Application-Level Traceroute: Adopting Mimetic Mechanisms to Increase Discovery Capabilities

Chiara Caiazza, Enrico Gregori, Valerio Luconi, Francesco Mione, Alessio Vecchio

▶ **To cite this version:**

Chiara Caiazza, Enrico Gregori, Valerio Luconi, Francesco Mione, Alessio Vecchio. Application-Level Traceroute: Adopting Mimetic Mechanisms to Increase Discovery Capabilities. 17th International Conference on Wired/Wireless Internet Communication (WWIC), Jun 2019, Bologna, Italy. pp.66-77, 10.1007/978-3-030-30523-9_6 . hal-02881737

HAL Id: hal-02881737
https://inria.hal.science/hal-02881737v1

Submitted on 26 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Application-level traceroute: adopting mimetic mechanisms to increase discovery capabilities

Chiara Caiazza[1,2][0000−0002−6212−4805], Enrico Gregori[3], Valerio Luconi[3][0000−0003−4933−8566], Francesco Mione, and Alessio Vecchio[2][0000−0001−6894−7338]

[1] University of Florence, Italy
[2] Dip. di Ingegneria dell'Informazione, University of Pisa, Italy,
chiara.caiazza@phd.unipi.it, alessio.vecchio@unipi.it
[3] IIT-CNR, Pisa, Italy
enrico.gregori@iit.cnr.it, valerio.luconi@iit.cnr.it

Abstract. Traceroute is a popular network diagnostic tool used for discovering the Internet path towards a target host. Besides network diagnostic, in the last years traceroute has been used by researchers to discover the topology of the Internet. Some network administrators, however, configure their networks to not reply to traceroute probes or to block them (e.g. by using firewalls), preventing traceroute from providing details about the internal structure of their networks. In this paper we present *camouflage traceroute* (camotrace), a traceroute-like tool aimed at discovering Internet paths even when standard traceroute is blocked. To this purpose, camotrace mimics the behavior of a popular TCP-based application-level protocol. We show preliminary results that confirm that camotrace is able to obtain additional information compared to standard traceroute.

1 Introduction

The typical traceroute application, in its many forms, sends IP packets with increasing time-to-live (TTL) to discover the network path towards a destination. Traceroute relies on the fact that when a router receives an IP packet with TTL equal to 1, the router should discard it and send to the source address an ICMP packet indicating that the TTL has expired before arriving at the destination (ICMP Time Exceeded) [19]. From that ICMP packet, traceroute is able to discover the IP address of the router at that distance from the source.

Originally designed as a diagnostic tool, traceroute has been widely used by researchers to discover Internet paths at various levels of abstraction, e.g. IP interface, router, point-of-presence, and autonomous system (AS) [7, 20, 17, 14]. The effectiveness of traceroute depends on the response rate to traceroute packets of routers across the Internet and it can be limited by several factors [16]. First, not all routers always send ICMP packets when receiving a datagram with TTL equal to 1. Some may be configured to never send ICMP packets, some others may be configured to give low priority to this operation, and send ICMP packets

only when their load is low. In these cases, traceroute is not able to discover part of the path. Another relevant factor that could affect traceroute performance is the presence of modern firewalls, traffic shapers, or similar machines. These devices are able to recognize traffic as belonging to different applications, basing classification on the pattern or the payload of traversing packets (via deep packet inspection). Then, certain classes of traffic can be blocked, shaped, or throttled according to the network operator's policies (in the EU this practice is prohibited as it goes against the network neutrality principles [4]). Some operators configure their firewalls to block incoming and/or outgoing traceroute traffic. Since traceroute does not belong to any end-to-end application, they consider it useless (as it wastes the bandwidth available for other traffic), or even potentially dangerous (e.g., DDoS attacks). If this happens, the last part of the Internet path between a source and a destination will be unreachable by measurement probes.

We devised *camouflage traceroute* (*camotrace*), a traceroute variant whose aim is to bypass firewalls and shapers and to possibly discover those parts of the network that are inaccessible to conventional traceroute tools. Camotrace mimics the behavior of common application-level protocols to confound traffic classification tools and avoid being blocked by firewalls. The idea is to establish a TCP connection between the measurement source and a server inside a firewall-protected domain, and then vary the TTL of some TCP packets during communication to discover the intermediate routers. We ran a validation measurement campaign that showed that the output of camotrace is correct. In addition, we ran a set of experiments on the Italian Internet that show that camotrace is able to obtain additional information in comparison with classic traceroute.

