Modern Traceroute Variants and Adaptations

Julius Niedworok, Johannes Zirngibl, Patrick Sattler*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: julius.niedworok@tum.de, zirngibl@net.in.tum.de, sattler@net.in.tum.de

***Abstract*—Traceroute is a widely used tool to perform measurements on the network path from a source to a destination. However, there are some issues when applying it to the recent structure of the Internet. Load balancing techniques are very common to encounter on today's Internet paths. This article describes a few approaches that can be used to adapt the original concept of traceroute to the current structure of the Internet. When looking at network topology discovery, the *Multipath Detection Algorithm (MDA)* and its lightweight version MDA-Lite come in place. For broader discovery *Yelling at Random Routers Progressively (Yarrp)* can help to speed up the process, which is advantageous for measurement of short living network topologies. However, when tracing a single application flow, Service traceroute provides more precise results. Choosing the right approach for a given use case is crucial in order to obtain appropriate results.**

***Index Terms*—Active Internet Measurements, Traceroute, Network Topology Analysis**

1. Introduction

Over the years traceroute became a well-known tool for network administrators to perform network diagnosis tasks. The original approach introduced by V. Jacobson in 1989 [1] is implemented in the standard Linux tool *traceroute*. The idea of traceroute is to provide the user insights on the path, which is taken through a network to a given destination. It creates active measurement probes for each hop along the path. This is done iteratively by increasing the probe's *Time To Live (TTL)* and inspect the ICMP response message of the host where the TTL expired [2]. The probe packets can be of different protocol types depending on the use case and on the network infrastructure. Although traceroute is widely used today, it does not fit the needs of the present structure of the Internet. Traditional traceroute is based on the assumption that there is a single forward path to the destination host. As shown in Figure 1, this can be problematic in load balanced networks. Traceroute would send a probe to node *A* with TTL n. When increasing the TTL to $n+1$, the probe could either traverse the upper or the lower path. In case of Figure 1 the probe reaches node *B*. Traceroute again increases the TTL to $n+2$. However, this time the probe traverses the lower path, through node *C*, and discovers node *F*. Given that the previous probe discovered node *B*, traceroute would assume that there exists a path between node *B* and *F*. The ICMP response message contains only

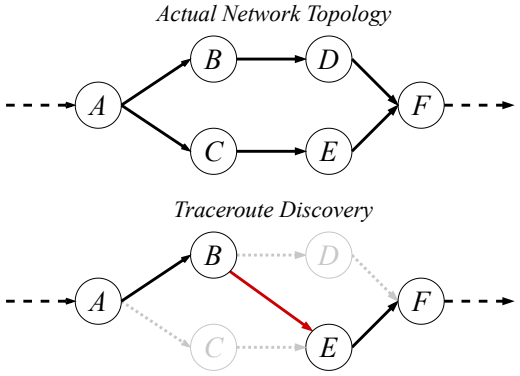

Figure 1: Problems using Traditional traceroute

the address of the node where the TTL expired. Therefore, traditional traceroute cannot detect, if the probe traversed through a different path than previous probes. Moreover, nodes *C* and *D* remain hidden.

The utilization of load balancing techniques has increased excessively over the last years [3]. In general there are different ways of performing load balancing on network traffic [4], [5]:

- **Per-destination** load balancing depends on the destination specified in the packet.
- **Per-flow** load balancing derives its decision from the packets flow identifier.
- **Per-packet** load balancing concentrates on keeping the load as equal as possible. No effort is made to keep packets of a single flow on the same path.

