Anycast Agility: Network Playbooks to Fight DDoS

A S M Rizvi, *USC/ISI;* Leandro Bertholdo, *University of Twente;*
João Ceron, *SIDN Labs;* John Heidemann, *USC/ISI*

https://www.usenix.org/conference/usenixsecurity22/presentation/rizvi

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

H Artifact Appendix

H.1 Abstract

In this artifact, we provide datasets and software tools related to our paper “Anycast Agility: Network Playbooks to Fight DDoS” [9]. Our artifact contains several datasets generated from our anycast experiments and analysis. Our datasets provide a snapshot of the results that we generated during our experiments. Some of our experimental results are dependent on the current state of the network interconnections and policies. However, due to the anycast stability, we expect to get similar results if we redo the experiments now. Our published datasets support our key results and are publicly available. We also provide tools and scripts that can be useful for other researchers.

H.2 Artifact check-list (meta-information)

- **Algorithm:** We provide an algorithm to select the best routing option from a BGP playbook containing multiple routing options and their impacts over traffic distribution (Section 3.4.2 of the paper [9]). We provide a working Python script for this selection algorithm. We include instructions about this tool in our anygility tool page [10].

- **Compilation:** We use shell/python script and java program for our tools. One needs to install Python and Java to run our tools. We depend on Verfploeter software, and we mention a series of other dependencies in the software READMEs [10].

- **Binary:** Some of our tools require extra binary files. We include those binary files with our software package, and provide instructions.

- **Data set:** We provide several datasets generated from our experiments and analysis [11]. Some software tools require extra datasets to run (e.g. IP hitlist). We include a sample dataset file with the software tool.

  However, we do not include large data files with our software tools. But these datasets can be downloaded separately (we provide the instruction in §H.3.1).

- **Run-time environment:** We tested our tools in Linux operating system. Peering toolbox on Fedora 34, and Tangled on Ubuntu 18.04 LTS and macOS 12. In some cases, our tools require root access. Our tools notify the users when it needs root access.

- **Run-time state:** Our key idea related to network playbook (Section 3.1 and 6.4 of the paper [9]) is dependent on the network interconnections and policies. We include the dates of experiments in our datasets. Since anycast is stable, we expect a similar outcome if we rerun the experiment.

- **Execution:** Some of our tools might need a long time to run. For example, our automated playbook builder announces different routing configurations, runs Verfploeter, and captures traces after a fixed interval. If we consider the whole process from measurement to playbook for 7 sites, it takes around 27-35 hours. For 3 sites it was around 17-24 hours. If we have more sites, or more routing policies, it would take even more time.

- **Security, privacy, and ethical concerns:** In the required cases, we anonymize IP addresses to prevent IP disclosure. As an example, we anonymize IP addresses in the DDoS attack datasets. For privacy reasons, we restrain ourselves from sharing certain attack data from Dutch national scrubbing center, and from an enterprise.

- **Metrics:** We provide datasets related to anycast catchments and DDoS attacks. Each dataset reports different metrics. We provide the details of these metrics in our README files. Our README files are included with the dataset packages.

- **Output:** We provide experimental outputs from Tangled and Peering testbeds. Tangled provides the measurement output in csv format while Peering provides raw captured traces in pcap format. These data files are parsed to generate output files in human-readable formats or graphs. The graphs are built using jupyter notebook and gnuplot scripts. We provide these scripts in our dataset webpage [11].

- **Experiments:** We provide scripts to automatically announce different routing configurations in both Peering and Tangled testbeds. We provide our generated datasets from these experiments. We provide some sample data to test our route selection process independent from running the whole measurement process.

- **How much disk space required (approximately)?:** Software tarballs are about 500KB. Our datasets related to the anycast experiments require around 100 GB disk space. Our attack datasets are large since we provide the whole day traffic captures (around 500 GB each). As our datasets are large, a user can download a portion of the datasets.

- **How much time is needed to complete experiments (approximately)?:** Some of the experiments may take a whole day (building a playbook with all routing options). Measurement process can take days depending the chosen measurement. Our decision maker can take decision within seconds. Parsing tools may need different times depending on the data size.

- **Publicly available (explicitly provide evolving version reference)?:** Our evolving datasets and software tools are publicly available at `https://ant.isi.edu/datasets/anycast/anycast_against_ddos/index.html`.

