Bradley Huffaker bradley@caida.org
Matthew Luckie mjl@caida.org
Long Tran lt@ucsd.edu

ROUTER NAMING

1. ABSTRACT

In this project, we focused on determining whether a set of hostnames belonged to the same router. The methodology involved finding patterns from those hostnames that could uniquely identify such a router in a domain. We developed an algorithm that could be used to automatically infer these patterns (regular expressions), which in turn could be used to identify routers in an organization's naming convention. We validated the resulting regular expressions against the router aliases found in the 201404 ITDK dataset, which was also used as input to the algorithm. We documented the regular expression that had the best result for each domain name, deriving a total of 8 regular expressions for 8 domains with at least 100 hostnames.

2. INTRODUCTION

Alias resolution is a process of mapping IP addresses to routers. It is a critical step in modeling Internet topology maps from traceroute data. Accurate and efficient alias resolution techniques result in more sophisticated network maps, which in turn aid other topological research topic such as network security, monitoring, congestion, etc.

Traditional alias resolution techniques like those discussed in "Internet-Scale IPv4 Alias Resolution with MIDAR" require extensive active measurements (i.e intentionally injecting data packets into a network or sending them to servers, following their routes, and monitoring them continuously). Although MIDAR is a significantly improved alias resolution technique, it still requires extensive infrastructure, specialized software, and deployment to actively monitor and measure the aliases [1]. If a need to study Internet-scale IP addresses arises, alias resolution techniques that involve active measurements prove to be expensive and time-consuming.

This project provided another approach to alias resolution since it relied solely on the inferred hostnames after active measurements have been performed once. While attempting to perform alias resolution on millions of hostnames, this project's algorithm would only examine the hostnames as strings, which would be far less expensive than continuing to actively monitor and measure millions of hostnames. This would also mean that hostnames obtained from the past could be alias-resolved without any additional measurements.

3. BACKGROUND INFORMATION

3.1 Basic topology

An **Internet Protocol (IP) address** is a string of digits and characters that uniquely identifies a device (e.g, an iPad, a Macbook) participating in a computer network. An IP address is used to identify a host or a network interface and location of the network. The most common standard is Internet Protocol Version 4 (IPv4), which defines an IP address as a 32-bit number. IPv4 addresses are often written in human-readable forms (e.g, 192.172.226.91).

A **hostname** is a human-readable identification assigned to a device participating in a computer network. Hostnames usually append the Domain Name System (DNS) domain with the host-specific label. Hostnames should not be confused with domain names.

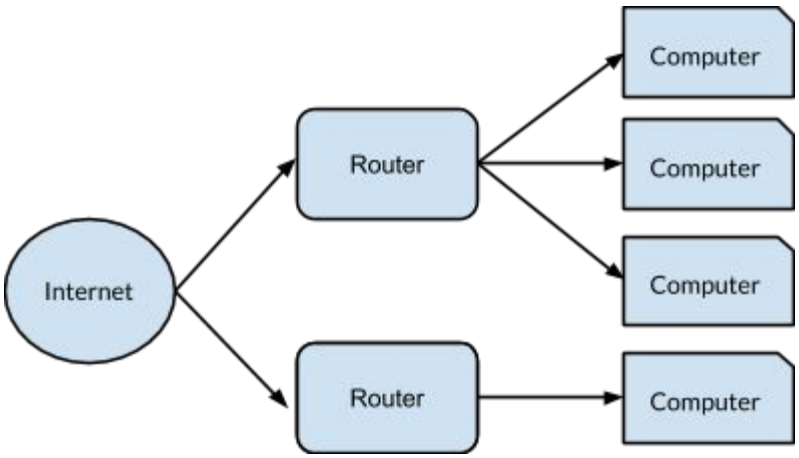
For example, the following are all hostnames of the domain name ‘virginmedia.net’.

```
hawk-cmts-12-gigaether-72.network.virginmedia.net
hawk-cmts-12-gigaether-73.network.virginmedia.net
hawk-cmts-12-gigaether-74.network.virginmedia.net

wolv-cmts-16-ge330.network.virginmedia.net
```

Note that although they might not all belong to the same router on the network, they are all considered hostnames of the same domain.

