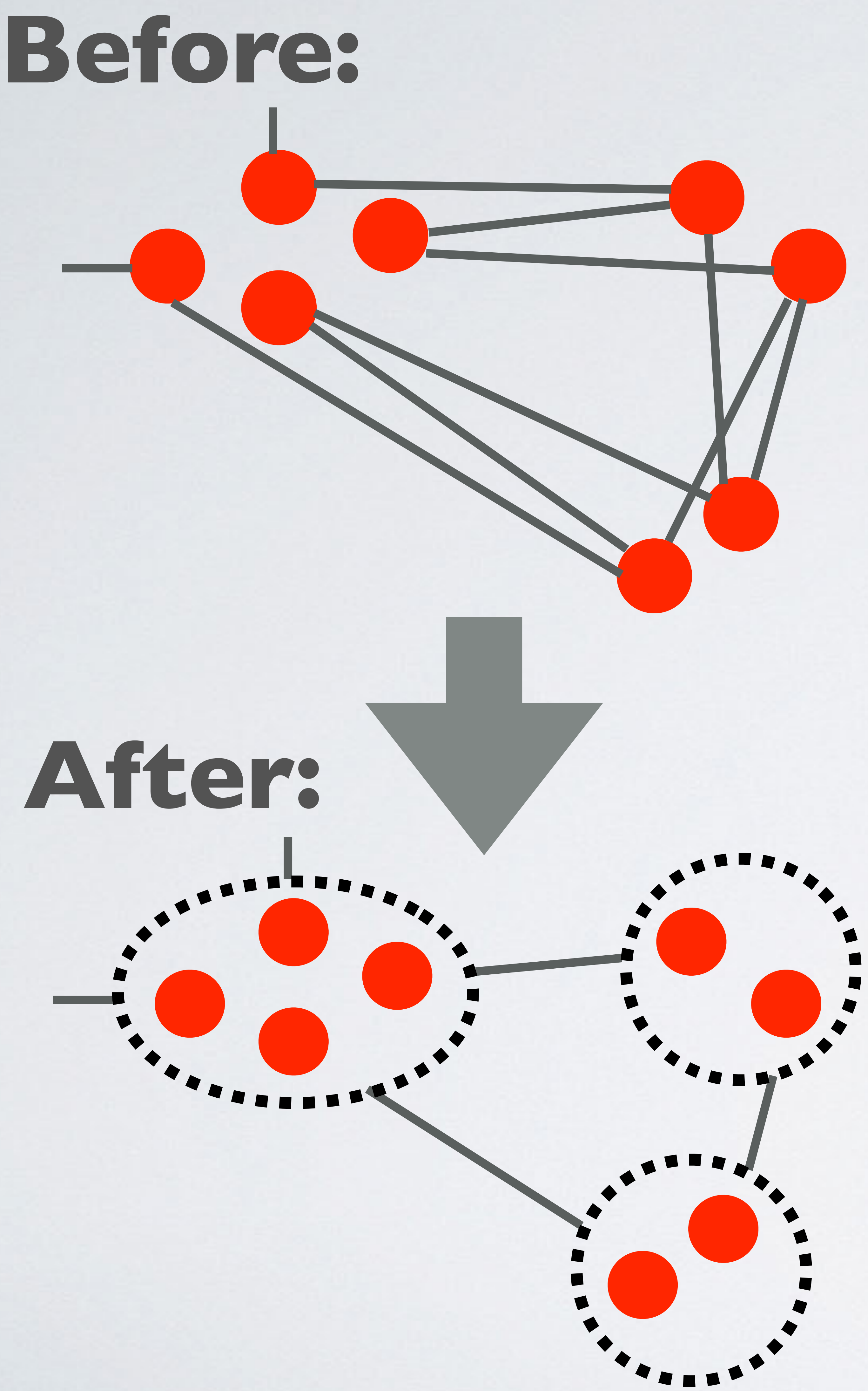
Learning to Extract Router Names from Hostnames

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: esr1|jfk2

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 #4: das1|nj2

das1-v3005.nj2.savvis.net
das1-v3006.nj2.savvis.net
das1-v3007.nj2.savvis.net

Router #2: esr2|pax

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 #5: das2|oc2

das1-v3005.oc2.savvis.net
das1-v3007.oc2.savvis.net
das1-v3008.oc2.savvis.net

Router #3: esr1|pax

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 #6: das2|nj2

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

Router #1: esr1|jfk2

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 #4: das1|nj2

das1-v3005.nj2.savvis.net
das1-v3006.nj2.savvis.net
das1-v3007.nj2.savvis.net

Router #2: esr2|pax

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 #5: das2|oc2

das1-v3005.oc2.savvis.net
das1-v3007.oc2.savvis.net
das1-v3008.oc2.savvis.net

Router #3: esr1|pax

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 #6: das2|nj2

das2-v3009.nj2.savvis.net
das2-v3010.nj2.savvis.net
das2-v3011.nj2.savvis.net

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

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

Intuition: Naming Conventions

Router #1: esr1|jfk2

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 #4: das1|nj2

das1-v3005.nj2.savvis.net
das1-v3006.nj2.savvis.net
das1-v3007.nj2.savvis.net

Router #2: esr2|pax

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 #5: das2|oc2

das1-v3005.oc2.savvis.net
das1-v3007.oc2.savvis.net
das1-v3008.oc2.savvis.net

Router #3: esr1|pax

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 #6: das2|nj2

das2-v3009.nj2.savvis.net
das2-v3010.nj2.savvis.net
das2-v3011.nj2.savvis.net

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

(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

Intuition: Naming Conventions

Router #1: esr1|jfk2

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 #4: das1|nj2

das1-v3005.nj2.savvis.net
das1-v3006.nj2.savvis.net
das1-v3007.nj2.savvis.net

Router #2: esr2|pax

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 #5: das2|oc2

das1-v3005.oc2.savvis.net
das1-v3007.oc2.savvis.net
das1-v3008.oc2.savvis.net

Router #3: esr1|pax

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 #6: das2|nj2

das2-v3009.nj2.savvis.net
das2-v3010.nj2.savvis.net
das2-v3011.nj2.savvis.net

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

(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 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

Hoiho Input Data

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

Contribution: Hoiho

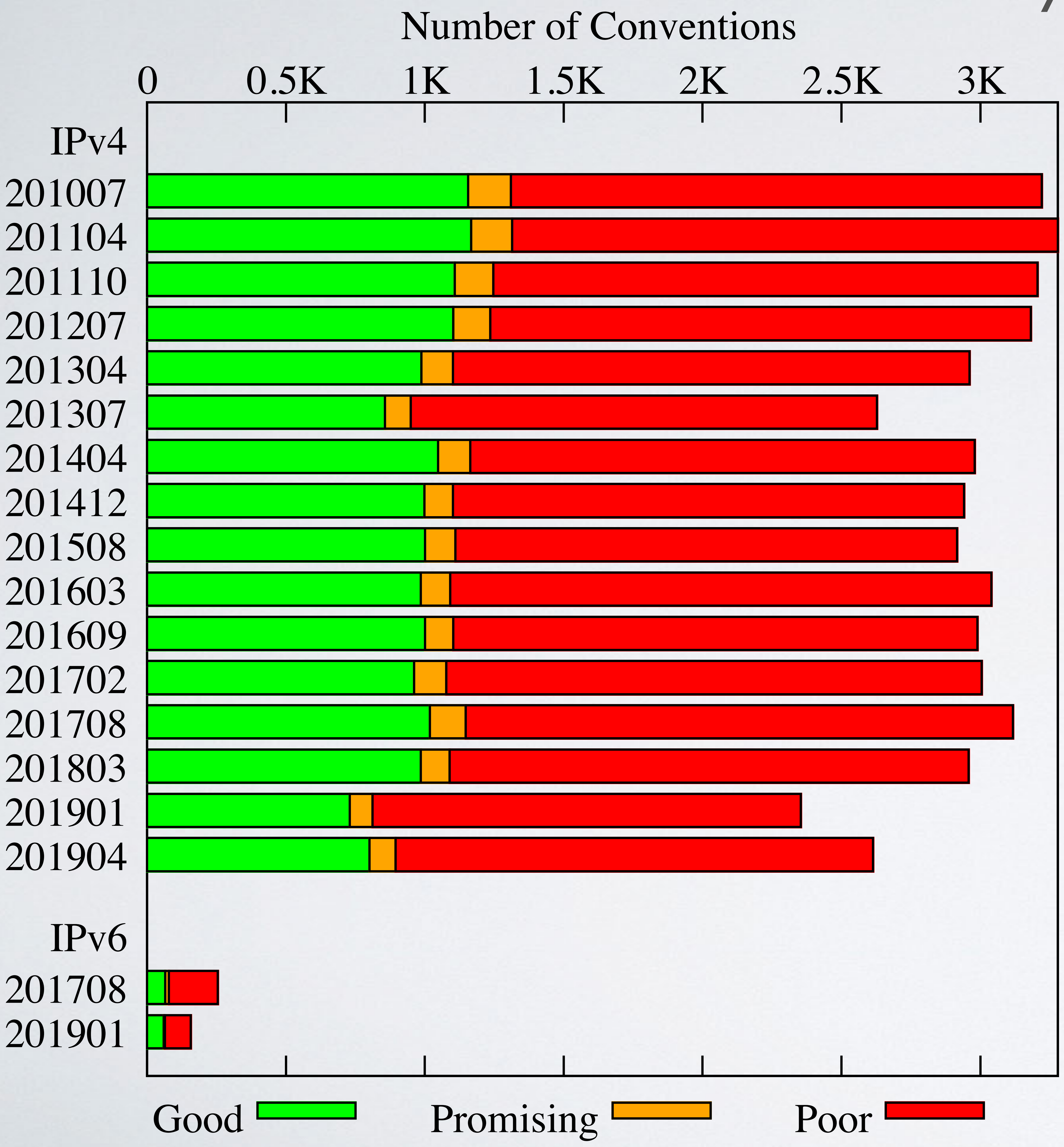
(Holistic Orthography of Internet Hostname Observations)

