Low-Cost Remapping of Internet Path Changes

Guilherme Capanema[1], Ítalo Cunha[1], Renata Teixeira[2], Darryl Veitch[3], Christophe Diot[4]

[1]Universidade Federal de Minas Gerais
{capanema, cunha}@dcc.ufmg.br

[2]CNRS and UPMC Sorbonne Universités
renata.teixeira@lip6.fr

[3]Dept. Elec. and Elec. Eng. University of Melbourne
dveitch@unimelb.edu.au

[4]Technicolor
christophe.diot@technicolor.com

ABSTRACT

Internet topology maps collected with traceroute may be incomplete or out-of-date because we cannot measure frequently enough to detect all routing changes without overloading the network. In this paper, we show that routing changes usually affect few routers. We exploit this property to design RemapRoute, a tool to locally remap Internet routing changes that only probes the few involved hops instead of the whole route. Our evaluation with trace-driven simulations and in a deployment shows that RemapRoute significantly reduces the number of probes needed to remap routes, with no impact on remapping accuracy nor latency. This reduction in remapping cost allows us to build more complete and up-to-date topology maps.

1. INTRODUCTION

Internet failure identification systems usually require information about network routes [9, 12, 15, 16]. Similarly, content distribution networks measure Internet routes to choose the "best" server to answer a request [10]. These and other systems measure routes frequently in an attempt to track routing changes as they happen.

Internet route measurements are usually performed with traceroute [1, 14, 25], which sends probes to identify a sequence of interfaces between a source and a destination. The bandwidth available to send probes is finite. Route measurements for topology mapping require a large number of probes and can take several minutes to a few days [4,6,22]. It is impossible to measure routes frequently enough to detect all routing changes without overloading the network. Thus, Internet topology maps may be outdated or inconsistent as routing changes may occur during the measurement process.

Our DTRACK system tracks Internet routing changes to keep Internet topology maps more up-to-date [7]. DTRACK separates the tasks of detecting and remapping routing changes. To detect changes, DTRACK uses a lightweight probing process that combines two main ideas: (1) redirect probes from stable paths, where routing changes are unlikely, to unstable paths, where routing changes are more likely; and (2) spreading probes uniformly over the network and over time to reduce redundant probes. DTRACK keeps a database with the previous route observed on each monitored path. DTRACK sends a probe to a hop in the path and compares the IP address in the reply with the IP address in the previous route. If the probe reply is incompatible with the previous route, e.g., the IP address in the probe reply is not in the previous route, a change is detected and the remapping process is triggered.

I would point out that these are multi-branch routes here so it doesn't make it some simpler than it is.

To remap changes, DTRACK uses Paris traceroute [1, 25] to measure the current route. Paris traceroute is an improved version of traceroute that identifies routers that perform load balancing. DTRACK uses Paris traceroute because it is impossible to differentiate routing changes from load balancing otherwise [6]. Currently, DTRACK remaps the whole path using Paris traceroute. This approach guarantees correct remapping of the current route, but wastes probes as most routing changes involve few hops (Sec. 3). In particular, this approach ignores two pieces of information available when the remapping process begins: the previous route and the hop where the change was detected.

In this paper we propose RemapRoute, a new tool to reduce the cost of remapping Internet routing changes (Sec. 4). Given the previous route observed before the change and a hop where a change was detected, RemapRoute strategically sends probes to locate the change and locally remap it. RemapRoute minimizes the number of wasted probes sent to hops that are not involved in the routing change. Our evaluation via trace-driven simulations shows that RemapRoute reduces by half the remapping cost of 88% of routing changes in our dataset (Sec. 5). Remap cost reduction is even larger for routes traversing routers that performs load balancing. Our evaluation of RemapRoute in a real deployment confirms our simulation results and demonstrates RemapRoute's accuracy (Sec. 6). We make the following contributions:

- We characterize Internet routing changes and show that they usually involve few hops (Sec. 3);
- We propose methods to locate and remap path changes that reduce wasted probes (Sec. 4);
- We show the efficacy of our tool with trace-driven simulations and in a real deployment (Secs. 5 and 6).

Reducing ~~probing cost to remap Internet routing changes~~ the remapping cost

increases the number of probes available for topology mapping. We can use the extra probes to monitor more paths and improve network coverage, or we can increase probing frequency and improve routing change tracking accuracy. RemapRoute is another step toward building more complete and consistent Internet maps.