A **router** is a networking device that transfers data between two or more data lines from different networks. When a data packet enters a network, the router interprets its address information to direct, or route, the packet to the final destination.



In this project, we were interested in determining whether a set of hostnames belonged to the same router on a network. Going back to the example of hostnames above: although we knew which hostnames belonged to which router based on the inferred 201404 ITDK dataset (ground truth), we would like to *programmatically* determine that the following strings of hostnames `hawk-cmts-12-gigaether-72.network.virginmedia.net`, `hawk-cmts-12-gigaether-73.network.virginmedia.net`, `hawk-cmts-12-gigaether-74.network.virginmedia.net` were indeed all on the same router, while `wolv-cmts-16-ge330.network.virginmedia.net` was on a different one.

3.2 Similar research

Looking for patterns or hints in a large set of hostnames has proven to be effective and resource-saving. In “DRoP:DNS-based Router Positioning”, Bradley Huffaker, Marina Fomenkov, and kc claffy discussed an automated approach to *geolocation*, the process of mapping Internet resources (e.g, routers, IP addresses, data centers, etc) to their physical locations. The approach involved creating a dictionary that mapped strings with geographical hints (e.g, airport codes, cities and states abbreviations) to the corresponding physical locations. This dictionary was then cross-referenced with a large set of Fully Qualified Domain Name(FQDN) to determine if the geographical-referencing strings actually correlated to the correct physical location for a particular domain name [2].

DRoP’s ultimate goal was to derive an extensive set of rules that could determine whether a set of particular hostnames belonged to the same geographical location. DRoP searched through the hostnames for the geographical-referencing strings built before hand and then inferred their physical locations. The results were then validated using Round Trip Time (RTT) and Time to Live (TTL) values.

Similar to DRoP, the algorithm presented here traversed through a large set of hostnames to extract patterns or useful hints. But unlike DRoP, this project's ultimate goal was to derive a set of rules that determined whether a set of particular hostnames belonged to the same router. Starting with the inferred hostnames, we looked for patterns (regular expressions) that were common among hostnames on the same routers, but were unique in that domain. Since the goal was not geolocation, physical locations were irrelevant and therefore the lack of need for the dictionary of geographical hints. We only looked for common string patterns (if they existed) among the hostnames, used such patterns to (hopefully) identify a unique router, and validated the result against the same dataset input. The resulting rules were meant to help with DNS resolution based on the hostnames' aliases.

3.3 Data source

The data source used in this project came from The CAIDA UCSD Internet Topology Data Kit - 201404, found at http://www.caida.org/data/internet-topology-data-kit/. We used the historical 2014-04 release, which included the related router-level topologies and DNS lookups for all observed IP addresses. We primarily used the Nodes File and the Hostnames File.

ITDK was produced from active measurements conducted on Archipelago (Ark) measurement infrastructure, which obtained raw topology by performing traceroutes to randomly-chosen destinations in each routed /24 BGP prefix using 87 Ark monitors located in 37 countries from April 6, 2014 to April 19, 2014.

The Nodes File (`midar-iff.nodes.bz2`) listed the sets of IP addresses which belonged to each router in the following format:

```
node N<node_id>: <ip_0> <ip_1> <ip_2> … <ip_n>
```

Each line indicates a node node_id has interfaces ip_0 to ip_n. Interface addresses in 224.0.0.0/3 (IANA reserved space for multicast) are not real addresses; they were artificially generated to identify potentially unqiue non-responding interfaces in traceroute paths.

The Hostnames File (`midar-run-20140421-dns-names.txt.bz2`) contains the hostnames for the each observed IP address in the following format:

```
<timestamp>        <ip> <hostname>
```

Hostnames *.in-addr.arpa should be ignored, since they show that no hostname was found for that IP address. Each hostname could be used to successfully perform a reverse DNS lookup.