Per-destination load balancing does not create a problem when tracing the forward path to a single destination host. Traditional traceroute changes the header to be able to match an ICMP response packet to its corresponding probe packet [2]. This is done by varying the destination port field for UDP probes and the sequence number for ICMP Echo probes. In case of per-flow load balancing such behavior leads to potential different paths for each probe packet. As described by Augustin et al. [6], this can be mitigated using Paris traceroute. Paris traceroute explicitly controls the packet header to direct the packet through a certain path. An ICMP response packet contains the discarded header of the probe, as well as the first eight octets of the payload. Instead of changin the flow identifier, Paris traceroute makes use of these eight octets for matching probes to the responses. In consequence, Paris traceroute can deal much better with topologies such as the one shown in Figure 1. Paris traceroute cannot get around

doi: 10.2313/NET-2020-04-1_06

the problems created by per-packet load balancing due to its randomness. However, in those cases where there is per-packet load balancing employed Paris traceroute can detect it.

These days traceroute is not only used for what it was initially intended. It is applied to a broader range of problems, i.e., detection of all load balanced paths or tracing specific services. Some approaches even separate from the idea of tracing towards a single destination host and rather try to get a bigger picture of the network topology [7], [8]. The following sections will look at recent traceroute-based approaches that deal with these additional use cases.

2. Network Discovery

This section describes approaches to discover the network in a broader sense. First, it explains an approach to discover all paths from a single source to a single destination. Afterwards, it diverges from the traditional scenario of tracing towards a single destination. A recent approach to discover the network topology is presented.

2.1. Detecting all Paths

After introducing Paris traceroute, the authors explored a way of getting a broader view of the network topology [9]. In 2009 Darryl Veitch et al. [7] present their final version of the *Multipath Detection Algorithm (MDA)*. MDA extends Paris traceroute to reveal all possible load balanced paths to a given destination. It runs iteratively through the paths and elicits all interfaces at each hop. In order to enumerate the paths from a node at hop h, it generates a number of probes with random flow identifiers and selects the ones which reach that node. Subsequently, it sends these probes to hop $h+1$ to discover all successors. It sends these probes under the assumption that hop h is a load balancer that evenly allocates traffic to k paths. MDA is using a statistical approach to compute the number of probes n_k which need to be sent over the node at hop h in order to enumerate all of its successors at a given level of confidence. If it is not possible to find more than k successors after sending n_k probes, MDA stops and assumes to have enumerated all possible successors. However, in case $k+1$ successors have been found, MDA continues with the assumption that there are at most $k+1$ successors. Correspondingly, it will generate and send n_{k+1} probes. With this approach MDA claims to find all load balanced paths between a source and a destination host.

There are different constructs that can be encountered when discovering the network topology. As soon as load balancing comes in place, so-called, *diamonds* will be exposed. According to the definition of Augustin et al. "a diamond is a subgraph delimited by a *divergence point* followed, two or more hops later, by a *convergence point*, with the requirement that all flows from source to destination flow through both points" [10]. Figure 2 shows two examples of diamonds. The upper diamond is an example of an *unmeshed* diamond and the lower one an example of a *meshed* diamond. Vermeulen et al. [3] define *meshing* between two hops to meet one of the following criteria:

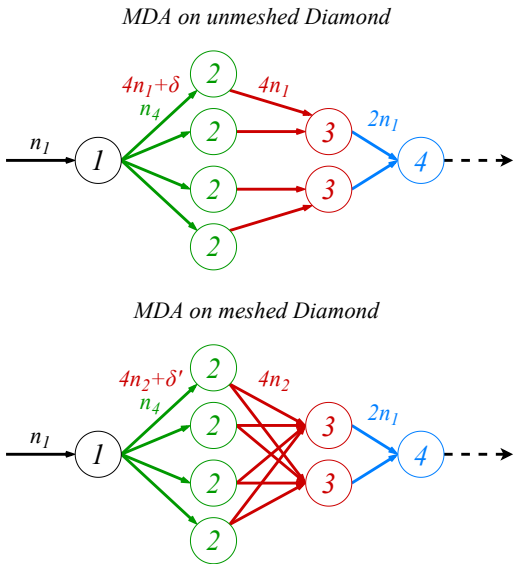

Figure 2: MDA on Meshed and Unmeshed Diamonds

- If the two hops h and $h+1$ have the same number of nodes, then the out-degree of at least one node of hop h must be two or more. Consequently, at least one node at hop $h+1$ will have an in-degree of two or more.
- If hop h has fewer nodes than hop $h+1$, then the in-degree of at least one node at hop $h+1$ must be two or more.
- If hop h has more nodes than hop $h+1$, then the out-degree of at least one node at hop h must be two or more.

