Streamlining traceroute by estimating path lengths

Tim Moors
School of Electrical Engineering and Telecommunications
University of New South Wales
Sydney, Australia
t.moors@unsw.edu.au

*Abstract***— Traditional traceroute determines the path through a network by sending probe packets with progressively increasing TTL/hop count values so that routers that are progressively further from the inquirer send ICMP Time Exceeded messages and so reveal their identity. This process can be slow (because traceroute must wait for a timeout or response to one probe before sending the next) and inefficient (through repeated probing of routers near the inquirer that lie on the intersection of paths leading to multiple targets). This paper shows how this process can be streamlined by the inquirer sending a scout packet to the (reachable) target before sending route tracing probes. The inquirer uses the TTL of the response to this scout packet to estimate the length of the path to the target (with tolerance for path asymmetry), and can then either expedite the route tracing process (by sending probes to each of the estimated number of routers on the path in quick succession) or reduce the number of probes needed (by inverting the direction of traditional traceroute, tracing the path from the target towards the inquirer, and terminating the process when it reaches a router on a known path from the inquirer).**

traceroute, raceroute, aceroute, network diagnostics, TTL

I. INTRODUCTION

Two indicators of the outstanding success of the Internet over the past decade are the increasing penetration of its use in the broad community, not just by technologists, and that these users are becoming increasingly reliant on the Internet as a communication medium. However, these users face a dilemma: They have an intense demand for resources offered on the Internet, but are often frustrated by their inability to access these resources. For example, recent measurements [1] suggest that prominent web servers are available to end-users 93% of the time; a far cry from the 'five nines' (99.999%) availability expected of telephone network nodes [2]. In this context, it can be useful for end-users to have access to software that can monitor the characteristics of paths (e.g. route and link delays) leading to targets of interest, including localising where the path breaks during network outages, and in collecting baseline measurements of path characteristics during normal operation. When such measurements are made routinely as part of normal operation, and hence run frequently by many users, it is essential that they be efficient.

One of the most popular tools for tracing the path from an inquirer towards a target is traceroute. Traditional traceroute [3] determines the path through a network by sending probe packets with progressively increasing values of the TTL field (for IPv4) or hop count field (for IPv6) so that routers that are progressively further from the inquirer send ICMP Time Exceeded messages and so reveal their identity. traceroute progressively increases the TTL because it does not know the distance to the target before it starts probing, and since paths average 15 hops in length [4] which is much shorter than the maximum of 255 hops permitted by the Internet Protocols it would be inefficient to immediately send probes to all possible hops. Variants of the traditional traceroute include an AS level traceroute [5] and tulip [6] which exploits the fact that many routers use a counter to set the IP Identification field, allowing end systems to determine which routers on the path cause mis-sequencing and loss.

Members of the North American Network Operators' Group (NANOG) have created a variant (known as "tracesroute") [ftp://ftp.login.com/pub/software/traceroute] that allows abortion of the route tracing after a specified number of hops have failed to respond, and "parallel probing" in a "spray mode" in which a set of probes is sent before waiting for feedback. While this paper also proposes allowing multiple probes to be waiting for responses, it improves on tracesroute by providing an algorithm to estimate *how many* probes should be included in the set.

Traditional traceroute can be slow because it incrementally learns the path length: It sends a probe a certain distance into the network and then waits for a timeout or response to this probe before deciding whether to send the next probe one hop further into the network. This slowness is particularly pronounced when traceroute times out waiting for a response, e.g. because a router did not send a Time Exceeded message or because this message was lost as it propagated towards the inquirer. This slowness wastes the time of human users of traceroute, and can also lead to confused traceroute output when the path changes during the trace (and the likelihood of this increases with the trace time): some responses will reflect one path to the target and other responses will reflect another path to the target.

To quantify the time that it takes to trace a path, consider a target that is a distance D hops away from the inquirer, and assume that each hop contributes equally to the round-trip time to the target, and that each router responds as soon as it receives the probe. With traditional traceroute, D probes must be sent, and each covers an average distance of $D/2$, making the time to trace the route of the order of $O(D^2)$. This can be improved by using a binary search for the path length [7], reducing the time to trace the route to $O(D\log_2 D)$. However, if the route tracing could estimate the number of hops before it starts probing, then it could send all probes immediately and the trace would complete as soon as the inquirer receives a response from the most deeply penetrating probe, taking a time of the order of $O(D)$. With path lengths averaging around 15 hops for typical paths [4], reducing the trace time from $O(D^2)$ to $O(D)$ would constitute a considerable improvement.