2. DEFINITIONS AND BACKGROUND

Following Paxson [20], we use *virtual path* to refer to the connectivity between a fixed source (here a monitor) and a destination d. At any given time, a virtual path is realized by a route which we call the *current route*. Since routing changes occur, a virtual path can be thought of as a continuous time process $P(t)$ which jumps between different routes over time. A *route* can be *simple*, consisting of a sequence of IP interfaces from the monitor toward d, or *branched*, when one or more *load balancing* routers are present, giving rise to multiple overlapping sequences (branched routes are called "multi-paths" in [25]). A route can be a sequence that terminates before reaching d. This can occur due to routing changes (e.g., transient loops), or the absence of a complete route to the destination.

Given two consecutive measurements of a path at instants t_i and t_{i+1}, we define a *path change* as a sequence of contiguous hops in the current route that differ from hops in the previous route. We compute path changes minimizing the edit distance (adding, removing or substituting interfaces) between the current and previous routes. We define a change's *divergence hop*, h_d, and *convergence hop*, h_c, as the hops immediately before and after the hops edited in the change, respectively. We say that the divergence hop, the convergence hop, and all hops in between are *involved* in the path change. To give an example, if $P(t_i) = \{a, b, c, d, e, f, g\}$ and $P(t_{i+1}) = \{a, b, e, x, y, g\}$, we have a change from $h_d = 1$ to $h_c = 2$ (removal of c and d), and another change from $h_d = 2$ and $s_c = 5$ (substitution of f for x and y). Finally, we refer to hop h in path $P(t)$ as $P(t)[h]$.

3. INTERNET PATH CHANGE CHARACTERIZATION

In this section we characterize Internet path changes and verify that most changes involve few hops in few ASes.

We deployed DTRACK [7] to track path changes from 72 PlanetLab nodes for one week starting March 4th, 2011. Each monitor chooses 1,000 random destinations from a list of 34,820 reachable destinations. At each monitor, we configure DTRACK to track changes using 8 probes per second, similar to the probing rate used by DIMES [21]. In one week we observed 1,202,960 path changes. The observed paths traverse 7,315 ASes and 97% of the ASes with more than 50 customers [19].

Fig. 1 shows the distribution of the number of hops involved in path changes in our dataset. Starting from the previous route, the number of hops involved in a change is the minimum number of hops we need to remap to build the current route. We see that 78% of changes involve four or less hops, a small number given that the median route length in our dataset is 17 hops. The most common type of path change is when the router in a single hop changes, which results in a change that involves three hops: the hop where the router changed, as well as the divergence and convergence hops. We note that 9% of path changes involve two hops. These changes only remove hops from the previous route (the current route is contained in the previous one), and the only involved hops are the divergence and convergence hops. Typical causes for path changes that involve two hops are connectivity failures and measurement errors where it is impossible to measure hops behind the failure or errors.

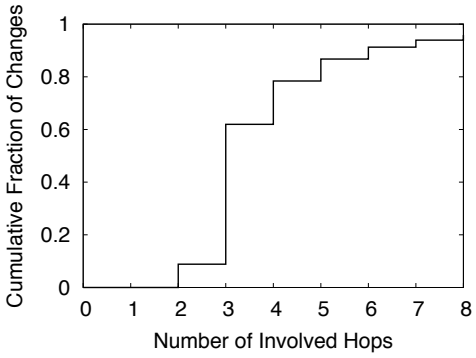

Figure 1: Distribution of the number of hops involved in path changes.

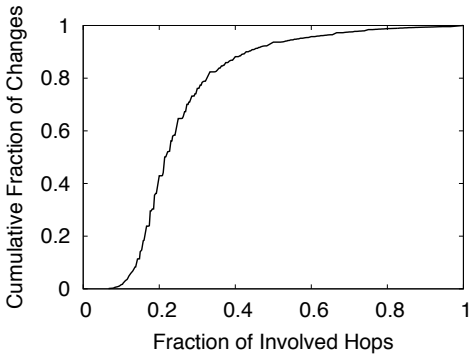

Figure 2: Distribution of the fraction of routers involved in path changes.

Fig. 2 shows the distribution of the fraction of hops involved in a change, i.e., the number of involved hops divided by the route length, for all changes in our dataset. We see that 76% of path changes involve less than 30% of the hops in the path. This shows the potential savings from local change remapping, when compared to the current approach of remapping the whole path. The curve starts at $x = 0.066 = 2/30$ as no change involves less than two hops and because DTRACK only measures up to 30 hops in a route (the default in Paris traceroute [1, 14]).