A diamond is considered to be meshed as soon as one hop is meshed. In the example of Figure 2 MDA would execute the following steps in order to detect the diamond topology [7]:

1) To discover hop 1, MDA sends n_1 probes with the assumption that there is only one node. As it cannot discover a second node, it will stop after n_1 probes.
2) In order to discover hop 2, MDA will send n_1 probes again. However, this time it will discover another node before it stops. Therefore, it will continue with the assumption that there are two nodes. While sending n_2, probes it will discover a third node and, after adjusting the number of probes, a fourth one. As it cannot discover a fifth node, it will stop after sending n_4 probes.
3) When discovering hop 3, MDA starts with the assumption that each of the four nodes has one successor. A number of n_1 probes need to be sent over each of the four nodes. As the set of n_4 probes that revealed these four nodes at hop 2 does not contain a number of n_1 probes per hop 2 node, MDA needs to generate more probe packets. When generating new probes, it is unlikely that the new probes are evenly spread over all four nodes. Therefore, an overhead of a few additional probes, denoted as δ will occur. In the case of the unmeshed diamond, MDA will stop after sending these probes as only one

doi: 10.2313/NET-2020-04-1_06

successor node is discovered for each node at hop 2. For a meshed diamond, however, MDA will detect a second node. It needs to send n_2 probes over each node at hop 2. Therefore, MDA generates another set of additional probes with a potential overhead of δ'. As no third node will be discovered, MDA will stop after sending these probes.

4) In the last step MDA will discover the node at hop 4 by sending a number of n_1 probes per node at hop 3. As there are already n_1 probes available per hop 3 node, there is no need to generate new probes. MDA will stop after sending a total number of $2n_1$ probes.

This procedure requires a lot of active measurement traffic to be generated. For this reason, Vermeulen et al. [3] came up with a lightweight version of the algorithm, called *MDA-Lite*. It makes use of the benefit that nearly half of all diamonds, which are found in the Internet, only have a single hop with multiple nodes [3]. In consequence, meshed diamonds are very uncommon. This allows MDA-Lite to minimize the situations where it needs to generate new probes for already discovered nodes, as seen in Step 3 of full MDA. Figure 3 illustrates the process of MDA-Lite on the same unmeshed diamond as in Figure 2. Discovery of hop 1 and 2 works the same way as for MDA. When revealing hop 3, the benefit of MDA-Lite comes into effect. Instead of generating and sending $4n_1 + \delta$ probes, MDA-Lite will consider the set of nodes at hop 2 as one node and proceed as usual. As two nodes are discovered, n_2 probes need to be sent. These probes can be taken from the set of n_4 probes, which were sent before. Similar to the third step, MDA-Lite will consider all nodes at hop 3 as one and send a total number of n_1 probes to discover hop 4. MDA-Lite will therefore send $2n_1 + n_2 + n_4$ probes in total. In contrast, full MDA will send $11n_1 + \delta$ probes for the unmeshed and $8n_2 + 3n_1 + \delta'$ probes for the meshed diamond. Keeping in mind that for most confidence levels, including the ones used by Vermeulen et al., $n_2 < 2n_1$ holds, this clearly shows the amount of probes saved by MDA-Lite. In their paper Vermeulen et al. compare the savings in more detail based on actual numbers by defining example values for all parameters of MDA-Lite [3].