4. USING THE DATA

In order to build a database from the ITDK dataset, the following steps were taken:

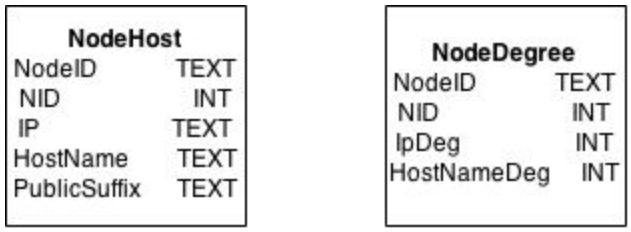
- We created a dictionary to map IP addresses to corresponding hostnames. This allowed constant lookup time when we needed to lookup an IP address' hostname.
    - For each line in the host file `midar-run-20140421-dns-names.txt`, we ignored all timestamps and hostnames that contained the substring "FAIL.NON-AUTHORITATIVE.in-addr.arpa" (or simply strings starting with an uppercase character). We created the mapping IP address to hostname in the said dictionary.
    - For example,

      ```
      1398912738      4.28.130.94     yahoo-inc.edge4.newyork1.level3.net
      1398920735      62.30.112.131   nmal-geam-v1.network.virginmedia.net
      1398912738      4.28.130.134    FAIL.NON-AUTHORITATIVE.in-addr.arpa
      ```

      would be used to create a dictionary:

```
{4.28.130.94       : yahoo-inc.edge4.newyork1.level3.net,
62.30.112.131      : nmal-geam-v1.network.virginmedia.net}
```

- We utilized Python's sqlite3 module to create a SQL database with the following schemas:



  Relation (table) NodeHost stored all IP addresses and their corresponding hostnames (found by looking up the dictionary built in step 1) of each NodeID (both in string and integer form) and the domain name (PublicSuffix).
  Relation (table) NodeDegree stored the number of IP addresses and the number of their corresponding hostnames for each NodeID (both in string and integer form)

- Once the IP-to-Hostnames dictionary and database with the schema above had been set up, data from the node file was ready to be put in the respective tables. For each line in the node file `midar-iff.nodes`, we ignored all commented-out lines and:
  - Determined the NodeID (i.e the substring that the regular expression `N\d*` matches).
  - For each of the IP address that was associated with this NodeID, we checked if it was in the dictionary created in step 1 and kept track of the number of successful lookups. We used Mozilla's public suffix library (https://publicsuffix.org/learn/) to determine the domain name of the IP address. Finally, we inserted all of these values into the relation NodeHost.
  - If the number of successful lookups in the previous step was at least 1, then we also inserted the following values into the relation NodeDegree: the number of IP addresses this NodeID had, the number of successful lookups, the NodeID (string form), and the NodeID (integer form).

5. METHODOLOGY

First, we compiled a large dictionary mapping a router identification number to a list of the router's hostnames. Then, we collected the longest common substring of each router's hostnames. We then derived a regular expression that matched each member of this longest common substring collection. We applied each regular expression to the original set of hostnames and benchmark how well it uniquely identified a router's hostnames.

With all the data stored into the SQL database, we performed a SQL query to sort the number of hostnames occurrences for each domain name in ascending order. Out of 4241 domain names captured, there were only 26 domains that had more than 150 hostnames. The validity and fidelity of the regular expressions generated would be more likely to be correct if more hostnames were sampled, therefore we picked out the top domains that had the highest hostnames occurrences count. They were comcast.net (789), rr.com (580), telia.net (557) , cogentco.com (427), virginmedia.net (409), ntt.net (402), alter.net (362), and yahoo.com (257).

For each domain name in this list, we:

- Performed a SQL query on NodeHost relation that returned all tuples (NodeID, the hostname, and the count of the hostnames) whose PublicSuffix column matched with the current domain being studied.

- Using these tuples, created a dictionary mapping a NodeID to a list of its associated hostnames and removed the substrings containing the domain name in all hostnames in this dictionary. Since we would need to find the longest common substring for each router (i.e NodeID or a key in this dictionary) that could potentially be the unique identification of the current router, the domain name would not be helpful since it appears in every hostname.

- For each NodeID in this dictionary, we:
    - computed the longest common substring of all of its hostnames. This was a rather brute-force solution since the algorithm was finding the longest common substring for every combination of every 2 hostnames, then put the longest one into a set. The found longest common substring for this router would (ideally) be this router's unique identification, yet still had a similar pattern with the longest common substring (and ideally unique identification) of other routers within this domain.
    - computed the leftover substring (all characters following the longest common substring in each hostnames.
    - generated a regular expression that matches the longest common substring and the leftover substring and store it in a set.

