Periscope: Unifying Looking Glass Querying

Vasileios Giotsas, Amogh Dhamdhere,kc claffy

CAIDA, UC San Diego, USA
vgiotsas,amogh,kc@caida.org

Abstract. Looking glasses (LG) servers enhance our visibility into Internet connectivity and performance by offering a set of distributed vantage points that allow both data plane and control plane measurements. However, the lack of input and output standardization and limitations in querying frequency have hindered the development of automated measurement tools that would allow systematic use of LGs. In this paper we introduce *Periscope*, a publicly-accessible overlay that unifies LGs into a single platform and automates the discovery and use of LG capabilities. The system architecture combines crowd-sourced and cloud-hosted querying mechanisms to automate and scale the available querying resources. Periscope can handle large bursts of requests, with an intelligent controller coordinating multiple concurrent user queries without violating the various LG querying rate limitations. As of December 2015 Periscope has automatically extracted 1,691 LG nodes in 297 Autonomous Systems. We show that Periscope significantly extends our view of Internet topology obtained through RIPE Atlas and CAIDA's Ark, while the combination of traceroute and BGP measurements allows more sophisticated measurement studies.

1 Introduction

Measurement and monitoring tools are essential to many Internet research and engineering tasks, ranging from topology discovery to detection of security threats and network anomalies. However, the development of such tools is challenged by the decentralized nature of Internet infrastructure. For years, researchers have attributed measurement artifacts to the limited coverage of available measurement vantage points [14, 17], which has motivated revision of Internet measurement practices. Large-scale distributed measurement projects either crowd-source the hosting of traceroute vantage points [28, 7, 27, 1], or leverage cooperation from academic networks [25]. Network operators deploy their own monitoring infrastructure, including Looking Glass (LG) servers, which enable remote execution of non-privileged diagnostic tools, such as traceroute, ping or BGP commands, through a web interface. Although the primary purpose of LGs is operational, i.e., to debug connectivity and performance issues, LGs have also expanded researchers' cartographic and monitoring capabilities [30, 15, 20, 23, 19, 29].

LGs have two characteristics that benefit Internet research. First, LGs often permit the execution of both traceroute and BGP queries, offering data and control plane views from the same location. Second, in contrast to crowd-sourced

traceroute monitors that are deployed at end-hosts (e.g. home clients), LGs are typically deployed near or at core and border routers. Despite these advantages, the use of LGs has been sporadic due to design features that limit their use for scientific studies that require systematic and repeatable measurement. First, LGs do not form a unified measurement network of homogeneous probes, such as the RIPE Atlas or Ark infrastructures. Each LG is independently owned and operated; there is no centralized index of available LGs, nor standardized querying or output formats. Furthermore, LG command sets change over time, there is attrition of LG infrastructure, and because LGs are generally intended for low-frequency (manual) querying, operators often configure query rate limits to mitigate the risk of DoS attacks against them (or using them).

In this paper we introduce *Periscope*, a platform that unifies the disparate LG interfaces into a standardized publicly-accessible querying API that supports on-demand measurements. The core of the Periscope architecture is a central controller that coordinates queries from multiple users to prevent concurrent requests to the same LG from violating rate limits configured by that LG. The controller dispatches LG requests to crowd-sourced and cloud-hosted querying instances, which scale as necessary to handle large bursts of queries. A parser transforms the LG results into a set of standardized output formats (JSON and iPlane), and aggregates them in a repository for future analysis. A daemon checks periodically for changes in the HTML interfaces of the LGs, and automatically extracts and updates the LG configurations. The Periscope API and the repository of raw data are publicly accessible to authenticated users.[1]

This paper describes the Periscope architecture and how each Periscope component tackles the challenges related to LG measurements. We compare Periscope's querying capabilities and coverage with those of two major measurement platforms (RIPE Atlas and Ark). Finally, we demonstrate the utility of having colocated BGP and traceroute vantage points with two case studies involving the validation of IP-to-AS mapping, and the geolocation of border router interfaces.

