Implementation and Analysis of Modern Traceroute

by

Hamish Hartley

In partial fulfilment of the
requirements for the degree of
MSc
in
Advanced Computer Science

Department of
Computer and Information Sciences

March, 2024

Contents

1 Introduction and Literature Review

1.1 Introduction

The importance and impact of the internet over the past 30 years cannot be overstated; in almost every area it has dramatically changed the efficiency with which data can be transmitted and received across the globe. In order to facilitate such a large system, many millions of nodes are interlinked, all working together in tandem to send and receive ”packets” of data.

Network topology is defined as the physical and logical layout of nodes and connections in a network, with nodes usually being either a router, switch or software emulating a switch or router.

Being able to map the topology of this structure is key not only to establish the most efficient routes for data transmission by reducing latency but also to ensure even traffic distribution across different network paths via Load balancing. There are also additional advantages of accurate mapping, which include mitigating potential security threats, fault detection and scalability. A common tool used to achieve this is Traceroute, with it being widely used, from the diagnosis of network problems to the assemblage of internet maps. [1]

1.2 Literature Review

Traceroute reports an IP address for each network-layer device along the path from a source to a destination host in an IP network[2]. However, traceroute fails in the presence of routes that employ load balancing on packet header fields. The failures lead to incorrect route inferences that may mislead operators during problem diagnosis and result in erroneous internet maps.[1][3]

Jacobson’s traceroute, released in 1989 was the *de facto* tool used for network diagnostics and internet topology mapping, however as stated above it cannot properly map routes which use load balancing for packet header fields. This severely impairs it’s ability and overall feasibility as load balancing is widely used.

Load balancing is used to increase dependability and enhance resource usage. With intra-domain routing protocols OSPF[4] and IS-IS[5] being used to implement this. Routers can spread their traffic across multiple equal-cost paths using a per-packet, per-flow or per-destination policy. [6][7] For per-flow balancing packet header information ascribes each packet to a flow, and the router forwards all packets belonging to a same flow to the same interface. A natural flow identifier is the classic five-tuple of fields from the IP header and either the TCP or UDP headers: Source Address, Destination Address, Protocol, Source Port, and Destination Port. Per-flow load balancing ensures that packets from the same flow are delivered in order. Per-packet load balanc- ing makes no attempt to keep packets from the same flow together, and focuses purely on maintaining an even load. Per-destination load balancing could be seen as a coarse form of per-flow load balancing, as it directs packets based upon the destination IP address. But, as it disregards source in-153 formation, there is no notion of a flow per se.[1]

When load balancing in employed there is no single route from a source to a destination, for example with per-load packet balancing any given packet could go any one of a selection of possible routes. Although, with per-flow load balancing a single route is still used for packets of a given flow, however even in this case differing flows for the same source and destination pair can follow different possible routes. As stated above, the original traceroute is insufficient to correctly recognize individual routes from a group of routes. This is due to how traceroute traverses routes in a network; it discovers hops along a route with a series of probe packets with increasing TTL, with each node in the network sending back an ICMP packet back in response. If these probe packets reach a router using load-balancing there is a possibility of the probe packets being directed along different paths, potentially resulting in missing nodes/links and false links as shown below in fig. 1.

Due to the large prevalence of routers using Load balancing this is highly problematic, in response to this, new approaches have been proposed to mitigate the issue's highlighted above. Paris traceroute, a new traceroute designed for networks with load balancing routers. Its key innovation is to control the probe packet header fields in a manner that allows all probes towards a destination to follow the same path in the

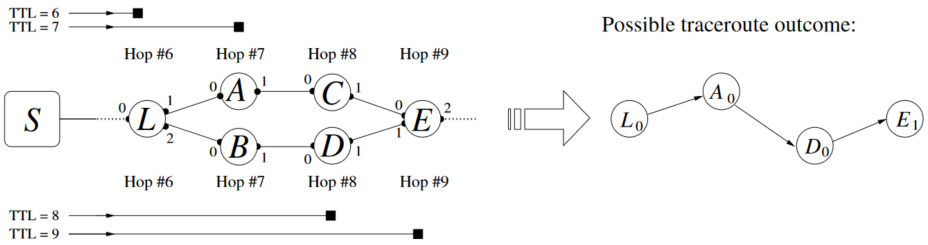

Figure 1: Missing nodes and links, and false links [1]

presence of per-flow load balancing. It also allows a user to distinguish between the presence of per-flow load balancing and per-packet load balancing. Unfortunately, due to the random nature of per-packet load balancing, Paris traceroute cannot perfectly enumerate all paths in all situations. But it can do considerably better than the classic traceroute, and it can flag those instances where there are doubts. [1]