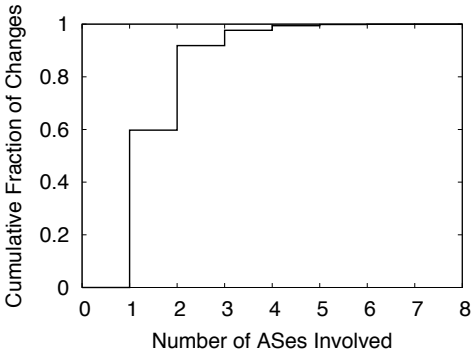

Figure 3: Distribution of the number of ASes involved in path changes.

Fig. 3 shows the distribution of the number of ASes involved in path changes. We translate interface IP addresses measured by DTRACK to AS numbers combining IP-to-AS mapping databases from Team Cymru[1] and iPlane [17]. Each IP address that does not appear in any database is associated with a unique AS number. This is a conservative decision that may overestimate the number of ASes involved in a change. Fig. 3 explains why path changes involve few hops. Even with a conservative mapping of IP addresses to AS numbers, 60% of path changes are internal to one AS number and only 7% involve more than two ASes.

4. REMAPROUTE

We now present the algorithm we use in RemapRoute to locally remap path changes. The algorithm receives as input the previous route observed before the path change, $P(t_{i-1})$, and the hop h' where the change was detected, i.e., $P(t_i)[h'] \neq P(t_{i-1})[h']$. To remap the path change, RemapRoute operates in two phases: first it locates where the change happened (Sec. 4.1) and then remaps it locally (Sec. 4.2).

The process of locating and remapping the path change compares hops on the current route with hops on the previous route. To remap each hop, RemapRoute sends multiple probes, systematically varying the IP header fields to identify routers that perform load balancing, similar to Paris traceroute [1,25].

4.1 Locating path changes

RemapRoute starts from the hop h' where the current route differs from the previous one, i.e., $P(t_i)[h'] \neq P(t_{i-1})[h']$. If the router in the current route at hop h' is not contained in the previous route, i.e., $P(t_i)[h'] \notin P(t_{i-1})$, then h' is one of the hops involved in the path change and RemapRoute proceeds to the next phase to locally remap the change (Sec. 4.2).

If the previous route contains the current router at hop h', i.e., $P(t_i)[h'] = P(t_{i-1})[h'']$, then we have a path change involving hops before h' that added or removed routers to the path. Fig. 4 shows an example of one path change where a router d was substituted for two routers x and y. A probe to the sixth hop detects a path change as the answer comes from router $f = P(t_i)[6]$, which is not the expected answer, $g = P(t_{i-1})[6]$.

[1]Team Cymru, IP to ASN Mapping, http://www.team-cymru.org/

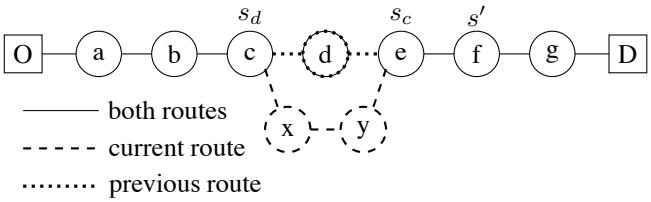

Figure 4: Example of a path change substituting d for x and y.

is h'' < h'? > h'? same flow id?

To find a hop involved in the path change, i.e., a router in the current route that is not in the previous route, RemapRoute performs a binary search in the path. RemapRoute initializes $h_{\text{left}} = 0$ and $h_{\text{right}} = h'$. At each iteration in the search, RemapRoute remaps hop $h = (h_{\text{left}} + h_{\text{right}})/2$ and looks for the detected router on the previous route. Again, if the router found at hop h is not in the previous route, i.e., $P(t_i)[h] \notin P(t_{i-1})$, the search finishes and RemapRoute goes to the next phase to locally remap the change. If the router at hop h in the current route is at hop h in the previous route, i.e., $P(t_i)[h] = P(t_{i-1})[h]$, then the path change is to the right of h and RemapRoute makes $h_{\text{left}} = h$. If the router at hop h on the current route is at another hop h'' in the previous route, i.e., $P(t_i)[h] = P(t_{i-1})[h'']$, the change is to the left of h and RemapRoute makes $h_{\text{right}} = h$.

We cannot compare the current route with the previous route if the router at hop h does not answer to probes. In this case we take a conservative approach, decrementing h and continuing the search at the previous hop without updating h_{left} and h_{right}.