- **Code licenses (if publicly available)?:** Our tools are free; so anyone can redistribute it and/or modify it under the terms of the GNU General Public License, version 2, as published by the Free Software Foundation. We include this license notice with every tools that we make publicly available.

- **Data licenses (if publicly available)?:** We follow the data sharing policy through the participation of the LACREND project in the DHS IMPACT program [5].

- **Archived (explicitly provide DOI or stable reference)?:** Our stable reference for this artifact is here: `https://zenodo.org/record/6473023` with DOI 10.5281/zenodo.6473023.

H.3 Description

We provide datasets and tools for measuring anycast agility against DDoS. Our datasets are available upon request [5]. We provide datasets about the traffic distribution after BGP changes in testbeds, attack data from a DNS root server and from a national scrubbing

Software tools	**Software dependencies**	**Software source**	**Dataset dependencies**	**Dataset source**
Traffic Estimator	Java	openjdk-11.0.13	pcap traces	With dataset
	tshark	Wireshark	RIPE IPs	Included
playbook builder	Access to Peering	Required Testbed access	Hitlitst	With dataset
	Pinger	Provided + open source		
playbook tuner	Python	Python 3.10.2	Playbook	Included
			Load	Included
load_parser+ ParsingLoad	shell+Java	openjdk-11.0.13	Dataset dir. with pcaps	With dataset
	pingextract	Provided + open source	Load file	With dataset
BGPTuner	Python bgptuner-requirements.txt	Python 3.8	Playbook with specific site list	Included
measurement scripts + tangler-cli	Bash Python Verfploter ExaBGP Access to Tangled	Bash 4.4 Python 3.8 Verfploter 0.1.42 ExaBGP 4.1.2	—	—
vp-cli	Python	Python 3.8	Verfploter 0.1.42 files	Included
make-playbook	Python	Python 3.8	stats files	Included
run-playbook	Python ExaBGP Access to Tangled	Python 3.8 ExaBGP 4.1.2	Routing Playbook	Included

Table 1: Software tools dependencies.

center, other data related to anycast catchment stability, and other supporting data for our software tools. We provide codes for traffic estimation, for reproducing experiments, and for parsing the collected data.

H.3.1 How to access

Our datasets are available from the institutional storage system [6]. We provide the datasets based on requests [5]. After getting a request, we provide the download instructions. Our software tools will be available to download from its own webpage [10].

H.3.2 Hardware dependencies

Our whole uncompressed datasets size is over 1 TB. However, a user can download the partial datasets [6]. An interested user may want to look over the meta data of each dataset (using the README files), and keep the required amount of free storage.

H.3.3 Software dependencies

We provide several tools for different purposes [10]. We tested our software tools in Linux operating system. Some of our tools are dependent on external data sources and binaries. In most cases, we provide a sample data source with the package, and for other cases one can download the datasets with our released dataset. We provide the required binaries with our tools. One might need to install dependencies like Python or Java. We detail dependencies on (Table 1).

H.3.4 Data sets

We provide a full list of datasets in our web page [11].

We release datasets related to catchment distribution after routing configuration changes. We announce different BGP options, run Verfploeter to ping millions of responsive targets, and then capture the responses at every site. Our dataset includes raw pcap files captured from these measurements, and parsed data files in human-readable format.

We also provide DDoS attack data collected from B-root and Dutch national scrubbing center from 2015 to 2021.

Within other datasets, we provide datasets for anycast stability, and other supporting datasets to run our software tools.

The READMEs for these datasets are available with the dataset package.

H.3.5 Models

N/A

H.3.6 Security, privacy, and ethical concerns

We see no privacy concerns with our shared datasets. In cases like the DDoS attack data, we only share the /24 prefixes to hide the exact IP.

H.4 Installation

Instructions for running the tools are available in the webpages [10].

H.5 Evaluation and expected results

We provide the key results of the paper by mentioning the figures and tables, and list the corresponding datasets and tools in Table 2. Next, we list the key results, then we describe how can one get these results, and possible variations in the results. Please check the detailed steps to regenerate the graphs from the provided datasets.