- 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

Hoiho: Yellow-eyed penguin

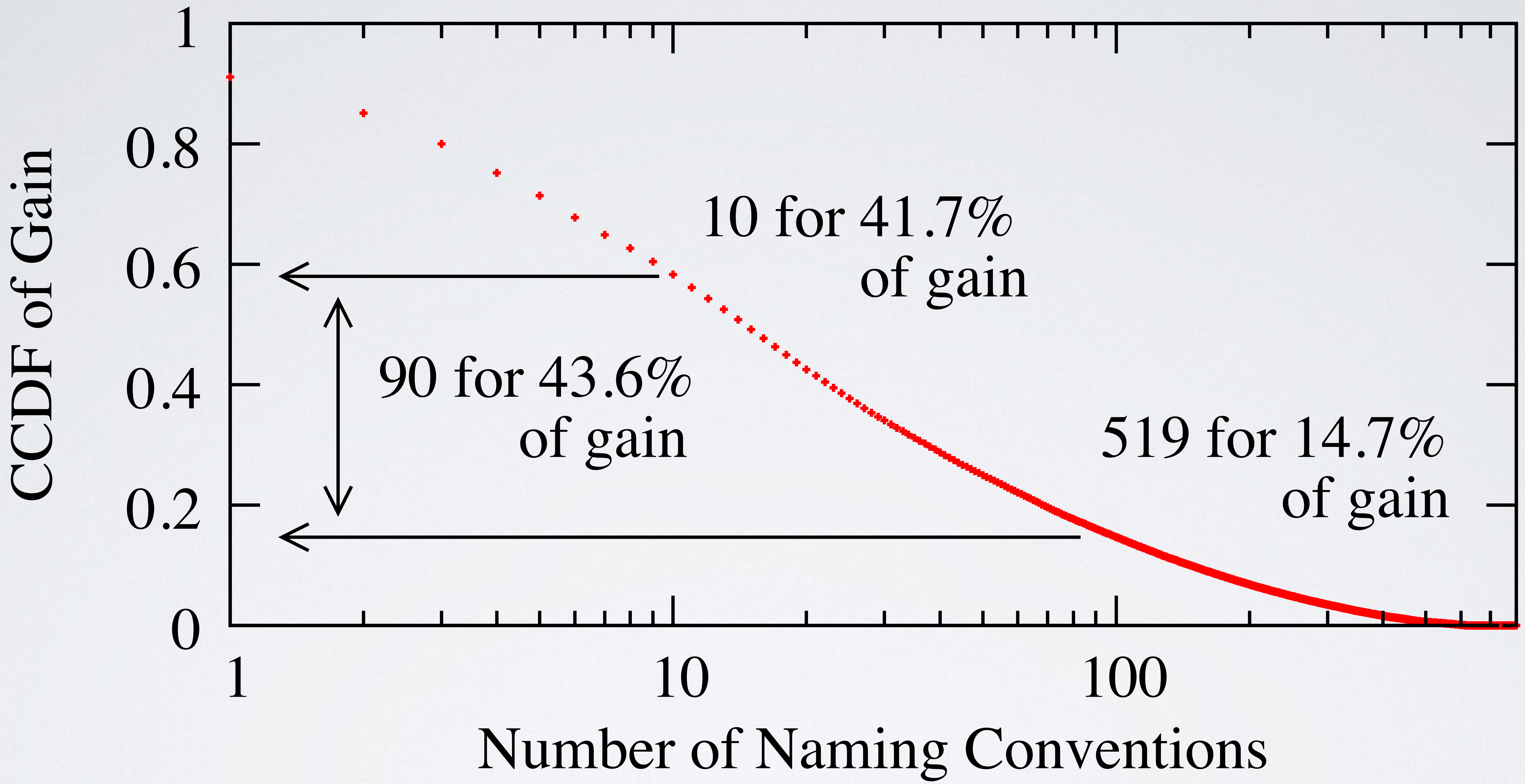
Image: Brent Beaven
Department of Conservation (New Zealand)

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
  - https://www.caida.org/publications/papers/2019/hoiho/

Suffixes:

012.net.il	201904	201901	201803	201708	201702
0rbitel.net	201904	201901	201803	201708	201702
100it.net				201708	201702
163data.com.cn	201904	201901	201803	201708	201702
2i3.net	201904	201901	201803	201708	201702
2iij.net	201904	201901	201803	201708	201702
31173.se	201904	201901	201803	201708	201702
360.net	201904	201901	201803	201708	201702
39122.as	201904	201901	201803	201708	201702
3rox.net		201901	201803	201708	
3s.pl	201904	201901	201803	201708	201702
3z.net	201904	201901	201803	201708	201702
4d-dc.com	201904	201901	201803	201708	201702

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

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

Router #4: unnamed

esnet.10gigabitethernet5-15.core1.ash1.he.net	4a

Router #5: unnamed

fastserv.core1.ash1.he.net	5a

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

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

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-z]+`	at least one alphabetic character
`[a-z\d]+`	at least one alphanumeric character
`[a-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

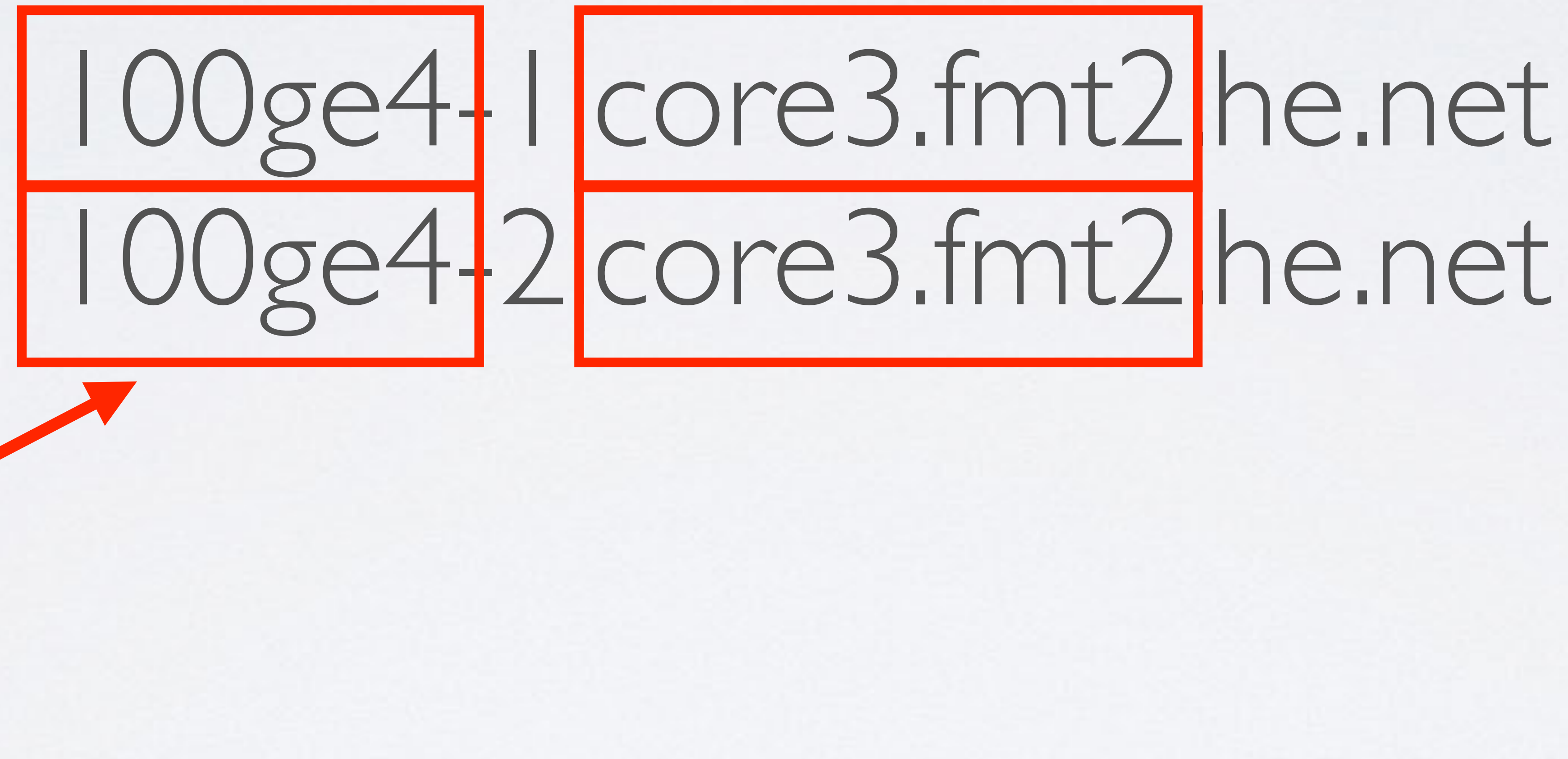
- We divide the ITDK into two portions, per suffix
- **Training Set**
  - These are routers we believe are responsive to alias resolution because the router had multiple IP addresses resolved
- **Application Set**
  - These are routers with a single interface in ITDK
  - This set is where we can infer 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

