Learning Regexes to Extract Network Names from Hostnames

Matthew Luckie
University of Waikato
Hamilton, New Zealand
mjl@wand.net.nz

Alexander Marder
CAIDA, UC San Diego
La Jolla, CA, USA
amarder@caida.org

Bradley Huffaker
CAIDA, UC San Diego
La Jolla, CA, USA
bradley@caida.org

k claffy
CAIDA, UC San Diego
La Jolla, CA, USA
kc@caida.org

ABSTRACT

We present the design, implementation, evaluation, and validation of a system that automatically learns regular expressions (regexes) to extract network names from Internet hostnames assigned by operators using their own conventions. Our fully automated method does not rely on a human to provide a starting regex, labeled examples of valid extractions, or a dictionary of network names. Our method first learns the dictionary of network names, and then automatically generates and evaluates regexes that extract these names. We validate our dictionary against ground truth, finding that 97.3% of the names our regexes extract are valid names for the networks.

CCS CONCEPTS

• **Networks → Naming and addressing**.

KEYWORDS

Regular expression learning, Internet topology

ACM Reference Format:
Matthew Luckie, Alexander Marder, Bradley Huffaker, and k claffy. 2021. Learning Regexes to Extract Network Names from Hostnames. In *Asian Internet Engineering Conference (AINTEC '21), December 14–16, 2021, Virtual Event, Japan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3497777.3498545

1 INTRODUCTION

Every Internet-connected network device requires an Internet Protocol (IP) address to communicate with other Internet devices. Since the numbers in IP addresses provide little context to people, the Domain Name System (DNS) provides mappings between IP addresses and *hostnames* – a sequence of alphanumeric characters and punctuation that people can understand. To participate in the DNS, an organization registers a *suffix* (e.g., zayo.com), and can use any sequence of characters they choose in the hostname, prior to the suffix.

zayo-**netflix**.iad10.us.zip.zayo.com	(A)
netflix.sgix.sg	(B)
netflix-gw.customer.alter.net	(C)
netflix1.fra.ecix.net	(D)
netflix1-lacp-100g.hkix.net	(E)
netflix-inc.ear2.sanjose1.level3.net	(F)
nflx1.ix.fl-ix.net	(G)
as2906.saopaulo.sp.ix.br	(H)
zayo.**vdms**.mpr4.atl6.us.zip.zayo.com	(I)
verizondigitalmedia-com.customer.alter.net	(J)
edgecast-gw.customer.alter.net	(K)
verizondms.jfk10.atlas.cogentco.com	(L)
as15133.cr2-nyc6.ip4.gtt.net	(M)

Figure 1: Example hostnames for connections with Netflix (A-H) and Verizon Digital Media Services Edgecast (I-M). Each operator, indicated with their underlined suffix, has its own convention. We highlight network names with a bold green font, and geographic locations with a blue font.

This paper focuses on extracting network names that network operators embed in hostnames to convey interconnection structure. When two networks interconnect in the Internet, one network will provide an IP address and associated hostname to the other network to facilitate interconnection. How operators embed a reference to the interconnecting network, and any other information they choose to disclose in the hostname, is solely up to the operator assigning the name. Figure 1 shows operators placing references to "Netflix" and "Verizon Digital Media Services Edgecast" in the hostnames. Each network operator – identified by their suffix – uses its own conventions to convey information; e.g., hostname A shows that Zayo connects with Netflix in the Washington, D.C. ("iad") area. These hostnames are important to operators when troubleshooting network-level problems, since diagnostic tools output IP addresses that require additional context. These hostnames also prove immensely valuable to Internet infrastructure researchers, as identifying the network that operates a router is critical to understand connectivity within and between organizations. For example, recent work that required accurate identification of the network that operated a router examined patterns of congestion between networks [8, 10, 31], the connectivity and performance of cloud providers [23, 26, 32, 33], macroscopic impacts of submarine cable deployments [9], load balanced paths within and between networks [25, 28], routing detours [12], and the ability of a certificate authority to defend against domain validation attacks [3].

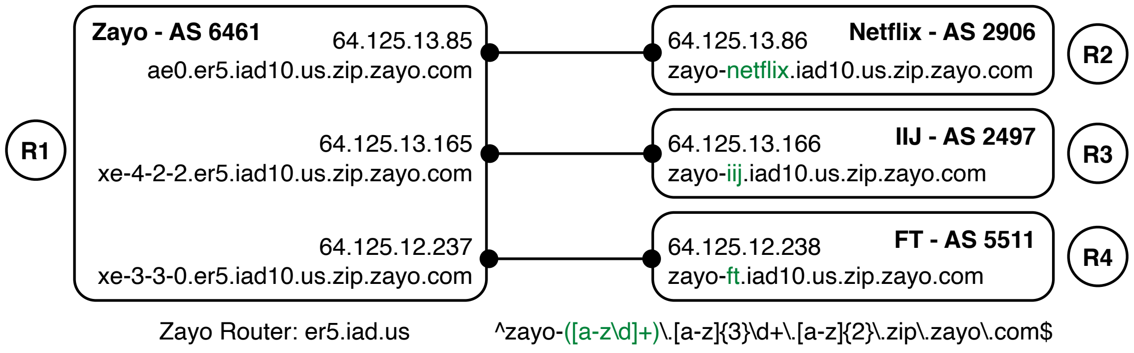

Zayo - AS 6461
64.125.13.85
ae0.er5.iad10.us.zip.zayo.com
64.125.13.165
xe-4-2-2.er5.iad10.us.zip.zayo.com
64.125.12.237
xe-3-3-0.er5.iad10.us.zip.zayo.com
R1
64.125.13.86
Netflix - AS 2906
zayo-netflix.iad10.us.zip.zayo.com
R2
64.125.13.166
IIJ - AS 2497
zayo-iij.iad10.us.zip.zayo.com
R3
64.125.12.238
FT - AS 5511
zayo-ft.iad10.us.zip.zayo.com
R4
Zayo Router: er5.iad.us
^zayo-([a-z\d]+)\.[a-z]{3}\d+\.[a-z]{2}\.zip\.zayo\.com$

Figure 2: A logical diagram showing how a single router operated by Zayo (R1) interconnects with routers operated by Netflix (R2), Internet Initiative Japan (R3, IIJ), and France Telecom (R4, FT). Each of these networks has their own Autonomous System (AS) number that uniquely identifies them. Our method first automatically learns the mapping of AS numbers to network names, and then automatically learns regexes to extract these network names.