This paper shows how route tracing can be streamlined by the inquirer sending a scout packet to the (reachable) target before sending tracing probes. The technique is only applicable when the target responds to scout packets (the tool reverts to the traditional incremental learning approach when there is a timeout waiting for such a response) and so is most applicable for baseline measurements of paths to targets that are reachable, rather than for localising faults when the target is disconnected. The inquirer uses the TTL of the response to this scout packet to estimate the length of the path to the target, and can then use this to expedite the route tracing process by sending probes to each of the estimated number of routers on the path in quick succession. To reflect the speed and traceroute heritage of this technique, we name the tool that implements this technique "raceroute". Software that implements the raceroute tool is available online at http://www.ee.unsw.edu.au/~timm/raceroute .

This paper also shows how prior knowledge of an estimate of the path length can be used to reduce the number of probes needed to trace a route when the trace process has memory of previous trace results. This is done by inverting the direction of traditional traceroute, tracing the path from the target towards the inquirer, and terminating the process when it reaches a router on a known path from the inquirer. To reflect the fact that this technique probes most routers only once, despite tracing multiple paths that may share routers, we name the tool that implements this technique is called "aceroute". Figure 1 graphically depicts the different approaches to route tracing.

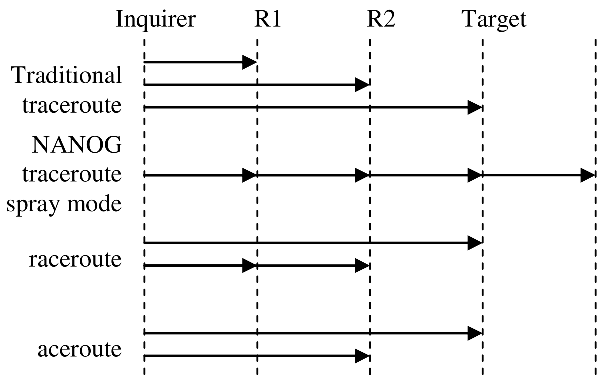

Figure 1. Various route tracing approaches take different durations (vertical dimension) when sending probes (arrows) from an inquirer, through multiple routers, to a target.

Because raceroute and aceroute require feedback from the target in order to work, they are not intended for the most blatant form of troubleshooting: determination of where connectivity is broken when the target is unreachable. However, they are intended for efficient measurement when the target *is* reachable: determination of where delays are occurring, and baseline collection of information about paths traversed, so that when the target is unreachable, the inquirer knows which network elements are likely to be unavailable and so can report the fault and know what to monitor to determine when service returns.

This paper first examines (§ II) what form scout packets should take in order to elicit a response from the target with high probability and assesses the accuracy of path length estimates made based on the TTL of such a response. It then examines (§ III) techniques for matching responses to probes when the inquirer sends large numbers of probes towards a target in rapid succession. This section shows the benefits of using source port numbers for such matching, and calls for a route tracing port in the well-known port range to prevent route tracing probes from being absorbed by processes on end systems. Section IV addresses some of the complications in tracing routes when using TTL fields of responses to estimate path lengths (e.g. arising from path asymmetry and TTL resetting) and the costs of basing probing on such estimates. Finally, § V describes the idea of inverting the traceroute process, tracing the path from routers that are furthest from the inquirer to those that are closest, in order to reduce the number of probes needed.

II. Scout packets

The purpose of scout packets is to elicit from the target a response from which the length of the path between inquirer and target can be estimated. The response can take three forms: It may be an ICMP error message, or a higher layer response to either a packet that the inquirer is naturally sending to the target (e.g. an ACK to a TCP SYN sent to a port that provides a service that the inquirer needs to use) or an artificial packet that exists only for the purpose of eliciting such a response.