H.5.1 Results with traffic estimation:

We propose a new technique to estimate the true offered load when we have loss in the upstreams (Section 3.3 and 4 [9]). We show our traffic estimation technique works well with the real world-attack events. For traffic estimation, we provide a tool named *TrafficEstimator* [14]. Using our traffic estimation tool, we show that we can correctly estimate the true offered load for real-world DDoS events.

To reproduce the same result, one needs to feed the attack traces to our program (provided as attack data in peering dataset [12]). One needs to have the pcap traces that we used, and needs to install tshark (with Wireshark) to feed the traffic content to our program. We used tracefiles for 2015-11-30 and 2016-06-25 events. A user needs to know the attack start time to use the right pcap files to observe the estimation outputs. The provided README tells the attack start time. We also need to provide a list of RIPE IPs that our program will use (already provided with the tool). We provide the instructions for running this tool in our webpage [14].

If running correctly, one can regenerate the same results that we showed in the paper. Depending on the start and end time of the attack trace, we might get a slightly different estimation. But on average we expect to get the same results.

Detailed steps:

We show the results generated for 2015 and 2016 events. This covers Figure 4, Figure 12, and Table 1. We use the following datasets:

1. Non-attack traffic 2015: B_Root_Anomaly-20151130/29/20151129-065024-00175689.pcap.xz,
2. Non-attack traffic 2016: B_Root_Anomaly-20160625/24/20160624-200008-00356777.pcap.xz,
3. Attack traffic 2015: B_Root_Anomaly-20151130/30/20151130-065209-00177422.pcap.xz,
4. Attack traffic 2016: B_Root_Anomaly-20160625/25/20160625-221823-00357641.pcap.xz,
5. RIPE IPs 2015: ripe-ips-2015-11-30.txt (provided with the tool),
6. RIPE IPs 2016: ripe-ips-2016-06-25.txt (provided with the tool).

The first step is to calculate the RIPE traffic rate during normal period (known-good traffic - normal column of Table 1). To get this value, we feed non-attack traffic to our estimator to get the RIPE traffic rate during normal period. We use the following command to get this:

For 2015 event: xzcat B_Root_Anomaly-20151130/29/20151129-065024-00175689.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2015-11-30.txt 192.228.79.131,2001:500:84::9077:f4f0

For 2016 event: xzcat B_Root_Anomaly-20160625/24/20160624-200008-00356777.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2016-06-25.txt 192.228.79.62,2001:500:84::ad9b:d590

Please wait for some time to see the generated output in the command prompt. The given addresses (192.228.79.* and 2001:500:84::*) are the B root server addresses (different because of the different anonymization keys). This command will generate an output like:

2015: “Time diff: 5.01 Counter-packets: 193 Rate: 38.47” 2016: “Time diff: 5.06 Counter-packets: 195 Rate: 38.52”

Key results [9, 11]	**Shared datasets**	**Related tools**
Figure 3	sample dataset provided with the tool	*tangled tools [15]* bgp-tuner
Figure 4, Table 1, Figure 12	*peering and root DNS dataset [12]* B_Root_Anomaly-20151130 B_Root_Anomaly-20160625	*TrafficEstimator and selection tools [14]* TrafficEstimator
Figure 5	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3 sites) 2020-02-24	*peering tools [15]* playbook_builder load_parser ParsingLoad
Figure 6	*tangled dataset [13]* Usenix_anygility_5_sites_2022-03-24_NEW	*tangled tools [15]* measurement scripts tangler-cli, vp-cli Anygility-Tangled-Catchment-load-distribution.ipynb
Figure 7	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224, community (3 sites) 2020-02-25	*peering tools [15]* playbook_builder load_parser ParsingLoad
Figure 8	*tangled dataset [13]* community dataset (3 sites)	*tangled tools [15]* measurement scripts tangler-cli, vp-cli
Table 5, Table 6	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3 sites) 2020-02-24, community strings (3 sites) 2020-02-25, poisoning (3 sites) 2021-04-09	*peering tools [15]* load_parser ParsingLoad
Figure 9	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3 sites) 2020-02-28	*peering tools [15]* playbook_builder load_parser ParsingLoad
Figure 10	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3,5,7 sites) 2020-02-24, 2020-04-07, 2020-04-08 Community (3, 5, 7 sites) 2020-02-25 and 2020-04-19	*peering tools [15]* playbook_builder load_parser ParsingLoad
Table 7	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: baseline (3 sites) 2020-02, 2020-04, and 2020-06	*peering tools [15]* load_parser ParsingLoad
Figure 11	*peering and root DNS dataset [12]* B_Root_Anomaly_message_question-20170306	*peering tools [15]* ParsingLoad TimeBasedPrefixLoad AnycastSiteLoad
Figure 13	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: poisoning (3 sites) 2021-04-09	*peering tools [15]* load_parser ParsingLoad
Figure 14	*tangled dataset [13]* poisoning dataset (3 sites)	*peering tools [16]* measurement scripts tangler-cli, vp-cli
Figure 15	*peering and root DNS dataset [12]* anycast_catchment_stability-20210701	-
Figure 16	*peering and root DNS dataset [12]* B_Root_Anomaly_message_question-20200214 B_Root_Anomaly_message_question-20210528	*peering tools [15]* ParsingLoad TimeBasedPrefixLoad AnycastSiteLoad

