Learning to Extract Router Names from Hostnames

Matthew Luckie - University of Waikato
Bradley Huffaker - CAIDA / UC San Diego
k claffy - CAIDA / UC San Diego

IMC 2019, October 22nd 2019
Motivation

• Router alias resolution possible on **subset of routers**
  - Techniques rely on implementation artifacts (hacks)
    • **Common source address** in ICMP error message
    • **IP-ID assignments** from a counter
    • **IP pre-specified timestamp** option behavior

What if we could learn properties of networks from the subset of routers where alias resolution works, and use that property to reason about other routers in those networks?
Intuition: Naming Conventions

Router #1: esr1jfk2
  esr1-ge-5-0-0.jfk2.savvis.net
  esr1-ge-5-0-6.jfk2.savvis.net
  esr1-ge-7-0-5.jfk2.savvis.net

Router #2: esr2pax
  esr2-xe-4-0-0.pax.savvis.net
  esr2-xe-4-0-1.pax.savvis.net
  esr2-xe-8-0-1.pax.savvis.net

Router #3: esr1pax
  esr1-xe-4-0-0.pax.savvis.net
  esr1-xe-4-0-1.pax.savvis.net
  esr1-xe-8-0-0.pax.savvis.net

Router #4: das1nj2
  das1-v3005.nj2.savvis.net
  das1-v3006.nj2.savvis.net
  das1-v3007.nj2.savvis.net

Router #5: das2loc2
  das1-v3005.oc2.savvis.net
  das1-v3007.oc2.savvis.net
  das1-v3008.oc2.savvis.net

Router #6: das2nj2
  das2-v3009.nj2.savvis.net
  das2-v3010.nj2.savvis.net
  das2-v3011.nj2.savvis.net

^([a-z]+[\d]+)-.+\.(\[a-z\d\]+).savvis\s.net$
## Intuition: Naming Conventions

(1) The regex extracts the same value from a set of hostnames associated with the same router.
**Intuition: Naming Conventions**

(1) The regex extracts the same value from a set of hostnames associated with the same router.