There is no universal convention for embedding information in these hostnames. They evolve over time to convey new or different information as networks and operators change. Hostnames often use abbreviations and common identifiers that other network operators understand. But, they lack a universal, well-defined format and dictionary that would make them easily decipherable by machines. Instead, each network typically decides on a small number of patterns and identifiers that they use when generating hostnames in their suffix.

Regular expressions (regexes) are the dominant method of practically extracting information from text that contains information-encoding patterns [15]. Most regexes are hand-crafted with a human in the loop (HITL) to account for complex expressions [34]. However, a HITL approach does not scale for our problem; there are tens of thousands of networks (suffixes) in the Internet with varied conventions, but only a few hundred networks (suffixes) embed interconnection information. In this paper, we describe a method for *automatically* learning regexes to extract and interpret the name of the network that operates a router in the Internet from hostnames. Unlike hand-crafted regexes, our approach is easily reproducible, does not rely on a set of manually labeled training data, and adapts to changes in suffix conventions over time.

Our primary contribution in this work is to design, implement, and validate an automated method that (1) learns a dictionary of network names, and (2) learns regexes that extract network names from hostnames. We apply this method to the most comprehensive Internet topology data available to researchers. In the interest of extensibility and reproducibility, we publicly release our software [17] and datasets [22].

2 BACKGROUND AND RELATED WORK

2.1 Internet Interconnections

This paper focuses on extracting information from Internet router hostnames; we briefly discuss how network operators interconnect routers, and thus the significance of the hostnames. We illustrate with an example using router R1 in Zayo's global ISP network on the left hand side of figure 2, which connects with three routers on the right hand side operated by Netflix, Internet Initiative Japan (IIJ), and France Telecom (FT), so that these networks can exchange traffic. Zayo controls IP addresses beginning with IP address prefix 64.125, and provides addresses starting with this prefix to Netflix, IIJ, and FT so that these networks can connect to Zayo.

Because Zayo controls IP addresses beginning with the address prefix 64.125, it also controls the hostnames associated with those addresses. Zayo's convention is to begin those hostnames with the string "zayo", followed by a string that identifies the neighbor network in the hostname, a three-character string identifying the geographic location where they connect, and the corresponding two-letter country code. There is no database that identifies the name of the network that controls a router with a given IP address or hostname, presenting a challenge to any method that seeks to automate learning these conventions.

The Internet is organized into Autonomous Systems (ASes) – each AS typically maps to a single organization and is identified by a unique number (an ASN). The Internet's routing protocol uses ASNs to organize dynamic routing; i.e., determine how to get traffic between two Internet users. There is no public ground truth that identifies which ASes use which IP addresses – e.g., that AS 2906 uses 64.125.13.86. There are heuristic algorithms that imperfectly infer this mapping, such as RouterToAsAssignment [13] (RTAA) and bdrmapIT [24]. bdrmapIT is the current state of the art, which inferred the correct ASN for 95.3% of addresses according to ground truth from four network operators [24].

While there is no database that identifies the name of the network that controls a router with a given IP address or hostname, there are two databases that record network names for ASNs. The first, known as "WHOIS", contains information that a network operator submits to their Regional Internet Registry (RIR) when the operator obtains an ASN, including contact information for the network, as well as an organization name. While every AS in the Internet has a record in WHOIS, operators update these records infrequently and so the records can become out of date [5]. The second, known as "PeeringDB", is self-reported by vetted operators at each network, who use it to help coordinate interconnections. PeeringDB contains more information than WHOIS, such as specific physical

W OrgName:	MCI Communications Services, Inc. d/b/a Verizon Business
W ASName:	EDGECAST
PDB Name:	Verizon Digital Media Services (EdgeCast Networks)
PDB Website:	https://www.verizondigitalmedia.com/
PDB A/K/A:	EdgeCast Networks
PDB IRR:	AS-EDGECAST

(a) AS 15133 - Verizon Edgecast

W OrgName:	Orange S.A.
W ASName:	Opentransit
PDB Name:	Orange
PDB Website:	https://wholesalesolutions.orange.com/
PDB A/K/A:	Opentransit - IP Transit 5511
PDB IRR:	AS-OPENTRANSIT

(b) AS 5511 - Orange France Telecom

Figure 3: Information available for ASes in the WHOIS (W) and PeeringDB (PDB) databases. The unstructured information in these databases is useful for cross-validating inferences, but is incomplete.

locations (buildings) where the network can interconnect. PeeringDB records are updated more regularly because the interface is easier to use [16]. However, there are fewer networks recorded in PeeringDB; as of March 2021, there are ≈19K routed networks in PeeringDB, compared to ≈71K routed networks in WHOIS.

Figure 3a shows WHOIS and PeeringDB information for Edgecast, the network referenced in hostnames I-M in figure 1. While the network names embedded in those hostnames relate to information in WHOIS and PeeringDB, hostnames I and L contain derivatives "vdms" and "verizondms". Figure 3b illustrates information available for FT, featured in router R4 in figure 2. Zayo uses an acronym (FT) as a historic name for Orange (France Telecom) but neither WHOIS nor PeeringDB contain a reference to France Telecom.

2.2 Regular Expression Grammar Induction

Learning structure from example text is known as *grammar induction* in machine learning. Techniques in the literature generally focus on identifying patterns in text, such as software names, phone numbers, and university course numbers. ReLIE (2008) reduced the manual effort in building a regex [15] but relied on a human providing a starting regex and input data, which their method would then improve. In 2010, Babbar and Singh [1] introduced a technique that could learn regexes even when the human providing the starting regex had lower domain expertise than that assumed by ReLIE. In 2012, Murthy et al. [27] presented a technique to improve recall of regexes that involved human feedback and improved input regexes for identifying these patterns in general text. In 2018, Simoes et al. [29] continued this line of work, where a human provides a seed regex that is 100% precise, and their method improved recall.

Because writing regexes requires expertise, an alternate line of work infers regexes using sample extractions [4, 11]. The current state of the art, RegexGenerator (2016), relies on a human to provide examples of valid extractions from a set of input data, for which their method builds a regex [2]. There are tens of thousands of suffixes in the Internet, each with their own convention; relying on a human to identify extractions is not scalable, as conventions and networks evolve.

