Large-Scale Classification of IPv6-IPv4 Siblings with Variable Clock Skew

Quirin Scheitle, Oliver Gasser, Minoo Rouhi, Georg Carle
Chair of Network Architectures and Services
Technical University of Munich (TUM)
Email: {scheitle,gasser,rouhi,carle}@net.in.tum.de

***Abstract*—Linking the growing IPv6 deployment to existing IPv4 addresses is an interesting field of research, be it for network forensics, structural analysis, or reconnaissance. In this work, we focus on classifying pairs of server IPv6 and IPv4 addresses as *siblings*, *i.e.,* running on the same machine. Our methodology leverages active measurements of TCP timestamps and other network characteristics, which we measure against a diverse ground truth of 682 hosts. We define and extract a set of features, including estimation of variable (opposed to constant) remote clock skew. On these features, we train a manually crafted algorithm as well as a machine-learned decision tree. By conducting several measurement runs and training in cross-validation rounds, we aim to create models that generalize well and do not overfit our training data. We find both models to exceed 99% precision in train and test performance. We validate scalability by classifying 149k siblings in a large-scale measurement of 371k sibling candidates. We argue that this methodology, thoroughly cross-validated and likely to generalize well, can aid comparative studies of IPv6 and IPv4 behavior in the Internet. Striving for applicability and replicability, we release ready-to-use source code and raw data from our study.**

I. Introduction

The emergence of IPv6 in the Internet offers interesting possibilities for studies to compare IPv6 and IPv4 structures and attributes in the Internet. Interesting questions are, *e.g.,* whether and to what extent IPv6 addresses are co-deployed on existing IPv4 hardware, whether correlated IPv6 and IPv4 attacks originate from the same hosts, what levels of redundancy can be achieved by co-deploying IPv6, or to conduct IPv6-IPv4 geolocation comparisons. An important prerequisite for such studies is the identification and classification of related IPv6 and IPv4 addresses. One such association can be gained from DNS queries, which yields IPv6-IPv4 address pairs that deliver the same service, but may be hosted on different machines. We address the problem of determining whether a set of IPv6 and IPv4 addresses are located on the same machine in a dual-stack setup. As in prior work [10], we use the term *sibling* for such a relation. This level of relation may help to draw deeper conclusions from service-level IPv6-IPv4 comparative studies, *e.g.,* on latency [7] or security comparisons [12]. We base our classification approach on active measurements of TCP timestamps, based on prior work by Kohno [19], Zander [32], and Beverly and Berger [10]. Our approach leverages novel features, such as the identification of unique nonlinear patterns caused by variable skew. Based on these features, we train and test various classifier models, using thorough train/test splits and cross-validation to avoid overfitting. Our contributions are:

- We identify 682 ground truth hosts, of which a large fraction exhibits variable clock skew
- We define novel features for sibling classification, capable of, *e.g.,* identifying and comparing variable clock skew
- We utilize thorough train/test methodology and machine-learning to build and evaluate classifier models
- We achieve excellent train and test performance even for hosts with variable clock skew
- We establish scalability through large-scale measurements and find 149k server siblings
- We publicly share our ground truth, code, and data

We structure this work as follows. In Section II, we discuss background and related work. We present our methodology in Section III, and define features in Section IV. Section V presents and evaluates our models, followed by their large-scale application in Section VI. We discuss outliers and influencing factors in Section VII, concluded by Section VIII.

II. Related Work

We introduce background and related work in four categories: remote clock skew estimation, remote identification, IPv6-IPv4 comparative studies, and IPv6-IPv4 sibling detection.

Remote Clock Skew Estimation: Accurate time-keeping on computing machinery is a notoriously difficult problem: precisely oscillating hardware is prohibitively expensive for most machines. The dominant protocol to synchronize low-precision machines, NTP, exhibits many difficulties even after decades of development [21], [31]. Hence, clocks in most devices in the Internet do not run in sync with true time, but deviate from it to an extent that is measurable over the Internet. This deviation is called *skew*, and can either be consistent over time (*constant skew*), or vary over time (*variable skew*). Protocols or protocol extensions that include timestamps from a remote machine allow for measuring clock skew with good accuracy by comparing local and remote timestamps over time. This skew can be used to remotely identify network devices. Foundations in this field were laid by Paxson [25] in 1998 and Moon et al. [22] in 1999. Kohno et al. [19] in 2005 first apply these techniques to TCP timestamps. They conduct a variety of case studies on the influence of external factors on timestamp behavior, e.g., power-saving or virtualization settings. Murdoch [23], and Zander and Murdoch [32] actively

induce variable skew on remote devices to identify Tor hidden services. However, their method of decision taking is tailored for few hosts and human inspection.

Other Remote Identification Techniques: Determining whether a set of IP addresses belong to the same router is an important and well-understood problem in Internet research. Scientific tools such as Ally [29], Radar Gun [8] and MIDAR [18] use IP Identification (*IP ID*) header values to answer this question, exploiting the fact that the *IP ID* counter is commonly shared between interfaces. Unlike IPv4, IPv6 only offers *IP ID* values in an extension header for fragmented packets (cf. RFC 2460). In 2013, Luckie et al. [20] publish *speedtrap*, which uses forced packet fragmentation for alias resolution in IPv6. In 2015, Beverly et al. [11] use IPv6 identification values to measure router uptime.

IPv6-IPv4 Comparative Studies: In 2015, Bajbai and Schönwälder [7] compare connection setup latency of domains for IPv6 and IPv4 addresses. They cite content being served from different machines as one of the potential reasons for latency differences. In 2016, Czyz et al. [12] compare security characteristics at service level, *i.e., AAAA* and *A* records of a domain. They find IPv6 addresses to frequently have worse security characteristics.