2 Related Work

The traceroute tool has been firstly developed for network diagnostic purposes by Van Jacobson. This original traceroute uses UDP probes with high destination port number, to maximize the chance of not finding a used one. Each probe is sent with a TTL value increased by one with respect to the previous probe. According to the ICMP protocol RFC [19], once a probe reaches an intermediate router with a TTL value of 1, the router should discard it and send back to the source an ICMP Time Exceeded reply, which notifies that the probe has stopped on that router. On the destination, under the assumption that the destination port is not in use, an ICMP Port Unreachable is instead sent back. This is the implementation of the classic UNIX system's traceroute. On modern systems also an ICMP version of traceroute is available, based on ICMP Echo Request probes instead of UDP ones.

Besides diagnostic purposes, traceroute has been used in several studies of the past 15-20 years to infer Internet paths at various level of abstraction [6]: i) IP interface level, ii) router level upon alias resolution [15], and iii) AS level upon IP-to-AS mapping [5]. Traceroute measurements have been used as a basis

by several Internet mapping projects, such as CAIDA Ark [7, 1], iPlane [17], DIMES [20], or Portolan [9, 13, 10].

In the meanwhile, also some issues of the classic traceroute implementation have been discovered. In particular, bias in the outcome of Internet mapping measurements could be introduced because of the presence of load balancers, firewalls, or other evolved network equipment commonly referred to as middleboxes [2, 8]. Modern traceroute variants have been implemented to prevent or reduce the impact of such issues. For example, Paris traceroute [2] is designed to avoid known issues due to load balancers. The multipath detection algorithm (MDA) has been subsequently added to Paris traceroute to integrate its ability to discover all possible paths between a source and a destination in the presence of load balancers [3]. Tracebox is instead a tool which is able to discover the presence of middleboxes (i.e., machines that operate at levels higher than the network level) along the path between the source and the destination [8]. These tools however are not able to bypass firewalls specifically configured to block traceroute executions, which instead is the purpose of camotrace.

TCP traceroute has been developed to be able to bypass firewalls configured to block UDP- and ICMP- based probes. However, it must be noticed that TCP traceroute behavior is different from the one we propose. TCP traceroute's probes are just TCP SYN packets, but a connection between source and destination is never established. If the target host is not listening for incoming connections, a TCP RST will be generated to indicate to the other endpoint that the port is not open. Conversely, if the selected port on the target host is open, a TCP SYN+ACK will be sent back to the host running TCP traceroute. The latter terminates the connection with a TCP RST (the three-way handshake is never completed). This makes TCP traceroute easily identifiable by modern sophisticated firewalls. In camotrace instead, a TCP connection is first established and only after that TCP segments are used as probes by varying their TTL. Moreover, the payload of TCP segments contains the application-level data of the protocol currently in use by camotrace (i.e. HTTP). These differences are not marginal: since a connection is effectively established, packets can be considered by stateful firewalls as belonging to the same flow; since the payload of TCP segments contains real application-level data, this may help in making the flow being classified as non-diagnostic by deep packet inspection mechanisms.

3 Method

To better understand the behavior of camotrace, we briefly recall the main concepts upon which traceroute is based. Traceroute probes are IP packets with either UDP, ICMP, or TCP payload. These probes are sent without establishing a connection with the target host. For UDP probes, the payload is empty or random; ICMP probes are ICMP Echo Requests; TCP probes are instead TCP SYNs. Probes are sent cyclically with increasing IP TTL values, starting from 1. Traceroute stops when the target host or a maximum TTL value (hereafter MAX_DEPTH) is reached. Common traceroute implementations use 30 as

Algorithm 1 Camouflage traceroute probing algorithm

```
1: MAX_TTL ← System default TTL
2: MAX_DEPTH ← 40
3: MAX_ATTEMPT ← 3
4:
5: for all x ∈ {1..MAX_DEPTH} do
6:     setTTL(x)
7:     start timer
8:     send an HTTP request
9:     setTTL(MAX_TTL)
10:    nAttempt = 0
11:    while ICMP Time Exc. not received && nAttempt < MAX_ATTEMPTS do
12:        while True do
13:            try
14:                listen for and consume ICMP Time Exceeded packets
15:                if ICMP Time Exceeded packets arrive then
16:                    restart timer
17:                    break
18:                end if
19:            catch timer expired
20:                break
21:            end try
22:        end while
23:        if the server closes the socket then
24:            connect to the server
25:        end if
26:        nAttempt = nAttempt + 1
27:    end while
28: end for
```

the default maximum TTL value[4]. At each iteration, a probe can reach either an intermediate router or the target host. Intermediate routers should send back an ICMP Time Exceeded packet, which indicates that the probe has reached the router with a TTL value of 1. The target host instead should respond with an ICMP Port Unreachable (if UDP probe), ICMP Echo Reply (if ICMP probe), or a TCP RST or SYN+ACK (if TCP probes), that will stop traceroute operations.