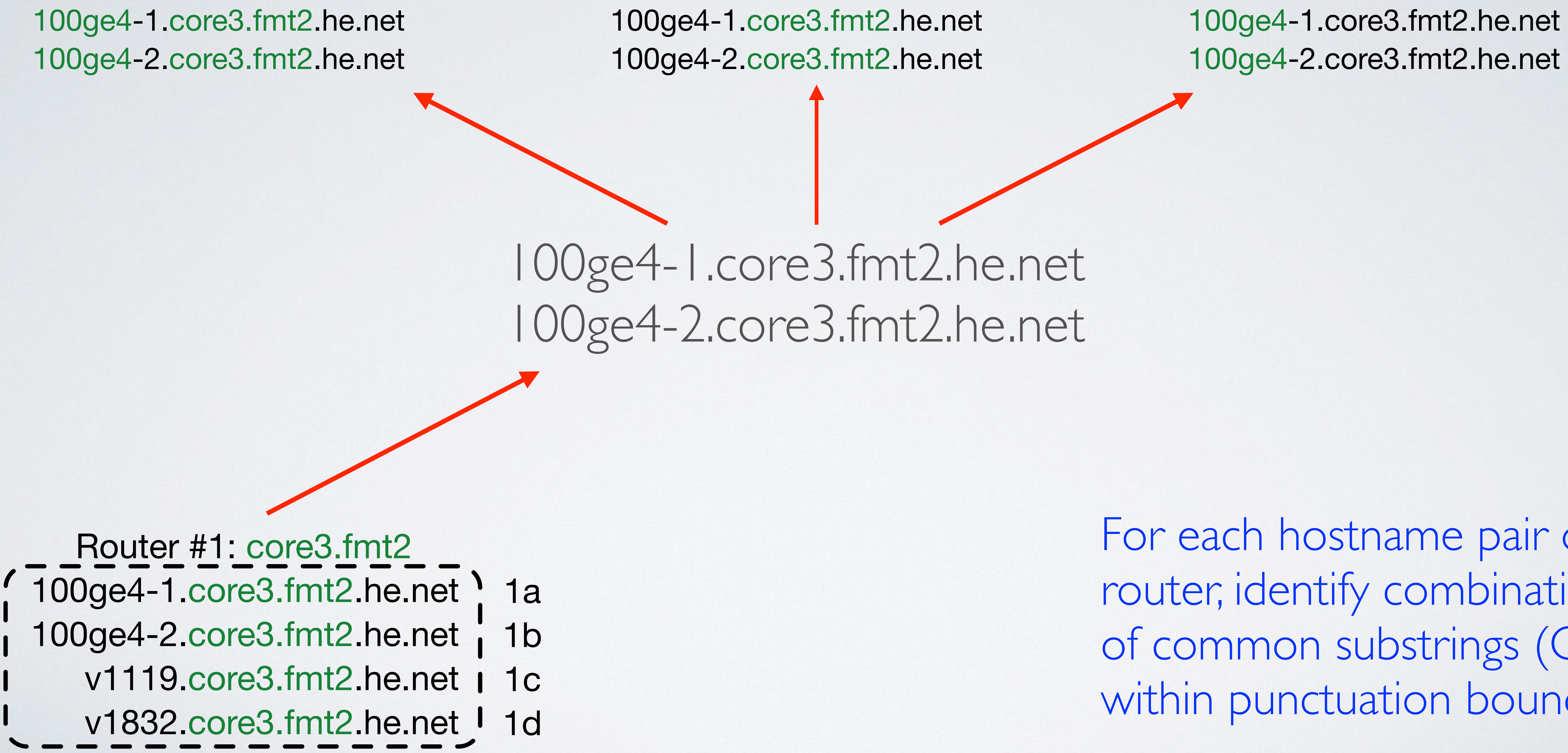


Router #1: core3.fmt2

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

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

100ge4-1.core3.fmt2.he.net
100ge4-2.core3.fmt2.he.net

100ge4-1.core3.fmt2.he.net
100ge4-2.core3.fmt2.he.net

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

100ge4-1.core3.fmt2.he.net
100ge4-2.core3.fmt2.he.net

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

◇ ▷ kept after removing redundant regexes

100ge4-1.core3.fmt2.he.net
100ge4-2.core3.fmt2.he.net

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

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 2: Refine True Positives

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

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

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

^[^\.]+\.([^\.]+\.[^\.]+)\.he\.net$ ▷

core3.fmt2 1a 1b 1c 1d
core1.atl1 2a 2b
core1.ash1 3a 3b 3c 5a

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

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

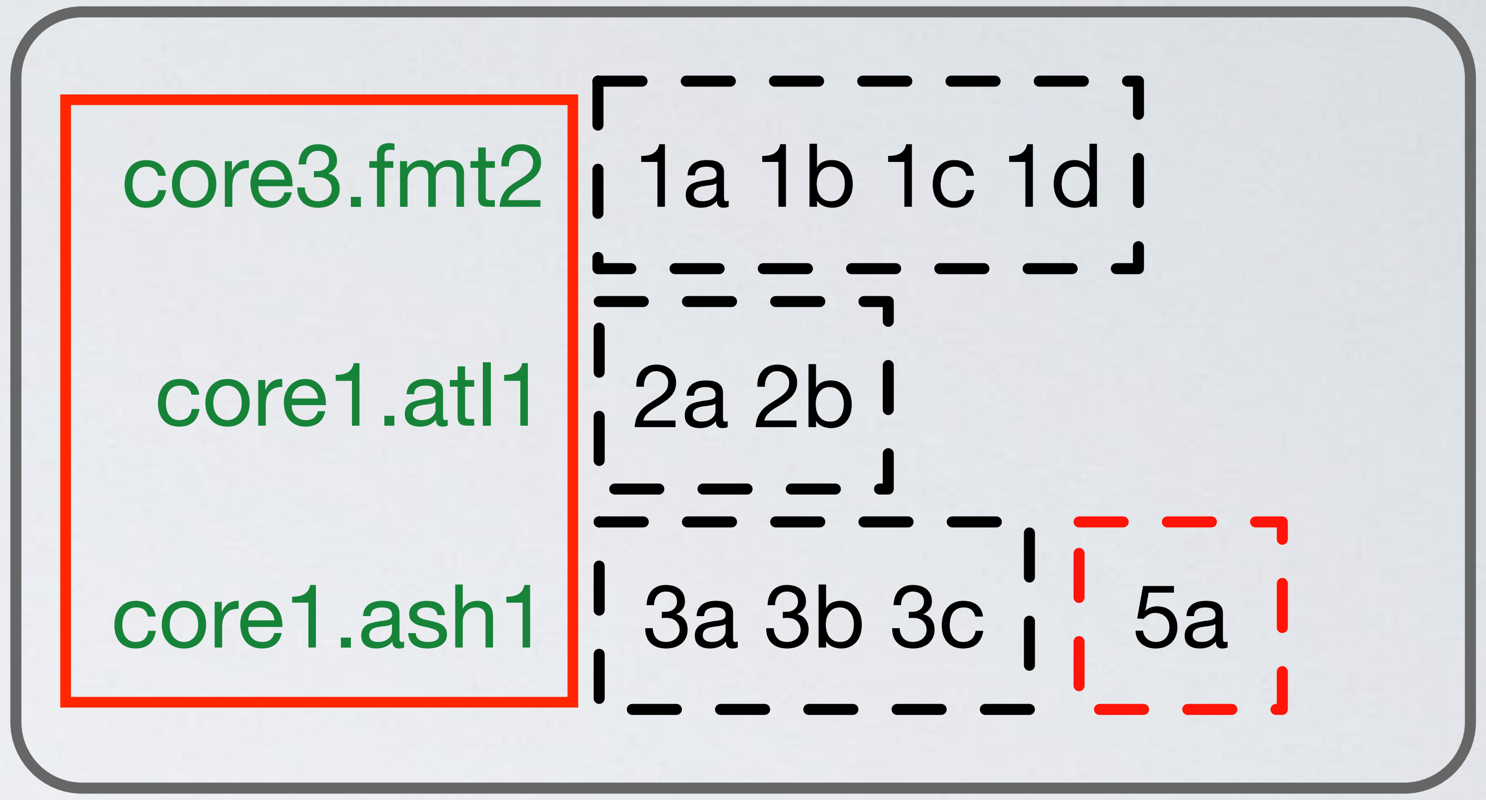
Router #4: unnamed

esnet.10gigabitethernet5-15.core1.ash1.he.net 4a

Router #5: unnamed

fastserv.core1.ash1.he.net 5a

^[^\.]+\.([^\.]+\.[^\.]+)\.he\.net$ ▷



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

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

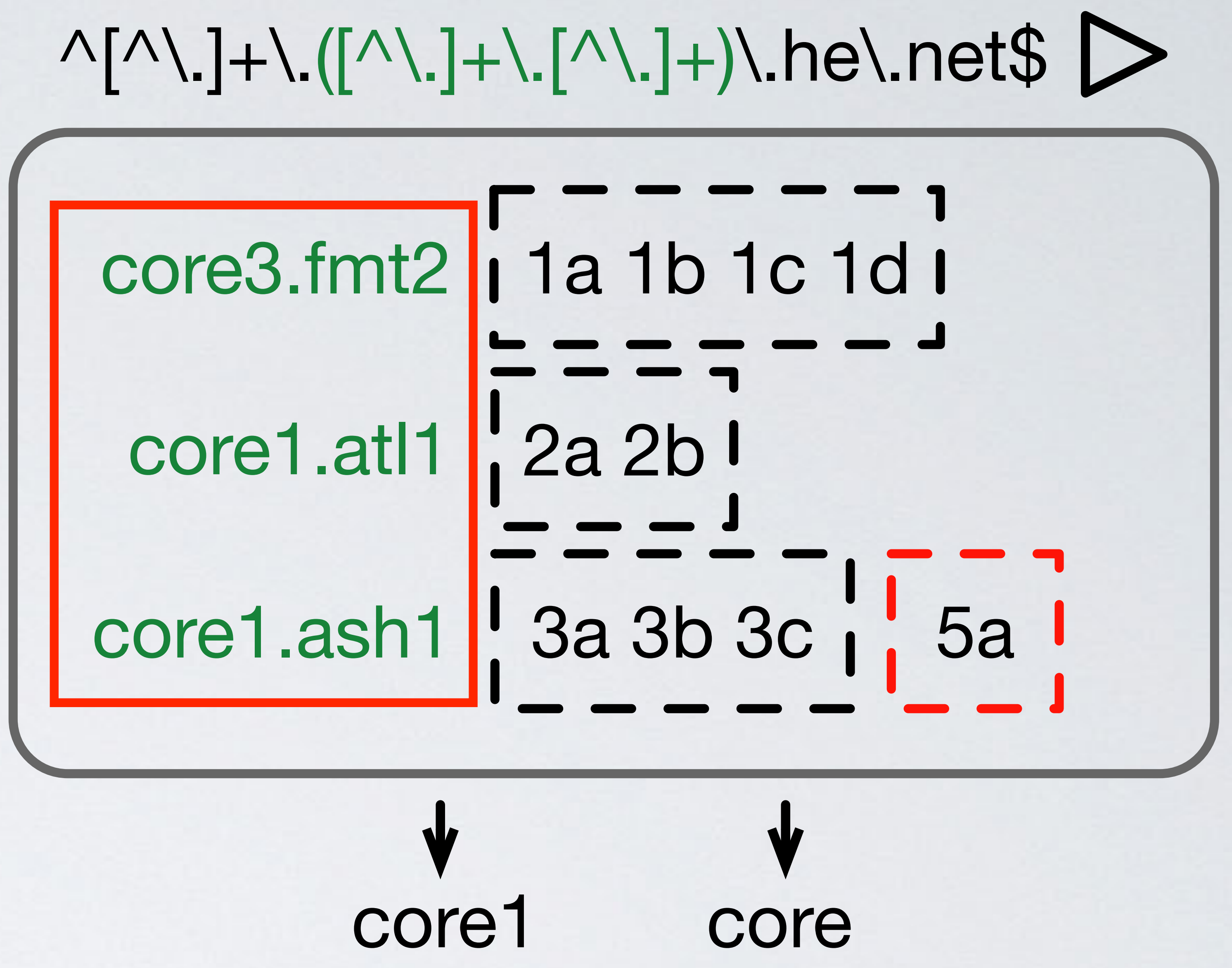
10ge16-5.core1.ash1.he.net 3a
10ge16-6.core1.ash1.he.net 3b
100ge5-1.core1.ash1.he.net 3c