IPv6-IPv4 Sibling Detection: The problem of classifying sibling relationships at a machine level has first been tackled by Berger et al. [9]. Using customized DNS replies, they associate DNS client resolvers through opportunistic passive probing and open DNS resolvers through active probing. This technique only works on DNS clients or open resolvers, and requires a DNS server backend infrastructure. In 2015, Beverly and Berger [10] refine prior work on remote clock skew estimation through TCP timestamps and apply it to actively probe IPv6-IPv4 servers for sibling classification. Their algorithm is as follows: First, they filter non-siblings based on different TCP option signatures. Second, they classify the kind of TCP timestamp behavior (e.g., random, monotonic, non-monotonic). Third, they compare the angle of two constant clock skews to determine a sibling/non-sibling relationship. They achieve very good metrics (99.6% precision) on their training data, but acknowledge their comparably small ground truth data set of 61 hosts might be prone to overfitting.

They highlight the existence of hosts with variable clock skew, for which we provide precise classification features and models in this work.

III. Methodology

We put great care into avoiding overfitting and providing a sibling classifier that generalizes well. First, we collect a sufficiently large and diverse ground truth, significantly exceeding that of prior work. We then conduct a series of traffic measurements against this ground truth and a large-scale data set. Next, we define features potentially suited to discern siblings and non-siblings. Subsequently, we develop sibling decision algorithms based on these features, leveraging both manual analysis and machine learning algorithms. We train and evaluate those algorithms based on a train/test split in 10-fold cross-validation.

Table I: Our ground truth data set covers diverse Autonomous Systems (ASes), Countries (CC), and clock skew characteristics. Subsets can have hardware and/or software diversity.

Data Set	Hosts	#AS	#CC	Skew	Div.
2016-03 ("*03*")	458	373	40	variable	sw+hw
2016-12 ("*12*")	682	536	80	variable	sw+hw
servers	31	9	5	variable	sw+hw
ring	430	383	56	variable	hw
RAv1	12	12	11	variable	-
RAv2	209	192	64	constant	-
Beverly [10]	61	34	19	constant	unkn.

Acquiring a Ground Truth Data Set: A critical success factor of this work is to obtain a ground truth data set with numerous and diverse hosts. As prior work does not publish their ground truth data set, we set out to construct our own by (i) collecting ground truth servers from personally known operators and (ii) leveraging public frameworks which enforce an IPv6 and IPv4 address to reside on the same machine. For the latter, we include *RIPE Atlas* anchors [26] and *NLNOG RING* probes [3]. Table I lists our ground truth data and compares it to related work.

Ripe Atlas anchors exist in two different hardware versions [6], which we split out as *RAv1* and *RAv2*. Within the groups of *RAv1* and *RAv2*, there is no hardware or software diversity. *NLNOG Ring* ("*ring"*) nodes are formed by installing a provided image onto a virtual or physical machine. The *ring* group hence offers hardware diversity, but no software diversity. Our *servers* group offers soft- and hardware diversity. The *RAv1*, *RAv2* and *ring* groups offer good geographical diversity, while the *servers* group centers on Germany.

Please note that this data set allows for testing both sibling and non-sibling relationships, as non-siblings can be created by mixing addresses from different servers.

Measurement Methodology: To obtain a sufficient amount of fingerprints, we repeatedly connect to every sibling candidate IP address in parallel for a duration of ten hours, with the goal of acquiring at least one TCP Timestamp per minute. We open a HTTP connection and issue a *GET research_scan* query. As this resource-heavy approach repeatedly requires a full TCP and HTTP connection for every IP address, we process batches of 10k IP address pairs for our large-scale measurements. Our ground-truth measurements fit into one batch. As our methodology aims to identify similarities in clock skew, measurements to all IP address of a sibling candidate need to be conducted in the same batch. We leverage the TCP keepalive option to avoid establishing a new connection every minute, but found many servers to quickly close our connections after few keepalive packets.

Our measurement stack consists of a Python3 master that dispatches work to several processes, which in turn start one

urllib3 thread per target IP address. Moving to a C library or high-speed packet-processing frameworks such as DPDK [1] or libmoon [15] might significantly reduce the kernel packet processing overhead and allow for larger batches. We acknowledge that our measurements might be considered intrusive, which we discuss later in this Section.

Measurement Runs: In March 2016, we conduct one run against our 2016-03 ground truth, referred to as *gt1*. In December 2016 and January 2017, we conduct six measurements against the 2016-12 ground truth, referred to as *gt2* through *gt6*. Notable are the runs *gt4* through *gt6*, which cover the time before, during and after the leap second on December 31, 2016. We also conduct a large-scale measurement campaign in August 2016, which we further discuss in Section VI. As the measured offset is relative to the offset of our own clock, we usually disable *ntpd* to avoid creating non-linear offsets from clock adjustments on our machine. We enable *ntpd* during the *gt2* measurement to test this hypothesis.

Training and Evaluation Methodology: Training an algorithm on a few hundred ground truth hosts that will later be applied on millions of hosts comes with two challenges: First, the ground truth obtained might not be a representative sample of the full population of hosts in the Internet. Second, classifier training may overfit the training data, achieving very good train/test metrics on the ground truth, but failing on large-scale application. We aim to mitigate the risk of training on a non-representative sample by establishing software, hardware, geographical, and administrative diversity in our ground truth. We find our ground truth to exhibit diverse TCP timestamping characteristics even when compared to our large-scale data set. To avoid overfitting our ground truth for both machine-learned and manually assembled decision algorithms, we deploy a strategy of train/test splits and cross-validation. We also aim to minimize model complexity to provide better generalization.