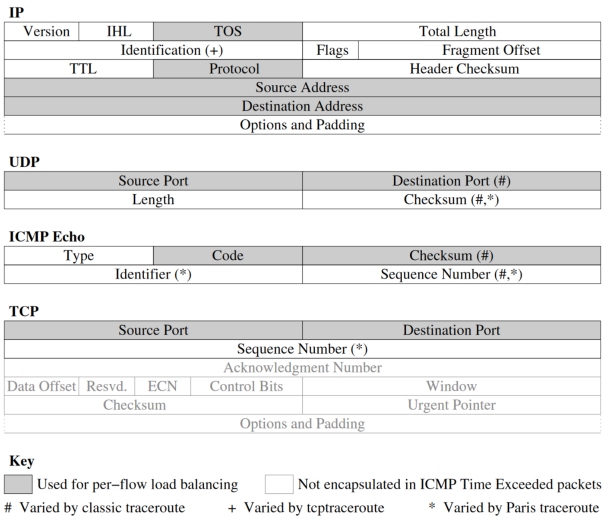

Figure 2: The roles played by packet header fields [1]

Paris traceroute controls the flow of each packet allowing them to follow the same path by varying the first eight octets of the transport-layer header, but that are not used for load balancing. For UDP probes, Paris traceroute varies the Checksum field. This requires manipulating the payload to yield the desired value, as packets with an incorrect checksum are liable to be discarded. For ICMP Echo probes, Paris traceroute varies the Sequence Number field, as does classic traceroute, but also varies the Identifier field, so as to keep constant the value for the Checksum field. Paris traceroute also

sends TCP probes, unlike classic traceroute, but like Toren's variant tcptraceroute. [1][8] Paris traceroute is a much more robust tool to map internet topology than traceroute as it can mitigate the effects of per-flow load balancing and ensure packets flow on the correct path through the network nodes, avoiding potential missing nodes/links and false links. However, nodes which use Network Address Translation(NAT) can still be problematic for paris traceroute due to the NAT modifying the source/destination IP addresses in the packet headers, obscuring the packet information and also therefore the actual routing path.

The Multipath Detection Algorithm (MDA) provides an extension to Paris traceroute called the MDA, a stochastic probing algorithm that adapts the number of probes to send on a hop-by-hop basis in order to enumerate all reachable interfaces at each hop. The number of probes required by the MDA is a function of a tunable parameter, which is an upper bound on the probability of failing to discover the entire explorable multipath route, or multipath. Upon completion, the MDA yields a level of confidence in its result. The MDA also has mechanisms to deal with unresponsive routers and for identifying per-packet load balancers and routing changes.[9] The MDA works on the basis of an open set of vertices, each of which has been discovered but has not yet had its successor vertices identified. A discovery round consists in choosing a vertex v from the open set and trying to find all of its successors. Where there is no load balancing, v has just one successor, but if v is the responding interface of a load balancing router, there will be two or more possible successors that can only be identified by stochastic probing. In the case that concerns us, per-flow load balancing, successors are found by varying the flow identifier from one probe packet to the next. [10][11][12] Rather than send a set amount of probes per hop like the original traceroute which sends 3, the MDA varies this amount based on a controllable parameter. The parameter is the upper limit of the probability of not revealing the entire multi-path network. This is highly advantageous as it can adapt to any given network structure, which in the "real" internet can be extremely complex and have many paths, but also due to it's adaptability it can still remain relatively lightweight. However, as it is a stochastic algorithm there is an element of uncertainty to the produced result, this is accounted for by "yeilding" the confidence of the final output to provide some further information

about the validity of the result.

The MDA-Lite, however, reserves node control for particular cases and proceeds hop by hop in the general case. At each hop it seeks to discover all of the vertices at that hop, and in doing so discovers some portion of the edges between that hop and the prior hop. It then seeks out the remaining edges. It operates on the assumption that the diamonds that it encounters will be uniform and unmeshed. If this assumption holds, hop-by-hop probing will maintain the MDA's failure probability bounds. Because these two topology assumptions might not hold, the MDA-Lite tests for a lack of uniformity and the presence of meshing using methods that are less costly than full application of the MDA. When it detects a diamond that does not adhere to one of the assumptions, it switches to the MDA.[10] MDA-lite is a further improvement on the MDA algorithm by reducing overhead, whilst still having the same performance. It achieves this by operating based on the assumption of uniform hops, with probes being sent at each hop without node control. One flow identifier from each previously discovered vertex in the preceding hop are reused, continuing with additional previously-used flow identifiers and then new ones. It applies the MDA's stopping rule to remain within the MDA's failure probability bounds for vertex detection. [10]