Router #4: unnamed

esnet.10gigabitethernet5-15.core1.ash1.he.net 4a

Router #5: unnamed

fastserv.core1.ash1.he.net 5a



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

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

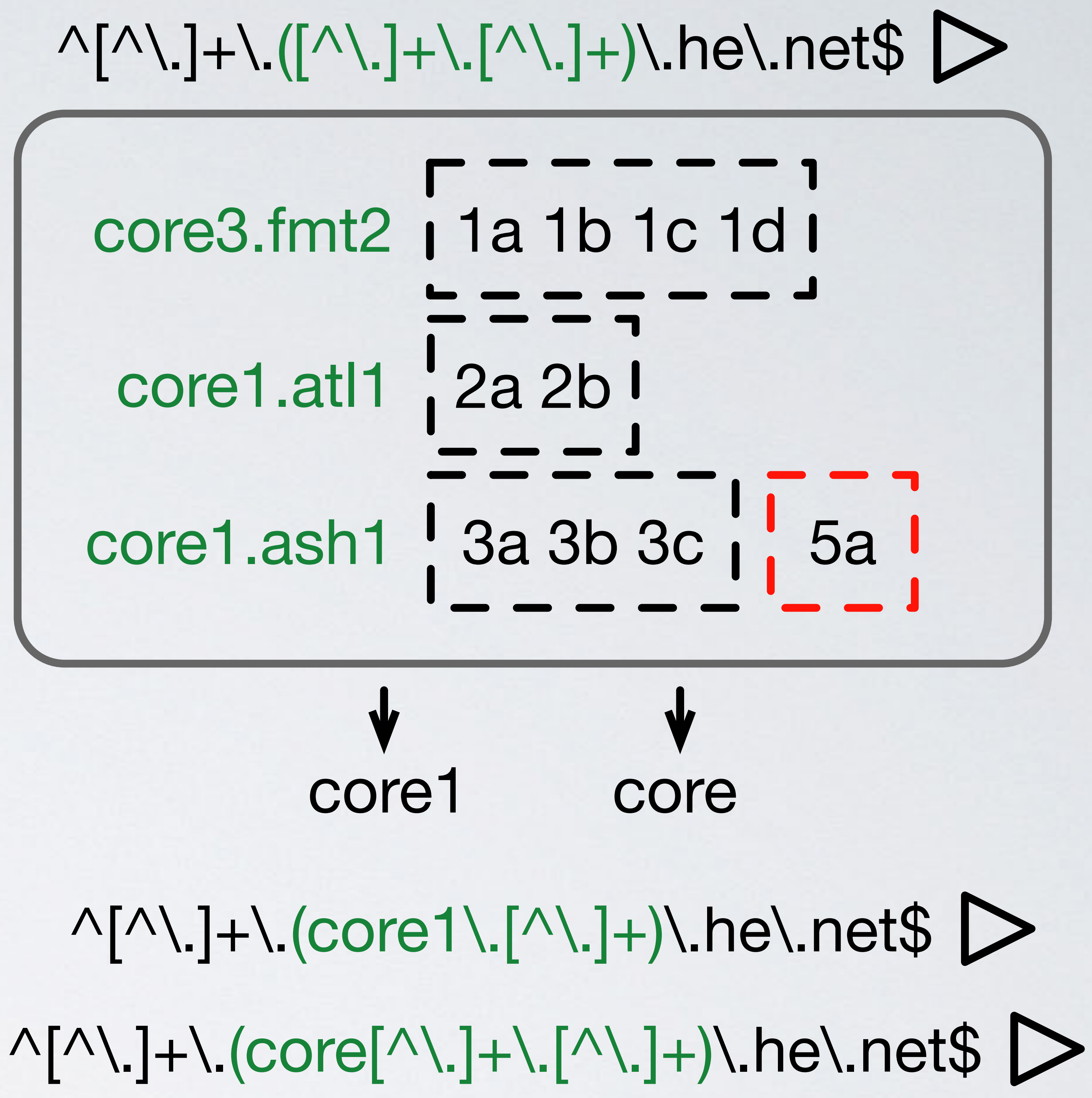
10ge16-5.core1.ash1.he.net 3a
10ge16-6.core1.ash1.he.net 3b
100ge5-1.core1.ash1.he.net 3c

Router #4: unnamed

esnet.10gigabitethernet5-15.core1.ash1.he.net 4a

Router #5: unnamed

fastserv.core1.ash1.he.net 5a



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

Router #1: core3.fmt2

Hostname	ID
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

Hostname	ID
ge2-9.core1.atl1.he.net	2a
ge6-7.core1.atl1.he.net	2b

Router #3: core1.ash1

Hostname	ID
10ge16-5.core1.ash1.he.net	3a
10ge16-6.core1.ash1.he.net	3b
100ge5-1.core1.ash1.he.net	3c

Router #4: unnamed

Hostname	ID
esnet.10gigabitethernet5-15.core1.ash1.he.net	4a

Router #5: unnamed

Hostname	ID
fastserv.core1.ash1.he.net	5a

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.

Stage 3: Refine False Negative Extractions

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

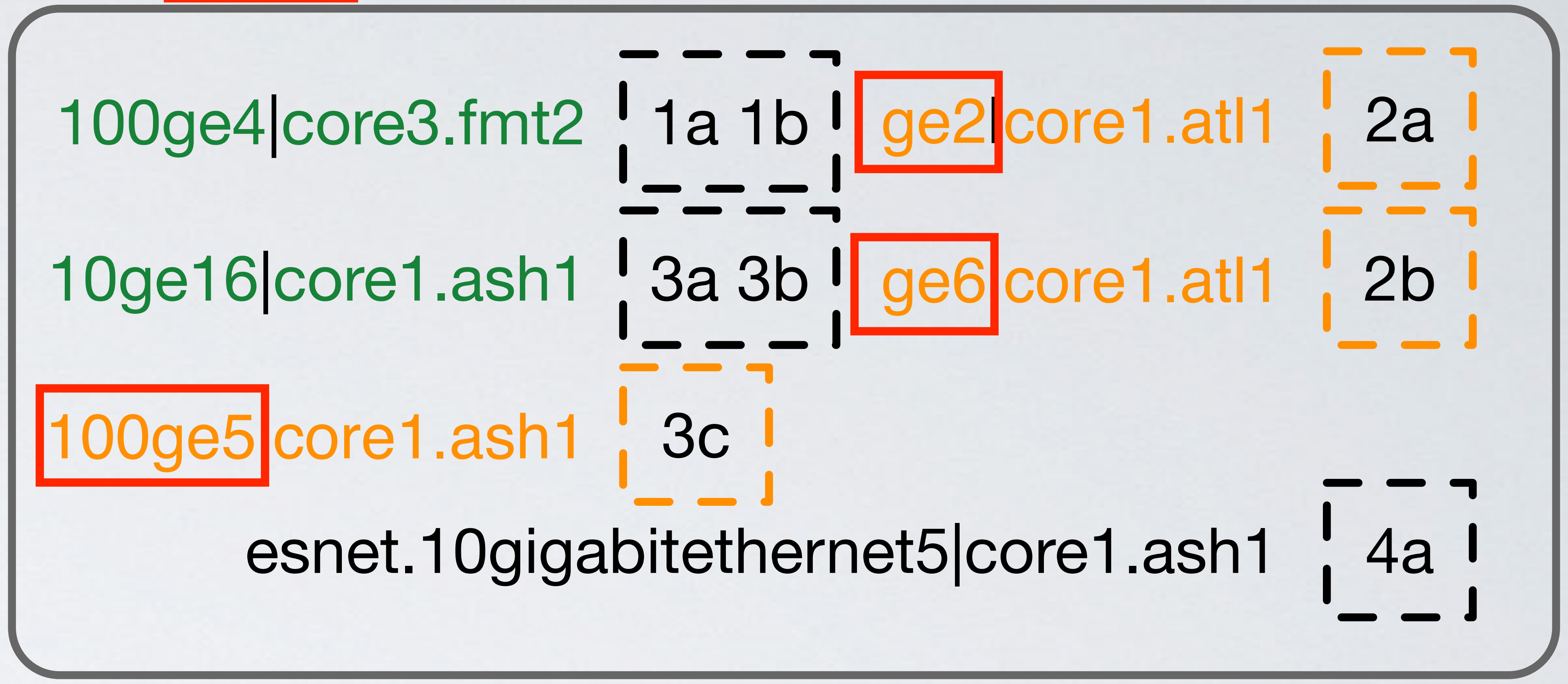
^([^-]+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

100ge4|core3.fmt2 1a 1b
ge2|core1.atl1 2a
10ge16|core1.ash1 3a 3b
ge6|core1.atl1 2b
100ge5|core1.ash1 3c
esnet.10gigabitethernet5|core1.ash1 4a

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.

Stage 3: Refine False Negative Extractions

^([^-]+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇



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

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.

Stage 3: Refine False Negative Extractions

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

^([^-]+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

100ge4|core3.fmt2 1a 1b
ge2|core1.atl1 2a
10ge16|core1.ash1 3a 3b
ge6|core1.atl1 2b
100ge5|core1.ash1 3c
esnet.10gigabitethernet5|core1.ash1 4a

↓

10ge16 100ge4 100ge5 ge2 ge6
esnet10.10gigabitethernet5

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.