The above steps of MDA-Lite do not reveal all edges of the graph even in unmeshed diamonds. However, the task to obtain the rest of the edges is deterministic rather than stochastic. It will therefore most likely require less probes for high levels of confidence. The three possible situations are handled as follows [3]:

- In case there are more nodes at hop h than at hop $h + 1$, additional probes are generated for each node at hop h and sent to hop $h + 1$. This will find all successors for the nodes at hop h and thus find the missing edges.
- If there are more nodes at hop $h+1$ than at hop h, the probes that have discovered the nodes at hop $h + 1$ will be sent to hop h. This will unmask the missing edges.
- Given that both hops have the same amount of edges, both of the above procedures are applied.

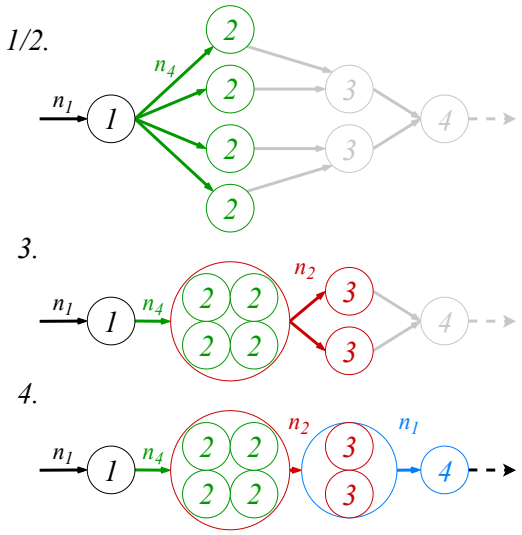

Figure 3: MDA-Lite on Unmeshed Diamond

Assuming there is no meshing, MDA-Lite will perform with much less probes than full MDA. In order to detect meshing MDA-Lite, applies stochastic probing for potential meshing. This includes the need for the costly generation of additional probes. The amount of these additional probes can be set by the user as a parameter introduced by MDA-Lite. For all meaningful values used by Vermeulen et al. the amount of additional generated probes is less than with full MDA [3]. In case meshing is detected, MDA-Lite will switch to full MDA.

2.2. Large Network Topology Discovery

Augustin et al. [10] already came up with the idea of getting a broader view on the network topology. They started using MDA to trace traffic to a network prefix rather than an address. However, the problem of extensive active measurement network traffic gets even more significant when tracing towards a network prefix. When requiring a lot of time to complete the measurement of a network topology, one can encounter changes of the topology, which will bias the result of the measurement. Due to that problem, Beverly introduced an implementation called *Yelling at Random Routers Progressively (Yarrp)* [8]. He determined that the problems of most traceroute implementations, when it comes to larger network topology discovery, to reside in the following:

- **Maintenance of State** for each probe that has not yet received a response: this state usually consists of timing information as well as the probes identifier.
- **Sequential Probing** of the path: this can lead to a significant execution time for large networks. Moreover, sending probes sequentially, hop by hop and node by node, can lead to intrusion detection mechanisms to identify the probe traffic as a security risk.

Yarrp deals with the sequential probing problem by adjusting the order of the probes by randomizing the destination and TTLs of the probes. To achieve pseudo randomness, the target IP addresses, as well as the TTLs, are encrypted

doi: 10.2313/NET-2020-04-1_06

by a keyed block cipher. This projects the input of IP addresses and TTLs onto a new pseudo random order. To eliminate the need to maintain state for each probe, Yarrp encodes all state information into the probe packet. This is done by overloading the TCP header of the probe. For example, the TTL of the original probe gets encoded in the IP identification field [11] and the elapsed time resides in the TCP sequence number. The approach of Yarrp can be beneficial, especially in situations when the measurement time is important [8]. It allows measurement and analysis of short living topologies.