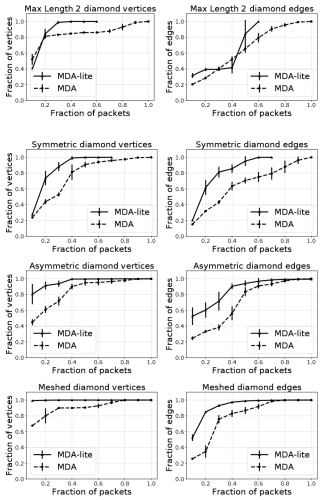

Figure 3: MDA-Lite versus MDA simulations[10]

D-Miner is designed to capture Internet topology snapshots inclusive of all load-balanced paths. At its heart, D-Miner uses Yarrp's randomized and stateless probing

to achieve high probing rates. To this, it adds probe set generation logic that keeps track, on a per-node basis, of whether all outbound load- balanced edges have been discovered with high probability. The logic guides Yarrp through multiple rounds until the full discovery criterion has been satisfied for almost all nodes. [11]

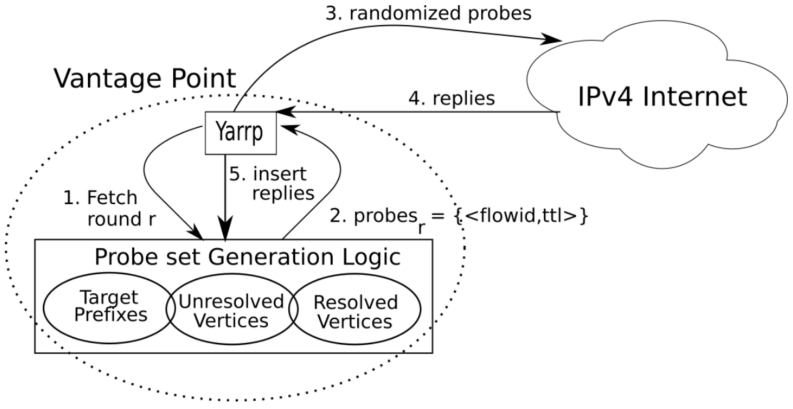

Figure 4: Diamond miner high-level conceptual overview [11]

Due to Diamond miner's utilisation of Yarrp's randomized and stateless probing, wide coverage of internet topology is achieved. It's incorporation of probe set generation logic allows tracking of all outbound load-balanced edges from each node with high certainty. D- Miner permits discovery of the Internet's multipath topology in 2.5 days when probing at 100kpps. This high speed allows us to characterize and quantify dynamic behaviors of the In- ternet induced by load balancing. Finally, D-Miner enables for the first time an Internet Scale survey of load balancing that shows its widespread prevalence, both in the core and at the edge. [11]

The high-level idea of Yarrp is: i) randomization of the probing order of the domain of network range(s) and TTLs; and ii) stateless operation, whereby all necessary state is encoded into the probes such that it can be recovered from the ICMP replies.[13]

Yarrp enables high speed collection of network topology data, due to it's stateless operation and randomization of the probing order of the domain of network ranges and TTL of ICMP packets sent.

To detect NATs, Dublin Traceroute forges a custom IP ID in the outgoing packets, and keeps track of them in the response packets. If the response packet references an

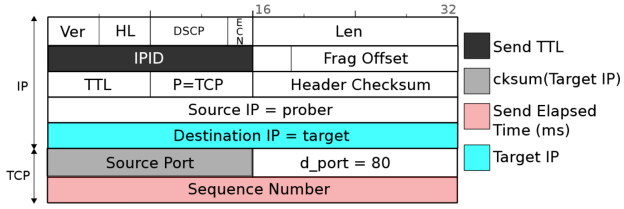

Figure 5: Yarrp encodes information in the IP and TCP field of outgoing proe packets in order to permit stateless operation [13]

outgoing packet with different source/destination IP addresses and ports, this may indicate the presence of a NAT. In that case, the packet referenced in the response will be different from the one that was sent, and cannot be correlated anymore. However, the IP ID is expected to be unchanged (thanks to the presence of the don't-fragment bit), and it will contain a value to correlate it to one of the outgoing packets.[14] [15] Dublin traceroute solves the NAT problem which was a limitation of paris-traceroute, it is also implmeneted in C++ for it's core and has also improved on further limitations of paris traceroute such as memory safety increasing it's robustness. Furthermore, it also offers a python library and interface as a wrapper on top of the C++ codebase. Results are exported as JSON data allowing ease of integration within other applications, adding to it's modularity and usability.