If a path change only removes hops from the previous route, then there is no hop in the current route that does not belong in the previous route. In this case, the search terminates when $h_{\text{left}} = h_{\text{right}} + 1$ and no local remapping is necessary (Sec. 4.2). what about possible further changes beyond this hop? or other topological changes to the branched path?

4.2 Local remap

Local remap starts from a hop h where the router in the current route does not belong to the previous route, i.e., $P(t_i)[h] \notin P(t_{i-1})$. RemapRoute sequentially remaps hops in the current route after h until if finds the convergence hop $h_c > h$ with a router that belongs to the old route, i.e., $P(t_i)[h_c] \in P(t_{i-1})$. If one of the routes does not reach the destination, the convergence hop may not exist. In this case, RemapRoute remaps the current route until it reaches the destination or up to the last reachable hop. In cases where a

route does not reach the destination, RemapRoute identifies the last reachable hop after finding three unresponsive hops (just like traceroute).

Similarly, RemapRoute sequentially remaps hops in the current route before h until it finds the divergence hop $h_d < h$ with a router contained in the previous route, i.e., $P(t_i)[h_d] \in P(t_{i-1})$. In the worst case, the search for the divergence hop terminates at the origin, which belongs to any route in the path. If hop h_d is not identical on both current and previous routes, i.e., $P(t_i)[h_d] \neq P(t_{i-1})[h_d]$, then there exists another path change before h_d that added or removed hops to the current route. RemapRoute goes back the first phase (Sec. 4.1), making $h' = h_d$, to locate it and then remap it. This process is repeated recursively until there is no path change to remap.

We note that we need to remap all hops between h_d and h_c sequentially because all these hops are involved in the path change. For path changes that only remove hops, we have $h_d = h_{\text{left}}$ and $h_c = h_{\text{right}}$ and no remap is necessary.

4.3 Remap example

Consider the path change shown in Fig. 4 is detected with a probe to the fifth hop in the path. We have $P(t_{i-1})[5] = f$ and $P(t_i)[5] = e$. As $e \in P(t_{i-1})$, a router was added before the fifth hop and RemapRoute starts the binary search to locate the change. RemapRoute starts remapping the second hop, where $P(t_{i-1})[2] = P(t_i)[2]) = c$, indicating that the router was added between the second and fifth hops. RemapRoute then remaps the third hop, where $P(t_{i-1})[3] = d$ and $P(t_i)[3] = x$. As $x \notin P(t_{i-1})$, RemapRoute proceeds to the local remap phase, remaps the fourth hop, then terminates.

5. EVALUATION WITH TRACE-DRIVEN SIMULATIONS

In this section we evaluate RemapRoute using trace-driven simulations. We use the same data set described in Sec. 3. We focus on comparing the probing cost of remapping path changes using Paris traceroute and RemapRoute.

RemapRoute's remapping cost varies according to the hop h' where the change was detected. We calculate RemapRoute's remapping cost for all hops in a path where the change can be detected. Fig. 5 shows the distribution of RemapRoute's remapping cost. We show the distributions for the minimum, average, and maximum costs, calculated over all hops where the change can be detected. We see that the distributions of minimum and maximum costs are similar. One reason for this is that many path changes involve few hops, so there are few hops where the path change can be detected and little variation in remapping cost. In the rest of this paper, we use the average cost as a representative of RemapRoute's remapping cost.

Fig. 6 compares remapping costs when using RemapRoute and Paris traceroute. Comparing with Fig. 1, we see that RemapRoute frequently probes more hops than the number of involved hops. This increase is due to path changes

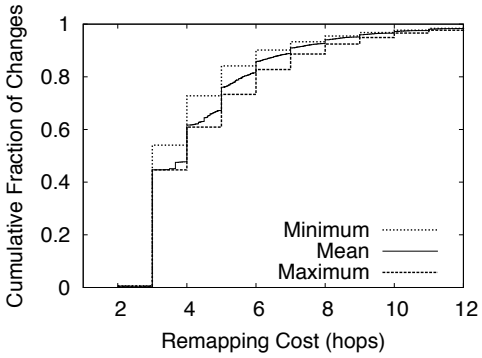

Figure 5: RemapRoute's remapping cost varying the hop h' where a change is detected.