Ethical Considerations: We follow an internal multi-party approval process, among others based on Partridge and Allman [24], before any measurement activities are carried out. We conclude that our measurements and the resulting data can not harm individuals, but may result in investigative effort for system administrators. We aim to minimize this effort by deploying scanning best-practice efforts of (i) using dedicated scan machines with explanatory websites, (ii) maintaining a blacklist, (iii) reply to every abuse e-mail (seven received, one asking for blacklisting, six curious), and (iv) request URLs preceded by */research_scan* to allow quick identification of our connections. Furthermore, based on user discussions, we will respect `robots.txt` and set a descriptive HTTP user agent in future work. To conclude, we argue that no individual was harmed by our active measurements. We also conclude that the gathered data bears little privacy intrusion, and hence release all data that was based on publicly available sources.

IV. Features

In this section, we present the set of features investigated and later leveraged by our algorithms for sibling detection. We present the features in the order they are calculated, as some features depend on the existence of others.

It is important to note that there is a distinction in the nature of these features. Namely, a feature can be either *falsifying* or *verifying*. *Falsifying* features may only help to determine a non-sibling relationship, whereas *verifying* features can actually determine a sibling relationship with confidence.

For every group of features, we explain their calculation and list their specific outputs, where $output_6$ indicates an IPv6 feature, and $output_4$ an IPv4 feature.

Network Level Features: We test various network level features, such as network latency, initial Time-to-Live values, OS predictions, or open ports. We find those features to have very low discriminative power and exclude them from further analysis. Our technical report gives details on these measurements and the obtained results [27].

TCP Options Fingerprint: Similar to Beverly and Berger [10], we leverage TCP options as a first falsifying feature. We compare the *presence* and *order* of options and the *no operation (NOP)* padding bytes. Additionally, if the *window scale* option is present, we consider its value for the process of falsifying non-siblings, as it has demonstrated high discriminative capability in our test data set. We find values of some options such as *MSS* to frequently differ by non-static offsets even for ground truth siblings. Thus, we do not include those in this filtering step. As an example, we frequently find the option fingerprint `MSS-SACK-TS-NOP-WS07` in our ground truth data set, and the `MSS-NOP-WS08-SACK-TS` in our large-scale data set. We highlight that asking for more exotic options slightly increases diversity in answers, but we did not find the effect strong enough to justify the additional measurement overhead. **Features**: $Options_4$, $Options_6$, $opts\_diff = !(Options_4 == Options_6)$.

Remote TCP Timestamp Clock Frequency: In a next step, we calculate the remote clock frequency as employed by Kohno et al. [19]. We first calculate relative remote TCP timestamps (32-bit unsigned integers) as $v_i = T_i - T_1$, where T_i is the TCP timestamp contained in the i-th received packet. Then, the relative received timestamps are calculated as $x_i = t_i - t_1$, where t_i is the time the i-th packet was observed by the prober and x_i is in seconds. We check the resulting array $[x, v]$ for monotonicity and fix wrap-arounds. We then solve a linear regression against $[x, v]$, where the resulting slope is the remote clock frequency *Hz*. We find typical values of 10, 100, 250 and 1000 *Hz*, all within the range of 1 to 1000 Hz as in RFC 7323. We also save the R^2 value of the linear regression, R^2_{Hz}. Low or different R^2_{Hz} values may indicate erratic time-stamping behavior such as randomized timestamps. We expect a sibling's clock to tick with the same frequency for IPv6 and IPv4. Hence, for each sibling candidate, we calculate the difference between Hz_4 and Hz_6 as a falsifying metric. This produces one false negative occurrence in the ground truth data set, which we discuss in Section VII. **Features**: Hz_4, Hz_6, *hz_diff*, R^2_{Hz4}, R^2_{Hz6}.

Raw TCP Timestamp Value: As a next step, we compare the raw TCP timestamp values T^4_1 and T^6_1 of a sibling can-

didate pair. As the TCP timestamp clock is, except for wrap-arounds, typically monotonically increasing, the raw value of the 32-bit timestamp offers a certain level of entropy across hosts. We expect the values for a sibling to be very close for IPv6 and IPv4 as they are generated from the same clock. Using the *Hz* values calculated in the previous paragraph, we can calculate the absolute difference between two remote clocks using Equation 1:

$$\begin{aligned} \Delta_{tcp} &= T_1^4/Hz_4 - T_1^6/Hz_6 && [s] \\ \Delta_{rec} &= t_i^4 - t_i^6 && [s] \\ \Delta_{tcpraw} &= |\Delta_{tcp} - \Delta_{rec}| && [s] \end{aligned} \quad (1)$$

First, we convert the raw TCP timestamps to seconds by dividing through *Hz*, Second, we calculate the difference between the local received timestamps. The final metric Δ_{tcpraw} is obtained by computing the absolute difference between the two values mentioned above. This metric can be interpreted as the time difference between the last TCP timestamp counter reset for IPv6 and IPv4. **Feature**: Δ_{tcpraw}.

Clock Offset and Skew Calculation: In a next step, we can estimate the remote clock offset, *i.e.,* the deviation of a remote clock from its expected values. To do so, we calculate the expected relative remote time $w_i = v_i/Hz$ and the offset to the observed time $y_i = w_i - x_i$, both measured for the i-th observed packet. The resulting array $[x_i, y_i]$ is then used for more fine-grained investigations such as clock skew estimation, which will be explained in the following paragraphs. This array is also used for plots in this work.

As noted by Kohno et al [19], the derivative of this array would theoretically be the *skew* of the remote clock. However, due to delay variances and various other effects, it is not sound to form the derivative of this array, but we rather use a more stable method to obtain the skew.

In this work, we use Robust Linear Regression [30] to obtain a robust and outlier-resistant regression, whose slope α we use as an estimation of remote clock skew. The rationale behind using this method is that offset points are in nature heavily impacted by various network dynamics [25] and hence prone to outliers. Additionally, we store R^2_{skew} which is the linear regression's coefficient of determination and is used to estimate the quality of the line fitted by the regression. **Features**: α_4, α_6, α_{diff}, R^2_{skew4}, R^2_{skew6}, $R^2_{skewdiff}$.