2 Architecture

We have four design goals to mitigate four key challenges related to using deployed LGs for systematic measurement:

- There is no authoritative list of active LGs. Periscope must automatically discover, extract and validate LG specifications from various sources.
- LGs are volatile, both in terms of availability and specification. Periscope must detect changes and automatically update LG specifications.
- There is no input/output standardization across LGs, so Periscope must translate query requests to the format supported by each individual LG and the output of individual LGs to a user-friendly format.

[1] A user requests access through email describing the intended use and we issue a unique security token which he/she uses to sign measurement requests.

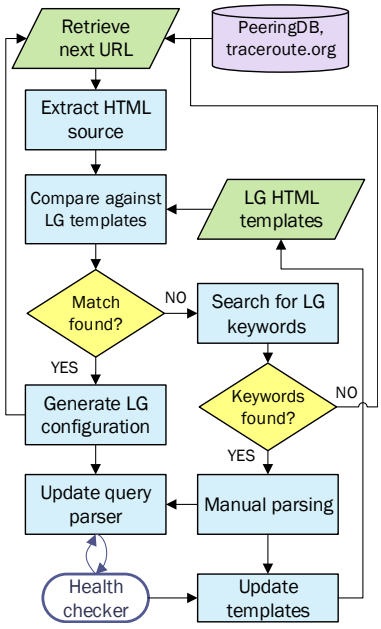

Fig. 1: End-to-end workflow to discover and extract looking glass specifications from web sources.

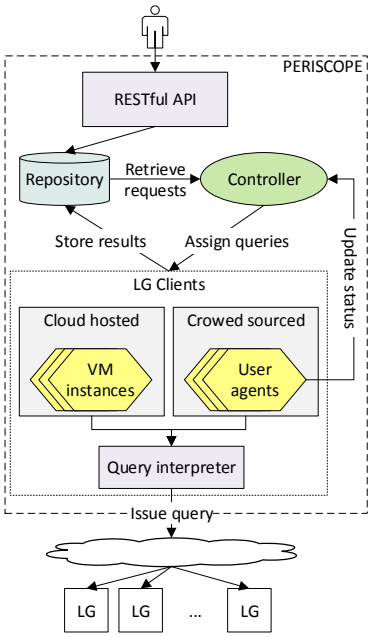

Fig. 2: The Periscope architecture, which sits on top of but does not itself include the LGs.

– LGs are intended for low-frequency querying and will block clients that exceed the configured querying rate limitations. Periscope should support multiple concurrent users without violating any LG limits.

2.1 Workflow of Periscope system

Figure 1 describes the Periscope workflow for integrating LGs into its querying system, which is repeated every month to update the list of supported LGs.The starting point of Periscope's workflow is the discovery of active LG servers, using public web sources that publish LG URLs, including PeeringDB [5] and `traceroute.org`. The system can easily integrate other listings of LG servers as they become available. Since these sources are non-authoritative, the published URLs may be stale or unresponsive. A **Web Crawler** visits each link and filters out pages that respond with HTTP errors.

To determine whether the collected URLs correspond to LGs, we attempt to automatically detect whether the HTML source contains web forms of LG interfaces. The automatic detection utilizes the fact that most LG deployments are based on open-source projects that determine the structure of the expected queries, the output format and the corresponding web interfaces [16]. We have processed seven popular open source projects [6, 13, 3, 12, 11, 10, 2] and created a template for each implementation, which describes the HTTP elements and

Table 1: Template for the input parameters of the Version6 LG [12].

Input name	Input type	Expected values	Meaning
query	radio	[bgp, trace, ping]	[sh ip bgp, traceroute, ping]
addr	text	*	Query target
router	select	*	Router identifier
protocol	select	[IPv4,IPv6]	IP version

the HTML parameters that comprise the input and output interfaces. Table 1 shows an example of such a template.