Table 2: Paper key results with datasets and tools. We provide the scripts to generate the graphs for our key results in our webpage [11].

We waited until 5 s to fix the final rate of the RIPE IPs. This rate is the cumulative rate measured from the start time. known-good traffic - normal column from Table 1 has a similar value.

The second step is to run the same TrafficEstimation java utility to find the estimated rate. We run the following commands for this:

2015 event:

```
xzcat B_Root_Anomaly-20151130/30/20151130-065209-00177422.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2015-11-30.txt 192.228.79.131,2001:500:84::9077:f4f0 38.47
```

2016 event:

```
xzcat B_Root_Anomaly-20160625/25/20160625-221823-00357641.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2016-06-25.txt 192.228.79.62,2001:500:84::ad9b:d590 38.52.
```

Please note that this command has an extra parameter (38.47 and 38.52) which we got from the previous command outputs. This command will generate two types of output lines. For 2015 event, we are showing a snapshot after 20 s, and for 2016 event we are showing a snapshot after 42 s.

2015 event output:

```
Time diff: 19.99 Counter-packets: 37 Rate: 1.85 1448866349.106 Count-packets: 1604914 Observed rate: 320982.8 Estimated: 6674713.88
```

2016 event outptut:

```
Time diff: 41.98 Counter-packets: 14 Rate: 0.33
1466893148.1316 Count-packets: 451957 Observed rate: 90370.72 Estimated: 11186300
```

Our program shows the RIPE rate when it finds new RIPE IPs in DNS traffic (starting after 1 minute). The observed rate line is printed at every 5 s. So, the users normally observe more number of first line.

The first line for 2015 event indicates that after 20 s during the attack period, our program receives 37 RIPE packets at a rate of 1.85 RIPE packets/s. This value corresponds to the known good traffic - observed column value from Table 1. Dividing by the prior normal rate of 38.47, we get the access fraction value, α. The first line for 2016 event indicates that the program gets 14 RIPE packets within 41.98 s with a rate of 0.33 RIPE packets/s. This value indicates the known good traffic - observed column value from Table 1. Dividing by the prior rate of 38.52 RIPE packets/s, we get the value of access fraction, α in Table 1. Please note that, because of a different RIPE IP list and measurement start time (using different pcap files), we are getting a slightly different value than what we have in the table.

Our program generates the second line at every 5 s. This line indicates the timestamp at every 5 s, packet count within that 5 s, the observed rate (packet count / 5.0), and the estimated traffic rate (observed rate / α). This observed rate corresponds to the offered load during attack - observed rate column of Table 1. For 2015 event, the sample output value is close to 0.32M packets/s, and for 2016 event this value is 0.09M packets/s. These two values are similar to what we have in Table 1 (0.37 and 0.10). A user will observe variable rates at different times. This observed rate is then divided by the calculated access fraction (α) to get the estimated offered load—offered load during attack - estimated column ($\sim$6.6M queries/second for 2015 event and $\sim$11M queries/second for 2016 event), which is close the reported rate of 5M queries/second and 10M queries/second, respectively [7, 8]. We use the estimated values from our TrafficEstimation program to generate the graphs—Figure 4 and Figure 12. Depending on the attack start time and RIPE IPs, the estimated values may vary slightly but we expect to get a similar trend. The offered load during attack - normal column indicates the normal traffic rate at a given time which we can measure from B root traffic (TrafficEstimator tool can measure this; we just need to feed the normal traffic with the known RIPE rate parameter) but we are skipping this detail since it is not directly related to the key outcomes. $\hat{\alpha}$ is calculated by dividing observed rate by the reported rate.

