various parameters
  • Detailed in the paper and README
• Output will be a list of services, dependencies, and confidence values
Going Beyond

- Extend and improve NSDMiner using the features of the 'ndminer' Python library.
- Use it in your own networks and let us know how it works for you in the SourceForge forum!
Conclusions

**NSDMiner**

- Non-intrusive and fairly accurate
- Open Source Python Module
  - [http://sf.net/p/nsdminer](http://sf.net/p/nsdminer)
- Future work includes
  - Making it work in real time
  - Identifying remote-remote dependencies
On the Accurate Identification of Network Service Dependencies in Distributed Systems

Barry Peddycord III  
NC State University  
bwpeddy@cncsu.edu  
@isharacomix

Dr. Peng Ning  
NC State University  
pning@ncsu.edu

Dr. Sushil Jajodia  
George Mason University  
jajodia@gmu.edu

Motivation

NSDMiner

Evaluation

Deployment

Conclusions

NSDMiner
- Non-invasive and fairly accurate
- Open Source Python Module
- http://netlab.gmu.edu/NSDMiner
- Future work includes
- Making it work in real time
- Identifying remote-to-remote dependencies

USENIX LISA '12

#NSDMiner

This work is supported by the U.S. Army Research Office (ARO) under MURI grant W911NF-09-1-0525