ICMP responses have the advantage of explicitly including the TTL of the scout packet when it arrived at the target, and hence accurately indicate the length of the path from inquirer to target. However, it can often be difficult to elicit an ICMP response from the target: ICMP messages may be blocked by firewalls, and to generate an ICMP port unreachable message requires a scout that is destined to an unused port, meaning that the scout introduces the overhead of an artificial packet and this artificial packet is less likely to be permitted to traverse firewalls on the path to the target. To maximise the chance of scouts penetrating firewalls, raceroute sends them to destination ports that identify services that external clients may have a legitimate need to access, e.g. ports 25 (SMTP), 53 (DNS), 80 (HTTP, although that can be cached as described below) and 443 (HTTP over TLS).

When the inquirer is already receiving responses from the target (e.g. for the intended application of monitoring performance of paths leading to information services used by an end-user) then those responses may be able to provide the required information without the need to send a separate scout packet. However, the most popular information service today is web access, and while transparent web caches can improve the performance of this service, they can also intercept packets as they propagate from origin servers to clients so that they can be cached, often by inserting themselves in the path from server to client, and so resetting the TTL when they forward the packet to the client, and so hiding the effect on the TTL of routers on the server side of the cache.

Since we are primarily interested in tracing the path to web servers, and because many domains have separate servers for SMTP and DNS traffic, we are limited to ports 80 and 443, with port 443 being preferred in order to work around caches. Scout packets should be sent with the TTL used for normal communication with the target (e.g. 64 or 255 as described below) so that they will likely reach the target if normal communication could also reach the target. A scout message that fully resembles the first packet of a natural exchange (e.g. a TCP SYN) will often establish state information in intermediate and end systems. This is a mixed blessing: It has the advantage that artificial scouts (e.g. an unsolicited TCP ACK) may be discarded by stateful firewalls or address translators, even when more natural traffic would have passed to the target. However, there is the disadvantage of having to "tidy up" and release the state information that is created in response to an apparently natural scout. In our measurements of 100 popular web sites [8], we found that after sending a SYN probe to port 443, 76% of servers responded: 43% with an ACK, 30% with a RST and 3% with an ICMP administratively prohibited message. In contrast, only 59% responded to an unsolicited ACK (all but one with a RST), and only 38% responded to traditional traceroute UDP probes. While these figures are biased by 43% of servers which support HTTPS, and for which SYN segments are clearly legitimate whereas unsolicited ACKs and UDP probes are questionable, they also demonstrate that the more natural a probe, the higher the chance that it will elicit a response.

raceroute takes the approach of using TCP SYN segments as scouts (so that they are just as likely as real traffic to reach the target), and sending a TCP FIN segment after the probing to clean up. If the scout elicits an ICMP unreachable response from the target, then the TTL of the reflected datagram is used to estimate the length of the path to the destination, otherwise it uses the TTL of the response itself to estimate the path length, as described below.

While the inquirer can read the TTL of the response (though implementing this requires access to raw sockets since the sockets API only allows setting the TTL of outgoing packets, not reading the received TTL), it cannot be certain of the initial setting of the TTL field of the response when it was transmitted by the target. RFC 1700 states that "The current recommended default time to live (TTL) for the Internet Protocol (IP) is … 64" although many systems use a TTL of 255 for sending normal traffic (Microsoft Windows systems) or for sending ICMP error messages. Other values also get used, e.g. BSD Unix systems sent new packets with a TTL of 30 (4.3 BSD) or 15 (4.2 BSD) and systems before the 4.3BSD Tahoe release set the TTL of an outgoing ICMP error message to equal the TTL of the incoming packet that caused that error, i.e. providing "TTL reflecting".

By adding the forward path length (determined using traceroute) to the TTL of scout responses, we can estimate the TTL being used by remote servers. When testing popular web services [8], we found this sum was around 64 for about half of the servers, and around 255 for one quarter of the servers, and around 128 for the remaining quarter of the servers. The sums were distributed "around" these common values presumably because of path asymmetries (§ IV). Figure 1 shows this distribution for the 33 popular servers that responded to both traceroute and TCP SYN probes sent to port 443. Thus, we can estimate the path length by measuring the difference between the TTL of a scout response and the next higher "magic number" (64, 128 or 255).