The computer network research community has previously built automated trial-and-error approaches to extract information from hostnames, as the punctuation-oriented patterns in hostnames are suited to this approach. In 2013, Chabarek and Barford [7] inferred simple regexes to extract link speeds from hostnames using a dictionary of known interface types. In 2014, Huffaker et al. [14] presented DRoP, which learned simple regexes to extract geographic locations from hostnames (the blue font in figure 1) using a dictionary of known city names and airport codes. There are no equivalent dictionaries for network names. In 2019, we introduced an automated approach to extract meaningful structure from router hostnames, which we called *hostname orthography* and embedded in a software module called *Hoiho* [19]. Our first goal was to identify the *router name* in hostnames, defined as substrings in common across all interfaces of the same router – "er5.iad.as" from R1 hostnames in figure 2. In 2020, we extended the Hoiho tool to automatically learn regexes to extract the ASN that operates the router [21] – AS 2906 from hostname H and AS 15133 from hostname M in figure 1. In 2021, we extended Hoiho to automatically learn regexes that extract geographic locations from hostnames [20] – "iad" from hostname A and "sanjose" from hostname F in figure 1.

The Hoiho tool provides an ideal platform to accelerate research on extracting information from hostnames, and we leveraged it when we implemented our method (§3). In this work, we focus on the problem of automatically building regexes to extract network names from hostnames with no human in the loop. Our method does not rely on a human providing a starting regex, examples of valid extractions, or an input dictionary of network names.

3 METHOD

Because there is no database that identifies the name of the network that controls a router with a given IP address or hostname, or a corpus detailing which network operators embed network names in hostnames and where they embed the names, our method must weigh up evidence that specific lexical tokens are network names. Our method has four phases: (1) automatically seeding a dictionary that maps network names to ASNs, (2) automatically learning seed regexes to extract these names for different networks, (3) automatically refining the dictionary based on likely network names extracted by those regexes, and (4) automatically refining the regexes that extract these names.

Our method is informed by the largest available source of router-level Internet topology data – CAIDA's Internet Topology Data Kit (ITDK) [6]. The ITDK contains a set of routers with their associated hostnames, and an owning AS for each router imperfectly inferred using a heuristic method (§2.1). Our method uses the inferred AS for each router as a label for the likely AS that operates the router. CAIDA has released 19 ITDKs between July 2010 and March 2021, and we use them all in §4.

We implemented all four phases of our method in Hoiho (§2.2) which contained code for processing the ITDK and a framework for generating and evaluating regexes from prior work [19–21].

Hostname	ASN	Suffix
zayo-netflix.iad10.us.zip.zayo.com	2906	zayo.com
zayo-level3.cdg11.fr.zip.zayo.com	3356	zayo.com
netflix-gw.skt.cw.net	2906	cw.net
level3-gw.dus.cw.net	3356	cw.net
netflix-ic-324205-hls-b2.c.telia.net	2906	telia.net
level3-ic-347052-hls-b1.c.telia.net	3356	telia.net

String	Mapping
netflix	2906 (3) zayo.com, cw.net, telia.net
level3	3356 (3) zayo.com, cw.net, telia.net
zayo	2906 (1) zayo.com — 3356 (1) zayo.com
iad10	2906 (1) zayo.com
cdg11	3356 (1) zayo.com
zip	2906 (1) zayo.com — 3356 (1) zayo.com

Figure 4: Seeding a dictionary to identify likely network names in hostnames. Our method adds strings correlated with an ASN to a seed dictionary (§3.1).

We designed and implemented new algorithms for learning the network name dictionary, and a new method to evaluate regexes that extract network names. We also modified Hoiho's regex builder to handle the extraction of network names. We integrated all of this functionality into the public version of Hoiho [17] to enable reproducibility, use, and extension by the research community.

3.1 Phase 1: Build Seed Name Dictionary

The goal of the first phase of our method is to learn strings in hostnames whose presence is associated with the labeled AS for the routers in the training data. Our intuition is that several operators will generally use the same string to refer to the same AS, and we can use this signal to seed a dictionary of likely network names.

Because operators use punctuation in hostnames to structure hostnames to help humans interpret them, our method splits each hostname into its component strings delimited by punctuation. For the first hostname in figure 4, our method extracts "zayo", "netflix", "iad10", "us", and "zip". Our method then tags each string with the labeled AS and suffix for the hostname. Our method repeats this process for all hostnames in the input dataset. Figure 4 shows a subset of the string mappings; the remainder we elide for space.

For the set of hostnames in figure 4, the string "netflix" is correlated with AS 2906 and "level3" is correlated with AS 3356. This is because three different network operators (Zayo, CW, and Telia) have each labeled "netflix" and "level3" routers with those strings. Other strings – "iad10", "cdg11", and "zip" – are not correlated with any ASN. For example, Zayo used "zip" in hostnames for routers operated by different ASes, and only Zayo used "iad10" in a hostname for a router operated by AS 2906.

For each string, our method ranks ASN mappings by the number of suffixes with that mapping – the number of suffixes is a proxy for the number of network operators using that mapping. Our method adds an entry to the seed dictionary that maps a string to an ASN provided at least three different suffixes have each used that string for that ASN, and that no other ASN associated with that string also had at least three different suffixes using that string. This limits adding common strings in hostnames to the seed dictionary, such as geolocation (many operators embed "iad" because they have routers in Washington, D.C). For the hostnames in figure 4, our method would add {netflix ⇒ 2906} and {level3 ⇒ 3356} to the seed dictionary. This heuristic will still inevitably include strings that, by chance, are correlated with an ASN but do not refer to its name. However, the seed dictionary is good enough for Hoiho to use in the next phase, which automatically builds and evaluates regexes that extract apparent network names from hostnames.