3. Service Oriented Approach

Tracing the path of a specific application's network traffic can be hard using (Paris) traceroute. The application can have multiple flows to different destination hosts. In order to solve this task with Paris traceroute, one would have to create probe packets with flow identifiers that match the flow identifiers of the applications network traffic [12]. But even if the flow identifier matches the applications flow identifier, for TCP connections many routers discard packets that do not belong to an already known open connection [13]. The idea that many service oriented tracing tools apply is that they place their probe packets into an existing flow. There are different implementations available for TCP connections [13]–[15]. However, they all work under the assumption that there is a single connection that is established for each application. A lot of applications fetch their content from multiple sources over different connections. This makes it hard to observe the paths, which the network traffic of an application is taking through the Internet.

For this reason, Morandi et al. [16] proposed an implementation called *Service traceroute*. This tool uses the idea of previous service oriented implementations, but it provides a feature to automatically select all flows to trace based on high level user input and provides the capability to trace them simultaneously. For example, the user can specify to trace all flows related to traffic of the streaming service *Youtube* by using its name as a service specifier. Moreover, the tool also works with UDP.

The execution of Service traceroute is split into two phases:

- The **observation phase** identifies the flows to be traced by observing the specified network interface. The flows are identified by the high level search terms of the user. To achieve this kind of functionality, a database of services signatures was created and is used to identify the services.
- The **path tracing phase** executes the probing on each identified flow. It stops probing as soon as the application closes the TCP connection or after a given timeout for UDP.

The probe packets are constructed from the observed packets of the actual application flow. For TCP packets, Service traceroute uses empty TCP Acknowledgements with the same flow identifier and sequence number as probe packets. In order to match the ICMP response messages to the probes, it uses the IP Identification field [11]. For UDP packets, Service traceroute constructs UDP packets with an empty payload using the same flow identifier as in the original packets.

Service traceroute can be a useful tool, especially when tracing short living flows, such as small web downloads [16]. On the one hand, the probe traffic will result in a higher overhead in these situations. On the other hand, other Tracing tools could be hard to launch in these situations, as they do not provide any reasonable fast way of identifying which flows to trace. As part of their work, Morandi et al. compared Service traceroute to Paris traceroute with and without MDA [16]. For Paris traceroute they used the same flow identifier as the flow identifier of the application traffic. The results show that 7% of the discovered paths by Service traceroute could not be discovered by Paris traceroute. When using MDA to discover all paths, more than 40% of the paths reported by Service traceroute are not found within all paths reported by MDA. The authors explain the reason for Paris traceroute performing worse than Service traceroute with the possibility that some routers drop packets that do not belong to an active flow. This proves the need for tools that inject their probes into active flows.

Morandi et al. also inspected potential interference with the application of the observed flows. They did not find any conflict with the packets being sent by the application.

4. Conclusion and Outlook

As the use cases of traceroute slowly expanded to a broad range of problems, the choice of the right implementation is important. When finding all possible existing paths towards a single destination, no matter what the packet's payload will look like, then MDA and its lightweight version MDA-Lite will be the right choice [3]. As a tool for large network topology discovery Yarrp tries to speed up the execution time. This makes it particularly useful to discover short living network topologies [8]. In case of tracing down a single application, the user might not necessarily be interested in parts of the network that are not taken by the applications network traffic. In this case the use of service oriented approaches provides the best results [16].

Besides the approaches discussed in this article, there exist many more solutions. Most traceroute-based solutions do not investigate the use of the *Record Route* field of the IP header [17]. This field is actually meant to allow tracing the paths of network traffic. Each router along the path will include its address in a list inside the IP header. *Tcpsidecar* is a tool integrating the Record Route field [14]. Unfortunately, the amount of hops being captured by the Record Route field is limited and most routers do not adhere to its specification. Moreover, some firewalls even block packets that have the record route option specified.

Instead of actually improving the core behavior of traceroute, there are tools available which try to focus on filtering out the wrong paths reported by traceroute [18], [19].