Calculation of Dynamic Range: Another feature we consider is the *dynamic range* of the offset array: While some hosts exhibit an offset of several seconds over the course of 10 hours, other hosts exhibit an offset of few milliseconds (cf. Figure 1). As this dynamic might be valuable information, we aim to extract it as a feature. To calculate this dynamic range in a manner that is stable against latency-caused outliers, we first prune the top and bottom 2.5% of offset arrays, and then store the difference between maximum and minimum as rng_4 and rng_6. **Features**: rng_4, rng_6, *rng_diff*=$|rng_4\text{-}rng_6|$, *rng_avg*=$(rng_4 + rng_6)/2$, *rng_diff_rel*=*rng_diff*/*rng_avg*.

Variable Skew Calculation: While α and R^2_{skew} fuel sibling/non-sibling classification for hosts with constant skew,

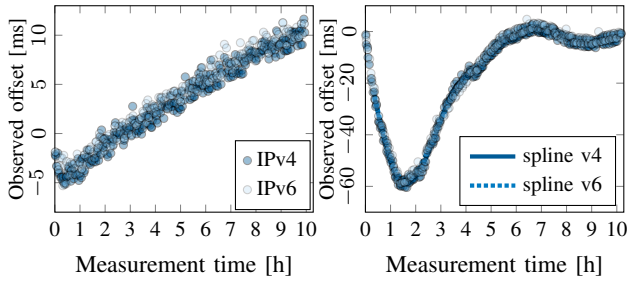

Figure 1: Siblings with constant skew and small dynamics (left), and variable skew and large dynamics (right).

additional steps are necessary to gain insight into the behavior of sibling candidates with variable skew.

To approach variable skew, we fit a polynomial spline against the $[x_i, y_i]$ arrays of a sibling candidate. Among various options to fit polynomial splines, we find it well-suited to pick 13 equidistant offset points and fit cubic splines between these candidate points in an approximative manner. Figure 2 shows the curve fitting approach for both siblings and non-siblings. In the next step, we minimize the area between the two splines by shifting the y-offset of one slope. This minimal area between the v4-spline and the v6-spline, *spl_diff*, is an output of our variable skew calculation. As the area between the two splines is also proportional to the dynamics of offsets, we also provide scaled version *spl_diff_scaled* = *spl_diff*/*rng_diff*. **Features**: *spl_diff*, *spl_diff_scaled*.

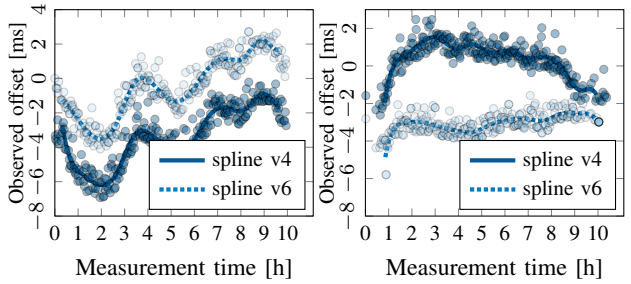

Figure 2: Splines for sibling (left) and non-sibling (right).

V. Classifier Model Training and Evaluation

Based on the discussed features, it is possible to train classifier *models*, that can be used to predict whether a pair of IP addresses is a sibling or not. In this section, we explain our approach to train and evaluate various such models, preceded by an explanation of our evaluation methodology.

A. Model Evaluation

To evaluate classifier models, a wide range of metrics exists, of which we focus on two: First, we prioritize high *precision*, defined as $tp/(tp + fp)$, *i.e.,* a low number of false positives. With higher precision, fewer non-siblings will be among predicted siblings. Second, we also want metrics for other aspects of performance to be stable to avoid overprioritizing precision.

For this, we use the Matthews Correlation Coefficient *(MCC)* [2] as defined in Equation 2. This coefficient nicely and robustly factors in all kinds of aspects of classifier performance for binary problems, where 1 would be a perfect score and 0 the worst, *i.e.,* a coin flip.

$$\text{MCC} = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}} \quad (2)$$

For training and evaluation purposes, we generate around 400k[1] non-siblings from our data set by mixing the IPv6 address of one sibling with the IPv4 addresses of other siblings. We generate the maximum possible number of non-siblings for each evaluation, and then equally weight both output classes for the classifier training. We generate non-siblings only for siblings actually used, *e.g.,* we first split siblings into train and test, and then form the non-siblings based on these splits.

Table II shows the evaluation results for our different models. Table III investigates whether a model's results are dependent on the evaluated subgroup.

Having established our evaluation approach, we now discuss filtering steps applied during feature calculation, followed by the individual models.

B. First-Order Filtering for All Models

Certain sanity checks on first-order filters apply to all models, *e.g.,* if the calculation of *Hz* fails, many dependent features can not be calculated.

Different TCP Options: If sibling candidates offer different TCP options, our algorithm stops with a non-sibling decision before continuing with feature calculation.

Hz Calculation: When calculating the remote clock frequency, the linear regression applied to do so may fail for reasons such as randomized timestamps. To only incorporate sound remote Hz frequencies, we require the R^2 parameter to be above 0.9. We classify sibling candidates with failed Hz calculations as non-siblings.

Different Hz Values: When a sibling candidate offers different *Hz* values for IPv6 and IPv4, calculation of dependent features becomes meaningless. We hence decide candidates with different Hz values as non-siblings.

Too Small Hz Values: When a Hz value below 1 Hz is calculated, we also stop further calculation and decide for a non-sibling relationship.

In all these cases, we decide for a non-sibling relationship and stop calculating other features. This works with a very low false negative rate (4 in 682) for our ground truth data set. We discuss those outliers in Section VII-A.

C. Beverly/Berger Algorithm

For comparison, we implement the sibling detection algorithm as proposed by Beverly and Berger [10]. The algorithm is designed for constant skew and works primarily by comparing the constant skew of sibling candidates.