(2) The values are unique to each router.

```
Router #1: esr1jfk2
esr1-ge-5-0-0.jfk2.savvis.net
esr1-ge-5-0-6.jfk2.savvis.net
esr1-ge-7-0-5.jfk2.savvis.net

Router #2: esr2pax
esr2-xe-4-0-0.pax.savvis.net
esr2-xe-4-0-1.pax.savvis.net
esr2-xe-8-0-1.pax.savvis.net

Router #3: esr1pax
esr1-xe-4-0-0.pax.savvis.net
esr1-xe-4-0-1.pax.savvis.net
esr1-xe-8-0-0.pax.savvis.net

Router #4: das1nj2
das1-v3005.nj2.savvis.net
das1-v3006.nj2.savvis.net
das1-v3007.nj2.savvis.net

Router #5: das2oc2
das1-v3005.oc2.savvis.net
das1-v3007.oc2.savvis.net
das1-v3008.oc2.savvis.net

Router #6: das2nj2
das2-v3009.nj2.savvis.net
das2-v3010.nj2.savvis.net
das2-v3011.nj2.savvis.net

^([a-z]+\d+)-+.+(^[a-z\d]+)\.savvis\.net$```
Intuition: Naming Conventions

(1) The regex extracts the same value from a set of hostnames associated with the same router

(2) The values are unique to each router

(3) The regex extracts names for multiple routers in the suffix

**Suffix examples:**
savvis.net  he.net
att.net    alter.net
High-level Approach

• Infer if an operator embeds information identifying individual routers in PTR hostname records for router interfaces

• **Input:**

  - Mozilla [public suffix list](https://publicsuffix.org) to identify where domains can be registered (.net, .org, .nz, .co.nz, .geek.nz)
  
  - Hostnames for router interfaces observed by traceroute (PTR records)
  
  - Router alias inferences MIDAR, mercator, speedtrap

• **Output:** regular expressions that extract router names
CAIDA Internet Topology Data Kit (ITDK)

- Heavily curated router-level topology dataset published roughly twice a year
  - IPv4 Routers, with aliases inferred by MIDAR and Mercator
  - Links between routers
  - Router geolocation
  - Router ownership
  - DNS hostnames

• 16 ITDK datasets between July 2010 to April 2019
  - 2 include IPv6 routers inferred by speedtrap (August 2017 and January 2019)
Contribution: Hoiho

(Holistic Orthography of Internet Hostname Observations)

• We design and implement a method to accurately infer regexes that extract router names from hostnames

• 8 stage learning process

• Implemented in C, parallel threads of execution

Image: Brent Beaven
Department of Conservation (New Zealand)
Key Results

• We applied Hoiho to 16 ITDKs across 9 years to infer “good” conventions for 2550 suffixes

  - **Good conventions:** PPV > 90% and correctly cluster interfaces on at least three routers.

  - **Poor conventions:** the suffix has no convention that embeds a router name in the hostname, or less than three routers.

• We validated 11 conventions with 10 network operators
Alias Resolution Gain on April 2019 ITDK

800 “good” conventions.
105% additional routers than originally present in ITDK.
Conventions for 181 (22.6%) suffixes provided no gain.
Inferring IPv6 and IPv4 aliases

- Naming conventions inferred using IPv4 topology (MIDAR and Mercator) usually predict IPv6 clustering (Speedtrap)
  - August 2017: 86.3% of 107 suffixes with no false positives
  - January 2019: 84.5% of 60 suffixes with no false positives

- 192 suffixes where IPv4 naming conventions applied
  - Went from 416 routers to 3757 routers, 9x multiplier
Contribution: Code and Data

• We publicly release the source code implementation
  - https://www.caida.org/tools/measurement/scamper/

• We publicly release inferred regexes, as well as webpages demonstrating how each regex applied to the training data

<table>
<thead>
<tr>
<th>Suffixes:</th>
<th>201904</th>
<th>201901</th>
<th>201803</th>
<th>201708</th>
<th>201702</th>
</tr>
</thead>
<tbody>
<tr>
<td>012.net.il</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orbitel.net</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>100it.net</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>163data.com.cn</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>2i3.net</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>2iij.net</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>31173.se</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>360.net</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>39122.as</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>3rox.net</td>