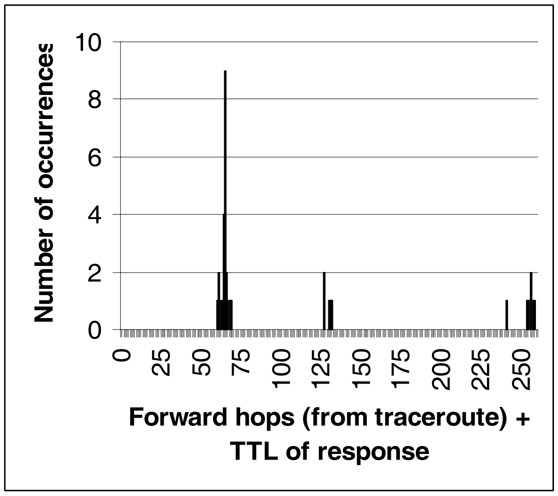

Figure 2. The sum of the forward path length (measured using traceroute) and the TTL of a response from the target is clustered around commonly used host TTL settings: 22 around 64 (9 of 65), 5 around 128, 6 around 255. Thus, the forward path length can be accurately estimated from the TTL of a response from the target.

III. MATCHING RESPONSES TO PROBES

Once the inquirer has an estimate of the path length, it can proceed to send probes to each of the multiple hops along that path. When an inquiring host receives a response to a tracing probe, it must first match the response to the route tracing process that generated the probe, since multiple route tracing processes may be running on an inquiring host. Then, that route tracing process must match the response to the probe that generated the response, so that it can infer how far the probe travelled, and to calculate the round-trip delay. When the response is an ICMP Time Exceeded (or Destination Unreachable) message, the only information available for this matching is the IP header of the probe and the bytes of IP data that the ICMP error message returns. RFC 1122 [9] states that error messages must return "at least the first 8 data octets of the datagram that triggered the error; more than 8 octets MAY be sent" and RFC 1812 [10] states that they "SHOULD contain as much of the original datagram as possible without the length of the ICMP datagram exceeding 576 bytes.". In practice, we found that the vast majority of routers that we tested (92% = 658/719) only returned 8 bytes of IP data in Time Exceeded messages. This forces the matching process to use information conveyed in the IP header or the first 8 bytes of higher layer protocol headers. We will concentrate on the 8 bytes of higher layer protocol headers: IP options could be used but may force probes along different paths than normal traffic would encounter. The Fragment Identifier field of the IPv4 header, or Flow Identifier field of the IPv6 header, could also be used, although that introduces complexity in dealing with different network layers. Finally, the IP length field could be used, but doing so could waste bandwidth, and sometimes it is useful to be able to vary probe lengths in order to measure link bandwidth (as is done by pathchar-like tools [11]).

ICMP echo requests contain two 16b fields that are well suited to such matching: an Identifier field (used to match to the appropriate route tracing process) and a Sequence field (used to match a response to a probe). However, many systems do not respond to echo requests, either for security reasons or because of an excessively literal interpretation of the original RFC 792 ICMP specification [12] that "no ICMP messages are sent about ICMP messages" (which was referring to the sending of ICMP error messages, not ICMP echo responses). This is the reason why the original LBL traceroute used UDP probes [13] and why the probes for most other variants of traceroute use transport protocol headers. While information in the second 32b word of the transport protocol header (i.e. the TCP sequence number, the SCTP verification number, or the UDP length and checksum) could be used for matching, doing so creates complexity in processing different fields for different transport protocols. Thus most traceroute programs use port numbers in the first 32b word for the matching process. In particular, the traditional traceroute uses the source port to match a response to the appropriate traceroute process, and the destination port to match a response to the appropriate probe.

Traditional traceroute's use of the port space can lead to two types of conflicts. Considering destination ports first, traceroute expects that no process on the target is bound to the destination port of a probe, so that the target will generate an error message (e.g. ICMP port unreachable) when it receives the probe. However this is only probably true: traceroute uses destination port numbers (32768+666 and above) that user processes can also bind to, since they are not "well-known" ports as defined by the Internet Assigned Numbers Authority. Thus it is possible, albeit unlikely, that a user process is bound to the port to which traceroute sends a probe, and that process will absorb the probe without generating a response. traceroute can send multiple probes for a specified hop length, each to a different destination port number, and so reduces (but does not eliminate) the possibility of all probes that reach the target not eliciting a response. The second form of conflict results from traceroute's use of source port numbers: To allow multiple traceroute processes to run concurrently on one host, traceroute includes the process ID in probes, and each traceroute process only handles responses that arose from probes that match its process ID. Such use of the process ID is appropriate for setting the Identifier field of the ICMP header of echo request probes, but UDP probes carry this identifier in the source port number. This can lead to a conflict if another process should happen to be bound to the port that matches the ID of a traceroute process. This second conflict could be readily avoided if traceroute used an ephemeral source port number for UDP probes. The use of the process ID is probably merely an artefact of traceroute initially being designed to use ICMP echo request probes and later being modified to use UDP probes [13]. However unlikely these conflicts may be in practice, they do point to the potential to improve traceroute's use of the port space.