A *Web Scraper* extracts the `<form>` elements from the HTML code of active LGs and compares the input fields with the corresponding fields of each template to test for matches. A match occurs when each input field in the form is described in the template. It is not necessary for the extracted HTML form to have all input fields in the template, because some LGs may support only a subset of commands. For example, the template of Table 1 has three parameters that must be implemented (query, addr, router) and one optional parameter (protocol version) which when omitted defaults to IPv4.

When a form matches an LG template, Periscope generates a JSON configuration file that describes the interface of the LG, including the request HTTP method, the input parameters and their permissible values, the mapping of input combinations to network commands and the HTML elements that enclose the reply. The JSON configuration is used by the *Query Parser* to translate measurement requests to the format supported by each LG. When a form does not match with a template, the Web Scraper searches for LG-specific keywords (such as the name of network commands), to determine if the form contains LG inputs. If such keywords are found, we parse the form manually and update the LG templates as necessary to enable the automatic processing of similar forms in the future.

The final step of the workflow is to test the correctness of the auto-generated LG configurations. A *Health Checker* uses the Query Parser to issue measurement requests and process the replies. If the output is empty or if an HTTP error code is returned, the Health Checker will signal the error and mark that LG for manual inspection. The Health Checker runs these tests periodically to detect changes in LG templates, input parameters, or the response HTTP status.

2.2 Components of Periscope architecture

Figure 2 illustrates how components of Periscope's architecture inter-operate to satisfy measurement requests. Periscope exposes a *RESTful API* that can be used to query the available LGs, request new measurements, and retrieve results. Every request is logged in the *Repository* which works as a broker between the API and the rest of the Periscope components.

An *LG Client* receives measurement requests submitted to the Repository and translates them to LG queries. The LG Client executes requests through Se-

lenium [9], a web browser automation suite[2] that interacts with the LGs through a headless (without screen) browser according to the JSON configuration file produced at the end of the Periscope workflow.

If LGs did not impose query rate limits, Periscope could transmit all measurement requests directly to LGs from a single LG client. But most LGs bound the number of requests a given client IP can submit during a given time interval. For example, the Telephone LG [11] software logs the time and IP address of queries in a database, and checks subsequent queries against the last query from the same IP address; if it is less than a configured timeout (e.g., 1 minute), the LG drops the query. If Periscope had only a single LG Client (or multiple LG Clients behind the same public IP address), concurrent Periscope users would be limited to single-user querying frequencies. Although Periscope aims to prevent query rate violations, we also want to avoid very limited querying frequencies that would make Periscope impractical. For Periscope to scale to multiple users while being faithful to the per-user LG query rate, the system runs multiple LG client instances, using one IP address per end user.

Our first approach of assigning different public IPs to LG clients is by crowd-sourcing their hosting as *User Agents* in end-user machines. As of December 2015 we had crowd-sourced 5 Periscope LG Clients. Because the Periscope client is software-based, we can extend coverage using cloud-hosted Virtual Machines (VMs), where each *VM instance* has a public IP address from the cloud provider's address space. Periscope uses two cloud platforms: Google Compute Cloud (GCC) and Amazon Web Services (AWS). Each VM Instance hosts a single LG Client. The elasticity of cloud resources allows Periscope to start VM instances only when needed to satisfy request volume, and terminate them when not in use. Periscope needs as many LG Clients as the maximum number of users that concurrently query a single LG. Periscope first attempts to satisfy the requests using the active crowd-sourced User Agents; if it needs more agents, it launches VM instances.

A central *Controller* assigns measurement requests to LG Clients; it has a global view of system resources and coordinates execution of LG queries so as to stay within the LG query limits. The controller manages the number of cloud-hosted instances, and every crowd-sourced instance sends a keep-alive message every 5 minutes to inform the Controller that they can still accept measurements. When Periscope receives a new measurement request, the Controller decides when to dispatch it and which Client instance will execute it. The Controller's logic is based on two LG-specific variables that restrict the maximum number of concurrent queries submitted to an LG[3]:

1. A *timeout* that expresses the minimum time interval between two consecutive LG queries by the same user

[2] Although most requests can be satisfied with simple HTTP requests, Selenium allows easier handling of HTTP sessions and cookies.