<td></td>
<td></td>
<td></td>
<td>201708</td>
<td></td>
</tr>
<tr>
<td>3s.pl</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>3z.net</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
<tr>
<td>4d-dc.com</td>
<td>201904</td>
<td>201901</td>
<td>201803</td>
<td>201708</td>
<td>201702</td>
</tr>
</tbody>
</table>

Evaluation against training data:

- ch2-core1.mad
  - 85.115.128.84 + xe-0-2-0.ch2-core1.mad.as34803.net
  - 85.115.128.86 + xe-1-2-0.ch2-core1.mad.as34803.net
  - 62.115.44.122 broadbandgibraltar-ic-306032-mad-b2.c.telia.net

- ep9-access1.gib
  - 85.115.140.17 + ae0.ep9-access1.gib.as34803.net
  - 85.115.140.21 + ae1.ep9-access1.gib.as34803.net

- ep9-core1.gib
  - 85.115.128.89 + xe-2-1-0.ep9-core1.gib.as34803.net
  - 85.115.128.46 + xe-3-2-0.ep9-core1.gib.as34803.net
  - 85.115.128.93 + xe-4-3-0.ep9-core1.gib.as34803.net
Challenges

1. **Heterogeneous Naming Conventions**
   - We do not a priori if a suffix has a convention
   - We do not know which components of a hostname make up its name

2. **Imperfect Naming Training Data**
   - Operators usually maintain zones manually
   - Typos, out-of-date names.

3. **Imperfect Router Training Data**
   - Alias resolution techniques may infer false negatives and false positives
Approach by example

Router #1: core3.fmt2
- 100ge4-1.core3.fmt2.he.net 1a
- 100ge4-2.core3.fmt2.he.net 1b
- v1119.core3.fmt2.he.net 1c
- v1832.core3.fmt2.he.net 1d

Router #2: core1.atl1
- ge2-9.core1.atl1.he.net 2a
- ge6-7.core1.atl1.he.net 2b

Router #3: core1.ash1
- 10ge16-5.core1.ash1.he.net 3a
- 10ge16-6.core1.ash1.he.net 3b
- 100ge5-1.core1.ash1.he.net 3c

R1, R2, R3 hostnames contain names for he.net routers
Approach by example

Router #1: core3.fmt2
- 100ge4-1.core3.fmt2.he.net (1a)
- 100ge4-2.core3.fmt2.he.net (1b)
- v1119.core3.fmt2.he.net (1c)
- v1832.core3.fmt2.he.net (1d)

Router #2: core1.atl1
- ge2-9.core1.atl1.he.net (2a)
- ge6-7.core1.atl1.he.net (2b)

Router #3: core1.ash1
- 10ge16-5.core1.ash1.he.net (3a)
- 10ge16-6.core1.ash1.he.net (3b)
- 100ge5-1.core1.ash1.he.net (3c)

Router #4: unnamed
- esnet.10gigabitethernet5-15.core1.ash1.he.net (4a)

Router #5: unnamed
- fastserv.core1.ash1.he.net (5a)

R1, R2, R3 hostnames contain names for he.net routers.

R4 and R5 hostnames label the neighbor and the he.net router they connect to.
Approach by example

Goal: learn regex to extract from R1, R2, R3, but not R4 or R5

R1, R2, R3 hostnames contain names for he.net routers

R4 and R5 hostnames label the neighbor and the he.net router they connect to
Regular Expressions: quick refresh

A regex defines a pattern that can be applied to a string to check if the string conforms to the structure expressed in the pattern.

- \.+  
  any sequence of characters
- \d*  
  zero or more digits
- \d+  
  at least one digit
- [a-zA-Z]+  
  at least one alphabetic character
- [a-zA-Z0-9]+  
  at least one alphanumeric character
- [a-zA-Z]+\d+  
  alphabetic characters followed by digits
Regular Expressions: quick refresh

A regex defines a pattern that can be applied to a string to check if the string conforms to the structure expressed in the pattern.

- `[^-]+` any sequence of characters except dash
- `[^\ .]+` any sequence of characters except dot
- `^` at start of regex, anchors match to start of string
- `$` at end of regex, anchors match to end of string
- `([a-z]+)` extracts a sequence of alphabetic characters
- `(?::foo|bar)` matches foo or bar, does not extract
Using the ITDK

• We divide the ITDK into two portions, per suffix

• **Training Set**
  - These are routers we believe are responsive to alias resolution because the router had multiple IP addresses resolved

• **Application Set**
  - These are routers with a single interface in ITDK
  - This set is where we can infer additional aliases with Hoiho.
Stage 1: Generate Base Regexes

Router #1: core3.fmt2
100ge4-1.core3.fmt2.he.net 1a
100ge4-2.core3.fmt2.he.net 1b
v1119.core3.fmt2.he.net 1c
v1832.core3.fmt2.he.net 1d
Stage 1: Generate Base Regexes