Hostname	ASN	
zayo-netflix.iad10.us.zip.zayo.com	2906	(A)
zayo-level3.cdg11.fr.zip.zayo.com	3356	(B)
zayo-tata.ams1.nl.zip.zayo.com	6453	(C)
zayo-sprint.er2.ord7.us.zip.zayo.com	1239	(D)
zayo-tata.mpr1.fra4.de.zip.zayo.com	6453	(E)
zayo-telefonica.er2.dfw2.us.zip.zayo.com	12956	(F)

Seed Name Dictionary netflix:2906 level3:3356 tata:6453 sprint:1239 telefonica:12956

`^[^-]+-([^\.]+)\.[^\.]+\.[^\.]+\.[^\.]+\.zayo\.com$` RE1
True Positives: 3 of 6 — hostnames A, B, C

`^[^-]+-([^\.]+)\.[^\.]+\.[^\.]+\.[^\.]+\.[^\.]+\.zayo\.com$`
True Positives: 3 of 6 — hostnames D, E, F RE2

`^[^-]+-([^\.]+)\..+\.zayo\.com$` RE3
True Positives: 6 of 6 — hostnames A, B, C, D, E, F

Figure 5: Building and evaluating seed regexes to identify where an operator embeds neighbor network names (§3.2). Our evaluation method prefers regexes that extract more congruent network names (e.g., RE3, 6 TPs) over regexes extracting fewer congruent names (RE1 and RE2, 3 TPs).

3.2 Phase 2: Build Seed Regexes

Because the seed dictionary that our method built in phase one will contain spurious entries, the goal of the second phase is to identify *where* in the hostnames the operators for each suffix embed network names. Learning this pattern will allow our method to automatically discover that dictionary entries derived from a different portion of the hostname are spurious. Therefore, this phase builds seed regexes that allow our method to infer the lexical semantics of hostnames for different suffixes.

We illustrate this phase in figure 5, which contains six hostnames for neighbors of Zayo, their labeled ASN, as well as the seed dictionary that our method inferred in phase one. This phase first identifies which strings in a hostname, if any, have an entry in the seed dictionary with an ASN matching the labeled ASN. Our method uses Hoiho's regex builder to build regexes that aim to extract the string using the punctuation in the hostname as a template. Hoiho builds structure in the regex recursively by using regex components that exclude specific punctuation depending on the punctuation at the beginning and end of each portion (e.g., [^\.]+ and [^-]+ match sequences of characters that do not contain a dot or hyphen, respectively), or match anything (i.e., .+) at most once per regex. For hostnames A-C in figure 5, our method builds regexes

RE1 and RE3, and for hostnames D-F, our method builds regexes RE2 and RE3, as well as other regexes that we elide for space.

We implemented a method in Hoiho that evaluates these regexes, examining the effectiveness of the regexes for extracting network names from all hostnames, using the following simple criteria. Our method assigns a true positive (TP) if the regex extracted a string whose dictionary ASN matched the labeled ASN. Our method assigns a false positive (FP) if the regex extracted a string whose dictionary ASN is different to the labeled ASN. Our method assigns a false negative (FN) if the regex did not extract a string that is in our dictionary whose ASN matches the labeled ASN. Our method does not penalize a regex that extracts a string that is not in the dictionary; the string might reflect a network name that our method did not learn because the training data lacked enough samples.

3.3 Phase 3: Refine Name Dictionary

The third phase uses the seed dictionary from phase one (§3.1) and the seed regexes from phase two (§3.2) to automatically remove spurious dictionary entries. Our method selects the *best* regex for each suffix, provided that regex found names for at least three unique networks – i.e., we have confidence that there is a convention for embedding network names, and our method did not extract a string coincidentally correlated with an ASN. We define the *best* regex as the regex that maximizes TP-FP; i.e., our method prefers a regex that extracts network names with dictionary ASNs matching the labeled ASN over a regex that matches fewer hostnames or extracts strings inconsistent with the labeled ASN. Using the example regexes in figure 5, our method selects RE3 as the best regex because it has the highest TP-FP sum of 6.

Our method then re-creates the dictionary from scratch using the strings *extracted by the regexes* using the following process. First, our method tags the extracted strings with the labeled ASN from their corresponding routers, as well as the hostname's suffix. Second, our method populates the dictionary using the same criteria as in §3.1 – i.e., our method adds a string to the dictionary provided at least three different suffixes have each used that string for routers with the same ASN, and that no other ASN associated with that string also had at least three different suffixes for that string. Third, our method considers strings with fewer than three suffixes suggesting the ASN mapping, adding a string if it is similar to another entry for the ASN already in the dictionary. For example, the operator has replaced a digit with its corresponding word form (e.g., "three" for "3") but the string is otherwise equivalent, or the string is a longer version of an entry in the dictionary (e.g., "flagtelecom" where "flag" is in the dictionary). Finally, our method adds strings with two suffixes suggesting the ASN mapping, provided there are no other ASNs tagged with the string. Our intuition is that these strings are very likely to be network names because phase 2 established *where* in a hostname each operator embeds network names, and two network operators used that string for that ASN. We show that our approach to refining the name dictionary is effective in §4.

3.4 Phase 4: Build Refined Regexes

The final phase of our method automatically builds refined regexes that extract network names from hostnames. This phase begins along the same lines as phase two (§3.2), building regexes to extract strings in the refined dictionary built in phase three (§3.3), only using simple punctuation-based regex segments, such as those in figure 5. Our method uses Hoiho's regex builder to refine these regexes, examining the strings covered by each punctuation-based regex segment. Figure 6 illustrates refining RE3, focusing on the two punctuation-based segments in the regex. The first always matches the word "zayo", while the second always matches sequences of alphanumeric characters ("netflix", "level3", and so on). Therefore, Hoiho creates a refined regex using the simple regex as a base, replacing the first segment with "zayo", and the second segment with a segment that matches a sequence of alphanumeric characters, emitting RE4 in figure 6.