Zeph proceeds across a series of cycles, with the directives for each cycle being based in large part upon the success of the directives that were used in the previous cycle. New directives are also tried out each cycle, with the overall aim being to improve the completeness of coverage over successive cycles. In reinforcement learning terms, the reuse of directives is exploitation and the trying out of new directives is exploration. Zeph's challenge is to best use the probing budgets of its agents, i.e., choose which directives to issue to each agent so as to obtain the overall most complete route trace picture possible within a cycle. There are many ways of conceiving of "completeness", and the one adopted here is to maximize coverage of the traceroute- style directed links that are available to be discovered, a directed link consisting in an ordered pair of IP addresses.[16]

By applying reinforcement learning for distributed tracing of network routes, zeph discovers the probing directives to allocate to the vantages points in order to maximize

topology discovery.

1.3 Possible work

Based on the studied literature above there are several avenues of research which could be carried out to provide potentially fruitful results. Such as expanding upon the existing dublin traceroute implementation to include features such as the Multipath Detection Algorithm and zeph algorithm, and also support for TCP, ICMP and DNS probes in order to create the optimal traceroute tool which improves upon the previous tools such as traceroute and paris traceroute. With these improvements implemented, there could be scope to conduct an internet-wise topology survey as additional work for the project to provide supporting evidence of the additional features merit.

Furthermore, the MDA and zeph alogrithms could also be extended to IPv6 as their existing implementations only support IPv4. However this could prove to be time-comsuming due to IPv6's address size of 128-bits.

2 Research aim, objectives and research questions

2.1 Research aim

The aim of this research proposal is to extend the functionality of the existing dublin traceroute tool in Go programming language.

2.2 Objectives

1. Implement the state-of-the-art algorithms available, namely MDA-lite and Zeph algorithm.
2. Add support for TCP, ICMP and DNS probes.
3. Extend the functionality of the MDA and zeph alogrithms to also use IPv6.
4. Evaluate the performance of the additional features added to dublin traceroute.

2.3 Research questions

1. How effective is the improved dublin traceroute compared with the original implementation?
2. What are the limitations of MDA-lite?
3. What are the limitations of Zeph algortihm
4. What benefit does extending the functionality of MDA-lite and Zeph to ipv6 with regards to available search space?
5. What are the limitations of traceroute?
6. What further improvements could be made?
7. What novel contexts/use-cases could such a tool be implemented?

3 Research Methods

3.1 Research methodology

The research methodology employed for the purpose of this research will be purely Quantitative. The design parameters for the implemented model are discrete, e.g. number of probes sent per hop. The measured data will be both discrete and continuous. The implemented software will be written in the Go language.

3.2 Research methods

The precision, recall, and numbers of probes sent using the improved dublin traceroute will be used as the basis of establishing model performance. They will be evaluated using ContainerLab [17] which is an open source network emulator which allows the creation of virtual network environments, it supports containerized router images of products offers by major companies such as Cicso, Juniper and Dell. This avoids ethical implications of evaluating performance on the real-world internet.

3.3 Data analysis

Analysis of the implemented model will be carried out using several metrics in order to illustrate comparisons between it's performance and the original model's performance. Several plot types such as scatter plots, line plots and violin plots will be used to visualize; negative and positive correlations and also to represent the standard deviation of results respectively. Furthermore, tables of data will also be used to provide accurate numerical data in a straightforward manner.

4 Research plan and deliverables

4.1 Research plan

Week 1	Become familiar with Dublin traceroute codebase, further exploration of literature
week 2	Conduct benchmark in ContainerLab of original dublin traceroute
week 3	Implement MDA algorithm
week 4	Testing and refactoring codebase
week 5	Implement Zeph algorithm
week 6	Testing and refactoring codebase
week 7	Extend for TCP, ICMP and DNS probes
week 8	Explore extending to IPv6
week 9	Final testing
weeks 10-12	Finalize thesis

4.2 Risk analysis

4.2.1 Existing Codebase

Unfamiliarity of the dublin traceroute codebase could lead to unexpected errors/delays in the development timeline of the project. This will be mitigated by allocating the first week of the project to read through the existing documentation and codebase. Also potentially reaching out to the origional developer for a meeting to discuss any concerns.