For each hostname pair on a router, identify combinations of common substrings (CSs) within punctuation boundaries.
Stage 1: Generate Base Regexes

For each hostname pair on a router, identify combinations of common substrings (CSs) within punctuation boundaries.
Stage 1: Generate Base Regexes

For each hostname pair on a router, identify combinations of common substrings (CSs) and build regexes that

1. Match the hostname structure with varying precision
2. Extract the CSs on punctuation boundaries
Stage 1: Generate Base Regexes

For each hostname pair on a router, identify combinations of common substrings (CSs) and build regexes that

1. Match the hostname structure with varying precision

2. Extract the CSs on punctuation boundaries

% kept after removing redundant regexes
Stage 2: Refine True Positives

This phase identifies common literals in correctly clustered hostnames, i.e., those that were true positives, and embeds those literals in the regex.
Stage 2: Refine True Positives

This phase identifies common literals in correctly clustered hostnames, i.e., those that were true positives, and embeds those literals in the regex.
Stage 2: Refine True Positives

This phase identifies common literals in correctly clustered hostnames, i.e., those that were true positives, and embeds those literals in the regex.
Stage 2: Refine True Positives

This phase identifies common literals in correctly clustered hostnames, i.e., those that were true positives, and embeds those literals in the regex.
Stage 2: Refine True Positives

This phase identifies common literals in correctly clustered hostnames, i.e., those that were true positives, and embeds those literals in the regex.
Stage 2: Refine True Positives

This phase identifies common literals in correctly clustered hostnames, i.e., those that were true positives, and embeds those literals in the regex.
Stage 3: Refine False Negative Extractions

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.
Stage 3: Refine False Negative Extractions

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.
Stage 3: Refine False Negative Extractions

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.
Stage 3: Refine False Negative Extractions

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.
Stage 3: Refine False Negative Extractions

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.
This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.
Stage 4: Embed Character Classes

This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.
This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.
Stage 4: Embed Character Classes

This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.
This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.
Stage 4: Embed Character Classes

This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.
Stage 4: Embed Character Classes

This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.
Stages 5-8: summary

5. Refine False Negatives Unmatched
   - Identify unmatched hostnames that contain an apparent name

6. Build Regex Sets
   - Combine regexes together to increase coverage

7. Build Filter Regexes
   - Identify patterns in hostnames that should not be matched

8. Select Best Convention
   - Identify convention that captures complexity within a suffix but without over-fitting to the training data

(see the paper for details)
Limitations

• It is well established that **hostnames can be stale**

• Can only resolve aliases in a **single domain suffix**
  - April 2019 ITDK: 18.9% of training routers with hostnames in more than one suffix

• Relies on the router name being **delimited by punctuation**
Opportunity: Overcome FNs in ITDK

<table>
<thead>
<tr>
<th></th>
<th>FNs in training</th>
<th>TNs in training</th>
<th>Unresponsive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training Set</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>98 (27.7%)</td>
<td>256</td>
<td>112 (24.0%)</td>
</tr>
<tr>
<td>Promising</td>
<td>28 (17.3%)</td>
<td>134</td>
<td>85 (34.4%)</td>
</tr>
<tr>
<td><strong>Application Set</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>6281 (75.1%)</td>
<td>2086</td>