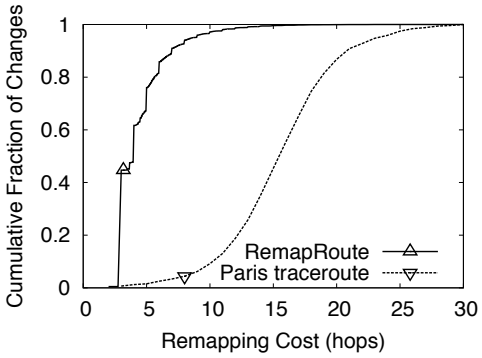

Figure 6: Comparison of remapping cost between RemapRoute and Paris traceroute.

that need to be located using binary search before being remapped. Despite the search process, RemapRoute has remapping cost significantly smaller than Paris traceroute. Note that RemapRoute's remapping cost is rarely smaller than three hops, even when 9% of path changes involve only two hops (Fig. 1). Path changes that involve two hops only remove hops in the old route, and RemapRoute needs to locate where hops were removed using binary search. Binary search requires at least three probes unless the path change was detected in the first three hops on the path, which happens only 0.4% of the time in our data.

Fig. 7 shows the distribution of remap cost savings when using RemapRoute instead of Paris traceroute. We compute remap cost savings as $(C_{\text{Paris}} - C_{\text{RemapRoute}})/C_{\text{Paris}}$, where C_{Paris} is the number of hops remapped by Paris traceroute and $C_{\text{RemapRoute}}$ is the number of hops remapped by RemapRoute. The solid line, computed for all path changes in our data set, shows that cost savings are significant. RemapRoute reduces remapping cost by more than half for 88% of path changes. The dashed lines in Fig. 7 show the remap cost savings for routes shorter than 10 hops (labelled "short") and routes longer than 20 hops (labelled "long"). Remap cost savings is higher for long routes, where Paris tracer-

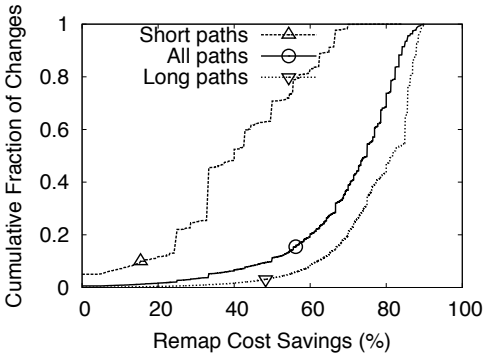

Figure 7: Remap cost savings of RemapRoute over Paris traceroute.

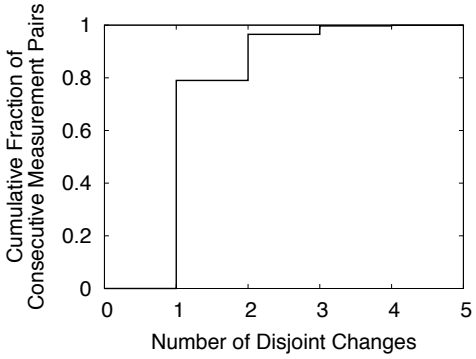

Figure 8: Distribution of the number of disjoint path changes between consecutive measurements pairs.

oute wastes probes in many hops that are not involved in the change. Also, RemapRoute brings cost savings even when remapping changes on short routes.

5.1 Remapping errors

RemapRoute remaps path changes given a hop h' where a change was detected. This may lead to inconsistencies if a path undergoes two disjoint changes before we detect the first one. For example, a path $\{a, b, c, d, e, f, g\}$ may change into $\{a, x, c, d, y, f, g\}$ before we detect a change. In this case, we can detect changes in hops $h' = 1$ and $h'' = 4$. Unfortunately, given h' or h'', RemapRoute will remap only one change.

To evaluate the gravity of this problem, Fig. 8 shows the distribution of the number of disjoint changes for consecutive path measurement pairs in our data set.[2] We have that 79% of consecutive measurements remap only a single disjoint change, which RemapRoute will correctly remap for any hop h' where the change may be detected. Only 3% of consecutive measurement pairs remap three or more disjoint changes, indicating that RemapRoute will correctly remap the current route in most cases.

Three other factors contribute to minimize the impact of disjoint changes. First, RemapRoute may remap disjoint changes before the detection hop h' if RemapRoute detects the change during the remap process. In particular, the probability that RemapRoute will remap a disjoint change before the detection hop h' is 43%. Second, a disjoint change what is not remapped when we run RemapRoute will be detected and remapped in the future (assuming the change tracking probing process is capable of detecting all possible changes). Any disjoint change that is not remapped causes only a temporary inconsistency in the data. Third, remap cost savings obtained with RemapRoute can be used to increase probing frequency and reduce the chance that two changes occur before we detect the first.