Our outcomes for known-traffic measurement, and estimated rate measurement may vary depending on the RIPE IPs we used and the traffic data we are using. We tried 5 s of traffic to find out the known traffic rate. This choice is arbitrary, a user can wait for some more time. Given the RIPE IPs that we provided, a user may expect to see 25-50 RIPE queries per second. Please note that, we used a subset of RIPE IPs. A larger RIPE IP set along with their consistent signal would ensure more stable RIPE query rate. We also provided some snapshots for the estimated rate measurement. Please note that, they are just snapshots. Estimated rates are dependent on the observed traffic rates (always varying), and the access fraction.

H.5.2 Building BGP playbook:

We propose a BGP playbook to fight against DDoS attacks. We build the BGP playbook with different routing options and their impacts over traffic distribution (Section 6 and 7 [9]). We show that BGP playbook can help the operators to select the right routing option during an attack event, and a playbook can provide a granular control over traffic distribution.

To reproduce the result, a user needs to announce different BGP configurations, and then run Verfploeter/pinger [3] to learn the prefixes to anycast site catchment. We provide scripts (*playbook_builder* in Peering and *tangler-cli* in Tangled) for our testbeds to makes these announcements automatically [15, 16]. One needs to have access to the testbeds to run this experiment. We used `Peering` [17] and `Tangled` [2] testbeds. These testbeds authorize an anycast prefix for a specific time period. One needs to ask for permission with a proposal to use these testbeds [1, 18]. Our script is dependent on verfploeter/pinger tool which is available online [3], and we provide a binary. This tool needs a target hitlist of IPs which we provided with our dataset (search for internet_address_history_it88w20191127 [6]). We provide a tool named getting_hitlist_ips to parse this raw hitlist file to get the list of responsive IPs. The instruction to run these tools is available in our webpage [15, 16].

To validate our results, we also provide the datasets that we got from our experiments. We include captured pcap files, and data in human-readable format for Peering [12], and in csv format for Tangled [13]. To reproduce results from the collected data, we also provide tools called *load_parser* and *ParsingLoad* in Peering [15], and *measurement scripts* in Tangled [16].

Our result is dependent on the stability of the network state. Since anycast catchment is fairly stable, we expect to get a slight variation but similar results if we rerun the experiment.

Detailed steps: We provide an example here to reproduce Figure 5 from our paper. Other similar graphs and tables like Figure 5—Figue 7, Figure 8, Table 5, Table 6, Figure 9, Figure 10, Table 7, Figure 13, Figure 14 can be generated using the similar process. Please note that figures for community strings and path poisoning (Figure 7, Figure 10, and Figure 13) for `Peering` utilizes only ParsingLoad utility alone (we provide the details later in this subsection).

At first, one needs to run *playbook_builder* tool to make BGP announcements for every prepending option. This step is dependent on getting access from the Peering testbed. Also, Internet routing changes, and we will not get the same outputs that we received while doing the experiment. As a result, we provide the collected data in pcap form to skip this step. Please find this dataset in peering and root DNS dataset—prepending (3 sites) 2020-02-24. The other datasets for other figures mentioned in prior paragraph are also provided.

To recreate Figure 5, we provide the following datasets:

1. The pcap files in peering and root DNS dataset: anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224,
2. The IP hitlist internet_address_hitlist_it88w-20191127/internet_address_hitlist_it88w-20191127.fsdb.bz2,
3. Some "load" data, provided with the software tool (we consider catchment in this figure so a full load data is not important).

After having these data, one needs to run anygility-peering/src/getting_histlist_ips/getting_hitlist_ips on the hitlist:

```
bzcat /data/internet_address_hitlist_it88w-20191127/internet_address_hitlist_it88w-20191127.fsdb.bz2 | python3 ./getting_hitlist_ips/data/ip_list_20191127.txt.
```

This will create a text file, ip_list_20191127.txt, containing one responsive IP address per line.