[1] $(n \cdot (n-1)), n = 682$, we use both the v6-v4 and v4-v6 combination

Table II: Hand-Tuned and Machine-Learned Classifiers train and test very well, speaking to good generalization. Beverly algorithm is not generalizing well to the more diverse data set.

Algo.	Train DS	Test DS	Prec.	MCC	Type
Bev.	Bev.	Bev.	99.6%	n/a	Train
Bev.	Bev.	03∪12	.9%	.17	Test
HT1	03	03	100%	.99	Train
HT1	03	12\03	99.49%	.98	Test
ML1	03∪12	03∪12	99.36%	1.0	Train
ML1	03∪12	03∪12	99.88%	1.0	Test

Legend: HT1 denotes our hand-tuned algorithm. 03 denotes the 2016-03 data set, 12 the 2016-12 data set. ML1 values are the arithmetic mean of 10-fold cross-validation. All tests against 2016-12 are arithmetic means over the results from measurement runs 2 through 7. Calculation of MCC for Bev. data set not possible from metrics given by Beverly and Berger [10].

Table III: Performance of Beverly algorithm slightly dependent on group, our hand-tuned and machine-learned (not shown) algorithm independent from group.

Algo.	Train DS	Test DS	Prec.	MCC	Type
Bev.	Bev.	03∪12-server	8.33%	.17	Test
Bev.	Bev.	03∪12-ring	1.09%	.09	Test
Bev.	Bev.	03∪12-rav1	8.35%	.00	Test
Bev.	Bev.	03∪12-rav2	.79%	.05	Test
HT1	03	12\03-server	100%	.99	Test
HT1	03	12\03-rav2	99.16%	.98	Test
HT1	03	12\03-ring	100%	.98	Test
HT1	03	12\03-rav1	100%	1.0	Test

The algorithm does not find siblings with high precision ($< 1\%$, see Table II) in our combined data set, which includes many hosts with variable clock drift. We find the algorithm's performance to vary slightly for different groups of our test set (see Table III). As this variation is not systematic (*i.e.,* due to overwhelming group characteristics), we argue that this is based on circumstantial existence of hosts that are well-fit to Beverly and Berger's algorithm.

D. Hand-Tuned Decision Algorithm

One of our classifiers is a hand-tuned decision algorithm similar to Beverly and Berger's [10]. We hand-tune our algorithm against our *2016-03* ground truth and test it against the newly added hosts of the *2016-12* ground truth. This results in a 40% train and 60% test set, with all subgroups achieving $>$40% test size.

The formalized algorithm is displayed in Algorithm 1. The following provides a terse description of its high-level decision taking steps. The algorithm offers many subtleties, and we recommend our source code and tech report [27] as a detailed reference. Similar to Beverly and Berger, we first eliminate candidates with different TCP options. Then, our algorithm performs first-order filtering a described in Section V-B. Third, we eliminate candidates with raw TCP timestamps too far

Algorithm 1 Hand-Tuned Sibling Classification Algorithm (trained and tested on disjunct data sets).

1: **if** *opts_diff* $\implies$ non-sibling.
2: **if** *Hz* calculation failed or invalid $\implies$ non-sibling.
3: **if** $\Delta_{tcpraw} > z_1 \implies$ non-sibling
4: ### Linear Testing (using Robust Linear Regression [30])
5: **if** $R^2_{skew4} \geq z_2 \wedge R^2_{skew6} \geq z_2$:
6: **if** $sign(\alpha_4) \neq sign(\alpha_6) \implies$ non-sibling.
7: **if** $|\alpha_{diff}| \leq z_4 \implies$ sibling.
8: **else if** $R^2_{skew4} \geq z_2 \oplus R^2_{skew6} \geq z_2$:
9: **if** $|R^2_{skewdiff}| \geq z_3 \implies$ non-sibling.
10: ### Non-Linear Testing:
11: **if** $rng_4 \leq z_5 \wedge rng_6 \leq z_5 \implies$ unknown.
12: **if** $rng_4 \geq z_5 \oplus rng_6 \geq z_5$:
13: **if** *rng_diff* $\geq z_6 \implies$ non-sibling.
14: **if** $rng_4 \geq z_7 \wedge rng_6 \geq z_7$:
15: **if** *spl_diff* $\leq y_1 \implies$ sibling.
16: **else** $\implies$ non-sibling.
17: **if** *spl_diff* $\leq y_2 \implies$ sibling.
18: **if** *spl_diff* $> y_3 \implies$ non-sibling.
19: **else** $\implies$ unknown.

Values used: $z_1 = 1$, $z_2 = .81$, $z_3 = .2$, $z_4 = .00005$, $z_5 = 1.5$, $z_6 = .47$, $z_7 = 14$, $y_1 = 2.3, y_2 = .6, y_3 = 4.0$

apart. In line 5, we test whether to apply linear logic by evaluating the R^2_{skew} values of robust linear regression. Skews with differing slope signs are classified as non-siblings (line 6), whereas small slope differences are classified as siblings (line 7). In line 8 and 9 we classify those cases as non-siblings if only one skew is clearly constant and there is a large difference in R^2_{skew} values.

If linear testing was not conducted or not decisive, we apply nonlinear testing. For this, we first test the dynamics of both signals to exclude cases with negligible (line 11) or very different dynamics (lines 12 and 13).

We take further decision based on the minimal area between non-linear splines in lines 14 to 18. Based on whether the overall dynamics are large (line 14-16) or small (line 17-19), we apply different thresholds. We found this simplistic distinction between large and normal dynamics to provide good results on our data set, but acknowledge that this step could potentially be improved by means of finer tuning, for example by scaling the threshold by the dynamics. Our algorithm, similar to Beverly's and Berger's, features some guard intervals. In those, we can not take a meaningful sibling/non-sibling decision and decide for unknown.

As visible in Tables II and III, our algorithm achieves very good ($>99\%$ precision, $\geq.98$ MCC) metrics in training and testing and is insensitive to subgroups. We argue that this algorithm likely generalizes well to new data sets.