<td>6866 (45.1%)</td>
</tr>
<tr>
<td>Promising</td>
<td>429 (69.8%)</td>
<td>186</td>
<td>1217 (66.4%)</td>
</tr>
</tbody>
</table>

We conducted focused alias resolution proving on FNs from April 2019 ITDK in May 2019.
Opportunity: Overcome FNs in ITDK

We conducted focused alias resolution proving on FNs from April 2019 ITDK in May 2019.

<table>
<thead>
<tr>
<th></th>
<th>FNs in training</th>
<th>TNs in training</th>
<th>Unresponsive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>98 (27.7%)</td>
<td>256</td>
<td>112 (24.0%)</td>
</tr>
<tr>
<td>Promising</td>
<td>28 (17.3%)</td>
<td>134</td>
<td>85 (34.4%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>~25% of apparent FPs were FNs in training set</td>
</tr>
<tr>
<td>Application Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>6281 (75.1%)</td>
<td>2086</td>
<td>6866 (45.1%)</td>
</tr>
<tr>
<td>Promising</td>
<td>429 (69.8%)</td>
<td>186</td>
<td>1217 (66.4%)</td>
</tr>
</tbody>
</table>

We conducted focused alias resolution proving on FNs from April 2019 ITDK in May 2019.
Opportunity: Overcome FNs in ITDK

<table>
<thead>
<tr>
<th></th>
<th>FNs in training</th>
<th>TNs in training</th>
<th>Unresponsive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training Set</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>98 (27.7%)</td>
<td>256</td>
<td>112 (24.0%)</td>
</tr>
<tr>
<td>Promising</td>
<td>28 (17.3%)</td>
<td>134</td>
<td>85 (34.4%)</td>
</tr>
<tr>
<td><strong>Application Set</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>6281 (75.1%)</td>
<td>2086</td>
<td>6866 (45.1%)</td>
</tr>
<tr>
<td>Promising</td>
<td>429 (69.8%)</td>
<td>186</td>
<td>1217 (66.4%)</td>
</tr>
</tbody>
</table>

~25% of apparent FPs were FNs in training set

~74% of interfaces with same inferred name were FNs in training set

We conducted focused alias resolution proving on FNs from April 2019 ITDK in May 2019
Related work

• **DDec** (CAIDA’s DNS Decoder) learns if the hostnames an operator assigns to a router contain geolocation hints.

• **Undns** (Rocketfuel’s DNS Decoder) contains manually assembled regexes that extract router names for 16 suffixes.

• **Validation** of alias resolution algorithms (MIDAR, speedtrap) used manually assembled regexes.

• **Grammar induction**: state of the art (TKDE 2016) can generate a regex given examples of extractions.
We designed, implemented, and validated a method to infer if operators embed router names in hostnames.

We publicly release the source code implementation - https://www.caida.org/tools/measurement/scamper/

We publicly release inferred regexes, as well as webpages demonstrating how each regex applied to the training data - https://www.caida.org/publications/papers/2019/hoiho/
Limitations: single domain suffix

Cannot always resolve aliases across domain suffixes.

The April 2019 ITDK had 18.9% of training routers with hostnames in more than one suffix.
Limitations: names delimited by punctuation

Router #1: fkhrw-01
- fkhrw-01gi1-1.nw.odn.ad.jp 1a
- fkhrw-01gi1-2.nw.odn.ad.jp 1b
- fkhrw-01gi3-1.nw.odn.ad.jp 1c
- fkhrw-01gi3-9.nw.odn.ad.jp 1d

Router #2: fkhrw-02
- fkhrw-02gi1-1.nw.odn.ad.jp 2a
- fkhrw-02gi1-2.nw.odn.ad.jp 2b
- fkhrw-02gi3-1.nw.odn.ad.jp 2c
- fkhrw-02gi3-9.nw.odn.ad.jp 2d

Router #3: kajrc-02
- kajrc-02te0-0-0-1.nw.odn.ad.jp 3a
- kajrc-02te0-0-2-2.nw.odn.ad.jp 3b
### Scoring Specificity of Candidate Regexes

<table>
<thead>
<tr>
<th>Regex component</th>
<th>Example</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anything</td>
<td>.+</td>
<td>0</td>
</tr>
<tr>
<td>Example specified punctuation</td>
<td>^[-]+[^]+</td>
<td>1</td>
</tr>
<tr>
<td>Specified classes</td>
<td>[a-z\d]+</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>[a-z]+</td>
<td>3</td>
</tr>
<tr>
<td>IP address</td>
<td>\d+</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>[a-f\d]+</td>
<td>3</td>
</tr>
<tr>
<td>Literal</td>
<td>infra.cdn</td>
<td>4</td>
</tr>
</tbody>
</table>

Regex builder generates regexes that might match, and chooses the most specific regex when breaking ties.
Penalizing Naming Convention Complexity

Under-specific

NC #1:

```
:^(^[^-]+).+\.(^[^-]+)\..savvis\..net$
```

- esr1jfk2, esr2pax, esr1pax, das1lnj2, das2loc2, das2lnj2

NC #2:

```
^([a-z]+\d+).+\.(^[a-z\d]+)\..savvis\..net$
```

- esr1jfk2, esr2pax, esr1pax, das1lnj2, das2loc2, das2lnj2

NC #3:

```
^(esr\d)-(?:ge|xe)-\d-\d\.(^[a-z]{3}\d*)\..savvis\..net$
```

- esr1jfk2, esr2pax, esr1pax

NC #4:

```
^((esr\d)-(?:ge|xe))-\d-\d\.(^[a-z]{3}\d*)\..savvis\..net$
```

- esr1gel0jfk2, esr2xel0pax, esr1xel0pax

Over-specific

```
^([^-]+).+\.(^[^-]+)\..savvis\..net$
```

- esr1-ge
- esr1-xe

```
^([^-]+).+\.(^[^-]+)\..savvis\..net$
```

- das1-v30lnj2, das2-v30loc2, das2-v30lnj2
IP Address Literals in Hostnames

- 66.161.134.161
- 154.126.82.122
- 94.199.152.9
- 92.60.81.5
- 2804:321c::1
- 2a00:aa40:0:235::96
- 2001:4060:1:3001::2

- 66-161-134-161.meyertool.com
- tgn.126.82.122.tgn.mg
- 152-9-f7m000p01cern.core.as8723.net
- 5.81.unused-addr.ncport.ru
- 2804-321c-0-0-0-0-0-1.nslink.net.br
- gum-core-rou-235-096.oberberg.ne
- prt-cbl-sw1-vlan-3001 gw.imp.ch
Evaluating a Regex Against Training Data

NC #1: ^([a-z\d]+)[^\.]\.[a-z]+\.[a-z]+\.[a-z]+\.[a-z]+\.[a-z]+\..level3.net$  

nc #2:  

FNU: 7c, 8b TP: 7, FP: 4, FIP: 1, FNE: 2, FNU: 2, SP: 3
Stage 5: Refine False Negative Unmatched

This phase identifies hostnames with the apparent router name embedded, but not extracted, and builds regexes to match those hostnames.
Stage 5: Refine False Negative Unmatched

This phase identifies hostnames with the apparent router name embedded, but not extracted, and builds regexes to match those hostnames.
This phase identifies hostnames with the apparent router name embedded, but not extracted, and builds regexes to match those hostnames.
Stage 5: Refine False Negative Unmatched

This phase identifies hostnames with the apparent router name embedded, but not extracted, and builds regexes to match those hostnames.
This phase identifies hostnames with the apparent router name embedded, but not extracted, and builds regexes to match those hostnames.
Stage 6: Build Sets

This phase increases coverage of suffixes where the operator has multiple conventions for hostnames on the same router by merging regexes in the working set into larger conventions.
This phase increases coverage of suffixes where the operator has multiple conventions for hostnames on the same router by merging regexes in the working set into larger conventions.
This phase increases coverage of suffixes where the operator has multiple conventions for hostnames on the same router by merging regexes in the working set into larger conventions.
Stage 7: Build Filter Regexes

This phase identifies filter regexes that match incorrectly clustered hostnames, so we do not use an extractor regex on those hostnames.
Stage 7: Build Filter Regexes

For hostnames that are incorrectly clustered by extraction regexes, we identify common substrings in the hostnames, and build filters.

This includes regexes that extract an apparent portion of an IP address from a hostname.
Stage 7: Build Filter Regexes

<table>
<thead>
<tr>
<th>Router #1: ar01.area4.il.chicago</th>
<th>as7272-1-c.ashburn.va.ibone.comcast.net</th>