Stage 3: Refine False Negative Extractions

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

`^([^-]+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$` ◇

100ge4|core3.fmt2	1a 1b	ge2|core1.atl1	2a
10ge16|core1.ash1	3a 3b	ge6|core1.atl1	2b
100ge5|core1.ash1	3c		
		esnet.10gigabitethernet5|core1.ash1	4a

↓

10ge16 100ge4 100ge5 ge2 ge6
esnet10.10gigabitethernet5

↓

`^\d+ge\d+-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$`
`^ge\d+-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$`
`^esnet\.\d+gigabitethernet\d+-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$`

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.

Stage 3: Refine False Negative Extractions

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

^([^-]+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

100ge4|core3.fmt2 1a 1b
ge2|core1.atl1 2a
10ge16|core1.ash1 3a 3b
ge6|core1.atl1 2b
100ge5|core1.ash1 3c
esnet.10gigabitethernet5|core1.ash1 4a

↓

10ge16 100ge4 100ge5 ge2 ge6
esnet10.10gigabitethernet5

↓

^\d+ge\d+-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$
^ge\d+-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$
^esnet\.\d+gigabitethernet\d+-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$

↓

^(?:\d+ge\d+|ge\d+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

core3.fmt2 1a 1b
core1.ash1 3a 3b 3c
core1.atl1 2a 2b

This phase identifies extraction components that separate hostnames from their training routers, replacing the extraction component with literals.

Stage 4: Embed Character Classes

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

^(?:\d+ge\d+|ge\d+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

core3.fmt2 1a 1b
core1.ash1 3a 3b 3c
core1.atl1 2a 2b

This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.

Stage 4: Embed Character Classes

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

^(?:\d+ge\d+|ge\d+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

core3.fmt2	1a 1b	core1.ash1	3a 3b 3c
core1.atl1	2a 2b		

This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.

Stage 4: Embed Character Classes

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

^(?:\d+ge\d+|ge\d+)-[^\.]+\.(core[^\.]+\.[^\.]+)\.he\.net$ ◇

core3.fmt2	1a 1b	core1.ash1	3a 3b 3c
core1.atl1	2a 2b		

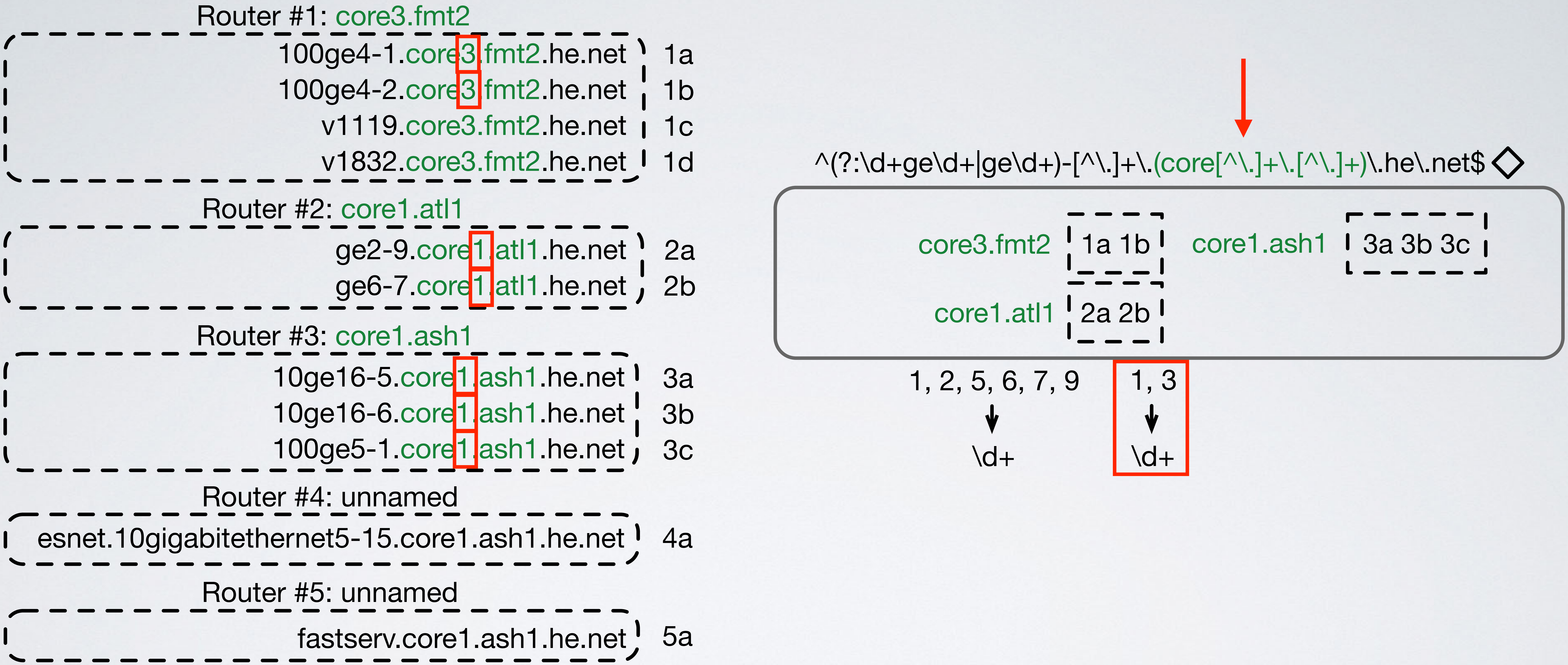
1, 2, 5, 6, 7, 9

↓

\d+

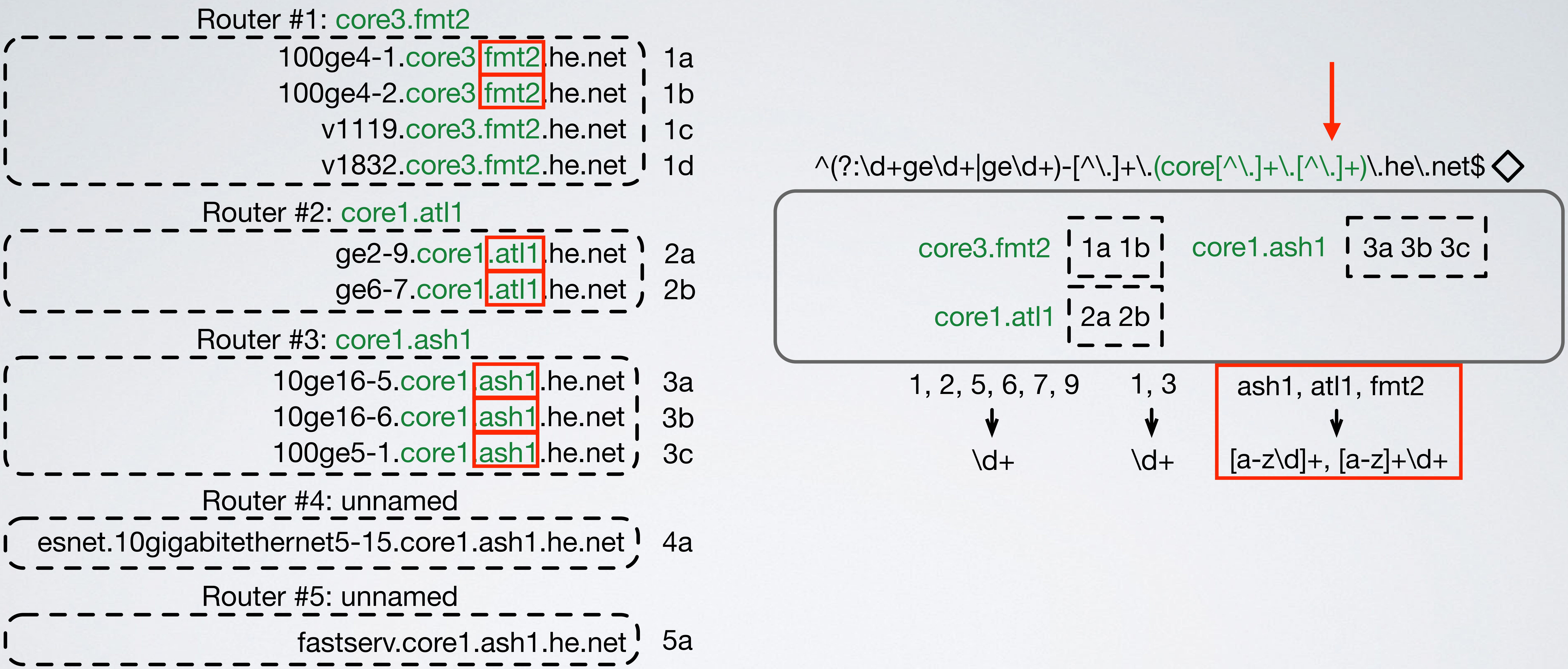
This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.