E. Decision Tree

Using scikit-learn [4], we train a CART Decision Tree on our features described in Section IV. For each of the seven measurement runs *gt1* through *gt7*, we do a 10-fold cross-validation with proportional selection from all subgroups (*servers*, *ring*, *rav1*, *rav2*). We find all models to consistently perform well with low variance, and report the arithmetic mean across validation folds and measurement runs for Table II. We also find all models' performance to be independent of groups.

For model selection, we export all generated decision trees and find very little variance: All trees contain just one branch, where they use a single threshold against the Δ_{tcpraw} feature to decide for sibling or non-sibling, *i.e.,* as a *verifying* metric. Our hand-tuned algorithm used a threshold of $>1s$ as a *falsifying* metric. We find the majority of models to pick a threshold of $>0.2557s$ for non-siblings and pick that value for our final model. Please note that the *ML1* model is preceded by the first-order filters described in Section V-B.

We argue that this model, due to its simplicity, will likely generalize best and recommend it for further use.

Table IV: Statistics of Large-Scale Domain Scans.

Source	#Domains	#IPv4 RRs	#IPv6 RRs	#Candidates
Alexa	1M	1.2M	108k	191k
biz	2.2M	2.0M	82k	104k
com	127M	134M	4.4M	6.7M
info	5.5M	5.0M	270k	299k
mobi	682k	560k	12k	15k
net	15.7M	14.3M	630k	898k
org	10.8M	10.6M	464k	700k
xxx	101k	178k	611	892
Total	162.4M	167.8M	5.9M	8.9M

VI. Large-Scale Application & Results

We apply our measurement methodology and classifier models to a large-scale data set to evaluate their scalability and suitability for finding sibling pairs for large-scale structural Internet studies. We first identify sibling candidates by resolving *A* and *AAAA* records for 162M domains, obtained from both registrars and "drop list" resellers. We filter blacklisted IP addresses, and form sibling candidates from all possible *A* and *AAAA* combinations per domain. Table IV details the statistics of this process by top-level domain. We find the number of sibling candidates to be bound by the relatively few *AAAA* records: We obtain only 6M *AAAA* records, compared to 168M *A* records. The number of sibling candidates, as the cross-product of *A* and *AAAA* records, will multiply with increased IPv6 deployment. Processing of the resulting 8.9M candidates is quantified in Table V.

After processing our sibling candidates to unique IPv6 and IPv4 addresses, we scan those addresses with zmap [14] on port TCP/80. We leverage our previously developed IPv6-capable version of zmap for this [16]. We find the majority of IP addresses to be responsive on port TCP/80. Discovery on more ports, such as TCP/443, might yield more responsive hosts. We next eliminate machines that do not offer the TCP Timestamp option, which is a prerequisite to applying our

Table V: Statistics of Large-Scale Domain Scans.

Processing Step	IPv4	IPv6
Sibling candidates	8,893,132	
Unique candidates	241,085	372,607
tcp80 responsive	226,419	315,782
Timestamp-capable	128,420	216,350
Consistent TCP options	128,146	216,073
Remaining candidates	6,619,100	
Unique candidate pairs	371,071	
Unique addresses in pairs	95,469	212,700
Pairs with measurement results	351,994	

Table VI: Large-Scale Domain Scan ($n = 351,994$)

Decision	HT	Bev.	ML1
Siblings	57k	203k	149k
Non-Siblings	126k	4k	57k
Unknown/Error	169k	145k	143k

technique. We find 57% (IPv4) and 69% (IPv6) of responsive IP addresses to offer the TCP Timestamp option. The higher percentage for IPv6 could be caused by IPv6 being offered by newer machines with more modern TCP configurations.

As we use the TCP option fingerprint of a remote host to filter for non-siblings, we extend zmap with TCP options capabilities. We chose to form a complex TCP options payload as this offers more possibilities for different TCP stacks to offer different replies. We ask for the set of options of: `<SACK permitted, Timestamps, Window Scale, TCP Fast Open, Unknown, MPTCP>`. We include an unknown TCP option (by using a reserved option identifier) as this may also trigger a range of different responses, from simple mirroring to correctly dropping the unknown option. However, this step removes only few hundred non-siblings in this large-scale data set.

We reassemble sibling candidates with both usable IPv6 and IPv4 addresses, resulting in 6.6M sibling candidates. Those 6.6M sibling candidates represent 371k unique IPv6-IPv4 address combinations.

In the next step, we measure the 371k unique candidate pairs by dividing them into batches of 10k addresses. Through these measurements, we obtain a sufficient count of timestamps for 351k of 371k address pairs. We then apply all discussed algorithms on this data set and display the results in Table VI. Several conclusions stem from its analysis: First, our *HT* algorithm was possibly tuned too conservatively and only identifies about a third of the siblings identified by our *ML1* algorithm. Second, our reproduction of Beverly's and Berger's algorithm produces the most siblings, but its very low precision numbers as evaluated before likely cause these to be mainly false positives. Third, the amount of unknown/error decisions is surprisingly high. In the ground truth evaluation, we mapped those decisions to a non-sibling decision to allow for binary evaluation. As we can not evaluate the large-scale measurement against a ground truth, we display the unknown/error category. Future work might dig into these unknown/error cases, try to find a ground truth, and perform further optimization. We investigate the contrast of siblings for *HT* and *ML1*, and find 57k siblings to intersect. Only few hundred of the *HT* siblings do not intersect, caused by the more aggressive Δ_{tcpraw} threshold in *ML1*.

Coming back to our initial goal, finding a confident set of siblings to study Internet-wide structural behavior, we conclude that both the *ML1* and the *HT* can find a significant number of siblings in Internet-wide scans. For model selection, we repeat our argument that the simplicity of the *ML1* makes it likely to generalize best, and the *HT* model likely suffers from a high false negative rate on our large-scale data set.