Then one needs to run anygility-peering/src/load_parser/load_parser.sh on the pcaps with the generated IP hitlist and sample load-file, and its corresponding load-date (e.g. –load=. –ldate=2022-02-01 to use the one provided with the tool):

```
bash load_parser.sh --numbers=3 --sites=AMS,BOS,CNF --date=2020-02-24 --dir=/data/anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224/ --load=.--ldate=2022-02-01 --hitlist=/data/ip_list_20191127.txt
```

Please note that the trailing / in the –dir argument is necessary.

This will run the ParsingLoad java utility for each announcement configuration, which will

- generate .dat files with ping responses from the .pcap files using pingextract utility.
- compute catchment data, both in terms of /24-blocks and "load" and store these as .txt files inside the data directory. For each announcement configuration, two files <DATE>-catchment-percentage.txt and <DATE>-load-percentage.txt are created. In addition, a combined all-<DATE>-load-<LOAD-DATE>.txt file is created in the data root directory.

The content of all-<DATE>-load-<LOAD-DATE>.txt consists of multiple blocks of this form:

```
<routing-configuration-path>
- <missing /24 count> <missing /24 relative>
site_1 <site_1 /24 count> <site_1 /24 relative> <site_1 /24 relative received>
[...]
site_n <site_n /24 count> <site_n /24 relative> <site_n /24 relative received>
multiple <multiple /24 count> <multiple /24 relative> <multiple /24 relative received>
- <missing load count> <missing load relative>
site_1 <site_1 load count> <site_1 load relative> <site_1 load relative received>
[...]
site_n <site_n load count> <site_n load relative> <site_n load relative received>
multiple <multiple load count> <multiple load relative> <multiple load relative received>
```

Figure 5 then shows bar-graphs created from the <site_x /24 relative received> values.

Using ParsingLoad alone: The script load_parser utilizes ParsingLoad for each of the path prepending configurations. When we are not parsing path prepending configurations, we can just utilize ParsingLoad utility alone. We utilize ParsingLoad alone for community strings and path poisoning (Figure 7 and Figure 13). We run ParsingLoad for each of these routing configuration separately.

```
java -jar ParsingLoad.jar 3 AMS,BOS,CNF anycast_catchment_distribution-20200224/Community_Strings_AMS,BOS,CNF-20200225/2020-02-25-AMS,BOS,CNF-AMS-ALL-PEERS/ /nfs/lander/traces/verfploeter/broot_verfploeter/Peering/Peering_Mapping/2020/community_strings/2020-02-25-AMS,BOS,CNF-AMS-ONLY-PEERS/ 2020-02-25 loads/ 2020-02-22
```

The output has the same format like all-<DATE>-load-<LOAD-DATE>.txt as we mentioned above. We combine these generated files to build Figure 7 and Figure 13. We use ParsingLoad separately for each routing configuration with community strings and path poisoning. But a script for all the community string and path poisoning options is also possible. For path poisoning, we used poisoning datasets (inside anycast_catchment_distribution-20200224) for AS174 (Tier-1), AS8283 (Transit-2), and AS12859 (Transit-1).

H.5.3 Selection from the playbook:

We provide a tool [14] to select the right routing configuration from the BGP playbook (Section 3.4.2 [9]). Using this tool, we show that an automated approach can be useful to select the right routing approach.

Our selection tool provides output based on the current playbook, and offered load. To show how the selection tool works, we provide a sample playbook (based on Table 5 [9]), and a load file. When the users run the tool with the given inputs, they can see the selection output. We also include a tool named *bgp-tuner* for showing the graphical interface [16].

Depending on the playbook and offered load, one can observe a different output, which can be a complete different policy selection.

Detailed steps: We provided a sample playbook and offered load file with the playbook_tuner tool. Please run the following command to see the outputs from this program:

```
cat load.txt | ./playbook_tuner –setup "playbook.txt"
```

This will result the following output:

```
Overloaded site: AMS
Suggested config: 1AMS, Estimated load distribution: 41292.64 29494.75 41292.64
Other configs: Poison-Tier-1, Estimated load distribution: 41292.64 29494.75 41292.64
Other configs: Poison-Tier-2, Estimated load distribution: 41292.64 29494.75 41292.64
```

This tells that prepending AMS by 1 would provide the best possible load distribution. Some other options are also possible.