- We applied each of these proposed regular expressions in this set on all the hostnames (all the values in the dictionary built in step 2) and built several more dictionaries, all of which would be used to validate the accuracy of the regular expression. These dictionaries were:

name	key	value
nid_host	NodeID	count of hostnames in this node (ground truth)
nid_host_re	NodeID	count of hostnames found by using supposed regex
rid_nid	router id	dictionary mapping NodeID to list of associated hostnames (representing a single router)
nid_rid	NodeID	dictionary mapping router ID to list of associated hostnames (representing a single router)

    - nid_host kept track of the number of hostnames each NodeID had. This dictionary was built by traversing through the dictionary built in step 2 and counting the length of the list of associated hostnames.

    - nid_host_re kept track of the number of hostnames matched by the proposed regular expression for each NodeID. This value was incremented every time the proposed regular expression matched a hostname for this NodeID.

- The matched substring was the (supposedly) unique router identification. We put this router ID into rid_nid, a dictionary that mapped the router ID to another dictionary that mapped the NodeID to the list of associated hostnames.

- We also put the current NodeID into nid_rid, a dictionary that mapped the NodeID to another dictionary that mapped the just-found router ID to the list of associated hostnames.

- If the router ID found by the regular expression was indeed unique, then we should have had a one-to-one mapping between rid_nid and nid_rid. In another word, each key in rid_nid should have only map to one NodeID (i.e one router) and each key in nid_rid should have only map to one routerID (i.e the unique router identification). In addition, the dictionaries that they keys map to must have the same list of associated hostnames as their values.

6. VALIDATIONS

The validation process involved cross-referencing the ground truth and the results of using the supposed regular expressions. We traversed through the dictionaries mentioned above:

- rid_nid: every router ID should be mapped to only one NodeID, since the router ID must be unique and must identify exactly one particular NodeID (i.e router).
  - If a router ID mapped to zero router, we classified this as a *no router error*.
  - If a router ID mapped to more than one router, we classified this as a *mixed error* (more than one router have the same router ID, making it not unique).
- nid_rid: every NodeID (i.e router) should be mapped to only one router ID, since each NodeID should have exactly one router ID.
  - If a NodeID (i.e router) mapped to more than one router ID, we classified this as a *split error* (one router has more than one router ID).
  - If a NodeID (i.e router) mapped to exactly one router ID, we performed additional checks for this NodeID:
    - If the number of hostnames found by the regular expression equaled the number of hostnames the router was supposed to have *and* this one router ID mapped to exactly one value in rid_nid, we declared this to be the *perfect* case.
    - If the number of hostnames found by the regular expression did not equal the number of hostnames the router was supposed to have, we classified this as a *partial error*.

7. RESULTS & ANALYSIS

Domain	# of router (ground truth)	# correct inferring	# missed	# split	# mixed	# partial	Accuracy
virginmedia.net	403	376	0	5	4	0	93.30%
alter.net	259	232	0	13	2	2	89.60%
comcast.net	740	659	0	45	30	0	89.10%
cogentco.com	294	214	0	8	30	0	72.80%
yahoo.com	257	155	0	6	2	21	60.30%
ntt.net	121	59	0	32	25	0	48.80%
rr.com	538	256	0	99	56	0	47.60%
telia.net	97	23	0	14	16	12	23.70%

virginmedia.net is the domain with the highest success rate (93.3%). We looked at virginmedia.net's hostnames and observed that the naming convention for the 403 routers were extremely consistent. Their hostnames were consistently in the following format:

<**string**>-<**string**>-<**string**>-<string>-<string>.<string>.virginmedia.net

The regular expression found for this domain suggested that the first 3 groups uniquely identify the routers in this domain. This consistency in the naming convention could also be observed in the case for alter.net, comcast.net, and cogentco.com

rr.com has a rather low success rate. Looking at the hostnames in this domain, we observed there were a lot of "split cases" (i.e one router ID identifies more than one router). For instance, the following routers do not have any router ID that is unique for this domain. This observation is very common among the domains with lower success rates.