Some operator conventions are more complex. Figure 7 shows example hostnames for BIX, where the operator sometimes embeds the string "-b" in the hostname, suggesting the neighbor network is connected to BIX with multiple routers. The punctuation-based

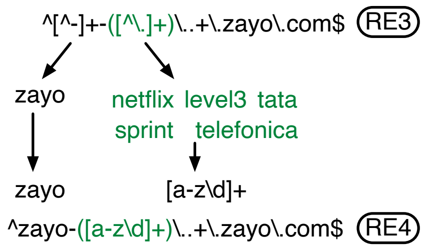
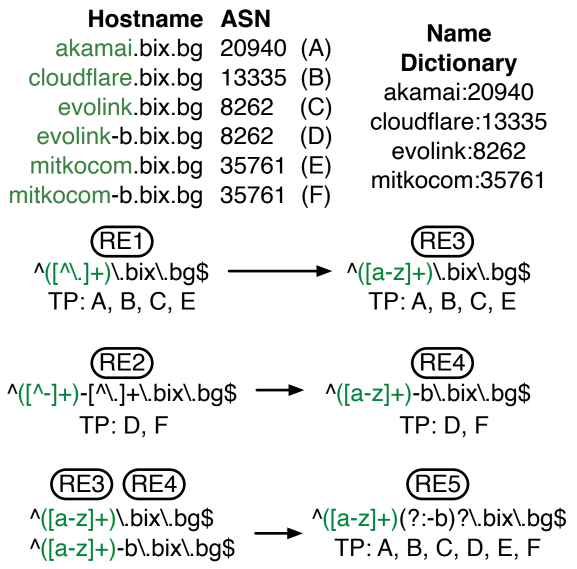

^[^-]+-([^\.]+)\..+\.zayo\.com$
RE3
zayo
netflix level3 tata
sprint telefonica
zayo
[a-z\d]+
^zayo-([a-z\d]+)\..+\.zayo\.com$
RE4

Figure 6: Refining punctuation-based regex segments based on the strings each segment matches to build a more-specific refined regex. In this example, the first segment always matches "zayo", and the second segment always matches alphanumeric characters.



Hostname	ASN	
akamai.bix.bg	20940	(A)
cloudflare.bix.bg	13335	(B)
evolink.bix.bg	8262	(C)
evolink-b.bix.bg	8262	(D)
mitkocom.bix.bg	35761	(E)
mitkocom-b.bix.bg	35761	(F)

Figure 7: Refining and merging regexes to capture naming variation in suffixes. Our method merges RE3 and RE4 to produce RE5, which covers the complexity in the bix.bg naming convention.

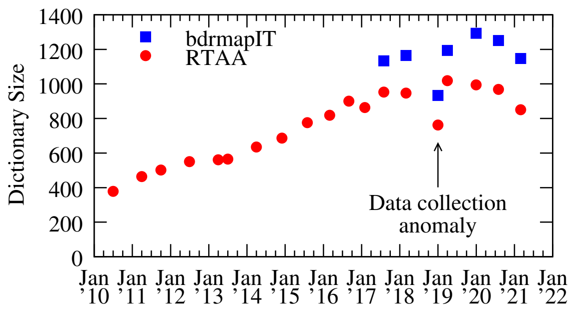


Figure 8: The size of the automatically learned network name dictionary grew over time as the Internet grew.

regexes (RE1, RE2) each capture a subset of the network names. After Hoiho refines these regexes to build RE3 and RE4, it is clear that the only difference between the two regexes is the string "-b". Therefore, Hoiho merges these regexes together to produce RE5, which includes a regex segment that optionally matches the "-b" so that a single regex covers variation in the hostnames.

Some suffixes have more complex differences, where there is no straight-forward way to merge the regexes that cover the differences. Hoiho emits a set of regexes to capture the complexity in the hostname conventions for these suffixes.

4 RESULTS

We applied our method to 19 ITDKs constructed by CAIDA (§3) between July 2010 and March 2021 that contain IPv4 routers annotated with an ASN inferred by RTAA or bdrmapIT (§2). Our implementation processed each IPv4 ITDK, which contained ≈850K–1.7M hostnames (≈53–62% of router interfaces were named) in ≈3–4 minutes using an 4-core Intel Core i7. Four ITDKs also contain IPv6 routers with an ASN inferred by bdrmapIT. Because IPv6 is not as widely deployed as IPv4, the IPv6 ITDKs are smaller – ≈68K-87K hostnames (15–16% of router interfaces were named). We examine our results for both IPv4 and IPv6, though we focus on the IPv4 results because the IPv4 ITDKs are larger.

We discuss the results in two parts. We first examine the dictionary our method learned over time on the input data (§4.1, §4.2). Then, we examine the regexes our method learned (§4.3).

4.1 Properties of Learned Dictionary

Figure 8 shows the size of the dictionary our method learned from IPv4 routers for each of the 19 ITDKs. The number of network names learned grew over time, as the Internet grew. We color the data points by the heuristic method CAIDA used to annotate router ASNs. Because the dictionary is inferred, the number of items in the dictionary increased when CAIDA changed the heuristic method it used to annotate router ASNs (RTAA) to a more accurate heuristic method (bdrmapIT) in August 2017. We also constructed ITDKs using RTAA beginning August 2017, and figure 8 shows the size of the dictionary inferred from a bdrmapIT ITDK is 17–35% larger than the dictionary inferred from an equivalent RTAA ITDK. There is one anomalous dip in the size of the dictionary inferred for the January 2019 ITDK caused by data collection issues for that edition of the ITDK, which used a different method to obtain hostnames (zdns) than the other ITDKs.

Customer cone size	Named / Total	Coverage
0 (stub)	357 / 60535	0.6% of 84.7%
1-9	308 / 8640	3.6% of 12.1%
10-99	221 / 1882	11.7% of 2.6%
100-999	87 / 322	27.0% of 0.5%
1000-9999	31 / 43	72.1% of 0.1%
≥ 10000	11 / 11	100% of 0.0%
	1015 / 71433	1.4% of 100%

Table 1: Coverage of named ASes by customer cone size. Our method is more likely to infer names for larger ASes.

Using the March 2021 IPv4 ITDK, our method learned a single name each for the vast majority of the 1,015 ASNs represented in our dictionary – 926 ASNs covering 91.2% of the ASNs. A further 63 (6.2%) and 15 (1.5%) had two or three names, and only 11 (1.1%) had four or more names. The maximum number of suffixes for a dictionary entry was 42, which was for Cloudflare, a large content distribution network that is densely connected to other networks in the Internet.

Table 1 shows the coverage of ASes our method inferred names for, by *customer cone size* – the number of ASes that are customers of a given AS, as well as those customers' customers, and so on [18]. The coverage of our method increases as customer cone size increases. The vast majority (84.7%) of ASes are stub ASes with no customers; our method infers names for only 0.6% of these ASes, because these ASes are typically not well connected to other networks and so our method is unlikely to infer names for these. At the other extreme, our method infers a name for 77.8% of ASes with a customer cone size of at least 1000.