If the only purpose for varying the destination port number of UDP probes was to reduce the chance that multiple probes are absorbed by processes running on the target host, then the distinct destination port numbers would only be needed for each probe sent with the same TTL; probes sent with differing TTLs could reuse destination port numbers. However, there is also a need to match a response with the probe that generated that response so traditional traceroute changes the destination port number for each probe that it sends, not just each probe that it sends with a certain TTL. Unfortunately this behaviour produces a large load on the destination port space: traceroute will by default send probes to as many as 90 different destination port numbers as it sends three probes for each of 30 hops. Furthermore, this load is incurred by potential targets of traceroute (who receive no benefit from the traceroute) rather than by the inquirer. Instead, it would be useful to shift the load to the *inquirer's source* port space, and allow all probes to be sent to a common destination port. raceroute does this by requesting ephemeral source port numbers from the operating system for each probe that it sends and using a hash table to map the source port numbers in responses to information about the corresponding probe. The hash table is necessary because the operating system may allocate port numbers from across the port space, compared to traditional traceroute that sends probe number *P* to destination port 32768+666+*P*.

We propose assigning a well-known port number that no process will bind to (by virtue of it being outside the range of ports available for processes in user space) for route tracing. Probes could be sent to such a destination address with certainty that they will not be absorbed by running processes, so they would be sure to elicit responses provided such responses are permitted by the security policy of the host. While this will not prevent existing traceroute implementations from sending probes to varied destination ports, the pressures of increasing path lengths and increasing port utilisation by NAT and the like will encourage use of route tracers that can use source ports and so are not restricted to 90 destination port numbers starting at 32768+666. Furthermore, assigning a specific port for route tracing activities would allow firewalls to explicitly indicate whether or not they permit route tracing activities. A considerate inquirer that discovers that route tracing to this port is blocked could then refrain from route tracing to other ports.

By using source ports to match responses to probes, an inquirer can trace the route to ports that are used for common services, e.g. HTTP or SMTP. The benefit of such tracing is that it traces the path used for normal traffic (rather than a traceroute-specific path) so probes and responses will be handled by address translators, load balancers and other network elements in the same way as normal traffic, and so report the behaviour observed by normal traffic. The risk of such tracing is that it may be used in an underhand manner to try to circumvent such network elements. TCP SYN segments have also been used for route tracing by the tcptraceroute tool [14].

In summary, raceroute uses source port numbers to match responses to probes, and uses a set of ephemeral source port numbers provided by the operating system, allowing multiple instances of raceroute to run concurrently on one machine.

IV. Complications and costs of path length estimation

Two factors complicate path length estimation: asymmetrical paths and TTL resetting.

The response to the scout packet suggests the length of the path from target to inquirer, but what the route tracing process really needs is the length of the path from *inquirer to target*. Because of asymmetrical paths [15] these two path lengths may not be identical. (Note that we are only concerned with the path *lengths*. The two paths may happen to have the same length but be asymmetrical because they pass through different sets of routers, however that is of no concern to us.) If the path from target to inquirer is *longer* than the forward path in the opposite direction, then the inquirer will overestimate the forward path length and send multiple probes that will reach the target, leading to waste of bandwidth. If this path is *shorter* than the forward path, then the inquirer will need to send additional probes after those that probe the estimated path length.

raceroute addresses path asymmetry by "bumping up" the path length estimate by one hop if the probe sent to the estimated path length does not result in a response from the target. raceroute will then operate in incremental fashion, like traditional traceroute, sending another deeper probe whenever it does not receive a response from the target for a probe. It continues in this fashion until either it reaches a maximum hop length (as in traceroute, with the same default of 30 hops) or until it fails to receive responses to probes sent to more than a certain range of path lengths. A default range of 3 seems effective in avoiding trailing strings of non-responsive probes while only rarely terminating the probing when a deeper probe would have elicited a response.