N7744 **rdc-24-93-12**-133.**wny.northeast**.rr.com
N7744 **rdc-24-93-12**-135.**wny.northeast**.rr.com
N7744 **rdc-24-93-12**-160.**wny.northeast**.rr.com
N7744 **rdc-24-93-12**-60.**wny.northeast**.rr.com

N7745 **rdc-24-93-12**-144.**wny.northeast**.rr.com
N7745 **rdc-24-93-12**-146.**wny.northeast**.rr.com
N7745 **rdc-24-93-12**-148.**wny.northeast**.rr.com
N7745 **rdc-24-93-12**-22.**wny.northeast**.rr.com

As seen in the example with rr.com above, there is no router ID that uniquely identifies these routers. However, our algorithm classifies this as a "split error", reducing the accuracy rate. We plan to overcome this problem by adding an additional checkpoint to disqualify routers whose computed longest common substring is not unique. Since the router ID should be unique and derived from routers' longest common substrings, these strings have to be unique as well. Such an improvement will distinguish actual "split cases" from instances without possible answers.

Domain	Supposed regular expression
virginmedia.net	([a-z\d]+)-([a-z\d]+)-([a-z\d]+)-([^\.]+\.){2}virginmedia.net$
alter.net	\.([a-z\d]+)\.([a-z\d]+)([^\.]+\.){1}alter.net$
comcast.net	-([a-z\d]+)\.([a-z\d]+)([^\.]+\.){3}comcast.net$
cogentco.com	([a-z\d]+)\.([a-z\d]+)([^\.]+\.){2}cogentco.com$
yahoo.com	\.([a-z\d]+)-([a-z\d]+)-([a-z\d]+)([^\.]+\.){2}yahoo.com$
ntt.net	([a-z\d]+)\.([a-z\d]+)\.[a-z\d]+\.([^\.]+\.){2}ntt.net$
rr.com	([a-z\d]+)\.([a-z\d]+)([^\.]+\.){1}rr.com$
telia.net	([a-z\d]+)-([a-z\d]+)-([^\.]+\.){2}telia.net$

The above table lists the domain names with the automatically generated regular expression that could extract their router IDs. For example, we could extract the following for the domain alter.net:

```
N23928 gigabitethernet6-0.gw3.tor2.alter.net
N23928 gigabitethernet6-1.gw3.tor2.alter.net
N23928 gigabitethernet7-0.gw3.tor2.alter.net

N23929 gigabitethernet6-0.gw5.tor2.alter.net
N23929 gigabitethernet6-1.gw5.tor2.alter.net
N23929 gigabitethernet7-0.gw5.tor2.alter.net
```

The bolded parts of the hostnames represent the router ID of their respective router: router N23928 is identified as “gw3.tor2”, while N23929 is identified as “gw5.tor2”.

Similarly, in the case of virginmedia.net, we have:

```
N5405 nmal-cmts-16-ge130.network.virginmedia.net
N5405 nmal-cmts-16-ge131.network.virginmedia.net
N5405 nmal-cmts-16-ge132.network.virginmedia.net

N5406 dund-cmts-11-ge130.network.virginmedia.net
N5406 dund-cmts-11-ge131.network.virginmedia.net
N5406 dund-cmts-11-ge132.network.virginmedia.net
```

Router N5405 is identified as “nma1-cmts-16” and N5406 is identified as “dund-cmts-11”.

8. CONCLUSION

We have developed an approach to determine whether a set of hostnames belong to the same router by examining only naming convention patterns. This eliminates the need to continuous active measurements, which are rather expensive and resource-consuming.

We have had developed an algorithm to derive regular expressions that could extract router IDs for a total of 8 domains. The most successful regular expression has a 93.3%.

The methodology involved organizing the 201404 ITDK dataset into a SQL database. Next, we picked out the domain names with the most router count. For each of these domains, we computed the longest common substring and any leftover substring in these hostnames. We then generated a set of regular expressions to match these substrings. This set of regular expressions were then applied on all the hostnames of the current domain and the one with the highest result was selected.

9. REFERENCES

[1] K. Keys, Y. Hyun, M. Luckie, and k. claffy, “Internet-Scale IPv4 Alias Resolution with MIDAR,” IEEE/ACM Transactions on Networking, vol. 21, Apr 2013.
[2] B. Huffaker, M. Fomenkov, and k. claffy, "DRoP:DNS-based Router Positioning", ACM SIGCOMM Computer Communication Review (CCR), vol. 44, no. 3, pp. 6--13, Jul 2014.