To disguise itself and bypass firewalls or other blocking entities, camotrace mimics the behavior of application-level protocols. In particular, we implemented camotrace to act as an HTTP speaker. Camouflage traceroute operates in two phases. In the first phase camotrace establishes a connection with the target host. Thus, to operate correctly, camotrace needs as a target for measurements a host listening for connections for the implemented protocol (i.e., an HTTP-based service or a Web-server). In its current implementation, to establish a connection, camotrace uses sockets of type stream, relying on the operating system support for the TCP protocol. In other words, to avoid implementing all the intricacies of TCP mechanisms, camotrace implementation uses just the functionalities offered by the stream socket interface. As a consequence, camotrace does not have the visibility at the packet level for both outgoing and incoming traffic. The second phase is the probing phase. Once connected to the server, camotrace changes dynamically the TTL associated with outgoing data to dis-

[4] In some preliminary tests conducted using standard TCP traceroute, we observed that 30 hops may be insufficient to reach all destinations. For this reason, we set MAX_DEPTH to 40 hops for the experiments described in Section 5.

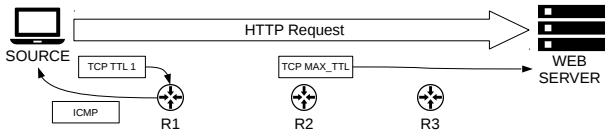


Fig. 1: Camotrace principle of operation.

cover routers along the path. In this phase, camotrace operates according to Algorithm 1. In detail, for values of x ranging from 1 to MAX_DEPTH camotrace executes the following steps: i) the time-to-live (TTL) associated with the socket is set to x (this is done to discover the router at x hops from the sender); ii) up to $MAX\_ATTEMPTS$ HTTP requests are sent through the socket: this data will elicit an ICMP error on the router at x hops from the sender (camotrace may stop before $MAX\_ATTEMPTS$ probes are sent if an ICMP error is received); iii) the TTL associated with the socket is reset to its default value (MAX_TTL), this is done to retransmit the HTTP request with a TTL that makes it reach the other endpoint; iv) ICMP errors are consumed; v) the state of the socket is checked: if it has been closed by the server, a new connection is established. An overview of camotrace operations is shown in Figure 1.

ICMP Time Exceeded errors are handled in a `while` cycle to cope with possible TCP re-transmissions. The execution exits from the `while` cycle when a timer associated with the socket expires (`catch` block). In other words, if no ICMP Time Exceeded errors are received for a given amount of time, camotrace assumes that the router at x distance is not responding and it proceeds to the next hop. The payload of probes is an HTTP 1.1 GET request with the following simple format:

```
GET / HTTP/1.1
Host: <target_host_name>
Connection: keep-alive
```

It must be noticed that camotrace is able to detect a hop only if an ICMP packet is received. Since the destination host does not send any ICMP packet, camotrace is not able to determine if, and eventually when, the target is reached. Therefore, in its current implementation, the algorithm always continues until MAX_DEPTH is reached. The default values for MAX_DEPTH and $MAX\_ATTEMPTS$ are 40 and 3, respectively.

3.1 Performance enhancements for some specific cases

Since the Web server on the target machine is not under camotrace's control, the latter has to deal with arbitrary decisions that may affect the connection. To cope with these events we modified the basic camotrace algorithm presented in the first part of this section. In particular, two improvements were introduced.
Managing non persistent connections. To successfully operate, the connection between the sender and the target has to be persistent. The HTTP 1.1

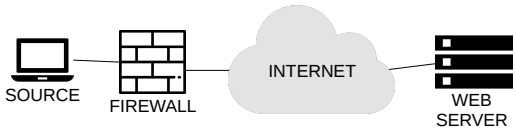


Fig. 2: Validation set up.

standard states that the connection between client and server should be persistent [11]; however, some HTTP servers still close the connection immediately after completing to serve a request. To bypass this problem, we implemented a camotrace variant that, instead of sending a complete HTTP request at each step, sends only a portion of the request (few bytes at a time). More precisely, the operations illustrated in Algorithm 1 are changed by sending just a few bytes of an HTTP request and not a complete one (Line 8).