[3] We derived empirically conservative values for the timeout and number of slots for each LG.

```
   Data: A set of measurement requests M for lg, and a set of active instances I
   Result: Assignment of a client instance i ∈ I' ⊇ I for each m ∈ M
 1 for m ∈ M do
   |  /* Timestamp of next permitted user query                          */
 2 |  m.ts ← lastQuery(m.user, lg) + lgTimeout(lg)
   |  /* Queue measurements in asceding m.ts order                       */
 3 |  mQueue.add(m)
 4 end
 5 while mQueue ≠ ∅ do
 6 |  measurement = mQueue.pop()
 7 |  slots ← totalSlots(lg) - activeSlots(lg)
   |  /* Wait until the next measurement can be executed                 */
 8 |  while (now() < measurement.ts) || (slots < 1) do
 9 |  |  wait()
10 |  end
11 |  assignedInstance ← false
12 |  for i ∈ I do
   |  |  /* Timestamp of next permitted instance query                   */
13 |  |  i.ts = lastQuery(i, lg) + lgTimeout(lg)
14 |  |  if i.ts > now() + lgTimeout(lg) then
15 |  |  |  assignedInstance ← i
16 |  |  |  break
17 |  |  end
18 |  end
19 |  if assignedInstance is false then
20 |  |  assignedInstance ← newCloudInstance()
21 |  end
22 end
```

Algorithm 1: The Controller's algorithm to assign concurrent measurement requests for an LG to the appropriate Client instances.

2. A number of *query slots* that indicate the maximum number of queries that Periscope will accept for an LG at any given moment.

Essentially, the timeout expresses a user-specific limit while the query slots impose a user-wide limit. If an LG has no available query slots it cannot be queried even if a user has not queried this LG for a period longer than the timeout. Algorithm 1 presents the Controller's decision process. For each query request the Controller calculates its execution time based on the timestamp of the last query from the same user toward the same LG, and the timeout of the LG (line 3). If the query does not conform to either of the two rate limits, it is queued inside the Controller (line 9) until the timeout expires and if at least one slot becomes available. When a query exits the queue, the Controller will choose an eligible Client instance to execute it. An instance is eligible if it has not executed a query to the same LG for a period longer than the timeout (line 15). If no active Client instance is eligible to execute the query, the Controller will request a new cloud-hosted instance (line 21). The required number of active Client instances will therefore depend only on the number of concurrent queries

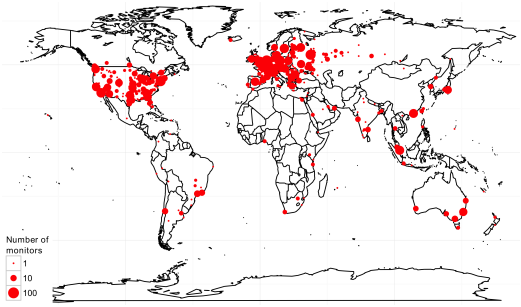

Fig. 3: Geographical distribution of LG VPs.

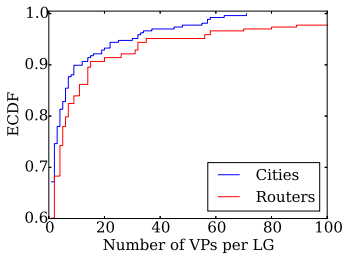

Fig. 4: CDF of router-level and city-level VPs per LG

to the same LG from different users, and not on the total number of active users or queried LGs.

3 Analysis

3.1 Coverage and Capabilities

As of December 2015 Periscope has extracted LGs for 297 Autonomous Systems. Periscope had automatically generated the configuration for 262 of these LGs; 35 LGs were not based on any of these initial templates and we parsed them manually. The LG-to-ASN mapping is not always readily available. In these cases we determined the IP address of the LG host and mapped it to an ASN using the longest prefix matching method. To get the IP address of the router that hosts each LG, we execute traceroutes against a machine under our control on which we run `tcpdump` to capture the incoming traceroute packets and extract the source address. We use the same technique to determine the traceroute protocol used by each LG. We found that 266 LGs use UDP probes, and 31 LGs use ICMP Echo Request probes. Whenever an LG supports both protocols, Periscope uses ICMP traceroute.