While raceroute does not know how many hops the scout traversed to reach the target, it does have information about how many hops *probes* traversed to reach the nodes (router or target) that generated responses, since it set the TTL field of these probes. Thus, raceroute can detect path length asymmetries by virtue of the TTL of a response suggesting that the response traversed a different number of hops than were traversed to reach the respondent. raceroute uses this to calibrate the path length estimate, increasing the estimate by this difference when a response appears to have traversed fewer hops than the probe, suggesting that the response to the scout may have traversed fewer hops than will be needed to reach the target over the forward path.

Another source of interference to the path length estimation is created by devices that reset the TTL of passing packets. Such devices are sometimes deployed at the outskirts of the Internet (deploying them at transit points could result in routing loops that the TTL field is intended to break) to limit route tracing *into* corporate networks. Such resetting can cause raceroute to underestimate the path length, forcing it to incrementally bump up the estimate when it finds that probes to the estimated path length do not generate responses from the target.

Another complication can arise from ICMP rate limiting, which can be used to protect a network from overload from ICMP traffic (e.g. by the W32/Welchia worm) by smoothing out bursts of such traffic or discarding excess ICMP traffic. Such rate limiting can cause the responses to probes that are sent in bursts (as raceroute does) to appear to have experienced longer delays than would be observed if the probes had been sent out more gradually (as in the traditional traceroute) or for no response to be received at all. At present, raceroute does not address ICMP rate limiting, other than providing an option for controlling the minimum interval between probes.

There are several costs to basing route tracing on estimated path lengths, and it is important that these not raise the overall cost of such tracing beyond that of existing tracing mechanisms. First, there is the overhead of sending the scout packet, in terms of the delay incurred in waiting for a response (or timeout), the bandwidth used by this packet, and the potential for the packet to initialise state information in the target which a polite inquirer should release (e.g. by sending a TCP RST if it receives an ACK to a SYN probe). Another overhead is the potential for excess probes to be sent when the inquirer overestimates the forward path length.

V. aceroute: Inverted route tracing

Another way in which a route tracer can exploit knowledge of an estimated path length is by inverting the direction of traditional traceroute: Rather than progressively *increasing* TTL values to reveal the identities of routers that are progressively *further* from the inquirer, the probes can be sent in order of *decreasing* TTL values, since an estimate of the maximum TTL needed is already known. The advantage of inverting the path trace is that the process can be terminated when it reaches a router that the inquirer already knows, since presumably the path from the inquirer to this target through this known router will be the same as the path from the inquirer to a previous target through this known router. For example, Table 1 shows measured paths from an inquirer to three web sites. If the path to www.irtf.org is traced before the path to rfc-editor.org, then the latter trace could terminate once it reaches the lowest highlighted router (137.164.25.2). We call this technique "inverted route tracing" to distinguish it from remote [16] or reverse [www.reversetraceroute.org] traceroute, in which traceroute is invoked on a remote host to trace the route from that host to the inquirer.

It is possible that different packets take different paths from an inquirer to reach the same router. For example, Table 1 shows that router 198.32.170.43 is on the paths from the inquirer to both www.irtf.org and rfc-editor.org, but the paths differ in which routers are traversed immediately before this router (192.231.212.49 followed by 192.231.212.42 and 192.231.212.162 for the path to www.irtf.org, but followed by only 192.231.212.34 for the path to rfc-editor.org). This can occur because of load balancing.

Traditional traceroute offers a `-f` option to allow control of the first TTL used. While this can avoid probing of routers near the source, traditional traceroute has no way of knowing how many such routers need not be probed. It is only by inverting the route tracing process that this can be determined.

Paths to multiple targets intersect at routers near the inquirer (boldface shows commonality with path to www.irtf.org). Inverting the trace process allows it to terminate when the first such router is identified, reducing probing of routers near the inquirer.