4.2 Validation of Learned Dictionary

For the March 2021 IPv4 bdrmapIT ITDK, which contains 1.5M router hostnames, our method inferred a dictionary containing 1,147 network names for 1,015 ASNs. We were able to confirm that 1,116 (97.3%) of these names were congruent with information in WHOIS, PeeringDB, and via manual inspection. Table 2 summarizes the outcome of our cross-validation against these databases.

To validate our dictionary, we compared the names our method learned for each ASN with two public sources of technical data – ASN records in the WHOIS and PeeringDB databases (§2.1, figure 3) – using a script. Our script compares names our method learned against strings in these sources – inferring a correct name if it finds (1) an exact match, (2) a match by forming an acronym from capital letters (e.g., "VDMS" from "Verizon Digital Media Services" in figure 3a), (3) or a match by contracting (abbreviating) the string. Table 2 shows that we validated 90.5% of the names using these databases, and that each field provided diminishing returns to our ability to validate.

We further performed a manual inspection when our script could not find a matching string in the WHOIS and PeeringDB databases, adding these entries to a manually-constructed database when we found evidence that the name was valid for the ASN, perhaps because the network re-branded, or acquired another network. This effort was valuable because it validated more names than the combined gain for all of the PeeringDB sources.

Source	Total	Gain	Cumulative	
W OrgName	841	841	841	75.4%
W ASName	780	108	949	85.0%
W Total	949			
PDB Name	750	27	976	87.5%
PDB Website	564	20	996	89.2%
PDB A/K/A	341	12	1,008	90.3%
PDB IRR	279	2	1,010	90.5%
PDB Total	843			
Manual	106	106	1,116	100%
Total		1,116		

Table 2: Contribution of each source to cross-validation. Overall, 1,116 of 1,147 (97.3%) learned names are a valid name for the corresponding ASN.

# Suffixes	Frequency		Correct
1	88	(7.7%)	97.7%
2	565	(49.3%)	95.6%
3	253	(22.1%)	98.4%
4	108	(9.4%)	100%
5	49	(4.3%)	100%
6	29	(2.5%)	100%
7+	55	(4.8%)	100%
Total	1,147		97.3%

Table 3: Number of suffixes in which we found a matching dictionary entry, and their validation. Every dictionary entry observed in at least four suffixes was correct.

Table 3 shows the number of suffixes in which we found a matching dictionary entry; the majority (79.0%) were found in three or fewer suffixes. Table 3 further shows that every dictionary entry found in at least four suffixes was correct in our validation data. The percentage of correct entries for dictionary entries found in one suffix is higher than those in two suffixes; to be included in the dictionary, an apparent network name found in one suffix had to be similar to an entry already in the dictionary (§3.3).

We examined the benefit of our multi-phase approach to constructing the dictionary by evaluating the seed dictionary at the end of Phase 1 (§3.1) with the same validation data. For the March 2021 IPv4 ITDK, the seed dictionary contained 787 entries, and 73 were wrong – 90.7% were correct overall. At the end of Phase 3 (§3.3) the dictionary validation is 97.3% for 1147 entries (31 are wrong) as shown in Table 2. The implication is the refined dictionary gets much larger and sheds most of the wrong entries.

4.3 Properties of Learned Regexes

Figure 9 summarizes the performance of the best regexes per suffix that our method learned across the 19 ITDKs. As with the dictionary (§4.1), the number of suffixes that embed network names for which our method learned regexes grew over time as the Internet grew.

We classified a regex as *good* if it extracted at least three unique ASes congruent with labeled ASes with a PPV ≥ 80% – the green portion of each bar in figure 9. Our method built 41 – 105 *good*

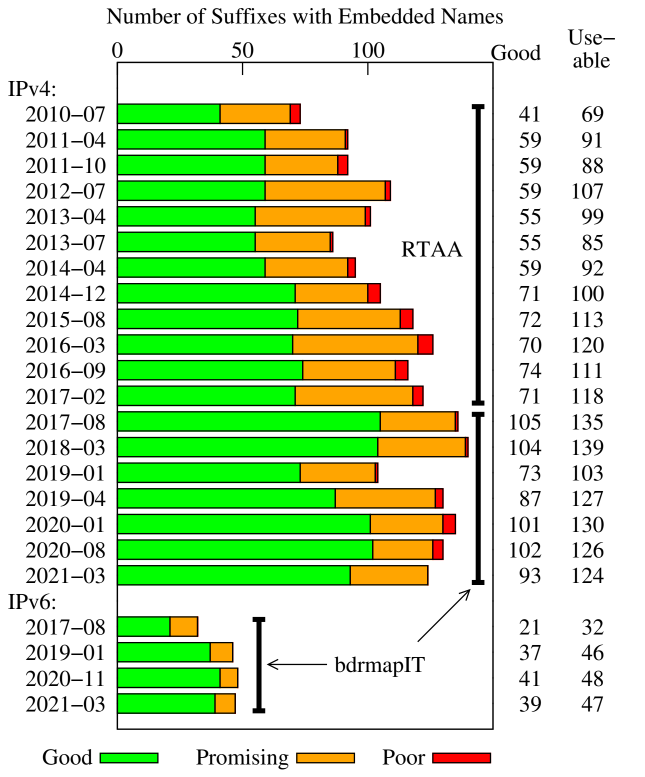


Figure 9: Classification of regexes. The number of network operators embedding network names grew over time as the Internet grew. Data collection issues for the January 2019 ITDK caused the dip in the number of regexes for that ITDK.

regexes per ITDK. In contrast, our prior work [21] extracted ASNs directly embedded by operators in hostnames – such as "2906" from "as2906.saopaulo.sp.ix.br" (hostname H in figure 1) – and built 12–55 *good* regexes per ITDK. That is, twice as many operators are embedding network names than are embedding ASNs directly.

We classified a regex as *promising* (the orange portion of each bar) if it extracted at least two unique congruent ASNs with a PPV ≥ 50%, and these are the bulk of the remaining suffixes for which our method inferred regexes. The good and promising regexes are *usable* because they usually extract a correct network name. We classified the remaining regexes with a PPV < 50% as *poor*. While our method learned ≈120 usable regexes per ITDK for recent ITDKs, our method learned usable naming conventions for 308 suffixes across all 19 ITDKs, reflecting the evolving structure of the Internet.