VII. Outliers & Discussion

In this section, we analyze outliers in our ground truth and discuss influence of several factors onto our measurements.

A. Analysis of Ground Truth Outliers

We discuss the few outliers in our ground truth measurements and evaluation. First, the Ripe Atlas node #6220 across measurements (i) returns different Hz values for IPv6 and IPv4 (100 and 1000), and (ii) has a significant ($\sim 2^{28}$) raw TCP timestamps difference, equaling $\sim$3 days difference at 1000 Hz). We have contacted the operators of this node to possibly obtain an explanation for this behavior.

Second, we find the hosts *ovh0X.ring.nlnog.net* to return varying TCP options for IPv4 addresses through a measurement, typically varying between only `MSS` and `MSS-SACK-TS-NOP-WS07`. Using tracebox [13], we typically receive responses that strip all but the MSS option at the last or penultimate hop. We conduct traceroutes path measurements from these machines and find the default IPv4 route traversing several RFC 1918 IP addresses, possibly indicating a NAT or tunnel techniques interfering with our measurements. It is unclear why the hosts sometimes proceeds with a full set of TCP options.

We consider both hosts legitimate cases for our classifiers to take a non-sibling decision. While Ripe Atlas and NLNOG Ring ensure that the associated IP addresses reside on the host, deployed middleboxes or proxies seem to distort this sibling relationship. Hence, we consider it positive that our models did not classify these cases as siblings.

B. Influence of Measurement Machine's Clock Skew

Irregularities in the measurement machine's clock may influence our measurements. We aim to minimize those irregularities by maintaining thermal conditions for the period of a measurement and by disabling the *ntpd* daemon during our measurements. We conduct one measurement run (*gt2*) with *ntpd* enabled and find *ntpd* interventions to be visible in manual analysis (through non-linear dynamics replicated across all hosts). However, all classifier models returned equally good results for this measurement run.

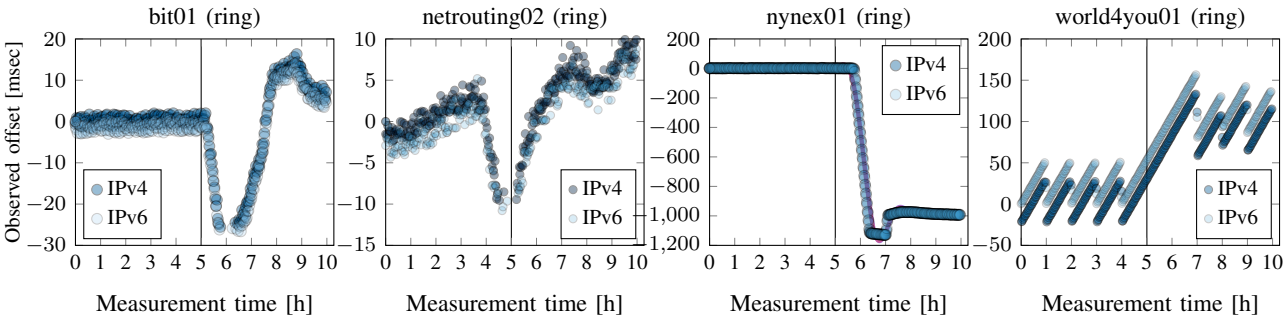

Figure 3: Leap second observations of four NLNOG ring siblings.

C. Influence of Leap Seconds

We conduct measurement runs before, during, and after the leap second on Dec 31, 2016. We find the metrics of our classifiers to be invariant to this circumstance, but interesting effects appear from visual inspection. Figure 3 shows clock offsets for the measurement during the leap second, which happens 5 hours into the measurement and is marked by a vertical line. We show four hosts with interesting behavior, most (>99%) servers show no effect from the leap second at all. This is expected behavior, as the TCP timestamping clock is supposed to monotonically tick without regard to leap seconds. Host *bit01* shows a typical reaction unaware of leap seconds, with *ntpd* adjusting clock speed after the leap second. Host *netrouting02* seems to smooth out the leap second by starting to slow its clock about an hour before the leap second, a technique similar to the *leap smear* deployed by Google [17]. Host *nynex01* reacts to the leap second with some delay, probably caused by periodic *ntpd* polling. Remarkably, it rapidly adjusts its clock by a full second with some minor corrections following. Host *world4you* periodically adjusts its clock by a hard change instead of changing the tick speed. For some time after the leap second, no clock change is conducted, likely until local time has surpassed its remote equivalent.

D. Influence of Ripe Atlas Hardware Version

Ripe Atlas anchors exist in two hardware versions which offer different characteristics [6]. Interestingly, remotely measuring the TCP timestamps of Ripe Atlas anchors reveals their hardware version, as all *v1* anchors exhibit variable skew, while all *v2* anchors offer constant skew (cf. Figure 4).

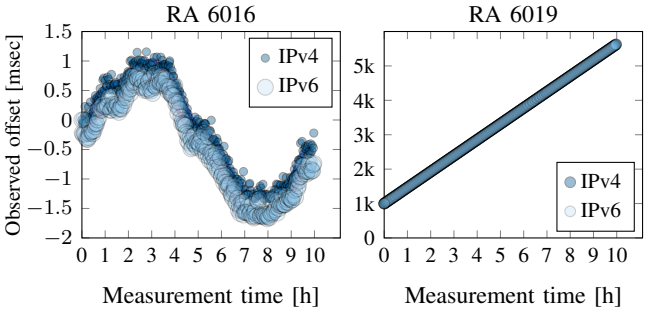

Figure 4: Ripe Atlas nodes with variable skew for *v1* (left) and constant skew for *v2* (right).