4.2.2 Virtual testing evironment

ContainerLab could potentially have compatibility issues with the existing dublin traceroute tool, which would negatively impact the ability to properly evaulate the inital and final model's performance. Inital benchmarking will be carrierd out early on in the project so as to avoid these issues, with alternative enviroments being an option if required.

4.2.3 Implementaiton of MDA and Zeph algorithms

Implmenation of the MDA and zeph algorithms are likely to take up the bulk of the given time for the project due to their complex nature. To avoid running out of time the implementation of both will be done separately and incrementally. Additionally time has been alloted in the middle of the project to conduct unit testing and refactoring to ensure error free code.

4.2.4 Extended requirements

The extended deliverables of adding TCP, ICMP and DNS probes and also IPv6 extension have been placed as a lower priority due to them potentially impacting the primary focus of this research to implement MDA and zeph alogrithms with dublin traceroute.

4.2.5 Final evaluation

Continual testing of the implemented model will be carried out from the start of development to avoid prior mentioned compatability issues with the virtual network environment.

4.3 Ethical considerations

The potential ethical concerns of sending out packets and scanning network topologies without prior owner's concent has been avoided by using a virtual network environment. However, if completed and publicly available, the implemented tool could also be used in a potentially malicious way to aid cyber crime. Further work is needed to determine how best to mitigate this potentially unsolvable issue.

4.4 Deliverables

1. Implementation of MDA/MDA-lite algorithm in dublin traceroute codebase
2. Implementation of zeph algorithm in dublin traceroute codebase
3. Extension for TCP probe
4. Extension for DNS probe
5. Extension for ICMP probe
6. Extension for IPv6
7. Evaluation of perfomance of implemented improved

References

[1] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, page 153–158, New York, NY, USA, 2006. Association for Computing Machinery.

[2] V Jacobson. Traceroute ftp://ftp. ee. lbl. gov/traceroute. tar. gz, 1989.

[3] Brice Augustin, Timur Friedman, and Renata Teixeira. Exhaustive path tracing with paris traceroute. In *Proceedings of the 2006 ACM CoNEXT Conference*, CoNEXT '06, New York, NY, USA, 2006. Association for Computing Machinery.

[4] John Moy. Ospf version 2. rfc2328, april 1998. *URL: http://tools. ietf. org/html/rfc2328*.

[5] Ross Callon. Use of osi is-is for routing in tcp/ip and dual environments. Technical report, 1990.

[6] Cisco. How does load balancing work? Available at `http://www.cisco.com/en/US/tech/tk365/technologiestechnote09186a0080094820.shtml`.

[7] Juniper. Configuring load-balance per-packet action. Available at `http://www.juniper.net/techpubs/software/junos/junos70/swconfig70-policy/html/policy-actions-config11.html`.

[8] M. Toren. tcptraceroute. Available at `http://michael.toren.net/code/tcptraceroute/`.

[9] D. Veitch, B. Augustin, R. Teixeira, and T. Friedman. Failure control in multipath route tracing. In *IEEE INFOCOM 2009*, pages 1395–1403, 2009.

[10] Kevin Vermeulen, Stephen D. Strowes, Olivier Fourmaux, and Timur Friedman. Multilevel mda-lite paris traceroute. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 29–42, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Kevin Vermeulen, Justin P Rohrer, Robert Beverly, Olivier Fourmaux, and Timur Friedman. {Diamond-Miner}: Comprehensive discovery of the internet's topology diamonds. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 479–493, 2020.

[12] Julius Niedworok, Johannes Zirngibl, and Patrick Sattler. Modern traceroute variants and adaptations. *network*, 4:5.

[13] Robert Beverly. Yarrp'ing the internet: Randomized high-speed active topology discovery. In *Proceedings of the 2016 Internet Measurement Conference*, pages 413–420, 2016.

[14] Andrea Barberio. Dublin-traceroute. Available at `https://dublin-traceroute.net/`.

[15] Andrea Barbeiro. Visualizing multipath networks with dublin traceroute. *Presentation at the MOCA Italian hacker camp*, 2016.

[16] Matthieu Gouel, Kevin Vermeulen, Maxime Mouchet, Justin P. Rohrer, Olivier Fourmaux, and Timur Friedman. Zeph & iris map the internet: A resilient reinforcement learning approach to distributed ip route tracing. *SIGCOMM Comput. Commun. Rev.*, 52(1):2–9, mar 2022.

[17] Containerlab Team. Containerlab. https://containerlab.dev/.