Our method extracted network names from 7,393 hostnames in the March IPv4 2021 ITDK (true positives), and found 205 additional hostnames with network names that were not extracted (false negatives). That is, our method built regexes that extracted network names from 97.3% of the hostnames where names in our dictionary were present. When we applied our method to extract ASNs [21] to the same ITDK, it extracted ASNs from 5,758 hostnames (true positives).

Finally, figure 9 also shows the results of our method on the four ITDKs that contain IPv6 routers. Even though the March 2021 IPv6 graph contains 17.5x fewer hostnames than the March 2021 IPv4

Hostname	ASN		Name Dictionary
akamai.plix.pl	20940	(A)	akamai:20940
cloudflare.plix.pl	13335	(B)	cloudflare:13335
m247.plix.pl	9009	(C)	m247:9009
nask.plix.pl	204679	(D)	nask:204679
nask2.plix.pl	204679	(E)	oath:10310
oath2.plix.pl	10310	(F)	p4:39603
p4.plix.pl	39603	(G)	

		TP	FN
RE1	^([a-z\d]+)\.plix\.pl$	A,B,C,D,G	E,F
RE2	^([a-z]+)\d+\.plix\.pl$	E,F	A,B,C,D,G

Figure 10: Not all operators use a convention that is suited to being parsed with a regex. Hostnames E and F indicate secondary connections to PLIX, but a regex cannot distinguish these from digits that appear at the end of "m247" and "p4" network names.

graph, our method infers only 2.6x fewer usable naming conventions. Our method extracted network names from 2,236 hostnames in the March IPv6 2021 ITDK (true positives). Our method found 47 additional hostnames with network names in our dictionary that were not extracted by regexes (false negatives). Similar to our March 2021 IPv4 results, our method built regexes that extracted network names from 97.9% of the hostnames where names in our dictionary were present.

5 DISCUSSION

Because identifying the network that operates a router is critical to understand connectivity within and between organizations (§1), and all methods for this task are based on heuristics (§2.1), our annotated hostnames serve as a large, heterogeneous source of validation data for future evidence-based router ownership inference techniques. For example, combining the 7,393 network names our method extracts from hostnames, with the 5,758 ASNs extracted from hostnames using the method we described in our prior work [21], results in a validation dataset of 12,987 hostnames (164 hostnames encoded both a network name and an ASN).

The naming conventions our method infers can assist in inferring geolocation information encoded in hostnames [20]. For example, a human might recognize that the router with hostname *francetelecom.lon01.atlas.cogentco.com* is operated by France Telecom, and that its interconnecting router is in London, UK. However, a geolocation algorithm might infer that the substring "fra" or "fran" could refer to "Frankfurt am main, HE, DE", or that the operator encoded "France" as a country-level hint. Because our method can establish the first portion of the hostname corresponds to a network name, an algorithm could use this information to guide its inference of geolocation conventions away from this portion of the hostname.

6 LIMITATIONS

Some operators build hostnames that are not designed to be interpreted entirely by machines. Figure 10 illustrates using an example from PLIX, where two networks have trailing numbers in their names ("m247" and "p4") and two networks ("nask" and "oath") are connected to PLIX twice. There is no regex solution that extracts all of these names correctly. RE1 extracts "nask2" and "oath2" for hostnames E and F, which are not the names of those networks and are scored by our method as false negatives because the regex did not extract "nask" and "oath". RE2 does not match hostnames A, B, D, and incorrectly extracts "m" and "p" from hostnames C and G; our method also scores these as false negatives because the regex did not extract the actual names in the dictionary.

Some networks might have a global presence, but be organized into independently operated subsidiaries each with their own ASN. Examples of these networks are Vodafone, NTT, Telefonica, and AT&T. However, our method infers a single ASN for each network name. Figure 11 illustrates the problem: our method learned a mapping from Vodafone to AS 1273, as this instance is the largest Vodafone instance. However, the Vodafone network name is shared with other networks in the Netherlands, New Zealand, Romania, and Italy, each with their own ASN.

Hostname	ASN	Country
vodafone-level3-sanjose.level3.net	1273	UK
vodafone.sjc03.atlas.cogentco.com	1273	UK
vodafone.interxiondus1.nl-ix.net	3209	NL
vodafone1.ape.nzix.net	9500	NZ
vodafone.ronix.ro	12302	RO
vodafone.mix-it.net	30722	IT

Figure 11: Independently operated networks with their own ASN may use the same network name in different countries.

7 CONCLUSION

The central goal of Internet cartography is to build a realistic and richly annotated representation of network structure and properties, using independent measurements and interpretation of captured data and meta-data. Our primary contribution in this work is the development and application of an efficient information extraction technique to the semantic structure embedded in Internet hostnames. This process required extraction of a lexicon through parsing tokens from hostnames, and inference of a dictionary that mapped these lexical tokens to real-world referents (operational networks). We successfully applied this method to the largest publicly available Internet topology map, and in the interest of extensibility and reproducibility, we publicly release our software and datasets.

Prior work has illustrated how to use ASNs embedded in hostnames to improve router ownership inference [21]. The method we described in this work amplifies the ability of hostnames to serve as a large, heterogeneous source of validation data for improving router ownership inference techniques.

Beyond this immediate application, we believe that recent work to automatically learn regexes to extract information from router hostnames ([7, 14, 19–21], and this work) suggests the measurement community will soon be able to automatically explain most information encoded in router hostnames. Prior work, *undns* [30], contained manually-built regexes which were last updated in 2014. A fully-automated capability will enable more sophisticated analyses of an increasingly complex router-level Internet topology.

ACKNOWLEDGMENTS

We thank Young Hyun and Ryan Koga for assistance with the ITDK, and the anonymous reviewers for their feedback. This work was partly supported by U.S. NSF awards CNS-2105393, 1925729, 1901517, and OAC-1724853, and the U.S. DoD Defense Advanced Research Projects Agency under CA-HR00112020014. It does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

REFERENCES

[1] Rohit Babbar and Nidhi Singh. 2010. Clustering Based Approach to Learning Regular Expressions over Large Alphabet for Noisy Unstructured Text. In *AND*. 43–50.

[2] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2016. Inference of Regular Expressions for Text Extraction from Examples. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (May 2016), 1217–1230.