Managing other unexpected connection closures performed by the server. Even when adopting the above described mechanisms, the server may still unexpectedly close the connection. This could be due to several reasons, such as a high workload on the server or the presence of timers associated with connections. In fact, if a request is not completed in a given amount of time, or if the time between two consecutive requests is too long, the server can close the connection with the client. This issue may affect both the default camotrace algorithm and the variant previously described. To cope with this problem, when a connection-close is detected camotrace connects again to the target and restarts probing activities from the last hop reached during the previous run.

4 Validation

To validate camotrace both in terms of principle of operation and implementation we ran a two-step validation.

We first checked if camotrace is able to correctly discover the path from a source to a destination. We ran a measurement campaign with target belonging to the GARR network. GARR is the Italian public research network that connects all the Italian Universities and Research Centers [12]. The map of the GARR network is publicly available[5], thus we have been able to check if the paths found by camotrace were correct. We ran measurements towards 17 machines hosting the Web sites of University institutions and spread all over Italy. For all targets we checked that the Web server was actually hosted in the network of the considered institution. For all targets we successfully verified that the path found by camotrace was equal to the one available on the network map.

Second, we checked the ability of camotrace to bypass firewalls that are configured to block traceroute traffic. This validation step was run in a controlled environment set up between the IIT-CNR and the University of Pisa. The machine running camotrace was located in the IIT-CNR network. In the same

[5] `https://gins.garr.it/xWeathermap/mapgen.php?slice=garrx_top`

network a Palo Alto firewall was in execution [18]. Such device is able to recognize traffic at the application level via deep packet inspection, and then to block, shape, and forward traffic according to policies defined by administrators. We also set up a web server on a machine located at the University of Pisa. The validation environment is shown in Figure 2.

The Palo Alto firewall was configured to block all traceroute applications between the machine hosted at IIT-CNR and the web server hosted at the University of Pisa. We ran three types of traceroute (UDP, ICMP, and TCP on port 80) and camotrace between the two hosts. The Palo Alto firewall was able to block all traceroute traffic except camotrace. Thus camotrace was the only traceroute application which always managed to discover the entire path between the source and the destination. In other words, camotrace was able to bypass the Palo Alto firewall configured to block traceroute.

5 Results

We evaluated the discovering capabilities of camotrace using a large set of Italian Web servers as targets. To generate the list of targets, we first collected from the Italian DNS system approximately one million domains belonging to the *.it* TLD. The list of domains was resolved to $\sim$ 800 k IPv4 addresses (the remaining $\sim$ 200 k names were registered but not associated with any IP address). We then removed duplicate addresses, thus obtaining a list of $\sim$ 92 k unique IPv4 addresses. The significant reduction from $\sim$ 800 k to $\sim$ 92 k addresses is due to the fact that many websites are actually hosted by the same physical machine. Finally, we selected a single IP address for each AS in the list, and this produced a final set of 3 260 targets, which were used for the experiments described in the following. For performing IP-to-AS mapping we used the Whois service provided by Team Cymru [21]. The rationale for the last step was to be able to carry out experiments in a reasonable amount of time (collecting traceroute results is time consuming) while preserving heterogeneity. We suppose that traceroute filtering policies may be quite different from organization to organization, whereas policies can be reasonably homogeneous within a single organization. The significant reduction of the set of targets caused by the last filtering step is due to the fact that a large fraction of websites is managed by a relatively small number of hosting providers. For each of these destinations we executed both camotrace with the reconnect option and TCP traceroute. Both were configured to explore paths with $MAX\_DEPTH = 40$ hops. We compared camotrace with TCP traceroute only, as the latter is known to have better discovering capabilities in comparison to ICMP and UDP traceroute.