H.5.4 Attack mitigation:

We show that BGP playbook is helpful to mitigate the real-world DDoS events.

To reproduce the same result, we provide the B-root attack traces in pcap and in message question formats [12]. Due to privacy reason, we cannot share the attack data from the Enterprise and Dutch National Scrubbing Center. We also provide the catchment distribution for different BGP changes [12,13]. Matching the attack prefixes and attack loads to the prefix-wise catchment gives us the traffic distribution at different sites. If one wants to test the B-root event, they need to run *TimeBasedPrefixLoad* tool to get the per prefix attack load [15]. Then one needs to run *AnycastSiteLoad* program to get the per anycast site load [15].

Since the attack and catchment mapping are fixed, we expect to get the same results that we showed in the paper.

Detailed steps: We show the detailed steps to generate Figure 11(a) here. All other subfigures of Figure 11 and Figure 16 can be generated using the similar process.

To generate Figure 11(a), we need the following datasets:

1. peering and root DNS dataset: B_Root_Anomaly_message_question-20170306/: Figure 11(a) shows 10000 s of traffic. To make the data processing faster, we recommend to use a subset of this whole timeframe. We recommend the user to download the datasets from 06:40:00 AM to 06:50:00 AM to reproduce a fraction of the whole timeframe combining both attack and non-attack period. The file names represent the dates and times (format: YYYYMMDD-HHMMSS-*).
2. peering and root DNS dataset: anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224/2020-02-24-AMS,BOS,CNF/
3. peering and root DNS dataset: /anycast_catchment_distribution-20200224/Community_Strings_AMS,BOS,CNF-20200225/2020-02-25-AMS,BOS,CNF-AMS-Transit-1-Trial-2/(update: this Trial-2 dataset is newly added. We also provided Trial-1 dataset for 2020-02-25-AMS,BOS,CNF-AMS-Transit-1 which will give a similar output, but we did not use that in the paper).

At first, run the TimeBasedPrefixLoad java utility on the downloaded message_question format data. We only need time, source IP and message length for our measurement. message_question formatted files have several attributes/columns. We used fsdb tool to retrieve the times, source IPs, and message length [4]. Please follow the instruction to install FSDB from here: https://www.isi.edu/~johnh/SOFTWARE/FSDB/perl-Fsdb-2.74_README.html. Next, use the following command to run TimeBasedPrefixLoad jar to generate the prefix-wise load for each 5 s:

```
xzcat B_Root_Anomaly_message_question-20170306/06/20170306-044* | dbcol time srcip msglen | java -jar TimeBasedPrefixLoad.java output-20170306/ 192.228.79.64,2001:500:84::bb26:87a2.
```

Here, dbcol is a utility from FSDB to select the right column from the message_question format dataset. output-20170306 will have multiple txt files named with a number indicating the time segment. This command will generate prefix-wise load at every 5 s in output-20170306 directory: <network_prefix> <number_load> <bytes>.

Then we run AnycastSiteLoad java utility to find out the per site load at every 5 s. We run this utility for two routing configurations—one without any routing change and one with announcing only to Transit-1.

```
java -jar AnycastSiteLoad.jar 3 AMS,BOS,CNF anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224/2020-02-24-AMS,BOS,CNF/ 2020-02-24 output-20170306/ 2017-03-06,
```

```
java -jar AnycastSiteLoad.jar 3 AMS,BOS,CNF anycast_catchment_distribution-20200224/Community_Strings_AMS,BOS,CNF-20200225/2020-02-25-AMS,BOS,CNF-AMS-Transit-1-Trial-2/ 2020-02-25 output-20170306/ 2017-03-06.
```

Please note that these two commands utilize output-20170306 that we generated in our previous step. These two commands generate two files in the corresponding catchment directory named as <CATCHMENT-DATE>-load-<ATTCK-DATE>-ingress.txt. The output format inside the file: <time> <site-1> <count-site-1> <bit-site-1> <...> <site-n> <count-site-n> <bit-site-n>. The first file contains load without any routing change, the second file contains load after announcing only to Transit-1. We combine these two files to show non-attack period (no policy deployed), and period when the route propagation is done (when we deployed Transit-1).