Each LG may allow the execution of its commands from different vantage points (VPs) inside the AS network, such as routers in different cities or routers that have different purposes (e.g. peering versus transit routers). We apply the same methodology we used for inferring the ASN of each LG, to geolocate an LG to a city whenever the LG interface does not reveal this information. After we determine the IP address of each vantage point, we map it to a city using NetAcuity's geolocation database [4]. Figure 3 shows the geographic distribution of LG vantage points that Periscope automatically parsed: 1,691 VPs distributed over 501 cities in 76 countries. As shown in Figure 4, 40% of the LGs have more than one city-level vantage point and 20% of the LGs have ten or more VPs. Figure 5 shows how many VPs support each LG command extracted by Periscope. Over 75% of the VPs offer both data and control plane measurements; 60% of the VPs support IPv6 commands in addition to IPv4. To determine which of the LG VPs are located in border routers, we check whether the AS of the first hop is different from the AS of the LG host. We examine the 416 VPs that

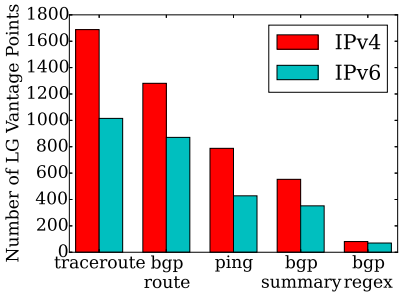

Fig. 5: Number of LG vantage points that support each command.

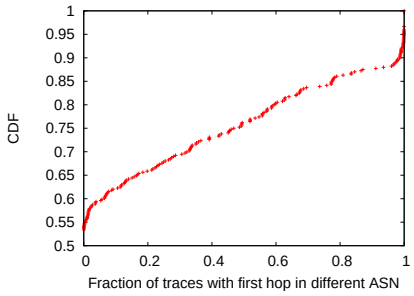

Fig. 6: CDF of fraction of traces with first hops (from LG VPs) that belong to a different AS, these LGs are likely deployed on border routers.

sourced at least 1000 traceroutes; of those, 222 had all traces going to an internal next hop, and 194 LGs had at least one trace that went directly to an IXP hop or a different ASN – these 194 are likely borders (Figure 6).

3.2 Comparison of topological coverage from LGs and Atlas

To compare the topology visible from our set of LGs, Atlas, and Ark VPs, we executed a traceroute campaign from each platform toward 2,000 targets in October 2015. At the time of our measurements, Atlas had 7,292 public probes in 2,779 different ASes across 160 countries, while Ark hadf 107 probes in 71 ASes across 41 countries.

To get an unbiased set of targets, we first collected the IP addresses found in the iPlane dataset [25], and executed a *ZMap* scan to keep only IPs that responded to both UDP and ICMP probes. We mapped IP addresses to their owner AS, and for each AS we randomly selected one IP address until we had a target set of 2,000 IP addresses each in a different AS, and spanning 151 countries [4]. This small sample is not necessarily representative of the global Internet, but it is required due to the probing rate restrictions on LG and Atlas infrastructure. We executed measurements from all Atlas probes, more than 6 million traceroutes in 2 months, using an account with elevated probing quota. With the default rate limit, this probing would have taken five years [8].