For 629 targets, camotrace was not able to successfully perform the connection to the server. There are several possible reasons behind this behavior: the target host may be disconnected, the target may be behind a completely blocking firewall, or a Web server may be unavailable on the target host. In fact, the presence of an entry in the DNS system does not imply that a Web server is necessarily running at that address. For 89.5% of the 629 targets, also

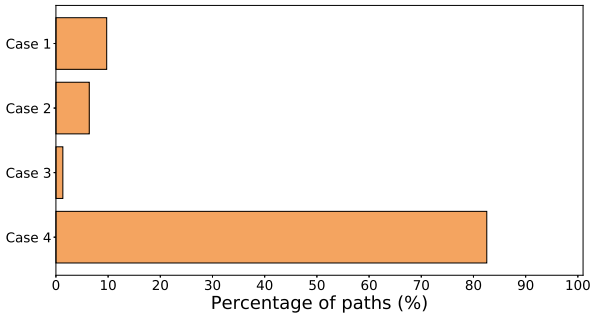


Fig. 3: Percentage of paths where camotrace finds more nodes (case 1), TCP finds more nodes (case 2), both find some nodes not found by the other but not all the nodes of the other (case 3), both find exactly the same nodes (case 4).

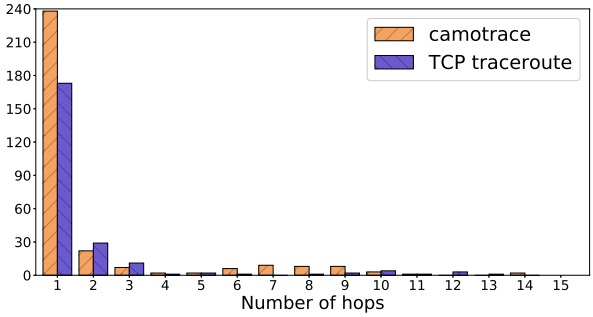
TCP traceroute was unable to reach the destination machine (even though it was able to collect information about the intermediate nodes along the path), thus suggesting that the target IP address is not allocated. Since camotrace requires a Web server in execution at the target machine, this subset of targets has been discarded, and hereafter only the targets for which camotrace is able to successfully perform the connect operation will be taken into account.

Figure 3 shows the fraction of paths in which camotrace is able to find additional information on intermediate nodes with respect to TCP traceroute (case 1) and vice-versa (case 2). The third column shows the fraction of paths where each algorithm is able to find some more hops with respect to the other algorithm, but at the same time is unable to find all the hops of the other one (case 3). Finally, the last column shows the fraction of paths where the two algorithms find the same set of hops (case 4). More formally, let us call I_{cam} and I_{tcp} the sets of intermediate nodes found by the two algorithms along the path[6]. The first and second columns represent the fraction of targets where $I_{cam} \supset I_{tcp}$ and $I_{cam} \subset I_{tcp}$, respectively. The third column corresponds to the case when $I_{cam} \neq I_{tcp}$ and $I_{cam}, I_{tcp} \subset (I_{cam} \cup I_{tcp})$. Finally, the last column represents the case when $I_{cam} = I_{tcp}$.

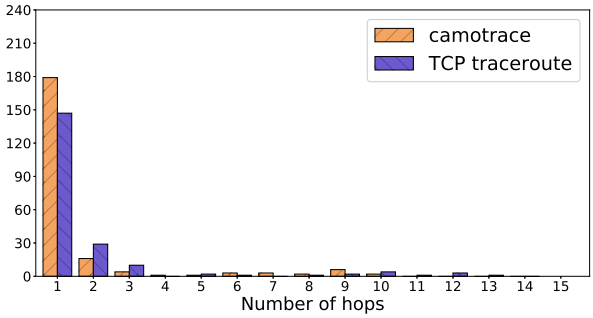
For approximately 83% of probed paths, camotrace and TCP traceroute found the same set of intermediate nodes. This means, conversely, that in approximately 17% of the paths the chosen algorithm influences the set of discovered routers. In particular, case 1 accounts for $\sim 10\%$, whereas case 2 accounts for $\sim 6\%$, demonstrating that camotrace may be able to provide more information. Case 3 covers a limited number of paths ($\sim 1\%$).

The above results provide an indication on the number of paths where camotrace performs better than TCP traceroute and vice-versa, but they do not measure the amount of additional information that is discovered. Figure 4a shows the number of additional IP interfaces that each algorithm is able to find in the paths

[6] For example, $I = \{1, 4, 5\}$ when the first, fourth, and fifth routers are found.



(a) Excluding the last occurrence of the destination address in TCP traceroute.



(b) Excluding all the occurrences of the destination address.