VIII. Conclusion and Future Work

We systematically approach the classification of IPv6-IPv4 servers siblings through active measurements, mainly reliant on TCP timestamping to estimate remote clock skew. We extract a variety of features from our active measurements and feed these into (i) a reproduction of existing work's algorithm, (ii) a hand-tuned algorithm developed by us, and (iii) machine-learning approaches. We find our algorithm, which significantly extends existing work by various features, to perform very well. Our machine-learning trained decision tree surprises with a very simple, but highly precise model. We apply our classifier models against a large-scale measurement and find different, but always significant, counts of siblings based on domain lists. We discuss outliers, likely caused by proxies or middleboxes, and the influence of leap seconds onto the TCP timestamping clock. We release our ground truth, code and data to the scientific community to allow for reproducibility and further research in this area.

Future Work: One direction of future work is the curation of larger sibling ground truth data sets. We hope to start this process with the release of our ground truth data set on GitHub. Another direction is the reduction of measurement effort in terms of duration, frequency, or both. Especially, the very discriminative raw TCP timestamp feature should work well with few data points, and hence only require few packets instead of hour-long measurements. Furthermore, the application of our technique on passive traffic captures to distinguish siblings sounds like a promising research goal.

Data Release: We publish our curated ground truth data set, acquired raw data, and developed source code for both reproducibility (cf. [5], [28]) and use by other researchers under:

`https://github.com/tumi8/siblings`

This website includes directions on how to obtain the large raw data set from an archival storage server.

Acknowledgments: We gratefully thank the various contributors of ground-truth server data. This work has been supported by the German Federal Ministry of Education and Research, project X-CHECK, grant 16KIS0530, and project AutoMon, grant 16KIS0411.

REFERENCES

[1] Data Plane Development Kit. http://dpdk.org/. Accessed 13 January, 2017.

[2] Matthews Correlation Coefficient. http://scikit-learn.org/stable/modules/model_evaluation.html#matthews-corrcoef. Accessed 13 January, 2017.

[3] NLNOG RING. https://ring.nlnog.net. Accessed 14 July, 2016.

[4] scikit-learn. http://scikit-learn.org/. Accessed 13 January 2017.

[5] ACM. Result and Artifact Review and Badging. https://www.acm.org/publications/policies/artifact-review-badging, Acc. Jan 18 2017.

[6] V. Bajpai, S. J. Eravuchira, and J. Schönwälder. Lessons Learned from Using the Ripe Atlas Platform for Measurement Research. *ACM SIGCOMM Computer Communication Review*, 2015.

[7] V. Bajpai and J. Schönwälder. IPv4 versus IPv6-Who connects faster? In *IFIP Networking, 2015*.

[8] A. Bender, R. Sherwood, and N. Spring. Fixing Ally's Growing Pains with Velocity Modeling. In *ACM SIGCOMM IMC*, 2008.

[9] A. Berger, N. Weaver, R. Beverly, and L. Campbell. Internet Nameserver IPv4 and IPv6 Address Relationships. In *ACM SIGCOMM IMC*, 2013.

[10] R. Beverly and A. Berger. Server Siblings: Identifying Shared IPv4/IPv6 Infrastructure Via Active Fingerprinting. In *Passive and Active Measurement*, 2015.

[11] R. Beverly, M. Luckie, L. Mosley, and K. Claffy. Measuring and Characterizing IPv6 Router Availability. In *Passive and Active Measurement*, 2015.

[12] J. Czyz, M. Luckie, M. Allman, and M. Bailey. Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy. In *NDSS*, 2016.

[13] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. Revealing Middlebox Interference with Tracebox. In *ACM SIGCOMM IMC*, 2013.

[14] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *USENIX Security*, 2013.

[15] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *ACM SIGCOMM IMC*, 2015.

[16] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle. Scanning the IPv6 Internet: Towards a Comprehensive Hitlist. In *Traffic Monitoring and Analysis*, 2016.

[17] Google. Leap Smear. https://developers.google.com/time/smear, Accessed Jan 27, 2017.

[18] K. Keys, Y. Hyun, M. Luckie, and K. Claffy. Internet-Scale IPv4 Alias Resolution with MIDAR. *IEEE/ACM Trans. Netw.*, 2013.

[19] T. Kohno, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2005.

[20] M. Luckie, R. Beverly, W. Brinkmeyer, et al. Speedtrap: Internet-Scale IPv6 Alias Resolution. In *ACM SIGCOMM IMC*, 2013.

[21] D. Malone. The Leap Second Behaviour of NTP Servers. In *Traffic Monitoring and Analysis*, 2016.

[22] S. B. Moon, P. Skelly, and D. Towsley. Estimation and Removal of Clock Skew from Network Delay Measurements. In *INFOCOM*, 1999.

[23] S. J. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. In *ACM Conference on Computer and Communications Security*, 2006.

[24] C. Partridge and M. Allman. Ethical Considerations in Network Measurement Papers. *Communications of the ACM*, 2016.

[25] V. Paxson. On Calibrating Measurements of Packet Transit Times. *ACM SIGMETRICS Perform. Eval. Rev.*, 1998.

[26] RIPE NCC. RIPE Atlas. https://atlas.ripe.net/, Accessed September 29, 2016.

[27] M. Rouhi. Path Tracing and Validation of IPv4 and IPv6 Siblings. Master's thesis, Technische Universität München, 2016.

[28] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, and G. Carle. Towards an Ecosystem for Reproducible Research in Computer Networking. In *ACM SIGCOMM 2017 Reproducibility Workshop*.

[29] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *ACM SIGCOMM Computer Communication Review*, 2002.

[30] H. Theil. A Rank-Invariant Method of Linear and Polynomial Regression Analysis. In *Henri Theil's Contributions to Economics and Econometrics*. Springer, 1992.

[31] D. Veitch and K. Vijayalayan. Network Timing and the 2015 Leap Second. In *Passive and Active Network Measurement*, 2016.

[32] S. Zander and S. J. Murdoch. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In *USENIX Security*, 2008.