Stage 4: Embed Character Classes



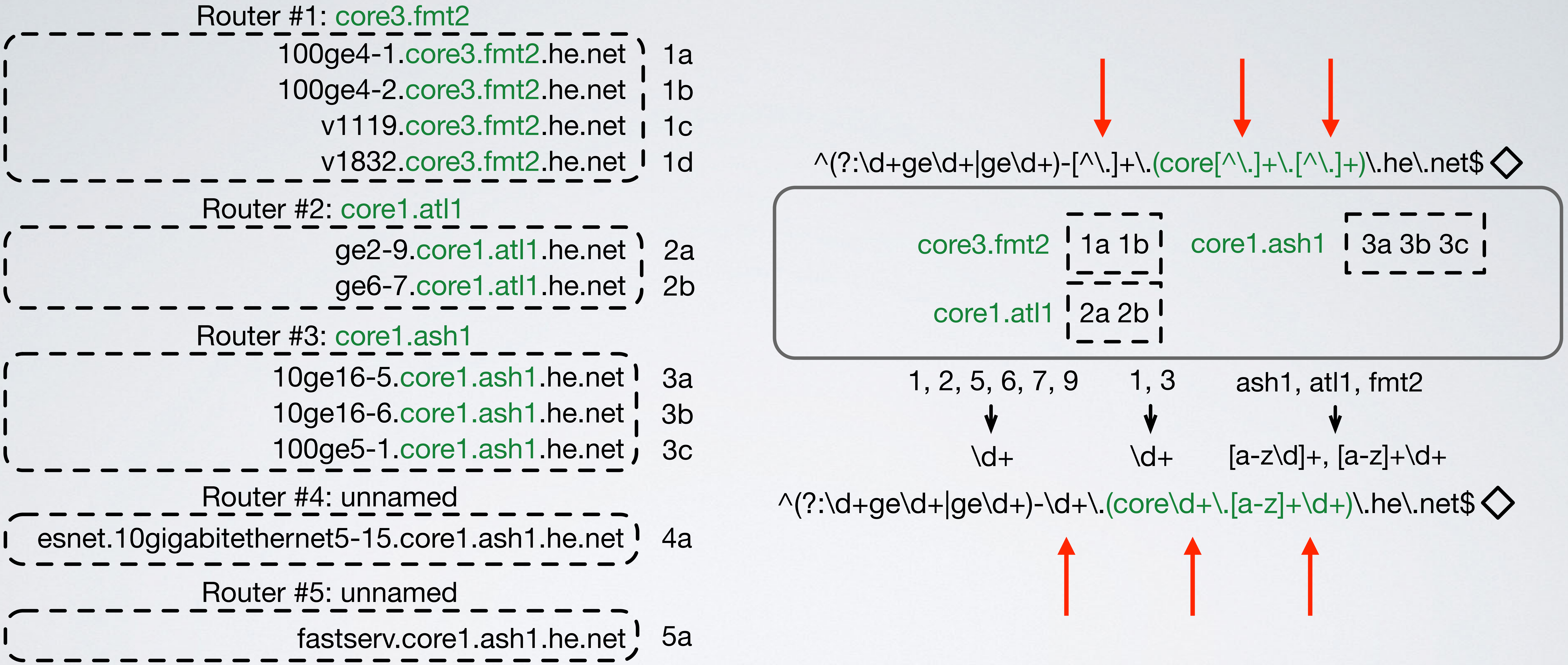
This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.

Stage 4: Embed Character Classes



This phase replaces components that only specify what they should not match (punctuation) with character classes for each component.

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

(see the paper for details)

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

Limitations

- It is well established that hostnames can be stale
  - Zhang et al. *How DNS Misnaming Distorts Internet Topology Mapping.* USENIX ATC 2006
- 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

	FNs in training	TNs in training	Unresponsive
Training Set			
Good	98 (27.7%)	256	112 (24.0%)
Promising	28 (17.3%)	134	85 (34.4%)
Application Set			
Good	6281 (75.1%)	2086	6866 (45.1%)
Promising	429 (69.8%)	186	1217 (66.4%)

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

Opportunity: Overcome FNs in ITDK

	FNs in training	TNs in training	Unresponsive
Training Set			
Good	98 (27.7%)	256	112 (24.0%)
Promising	28 (17.3%)	134	85 (34.4%)
Application Set			
Good	6281 (75.1%)	2086	6866 (45.1%)
Promising	429 (69.8%)	186	1217 (66.4%)

~25% of apparent FPs were FNs in training set

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

Opportunity: Overcome FNs in ITDK

	FNs in training	TNs in training	Unresponsive
Training Set			
Good	98 (27.7%)	256	112 (24.0%)
Promising	28 (17.3%)	134	85 (34.4%)
Application Set			
Good	6281 (75.1%)	2086	6866 (45.1%)
Promising	429 (69.8%)	186	1217 (66.4%)

~25% of apparent FPs were FNs in training set

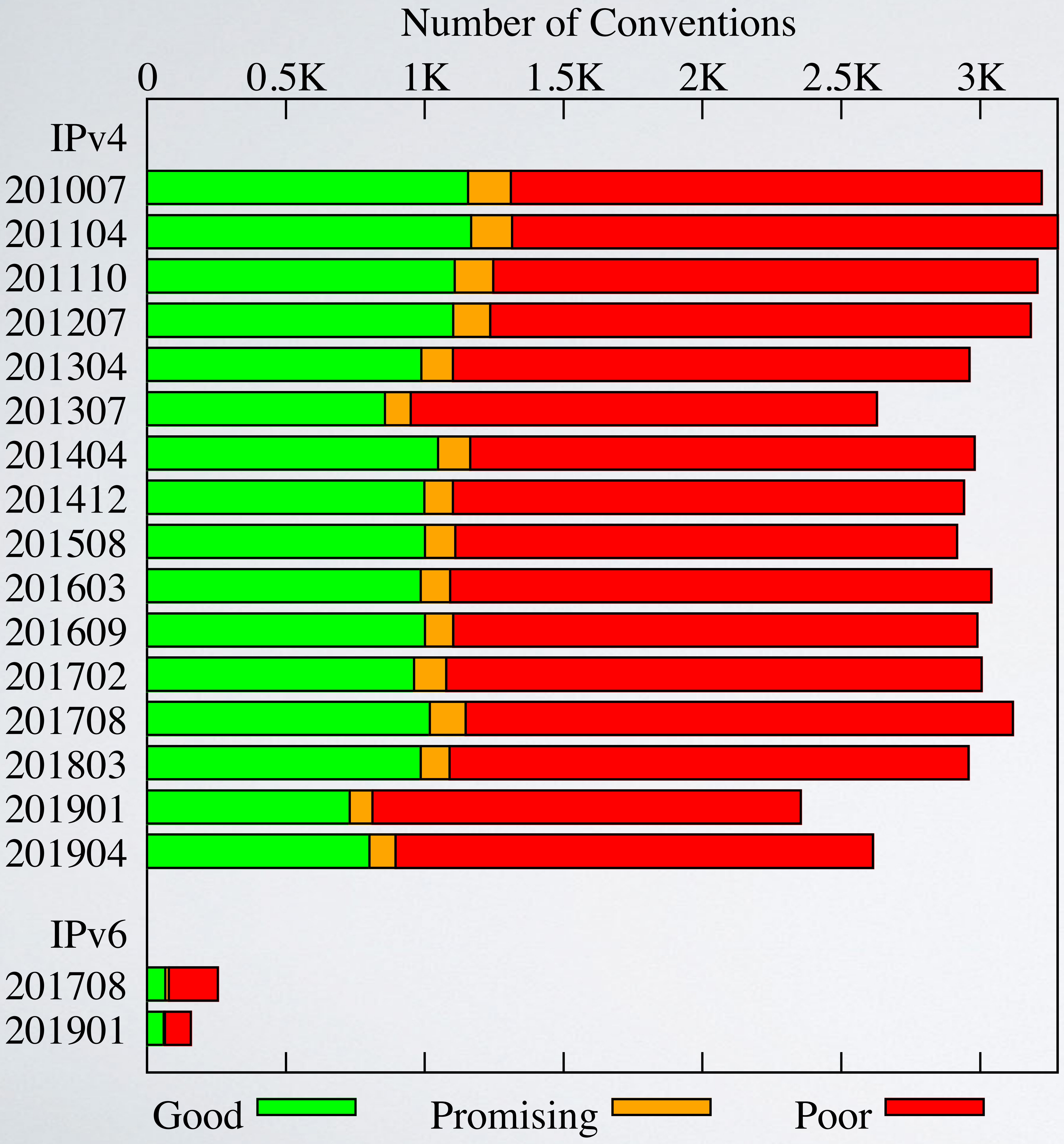
~74% of interfaces with same inferred name were FNs in training set

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.

Summary



- 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

Router #1: msr2.aue

xe-0-0-0.msr2.aue.yahoo.com
xe-2-1-0.msr2.aue.yahoo.com
yah2817952.lnk.telstra.net
as17457.bdr01.syd03.nsw.vocus.net.au

Router #2: pat1.atz

ae0.pat1.atz.yahoo.com
ae1.pat1.atz.yahoo.com
ae2.pat1.atz.yahoo.com
verizon.com.customer.alter.net
yahoo-inc.ear1.atlanta2.level3.net
yahoo-ic-325257-atl-b22.c.telia.net

^[^\.]+\.([a-z]+\d+\.[a-z]+)\.yahoo\.com$

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

^([a-z]+-[a-z\d]+)-[^\.]+\.nw\.odn\.ad\.jp$

fkhrw-01gi1: 1a 1b
fkhrw-02gi1: 2a 2b
fkhrw-01gi3: 1c 1d
fkhrw-02gi3: 2c 2d
kajrc-02te0: 3a 3b

^([a-z]+-\d+)[a-z]+\d+-[^\.]+\.nw\.odn\.ad\.jp$

fkhrw-01: 1a 1b 1c 1d
fkhrw-02: 2a 2b 2c 2d
kajrc-02: 3a 3b

Scoring Specificity of Candidate Regexes

Regex component	Example	Specificity
Anything	.+	0
Example specified punctuation	[^-]+ [^\.]+	1 1
Specified classes	[a-z\d]+ [a-z]+	2 3
IP address	\d+ [a-f\d]+	3 3
Literal	f infra\.cdn	4 36

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$
esr1ljfk2, esr2lpax, esr1lpax, das1lnj2, das2loc2, das2lnj2

NC #2:
^([a-z]+\d+)-.+\.([a-z\d]+)\.savvis\.net$
esr1ljfk2, esr2lpax, esr1lpax, das1lnj2, das2loc2, das2lnj2

NC #3:
^(esr\d)-(?:ge|xe)-\d-\d-\d\.([a-z]{3}\d*)\.savvis\.net$
esr1ljfk2, esr2lpax, esr1lpax
^(das\d)-v30\d{2}.+\.([a-z]{2}\d)\.savvis\.net$
das1lnj2, das2loc2, das2lnj2

NC #4:
^(esr\d-(?:ge|xe))-\d-(\d)-\d\.([a-z]{3}\d*)\.savvis\.net$
esr1-gel0ljfk2, esr2-xel0lpax, esr1-xel0lpax
^(das\d-v30)\d{2}.+\.([a-z]{2}\d)\.savvis\.net$
das1- v30lnj2, das2-v30loc2, das2-v30lnj2

Over-specific

IP Address Literals in Hostnames

66.161.134.161	66-161-134-161.meyertool.com
154.126.82.122	tgn.126.82.122.tgn.mg
94.199.152.9	152-9-f7m000p01cern.core.as8723.net
92.60.81.5	5.81.unused-addr.ncport.ru
2804:321c::1	2804-321c-0-0-0-0-0-1.nslink.net.br
2a00:aa40:0:235::96	gum-core-rou-235-096.oberberg.ne
2001:4060:1:3001::2	prt-cbl-sw1-vlan-3001.gw.imp.ch