Fig. 4: Number of additional nodes.

containing differences (which are, as mentioned, $\sim 17\%$ of the total number of paths). For each number of additional IP interfaces, the number of occurrences for both camotrace and TCP traceroute are presented. As expected, camotrace is able to find a higher number of IP interfaces. For example, camotrace is able to find one additional IP interface in comparison to TCP traceroute on the path towards approximately 240 targets, while the opposite occurs approximately 170 times. For two and three additional interfaces TCP performs slightly better, then for higher numbers of interfaces camotrace is again better. In few cases, TCP traceroute finds almost all the IP interfaces along the path whereas camotrace is unsuccessful. This explains the long tail of the TCP traceroute distribution. These limited number of cases are due to some anomalous behaviors that are analyzed in Section 5.1.

As previously mentioned, camotrace, differently from TCP traceroute, is not able to detect the target machine. Thus, the above results have been computed not considering the last hop found by TCP traceroute, i.e. the target itself. However, during the analysis, we noticed that in some cases some intermediate

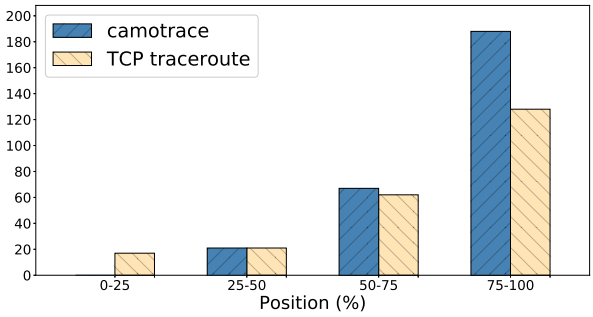


Fig. 5: Number of new IP interfaces, or groups, found by the two methods against the normalized position.

nodes were replying using the target's address. This could indicate that such hosts are behind a NAT connected to the public Internet using the target's address. We thus computed again the number of additional IP interfaces but excluding the target address, to show that this phenomenon marginally affects the previously discussed results (Figure 4b).

In addition, we computed the position along the path of the additional (or groups of additional) IP interfaces found by the two methods (for groups, just the starting position is considered). Since the length of the path is different from target to target, the position is expressed as a percentage from the beginning of the path. Figure 5 shows that for camotrace the newly discovered IP interfaces are mostly in the second half of the path, and in particular in the last 25%. This is rather expected, as it is reasonable to suppose that the majority of classification and filtering systems are placed in non-transit networks. It must be noticed that in the first 25% of the considered paths, only TCP traceroute is able to discover more interfaces than camotrace, whereas the opposite does not occur. This situation takes place just a few times, for some atypical behaviors described in Section 5.1.

5.1 Analysis of some atypical situations

Figure 4 includes a limited number of cases where TCP traceroute finds a rather large number of hops unseen by camotrace. We analyzed these cases in detail using a packet sniffing tool (Wireshark) and we found two main anomalous behaviors.

The first situation takes place right after the connection is established. The target server sends a TCP window update message that resets the TTL value to its maximum value. This means that subsequent messages are sent directly to the server and camotrace is unable to receive any ICMP message from the intermediate nodes. After the first message has been received these anomalous servers send a new TCP window update message or close the connection. This forces camotrace to open a new connection, thus starting the same behavior

again. In the end, camotrace is not able to send any message with TTL lower than 64 and no hop along the path can be discovered.

In the second case, the server accepts the connection but an HTTP reply is never sent. Camotrace sends the first message with TTL=1 receiving an ICMP message from the first router. Then it sends a message to the destination with TTL = 64 but no response is received. The underlying TCP layer starts to retransmit the packet and all following requests are queued. No ICMP message from the intermediate routers is received (with the exception of the first one). In some cases, after a while, the server sends a TCP reset and the connection can be re-established. However, since the server keeps being not responsive to HTTP requests, only an additional intermediate node can be discovered.

6 Conclusion

Traceroute is the most widely used tool for obtaining information about the topology of the Internet, and in the last decades it has been the cornerstone of countless research works. Camouflage traceroute tries to expand the amount of information collected in those portions of the network where operators apply restricting policies to diagnostic traffic. This is done by mimicking the behavior of application-level traffic, thus reducing the probability of being classified and consequently restricted. The main limitation of camotrace is that a server is required to be running on the target machine, as camotrace needs that a connection is open for delivering probes containing application-level traffic. Current implementation of camotrace only supports HTTP traffic, but other application protocols can be added to further improve its discovering capabilities and increase the set of possible targets.