We compared the number of ASes, AS links and IXPs (based on a list of IXP prefixes extracted from PeeringDB [5]) observed in each dataset. Traces from LG vantage points to the target destinations traversed 3109 ASes, 29525 AS links, and 167 IXPs. The traces from Atlas probes to the same targets traversed 3369 ASes, 55936 AS links, and 171 IXPs, while traces from Ark traversed 1608 ASes, 10237 AS links, and 136 IXPs. Table 2 shows the number of ASes, AS links and IXPs per dataset, including those uniquely observed in each dataset. Interestingly, close to half (47%) of AS links seen in the LG traces (13,969 out of 29,525) did not appear in the Atlas or Ark traces, while 26% (809 out of 3109)

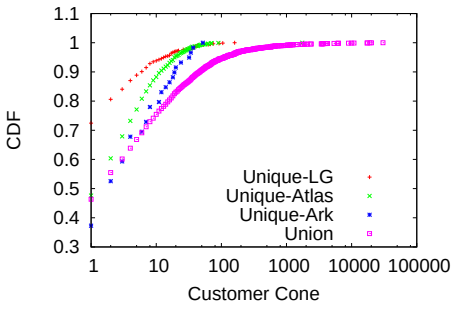

Fig. 7: CDF of the customer cones of ASes observed in LG, Atlas and Ark traces. The ASes uniquely observed in each dataset have significantly different customer cone sizes.

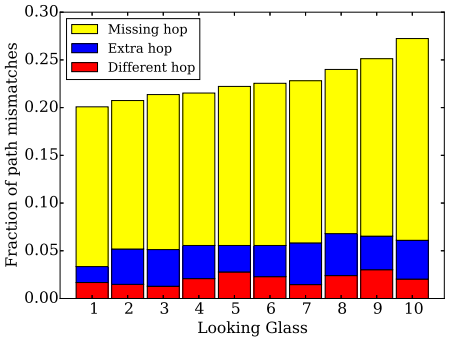

Fig. 8: Fraction of mismatches between traceroute and BGP paths, when longest prefix matching is used to map traceroute IP interfaces to ASes.

Dataset	ASes		AS Links		IXPs	
	Observed	Unique	Observed	Unique	Observed	Unique
LG	3109	809	29525	13969	167	16
Atlas	3369	1464	55936	40620	171	21
Ark	1608	59	10237	1625	136	8
All	4657	-	73348	-	202	-

Table 2: Number of ASes, AS links and IXPs observed in LG, Atlas, and Ark traces. Many AS nodes and links are uniquely observed in the LG dataset.

of ASes observed in the LG traces were not in Atlas or Ark traces. Finally, 16 IXPs observed in the LG traces were not observed in Atlas or Ark traces.

We compared ASes in each dataset using the *customer cone* as a metric of AS size. The customer cone is the number of ASes in the downstream path of a given AS, namely the number of ASes that can be reached through a customer, and it expresses the influence of an AS in the transit market [24]. Figure 7 shows the distribution of the customer cone sizes of ASes uniquely visible in the LG, Atlas and Ark datasets. ASes unique to each of the datasets significantly differ in cone size. LGs tend to capture more peripheral and stub ASes, while Ark and Atlas capture ASes with larger customer cones, due to the differences in the ASes that host the VPs of each platform. LGs are typically hosted in large transit providers that mainly access destination addresses through downstream paths. In contrast, Atlas and Ark VPs tend to be in eyeball ASes that traverse upstream paths to reach the same destinations. Comparison of topology visible from the LGs, Atlas VPs and Ark VPs reinforces our observation that the LG infrastructure provides a complementary view of topology compared to that visible from the existing Atlas and Ark infrastructures.

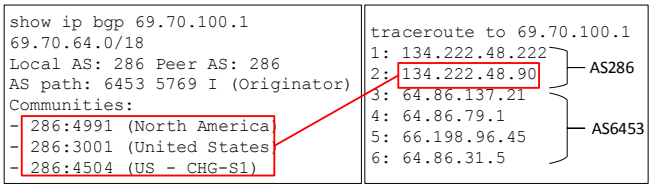

Fig. 9: When BGP community strings annotate the entry point of a route, combining them with traceroutes can enable city-level geolocation of IP interfaces.

4 Case studies

The ability to run BGP and traceroute measurements from the same LG VPs enables sophisticated studies that may not be feasible without combining control-plane and data-plane routing data.