Inquirer's end of path		
149.171.92.2	149.171.92.2	**149.171.92.2**
129.94.255.181	129.94.255.181	**129.94.255.181**
138.44.1.37	138.44.1.37	**138.44.1.37**
202.158.202.1	192.231.212.49	**192.231.212.49**
202.158.194.74	192.231.212.42	192.231.212.34
129.250.10.225	192.231.212.162	**198.32.170.43**
129.250.3.84	198.32.170.43	**198.32.8.10**
129.250.5.49	198.32.8.10	**198.32.8.94**
129.250.2.240	198.32.8.94	**137.164.25.2**
129.250.9.182	137.164.25.2	137.164.22.12
152.63.57.50	137.164.22.20	137.164.23.252
152.63.1.61	137.164.22.18	128.9.0.7
152.63.10.61	137.164.22.127	128.9.160.27 (rfc-editor.org)
152.63.144.30	137.164.22.110	
152.63.40.2	137.164.22.36	
152.63.39.97	137.164.22.30	
157.130.44.142	137.164.23.66	
132.151.6.21 (www.ietf.org)	128.32.0.99	
	169.229.0.30	
	192.150.187.18 (www.irtf.org)	
Target's end of path		

VI. Conclusion

This paper has shown how an inquirer, which seeks to trace the route through a network to a target, can use the TTL of a response from the target to estimate the length of the path between inquirer and target. With this knowledge, the inquirer can expedite the route tracing process, without the risk of sending large numbers of unnecessary probes. For example, tracing the 17 or 18 hop route from unsw.edu.au to www.ietf.org takes 2.8s using traceroute but only 0.5s using raceroute. This paper assessed the suitability of different types of probes, in terms of their ability to reach a target through middleboxes such as firewalls and address translators, and found that TCP probes addressed to port 443 were the most suitable. It also proposed the establishment of a route tracing port in the well-known port range to prevent route tracing probes from being absorbed by processes on end systems. Finally, the paper outlined how the route tracing process can be inverted, to probe routers from the target end of the path to the inquirer's end of the path, and terminating the route tracing process when it reaches a router that is familiar to the inquirer, thus reducing the number of probes needed.

Acknowledgements

Thanks to Matthew Luckie for suggesting the use of the TTL in ICMP responses to scout packets.

References

[1] M. Merzbacher and D. Patterson: 'Measuring end-user availability on the Web: practical experience', Proc. Int'l Conf. on Dependable Systems and Networks, pp. 473-7, Jun. 2002

[2] H. Malec: 'Communications reliability: a historical perspective', IEEE Trans. Reliability 47(3):333-45, Sep. 1998

[3] V. Jacobson: '4BSD routing diagnostic tool available for ftp', 1988, email sent to the ietf@venera.isi.edu and end2end-interest@venera.isi.edu mailing lists

[4] F. Begtaševic and P. V. Mieghem: 'Measurements of the Hopcount in Internet', Proc. Passive and Active Measurements Workshop, Apr. 2001

[5] Z. Mao, J. Rexford, J. Wang and R. Katz: 'Towards an accurate AS-level traceroute tool', Proc. SIGCOMM, pp. 365-78, Aug. 2003

[6] R. Mahajan, N. Spring, D. Wetherall and T. Anderson: 'User-level Internet path diagnosis', Proc. 19th ACM Symp. on Operating Systems Principles, pp. 106-19, Oct. 2003

[7] O. Marce: 'Dichotomy-based method of tracing a route between two nodes of a data network', US Patent application number 20020131367, Sep. 12 2002 2002

[8] 'Top 101 Most Incredibly Useful Sites', PC Magazine Oct. 14 2003, http://www.pcmag.com/article2/0,1759,1335911,00.asp

[9] R. Braden: 'Requirements for Internet hosts - communication layers', IETF RFC 1122, 1989

[10] F. Baker: 'Requirements for IP Version 4 Routers', IETF RFC 1812, Jun. 1995

[11] A. B. Downey: 'Using pathchar to estimate Internet link characteristics', Proc. SIGCOMM, pp. 241-50, 1999

[12] J. Postel: 'Internet Control Message Protocol', IETF RFC 792, Sep. 1981

[13] V. Jacobson: 'Re: traceroute history: why UDP?' 1999, posting to the comp.protocols.tcp-ip newsgroup

[14] M. Toren: 'tcptraceroute', http://michael.toren.net/code/tcptraceroute/ 2004

[15] V. Paxson: 'End-to-end routing behavior in the Internet', Proc. ACM SIGCOMM, pp. 25-38, 1996

[16] K. White: 'Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations', IETF RFC 2925, Sep. 2000