[3] Henry Birge-Lee, Liang Wang, Daniel McCarney, Roland Shoemaker, Jennifer Rexford, and Prateek Mittal. 2021. Experiences Deploying Multi-Vantage-Point Domain Validation at Let's Encrypt. In *USENIX Security*. 4311–4328.

[4] Falk Brauer, Robert Rieger, Adrian Mocan, and Wojciech M. Barczynski. 2011. Enabling Information Extraction by Inference of Regular Expressions from Sample Entities. In *CIKM*. 1285–1294.

[5] Xue Cai, John Heidemann, Balachander Krishnamurthy, and Walter Willinger. 2010. Towards an AS-to-Organization Map. In *IMC*. 199–205.

[6] CAIDA. 2021. Macroscopic Internet Topology Data Kit (ITDK). https://www.caida.org/data/internet-topology-data-kit/.

[7] Joseph Chabarek and Paul Barford. 2013. What's in a Name? Decoding Router Interface Names. In *HotPlanet*. 3–8.

[8] Amogh Dhamdhere, David D. Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky K.P. Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C. Snoeren, and k claffy. 2018. Inferring Persistent Interdomain Congestion. In *SIGCOMM*. 1–15.

[9] Rodérick Fanou, Bradley Huffaker, Ricky Mok, and k claffy. 2020. Unintended Consequences: Effects of Submarine Cable Deployment on Internet Routing. In *PAM*. 211–227.

[10] Rodérick Fanou, Francisco Valera, and Amogh Dhamdhere. 2017. Investigating the causes of congestion on the african IXP substrate. In *IMC*. 57–63.

[11] Henning Fernau. 2009. Algorithms for learning regular expressions from positive data. *Information and Computation* 207, 4 (April 2009), 521–541.

[12] Julián M. Del Fiore, Valerio Persico, Pascal Mérindol, Cristel Pelsser, and Antonio Pescapè. 2021. The Art of Detecting Forwarding Detours. *IEEE Transactions on Network and Service Management* 18, 3 (Sept. 2021), 3619–3632.

[13] Bradley Huffaker, Amogh Dhamdhere, Marina Fomenkov, and kc claffy. 2010. Toward Topology Dualism: Improving the Accuracy of AS Annotations for Routers. In *PAM*.

[14] Bradley Huffaker, Marina Fomenkov, and kc claffy. 2014. DRoP: DNS-based Router Positioning. *CCR* 44, 3 (July 2014), 6–13.

[15] Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and H. V. Jagadish. 2008. Regular Expression Learning for Information Extraction. In *EMNLP*. 21–30.

[16] Aemen Lodhi, Natalie Larson, Amogh Dhamdhere, Constantine Dovrolis, and kc claffy. 2014. Using PeeringDB to Understand the Internet Peering Ecosystem. *CCR* 44, 2 (April 2014), 21–27.

[17] Matthew Luckie. 2010. Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet. In *IMC*. 239–245.

[18] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and k claffy. 2013. AS Relationships, Customer Cones, and Validation. In *IMC*. 243–256.

[19] Matthew Luckie, Bradley Huffaker, and k claffy. 2019. Learning Regexes to Extract Router Names from Hostnames. In *IMC*. 337–350.

[20] Matthew Luckie, Bradley Huffaker, Alexander Marder, Zachary Bischof, Marianne Fletcher, and k claffy. 2021. Learning to Extract Geographic Information from Internet Router Hostnames. In *CoNEXT*.

[21] Matthew Luckie, Alexander Marder, Marianne Fletcher, Bradley Huffaker, and k claffy. 2020. Learning to Extract and Use ASNs in Hostnames. In *IMC*.

[22] Matthew Luckie, Alexander Marder, Bradley Huffaker, and k claffy. 2021. Data supplement for "Learning to Extract Network Names from Hostnames". https://www.caida.org/publications/papers/2021/hoiho-asnames/.

[23] Alexander Marder, k claffy, and Alex C. Snoeren. 2021. Inferring Cloud Interconnections: Validation, Geolocation, and Routing Behavior. In *PAM*. 230–246.

[24] Alexander Marder, Matthew Luckie, Amogh Dhamdhere, Bradley Huffaker, Jonathan M. Smith, and kc claffy. 2018. Pushing the Boundaries with bdrmapIT: Mapping Router Ownership at Internet Scale. In *IMC*. 56–69.

[25] Ricky K.P. Mok, Vaibhav Bajpai, Amogh Dhamdhere, and k claffy. 2018. Revealing the Load-Balancing Behavior of YouTube Traffic on Interdomain Links. In *PAM*. 228–240.

[26] Ricky K.P. Mok, Hongyu Zou, Rui Yang, Tom Koch, Ethan Katz-Bassett, and k claffy. 2021. Measuring the network performance of Google cloud platform. In *IMC*. 54–61.

[27] Karin Murthy, Deepak P., and Prasad M. Deshpande. 2012. Improving Recall of Regular Expressions for Information Extraction. In *WISE*. 455–467.

[28] Yibo Pi, Sugih Jamin, Peter Danzig, and Feng Qian. 2020. Latency Imbalance Among Internet Load-Balanced Paths: A Cloud-Centric View. In *SIGMETRICS*.

[29] Stanley Simoes, Deepak P, Manu Sairamesh, Deepak Khemani, and Sameep Mehta. 2018. Content and Context: Two-pronged Bootstrapped Learning for Regex-formatted Entity Extraction. In *AAAI*. 5924–5931.

[30] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. In *SIGCOMM*. 133–145.

[31] Srikanth Sundaresan, Danny Lee, Xiaohong Deng, Yun Feng, and Amogh Dhamdhere. 2017. Challenges in Inferring Internet Congestion Using Throughput Measurements. In *IMC*. 43–56.

[32] Bahador Yeganeh, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. 2019. How Cloud Traffic Goes Hiding: A Study of Amazon's Peering Fabric. In *IMC*. 202–216.

[33] Bahador Yeganeh, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. 2020. A First Comparative Characterization of Multi-cloud Connectivity in Today's Internet. In *PAM*. 193–210.

[34] Shanshan Zhang, Lihong He, Eduard C. Dragut, and Slobodan Vucetic. 2019. How to Invest my Time: Lessons from Human-in-the-Loop Entity Extraction. In *KDD*. 2305–2313.