4.1 Validation of IP-to-AS mapping

Validation of IP-to-AS mapping techniques typically requires comparison of BGP and traceroute paths obtained from VPs inside the same AS [26]. However, even among the PoPs of one AS, intra-domain routing may induce different paths to the same destination. Having traceroute and BGP VPs as closely located as possible, minimizes this risk, and LGs often support both functions from the same router. To investigate this potential, we used Periscope to study the accuracy of IP-to-AS translation when using longest prefix match to map IP interfaces to ASNs. We randomly selected 500 addresses from the experiments in section 3.1, and executed concurrent traceroute and `show ip bgp` measurements from 10 geographically diverse LGs. We sanitized the collected BGP paths by removing AS loops, private and reserved ASNs, and we discarded traceroute paths with unresponsive or unresolved interfaces. We compared the sanitized BGP and traceroute paths toward a given destination, ignoring IXP hops and repeated AS hops. Most path mismatches derived from traceroute missed the last AS-level hop that appears in the corresponding BGP path (Figure 8), which typically happens when a router interface in a customer AS has an address from its provider's IP range [22].

4.2 Geolocation of IP interfaces of border routers

Network operators often use the optional BGP communities attribute to tag a BGP route with the entry point where it was received by an external peer [18]. However, BGP communities provide only geographical location but not actual IP interfaces of the border routers. Combining BGP communities with traceroute paths from the same VP allows us to associate the locations encoded in the communities values to router interfaces, by identifying the interface that corresponds to the border between two ASes (Figure 9). We applied this technique for the AS286 LG, by executing simultaneous BGP and traceroute queries toward the same targets used in section 4.1. We pinpointed 89 border interfaces,

between AS286 and 58 of its AS-level neighbours, in 18 different cities. All the inferences agreed with DNS-based geolocation [21], although 23 interfaces had no corresponding hostname. In contrast, only 38% of the locations derived from the communities agreed with the NetAcuity database. Through follow-up RTT measurements we confirmed that the errors in the NetAcuity database.

5 Discussion and Future Work

We presented Periscope, a system that provides a unified interface to thousands of Looking Glass servers hosted by ISPs around the world. Periscope offers the capability for users to query any LG server without having to interact with individual LGs themselves, deal with timeouts and rate-limit issues, or develop code to automate issuing queries and parse LG responses. We showed that the topological view obtained from Periscope complements Atlas and Ark, serving as a valuable addition to the set of measurement platforms. Periscope respects the user-level limitations imposed by LGs, (a minimum time between successive queries by the same user to a given LG, and a maximum number of concurrent queries on the LG) and does not allow users to query at a rate faster than the LGs allow. Persicope distributes query instances, but measurements are dispatched through the API and a central Controller, which enforce LG rate limitations that cannot be overridden by querying instances. Preventing abuse is important, not only ethically but also because overwhelming the LGs would likely lead to their decommissioning from public use.

We plan to open Periscope for use by the research and operational community. We expect that allowing users into the system will be a (somewhat) manual process initially, mostly to prevent users from gaming the system by registering multiple user accounts. Beyond that we believe that the system can scale to many users, primarily because Periscope enforces the same per-user query quotas that the LGs themselves impose. Consequently, as long as Periscope can employ more LG clients than the typical number of query slots on a LG, the system can service user requests at the same rate offered by the LG. CAIDA's Archipelago [1] infrastructure already provides 132 active VPs that could be employed as LG clients. Cloud-hosted and crowdsourced LG clients can augment the set of clients, and reduce the querying load on each client. We provide documentation on how to obtain access and use the Periscope API at `http://www.caida.org/tools/utilities/looking-glass-api/`.

Acknowledgements

This work was supported in part by NSF CNS-1414177. The work was also funded by the DHS Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD) BAA 11-02 and SPAWAR Systems Center Pacific via contract N66001-12-C-0130, and by Defence R&D Canada (DRDC) pursuant to an Agreement between the U.S. and Canadian governments for Cooperation in Science and Technology for Critical Infrastructure Protection and Border Security. The work represents the position of the authors and not necessarily that of DHS or DRDC.