Experiments carried out on the Italian Internet show that camotrace is able to provide more information than TCP traceroute in approximately 10% of the paths, while the opposite occurs in 6% of the paths. We believe that this is a good improvement, especially considering that the traceroute tool is well consolidated. Moreover, it is possible to conceive a tool that first operates as the classical TCP traceroute and then as camotrace, to finally produce a set of intermediate routers that corresponds to the union of the results obtained by the two methods.

Future work will focus on studying how to cope with camotrace limitations, mainly the ability to discover the target IP address. In addition, we plan to execute a world-wide measurement campaign to evaluate both the soundness of camotrace and the diffusion of traceroute blocking mechanisms at a planetary scale.

Acknowledgment

This work was partially funded by the University of Pisa (project PRA 2017_37 - “IoT e Big Data”), and the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence).

References

1. The Cooperative Association for Internet Data Analysis Archipelago Measurement Infrastructure (CAIDA Ark). http://www.caida.org/projects/ark/
2. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding Traceroute Anomalies with Paris Traceroute. In: Proc. ACM SIGCOMM IMC. pp. 153–158 (2006)
3. Augustin, B., Friedman, T., Teixeira, R.: Multipath tracing with Paris traceroute. In: Proc. IEEE/IFIP E2EMON '07. pp. 1–8 (2007)
4. BEREC Guidelines on the Implementation by National Regulators of European Net Neutrality Rules. http://berec.europa.eu/eng/document_register/subject_matter/berec/download/0/6160-berec-guidelines-on-the-implementation-b_0.pdf (2016)
5. Chang, H., Jamin, S., Willinger, W.: Inferring AS-level Internet Topology from Router-Level Path Traces. In: Proc. SPIE ITCom '01. pp. 196–207 (2001)
6. Cheswick, B., Burch, H., Branigan, S.: Mapping the Internet. IEEE Computer **32**(4), 97–98, 102 (1999)
7. claffy, k., Hyun, Y., Keys, K., Fomenkov, M., Krioukov, D.: Internet Mapping: From Art to Science. In: Proc. CATCH '09. pp. 205–211 (2009)
8. Detal, G., Hesmans, B., Bonaventure, O., Vanaubel, Y., Donnet, B.: Revealing Middlebox Interference with Tracebox. In: Proc. IMC '13. pp. 1–8 (2013)
9. Faggiani, A., Gregori, E., Lenzini, L., Luconi, V., Vecchio, A.: Smartphone-based crowdsourcing for network monitoring: Opportunities, challenges, and a case study. IEEE Comm. Mag. **52**(1), 106–113 (2014)
10. Faggiani, A., Gregori, E., Lenzini, L., Mainardi, S., Vecchio, A.: On the feasibility of measuring the Internet through smartphone-based crowdsourcing. In: Proc. WiOpt '12. pp. 318–323 (2012)
11. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (1999)
12. Consortium GARR Home Page. https://www.garr.it/
13. Gregori, E., Lenzini, L., Luconi, V., Vecchio, A.: Sensing the Internet through crowdsourcing. In: Proc. PerMoby '13. pp. 248–254 (2013)
14. Gregori, E., Luconi, V., Vecchio, A.: Studying forwarding differences in European mobile broadband with a net neutrality perspective. In: Proceedings of the 24th European Wireless Conference. pp. 81–87 (May 2018)
15. Keys, K., Hyun, Y., Luckie, M., claffy, k.: Internet-Scale IPv4 Alias Resolution with MIDAR. IEEE/ACM Transactions on Networking **21**(2), 383–399 (2013)
16. Luckie, M., Hyun, Y., Huffaker, B.: Traceroute Probe Method and Forward IP Path Inference. In: Proc. ACM SIGCOMM IMC '08. pp. 311–324 (2008)
17. Madhyastha, H.V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., Venkataramani, A.: iPlane: An Information Plane for Distributed Services. In: Proc. USENIX OSDI '06. pp. 367–380 (2006)
18. Palo Alto Networks. https://www.paloaltonetworks.com/
19. Postel, J.: Internet Control Message Protocol – DARPA Internet Program Protocol Specification. RFC 792 (1981)
20. Shavitt, Y., Shir, E.: DIMES: Let the Internet Measure Itself. ACM SIGCOMM Comput. Commun. Rev. **35**(5), 71–74 (2005)
21. Team cymru: IP to ASN mapping. http://www.team-cymru.com/IP-ASN-mapping.html