To match the results with the Figure 11(a), the first output file will contain (<count-site-n> column) traffic load during normal period (before 0 s from the graph with around 20k packets/s). The first output file also contains the attack traffic (AMS load over 60k packets/s after 160 s of the first file). This is similar to the traffic from 0 s to 300 s of Figure 11(a). After that we announce only to Transit-1 (after 300 s of Figure 11(a)). The second output file contains this data (after 160 s from the file).

H.6 Notes

Considering the real datasets are big, and time expensive to run, we include smaller datasets collected using a small hitlist fraction (0.1% of original size) in experiments with Tangled. While the produced playbook will differ from paper results, we believe it can help for testing purpose. For Peering tools, we sometimes include smaller sample supporting data files.

If desired, we can provide access to the Tangled testbed. Access to Peering testbed is dependent on the approval from Peering admins.

H.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.

References

[1] Tangled admins. Tangled anycast testbed. https://anycast-testbed.nl/, 2019. [Online; accessed 15-Feb-2022].

[2] Leandro M Bertholdo, Joao M Ceron, Wouter B de Vries, Ricardo de Oliveira Schmidt, Lisandro Zambenedetti Granville, Roland van Rijswijk-Deij, and Aiko Pras. Tangled: A cooperative anycast testbed. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 766–771. IEEE, 2021.

[3] Wouter De Vries. Verfploeter/pinger: Active measurement of anycast catchements. https://ant.isi.edu/software/verfploeter/pinger/index.html, 2019. [Online; accessed 15-Feb-2022].

[4] John Heidemann. John heidemann / software / fsdb. https://www.isi.edu/~johnh/SOFTWARE/FSDB//, 1991. [Online; accessed 19-Mar-2022].

[5] Analysis of Network Traffic (ANT) group. Ant dataset requests. https://ant.isi.edu/datasets/requests.html, 2022. [Online; accessed 15-Feb-2022].

[6] Analysis of Network Traffic (ANT) group. Ant datasets. https://ant.isi.edu/datasets/index.html, 2022. [Online; accessed 15-Feb-2022].

[7] Root Server Operators. Events of 2015-11-30. https://root-servers.org/media/news/events-of-20151130.txt, 2015. [Online; accessed 12-Oct-2021].

[8] Root Server Operators. Events of 2016-06-25. https://root-servers.org/media/news/events-of-20160625.txt, 2016. [Online; accessed 12-Oct-2021].

[9] A S M Rizvi, Leandro Bertholdo, João Ceron, and John Heidemann. Anycast agility: Network playbooks to fight DDoS. In *Proceedings of the 31st USENIX Security Symposium*, page to appear. USENIX, August 2022.

[10] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. anygility - anycast agility tools: playbook builder and decision maker. https://ant.isi.edu/software/anygility/index.html, 2022. [Online; accessed 2-Mar-2022].

[11] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Artifacts about anycast agility against ddos. https://ant.isi.edu/datasets/anycast/anycast_against_ddos/index.html, 2022. [Online; accessed 2-Mar-2022].

[12] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Datasets about anycast agility against ddos in peering testbed. https://ant.isi.edu/datasets/anycast/anycast_against_ddos/peering/index.html, 2022. [Online; accessed 2-Mar-2022].

[13] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Datasets about anycast agility against ddos in tangled testbed. https://ant.isi.edu/datasets/anycast/anycast_against_ddos/tangled/index.html, 2022. [Online; accessed 15-Feb-2022].

[14] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Tools about anycast agility against ddos. https://ant.isi.edu/software/anygility/system/index.html, 2022. [Online; accessed 2-Mar-2022].

[15] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Tools about anycast agility against ddos in peering testbed. https://ant.isi.edu/software/anygility/peering/index.html, 2022. [Online; accessed 2-Mar-2022].

[16] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Tools about anycast agility against ddos in tangled testbed. https://ant.isi.edu/software/anygility/tangled/index.html, 2022. [Online; accessed 2-Mar-2022].

[17] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. PEERING: Virtualizing BGP at the Edge for Research. In *Proc. ACM CoNEXT*, Orlando, FL, December 2019.

[18] Peering The BGP Testbed. Peering the bgp testbed. https://peering.ee.columbia.edu/, 2019. [Online; accessed 15-Feb-2022].