Evaluating a Regex Against Training Data

Group	Hostname	ID
1	ae-0-11.bar1.toronto1.level3.net	1a
	ae-1-9.bar1.toronto1.level3.net	1b
	ae-13-13.bar1.toronto1.level3.net	1c
	ae6-1038.bar1.toronto1.level3.net	1d
	xe-8-3-2.bar1.toronto1.level3.net	1e
2	fiber-tech.bar1.toronto1.level3.net	2a
3	nobel-ltd.bar1.toronto1.level3.net	3a
4	ae-1-51.ear2.miami1.level3.net	4a
	ae-2-52.ear2.miami1.level3.net	4b
5	trinity-com.ear2.miami1.level3.net	5a
	trinity-com.ear2.miami1.level3.net	5b
6	trinity-com.ear2.miami1.level3.net	6a
	trinity-com.ear2.miami1.level3.net	6b
7	ae-14-51.car4.miami1.level3.net	7a
	ae-24-52.car4.miami1.level3.net	7b
	vlan600.car4.miami1.level3.net	7c
8	ae-5-5.car1.houston1.level3.net	8a
	vlan434.car1.houston1.level3.net	8b
9	4-35-237-150.edge1.washington1.level3.net	9a

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

Extracted	IDs	Extracted	IDs
ae|bar1.toronto1	1a 1b 1c	fiber|bar1.toronto1	2a
ae6|bar1.toronto1	1d	nobel|bar1.toronto1	3a
xe|bar1.toronto1	1e	ae|ear2.miami1	4a 4b
trinity|ear2.miami1	5a 5b 6a 6b	ae|car4.miami1	7a 7b
ae|car1.houston1	8a	4|edge1.washington1	9a

FNU: 7c, 8b

TP: 7, FP: 4, FIP: 1, FNE: 2, FNU: 2, SP: 3

NC #2: ^(?:ae|xe)-[^\.]+\.([a-z]+\d+\.[a-z]+\d+)\.level3.net$
^vlan\d+\.([a-z]+\d+\.[a-z]+\d+)\.level3.net$

Extracted	IDs	Extracted	IDs
bar1.toronto1	1a 1b 1c 1d 1e	ear2.miami1	4a 4b
car4.miami1	7a 7b 7c	car1.houston1	8a 8b

FNU: 5a, 5b, 6a, 6b. SN: 2a, 3a, 9a.

TP: 12, FNU: 4, SN: 3

Stage 5: Refine False Negative Unmatched

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

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

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

`^(?:\d+ge\d+|ge\d+)-\d+\.(core\d+\.[a-z]+\d+)\.he\.net$` ◇

core3.fmt2: 1a 1b
core1.ash1: 3a 3b 3c
core1.atl1: 2a 2b

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

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

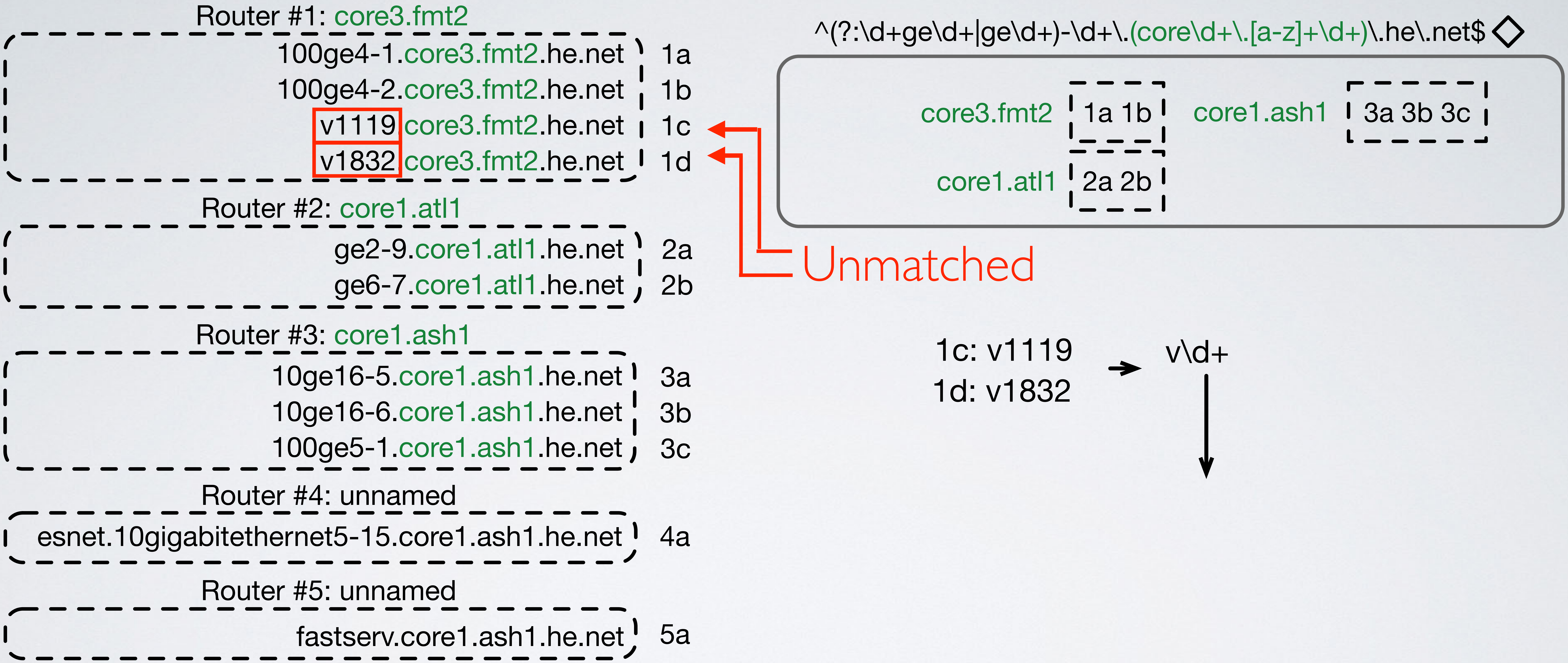
`^(?:\d+ge\d+|ge\d+)-\d+\.(core\d+\.[a-z]+\d+)\.he\.net$` ◇

core3.fmt2	1a 1b	core1.ash1	3a 3b 3c
core1.atl1	2a 2b		

Unmatched (1c, 1d)

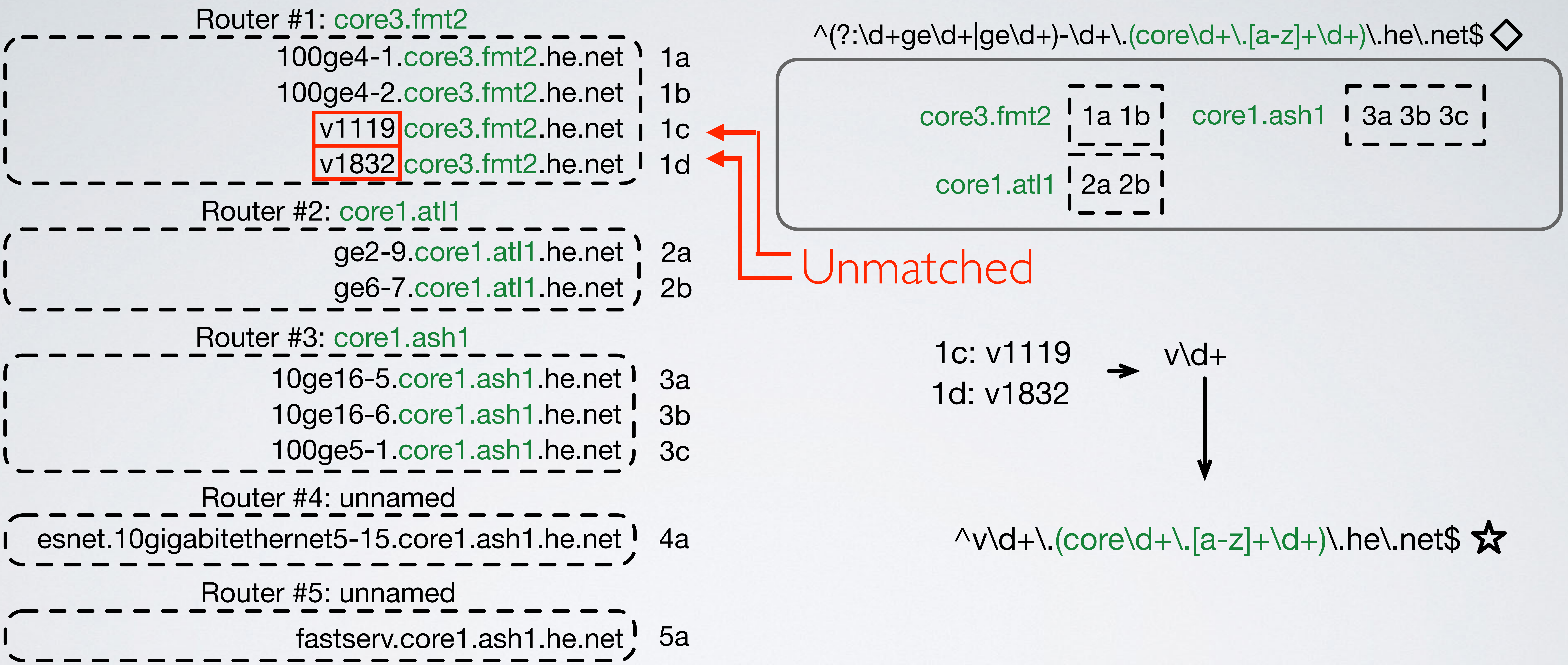
This phase identifies hostnames with the apparent router name embedded, but not extracted, and builds regexes to match those hostnames.

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.