References

1. CAIDA Archipelago (Ark). http://www.caida.org/projects/ark/
2. Kewlio Looking Glass. http://sourceforge.net/projects/klg/
3. Multi-Router Looking Glass. http://mrlg.op-sec.us/
4. Netacuity. http://www.digitalelement.com/solutions/
5. PeeringDB. http://www.peeringdb.com
6. RANCID Loooking Glass. http://www.shrubbery.net/rancid/
7. RIPE Atlas. https://atlas.ripe.net/
8. RIPE Atlas rate limits. https://atlas.ripe.net/docs/udm/#rate-limits
9. Selenium browser automation suite. http://www.seleniumhq.org/
10. Stripes Looking Glass. https://www.gw.com/sw/stripes/
11. Telephone Looking Glass. https://github.com/telephone/LookingGlass
12. Version6 Loooking Glass. https://github.com/Cougar/lg
13. Vyatta. https://github.com/MerijntjeTak/vyattaLookingGlass
14. Achlioptas, D., Clauset, A., Kempe, D., Moore, C.: On the Bias of Traceroute Sampling: Or, Power-law Degree Distributions in Regular Graphs. In: STOC 2005
15. Augustin, B., Krishnamurthy, B., Willinger, W.: IXPs: mapped? In: IMC '09 (2009)
16. Bruno, L., Graziano, M., Balzarotti, D., Francillon, A.: Through the Looking-Glass, and What Eve Found There. In: WOOT 2014
17. Cohen, R., Raz, D.: The Internet Dark Matter - on the Missing Links in the AS Connectivity Map. In: IEEE INFOCOM 2006 (April 2006)
18. Donnet, B., Bonaventure, O.: On BGP Communities. SIGCOMM Comput. Commun. Rev. 38(2) (2008)
19. Giotsas, V., Zhou, S., Luckie, M., claffy, k.: Inferring Multilateral Peering. In: CoNEXT '13 (2013)
20. He, Y., Siganos, G., Faloutsos, M., Krishnamurthy, S.: Lord of the Links: A Framework for Discovering Missing Links in the Internet Topology. IEEE/ACM Transactions on Networking 17(2), 391–404 (2009)
21. Huffaker, B., Fomenkov, M., claffy, k.: DRoP:DNS-based Router Positioning. SIGCOMM Comput. Commun. Rev. 44(3) (Jul 2014)
22. Huffaker, B., Dhamdhere, A., Fomenkov, M., et al.: Toward Topology Dualism: Improving the Accuracy of AS Annotations for Routers. In: PAM '10 (2010)
23. Khan, A., Kwon, T., Kim, H.c., Choi, Y.: AS-level Topology Collection Through Looking Glass Servers. In: IMC '13 (2013)
24. Luckie, M., Huffaker, B., claffy, k., Dhamdhere, A., Giotsas, V.: AS Relationships, Customer Cones, and Validation. In: ACM IMC '13 (2013)
25. Madhyastha, H.V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., Venkataramani, A.: iPlane: An Information Plane for Distributed Services. In: USENIX NSDI '06 (2016)
26. Mao, Z.M., Rexford, J., Wang, J., Katz, R.H.: Towards an Accurate AS-level Traceroute Tool. In: ACM SIGCOMM '03 (2003)
27. Sánchez, M.A., Otto, J.S., Bischof, Z.S., Choffnes, D.R., Bustamante, F.E., Krishnamurthy, B., Willinger, W.: Dasu: Pushing Experiments to the Internet's Edge. In: USENIX NSDI '13 (April 2013)
28. Shavitt, Y., Shir, E.: DIMES: Let the Internet Measure Itself. SIGCOMM Comput. Commun. Rev. 35(5) (2005)
29. Shi, X., Xiang, Y., Wang, Z., Yin, X., Wu, J.: Detecting Prefix Hijackings in the Internet with Argus. In: IMC '12 (2012)
30. Zhang, B., Liu, R., Massey, D., Zhang, L.: Collecting the Internet AS-level Topology. ACM SIGCOMM CCR 35(1) (Jan 2005)