There are also a few attempts to deploy active measurement infrastructure to deal with the problem of measuring the Internet. One of the most famous ones is *RIPE Atlas* [20]. Relying on global infrastructure brings some

doi: 10.2313/NET-2020-04-1_06

benefits as well as some other challenges, which are not discussed as part of this paper.

All in all, traceroute remains a useful tool for ad-hoc measurements. There are a few issues with the traditional approach that can be mitigated by choosing the right improvement.

References

[1] *MAN(8) Traceroute for Linux*, Linux 5.4, 2019, accessed Dec 09 2019. [Online]. Available: http://man7.org/linux/man-pages/man8/traceroute.8.html

[2] D. Malone and M. Luckie, "Analysis of ICMP Quotations," in *Passive and Active Network Measurement*, S. Uhlig, K. Papagiannaki, and O. Bonaventure, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 228–232.

[3] K. Vermeulen, S. D. Strowes, O. Fourmaux, and T. Friedman, "Multilevel MDA-Lite Paris Traceroute," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: ACM, 2018, pp. 29–42.

[4] Juniper, "Configuring Per-Packet Load Balancing," 2018, accessed Dec 09 2019. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/policy-configuring-per-packet-load-balancing.html

[5] Cisco, "How Does Load Balancing Work?" 2015, accessed Dec 09 2019. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5212-46.html

[6] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 153–158.

[7] D. Veitch, B. Augustin, R. Teixeira, and T. Friedman, "Failure Control in Multipath Route Tracing," in *IEEE INFOCOM 2009*, April 2009, pp. 1395–1403.

[8] R. Beverly, "Yarrp'Ing the Internet: Randomized High-Speed Active Topology Discovery," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: ACM, 2016, pp. 413–420.

[9] B. Augustin, T. Friedman, and R. Teixeira, "Multipath tracing with Paris traceroute," in *2007 Workshop on End-to-End Monitoring Techniques and Services*, May 2007, pp. 1–8.

[10] B. Augustin, T. Friedman, and R. Teixeira, "Measuring Load-balanced Paths in the Internet," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 149–160.

[11] J. Touch, "Updated Specification of the IPv4 ID Field," Internet Requests for Comments, RFC 6864, February 2013, accessed Dec 09 2019. [Online]. Available: https://tools.ietf.org/html/rfc6864

[12] M. Luckie, Y. Hyun, and B. Huffaker, "Traceroute Probe Method and Forward IP Path Inference," in *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '08. New York, NY, USA: ACM, 2008, pp. 311–324.

[13] J. Edge, "Tracing behind the firewall," *LWN.net*, Jan 2007, accessed Dec 09 2019. [Online]. Available: https://lwn.net/Articles/217076/

[14] R. Sherwood and N. Spring, "Touring the Internet in a TCP Sidecar," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 339–344.

[15] D. Kaminsky, *Paratrace Man Page*, Linux 5.4, 2019, accessed Dec 09 2019. [Online]. Available: https://man.cx/paratrace

[16] I. Morandi, F. Bronzino, F. Bronzino, and S. Sundaresan, "Service Traceroute: Tracing Paths of Application Flows," in *Passive and Active Measurement*, D. Choffnes and M. Barcellos, Eds. Cham: Springer International Publishing, 2019, pp. 116–128.

[17] J. Postel, "Internet Protocol," Internet Requests for Comments, RFC 791, September 1981, accessed Dec 09 2019. [Online]. Available: https://tools.ietf.org/html/rfc791

[18] A. Marder and J. M. Smith, "MAP-IT: Multipass Accurate Passive Inferences from Traceroute," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: ACM, 2016, pp. 397–411.

[19] N. Brownlee, "On Searching for Patterns in Traceroute Responses," in *Passive and Active Measurement*, M. Faloutsos and A. Kuzmanovic, Eds. Cham: Springer International Publishing, 2014, pp. 67–76.

[20] "Ripe Atlas," RIPE Network Coordination Centre, accessed Dec 09 2019. [Online]. Available: https://atlas.ripe.net/