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 6: Build Sets

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

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 6: Build Sets

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

^(?:\d+ge\d+|ge\d+)-\d+\.(core\d+\.[a-z]+\d+)\.he\.net$ ◇

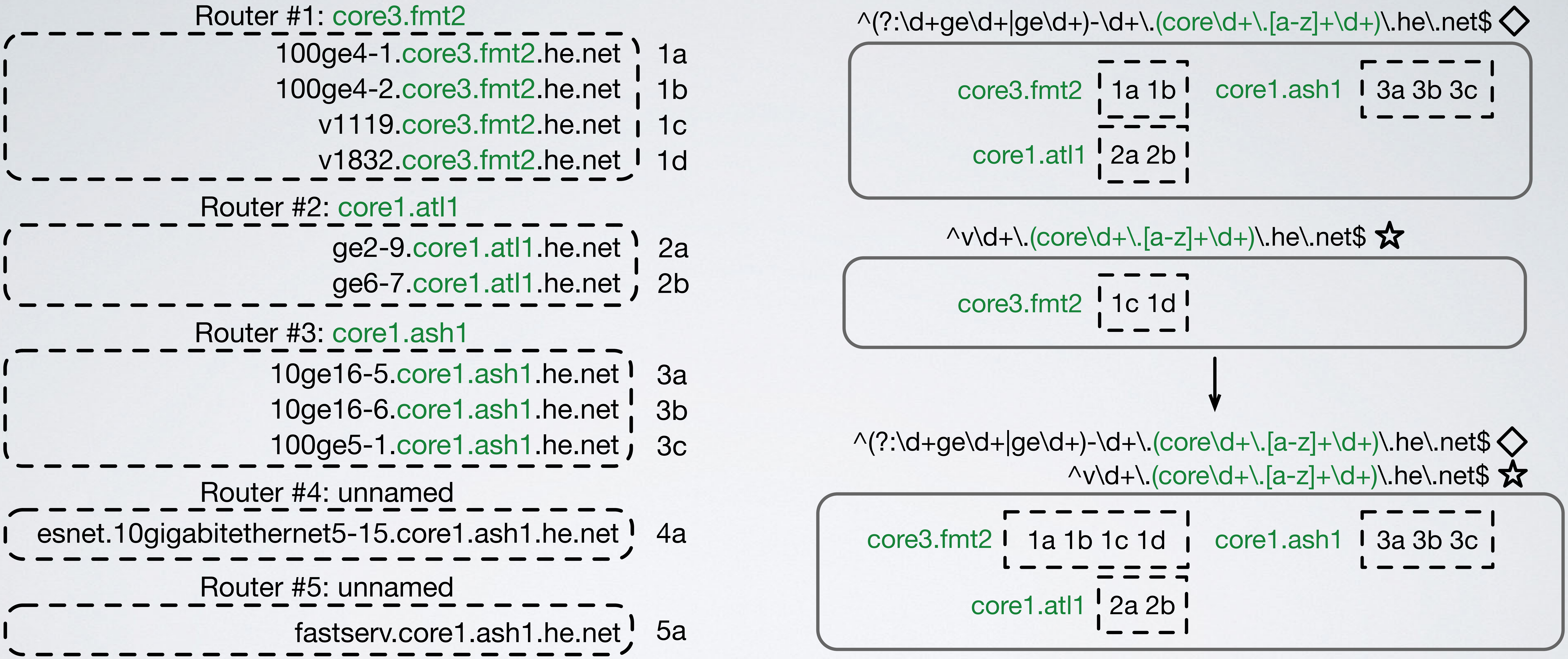
- core3.fmt2: 1a 1b
- core1.ash1: 3a 3b 3c
- core1.atl1: 2a 2b

^v\d+\.(core\d+\.[a-z]+\d+)\.he\.net$ ☆

- core3.fmt2: 1c 1d

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 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.

Stage 7: Build Filter Regexes

Router #1: ar01.area4.il.chicago

- ar01.area4.il.chicago.comcast.net 1a
- he-0-10-0-0-ar01.area4.il.chicago.comcast.net 1b
- he-0-12-0-0-ar01.area4.il.chicago.comcast.net 1c

Router #2: pe04.ashburn.va.ibone

- be-10-pe04.ashburn.va.ibone.comcast.net 2a
- be-11-pe04.ashburn.va.ibone.comcast.net 2b
- te-0-6-0-0-pe04.ashburn.va.ibone.comcast.net 2c

Router #3: cr01.miami.fl.ibone

- be-10-cr01.miami.fl.ibone.comcast.net 3a
- be-11-cr01.miami.fl.ibone.comcast.net 3b

- as7272-1-c.ashburn.va.ibone.comcast.net 4a
- as7272-1-c.chicago.il.ibone.comcast.net 4b

- as13385-10-c.chicago.il.ibone.comcast.net 5a
- as13385-17-c.ashburn.va.ibone.comcast.net 5b

- as13385-10-c.ashburn.va.ibone.comcast.net 6a
- as13385-2-c.miami.fl.ibone.comcast.net 6b

- c-98-233-46-230.hsd1.md.comcast.net 7a

- c-174-52-116-77.hsd1.ut.comcast.net 8a

([^-]+)\.comcast\.net$ ☐

ar01.area4.il.chicago	1a 1b 1c	pe04.ashburn.va.ibone	2a 2b 2c		
cr01.miami.fl.ibone	3a 3b	c.ashburn.va.ibone	4a 5b 6a	230.hsd1.md	7a
c.chicago.il.ibone	4b 5a	c.miami.fl.ibone	6b	77.hsd1.ut	8a

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

as7272-1-c.ashburn.va.ibone.comcast.net 4a
as7272-1-c.chicago.il.ibone.comcast.net 4b

as|c|ibone
^as\d+-\d+-c\.[a-z]+\.[a-z]+\.ibone\.comcast\.net$

as13385-10-c.chicago.il.ibone.comcast.net 5a
as13385-17-c.ashburn.va.ibone.comcast.net 5b

as13385-10-c.ashburn.va.ibone.comcast.net 6a
as13385-2-c.miami.fl.ibone.comcast.net 6b

c|hsd1
^c-\d+-\d+-\d+-\d+\.hsd1\.[a-z]+\.comcast.net$

c-98-233-46-230.hsd1.md.comcast.net 7a

c-174-52-116-77.hsd1.ut.comcast.net 8a

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

Router #1: ar01.area4.il.chicago

ar01.area4.il.chicago.comcast.net 1a
he-0-10-0-0-ar01.area4.il.chicago.comcast.net 1b
he-0-12-0-0-ar01.area4.il.chicago.comcast.net 1c

Router #2: pe04.ashburn.va.ibone

be-10-pe04.ashburn.va.ibone.comcast.net 2a
be-11-pe04.ashburn.va.ibone.comcast.net 2b
te-0-6-0-0-pe04.ashburn.va.ibone.comcast.net 2c

Router #3: cr01.miami.fl.ibone

be-10-cr01.miami.fl.ibone.comcast.net 3a
be-11-cr01.miami.fl.ibone.comcast.net 3b

as7272-1-c.ashburn.va.ibone.comcast.net 4a
as7272-1-c.chicago.il.ibone.comcast.net 4b

as13385-10-c.chicago.il.ibone.comcast.net 5a
as13385-17-c.ashburn.va.ibone.comcast.net 5b

as13385-10-c.ashburn.va.ibone.comcast.net 6a
as13385-2-c.miami.fl.ibone.comcast.net 6b

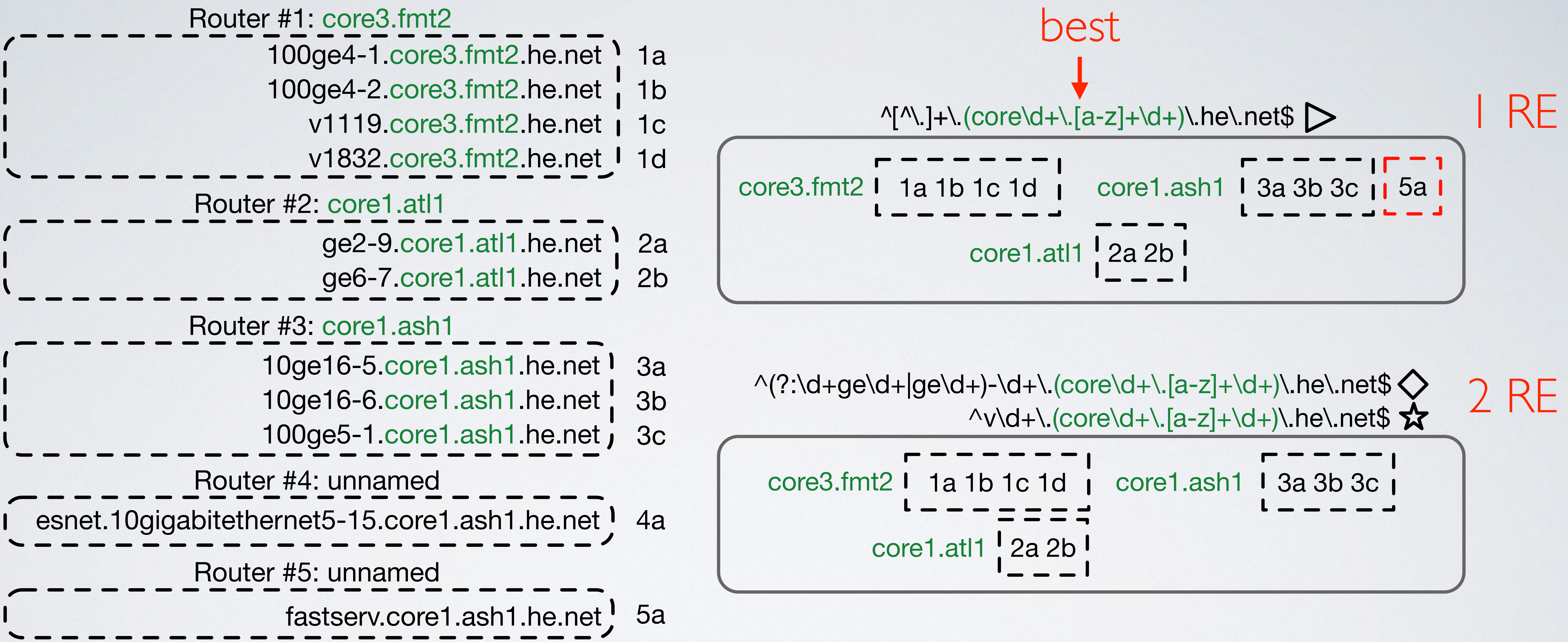
c-98-233-46-230.hsd1.md.comcast.net 7a

c-174-52-116-77.hsd1.ut.comcast.net 8a

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

ar01.area4.il.chicago 1a 1b 1c
pe04.ashburn.va.ibone 2a 2b 2c
cr01.miami.fl.ibone 3a 3b

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.