<th>4a</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>as7272-1-c.chicago.il.ibone.comcast.net</td>
<td>4b</td>
</tr>
<tr>
<td>he-0-10-0-0-ar01.area4.il.chicago.comcast.net</td>
<td>as13385-10-c.chicago.il.ibone.comcast.net</td>
<td>5a</td>
</tr>
<tr>
<td>he-0-12-0-0-ar01.area4.il.chicago.comcast.net</td>
<td>as13385-17-c.ashburn.va.ibone.comcast.net</td>
<td>5b</td>
</tr>
<tr>
<td></td>
<td>as13385-10-c.ashburn.va.ibone.comcast.net</td>
<td>6a</td>
</tr>
<tr>
<td></td>
<td>as13385-2-c.miami.fl.ibone.comcast.net</td>
<td>6b</td>
</tr>
<tr>
<td></td>
<td>c-98-233-46-230.hsd1.md.comcast.net</td>
<td>7a</td>
</tr>
<tr>
<td></td>
<td>c-174-52-116-77.hsd1.ut.comcast.net</td>
<td>8a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Router #2: pe04.ashburn.va.ibone</th>
<th>----------------------------------</th>
<th>----</th>
</tr>
</thead>
<tbody>
<tr>
<td>be-10-pe04.ashburn.va.ibone.comcast.net</td>
<td>as7272-1-c.ashburn.va.ibone.comcast.net</td>
<td>4a</td>
</tr>
<tr>
<td>be-11-pe04.ashburn.va.ibone.comcast.net</td>
<td>as7272-1-c.chicago.il.ibone.comcast.net</td>
<td>4b</td>
</tr>
<tr>
<td>te-0-6-0-0-pe04.ashburn.va.ibone.comcast.net</td>
<td>as13385-10-c.chicago.il.ibone.comcast.net</td>
<td>5a</td>
</tr>
<tr>
<td></td>
<td>as13385-17-c.ashburn.va.ibone.comcast.net</td>
<td>5b</td>
</tr>
<tr>
<td></td>
<td>as13385-10-c.ashburn.va.ibone.comcast.net</td>
<td>6a</td>
</tr>
<tr>
<td></td>
<td>as13385-2-c.miami.fl.ibone.comcast.net</td>
<td>6b</td>
</tr>
<tr>
<td></td>
<td>c-98-233-46-230.hsd1.md.comcast.net</td>
<td>7a</td>
</tr>
<tr>
<td></td>
<td>c-174-52-116-77.hsd1.ut.comcast.net</td>
<td>8a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Router #3: cr01.miami.fl.ibone</th>
<th>----------------------------------</th>
<th>----</th>
</tr>
</thead>
<tbody>
<tr>
<td>be-10-cr01.miami.fl.ibone.comcast.net</td>
<td>as7272-1-c.ashburn.va.ibone.comcast.net</td>
<td>4a</td>
</tr>
<tr>
<td>be-11-cr01.miami.fl.ibone.comcast.net</td>
<td>as7272-1-c.chicago.il.ibone.comcast.net</td>
<td>4b</td>
</tr>
<tr>
<td></td>
<td>as13385-10-c.chicago.il.ibone.comcast.net</td>
<td>5a</td>
</tr>
<tr>
<td></td>
<td>as13385-17-c.ashburn.va.ibone.comcast.net</td>
<td>5b</td>
</tr>
<tr>
<td></td>
<td>as13385-10-c.ashburn.va.ibone.comcast.net</td>
<td>6a</td>
</tr>
<tr>
<td></td>
<td>as13385-2-c.miami.fl.ibone.comcast.net</td>
<td>6b</td>
</tr>
<tr>
<td></td>
<td>c-98-233-46-230.hsd1.md.comcast.net</td>
<td>7a</td>
</tr>
<tr>
<td></td>
<td>c-174-52-116-77.hsd1.ut.comcast.net</td>
<td>8a</td>
</tr>
</tbody>
</table>

| ^c-\d+-\d+-\d+-\d+-\d+-hsd1.[a-z]+\.comcast.net$ | △ |
| ^as\d+-\d+-c.[a-z]+\[a-z]+ibone\.comcast\net$ | ○ |
| (^[^-]+)\.comcast\net$ | □ |

```
<table>
<thead>
<tr>
<th>Router #1: ar01.area4.il.chicago</th>
<th>pe04.ashburn.va.ibone</th>
<th>cr01.miami.fl.ibone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>2a</td>
<td>3a</td>
</tr>
<tr>
<td>1b</td>
<td>2b</td>
<td>3b</td>
</tr>
<tr>
<td>1c</td>
<td>2c</td>
<td></td>
</tr>
</tbody>
</table>
```
Stage 8: Choose Best Convention

This phase chooses a naming convention from the working set. Naming conventions with fewer regexes are preferred over conventions with more regexes if they perform similarly.