[2]In our data set we only measure paths with Paris traceroute when a change is detected. All consecutive measurement pairs remap at least one disjoint change.

Another limitation of RemapRoute is that the binary search mechanism may fail when the relative order of two hops change between the previous and current routes. An extreme, but illustrative, example, is a change from $P(t_{i-1}) = \{a, b, c, d, e, f\}$ to $P(t) = \{a, e, d, c, b, f\}$. Only 0.9% of path changes in our datasets reorder hops. As hop reordering is rare, we take the conservative approach of remapping the whole path (as in Paris traceroute) when a reordering is detected during the remap process. As the previous results show, this limitation does not compromise RemapRoute's utility.

6. EVALUATION IN PLANETLAB

In this section we evaluate RemapRoute in PlanetLab. We deployed a modified version of DTRACK in 140 PlanetLab nodes and collected measurements for 18 hours on November 30th, 2012. Our modified DTRACK runs RemapRoute and Paris traceroute whenever it detects a change. This allows us to compare RemapRoute and Paris traceroute running in parallel on the same path change. As in the data set used in the previous sections, each node has a detection budget of 8 probes per second and monitors 1,000 destinations chosen randomly from a list of 34,820 reachable destinations in the Internet. Given the short duration of the dataset, we only detect 87,848 path changes. The measured paths traverse 7,143 ASes and 95% of the ASes with more than 50 customers [19]. **[[We are currently redeploying the measurements to get a larger data set.]]**

Fig. 9 shows remap cost savings when DTRACK uses RemapRoute instead of Paris traceroute. Comparing with Fig. 7, the average cost savings in the real deployment is quantitatively similar to the results we obtained via simulation (solid line). For example, RemapRoute reduces remapping cost by more than half for 90% of path changes in the real deployment.

Remap cost savings for short and long routes are more similar to the average savings in the real deployment than in the simulations. In other words, the dashed lines in Fig. 9 are closer to the solid line than in Fig. 7. We attribute this

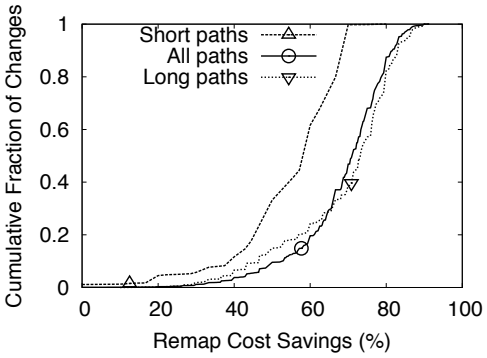

Figure 9: Remap cost savings with RemapRoute in real scenarios.

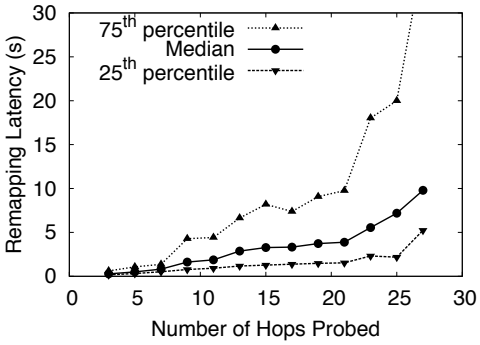

Figure 10: RemapRoute remap latency in the real deployment.

change to two factors: (i) the smaller number of observed changes may limit the variety of observed changes and (ii) differences in the set of monitored paths.

Fig. 10 shows the 25^{th}, the 50^{th}, and the 75^{th} percentiles of RemapRoute's remap latency as a function of the number of hops probes during the remapping process. We studied the remap latency because RemapRoute probes hops sequentially: the next hop to probe is computed based on the result of the last probed hop. Paris traceroute, however, could parallelize probing of different hops (but the classic implementation does not). As most RemapRoute remaps require probing only a few hops (Fig. 6), remapping latency is generally less than 5 seconds. Fig. 11 shows the 25^{th}, the 50^{th}, and the 75^{th} percentiles of Paris traceroute's remapping latency in the real deployment. Our objective is not to compare RemapRoute's and Paris traceroute's remapping latency, as latency is heavily affected by implementation choices. Our objective is to show that RemapRoute's remapping latency is acceptable for use in real systems. We also note that a topology mapping system like DTRACK can execute RemapRoute simultaneously on different paths if more than one path change is detected in a short time interval.

Finally, we also evaluate whether remaps performed by

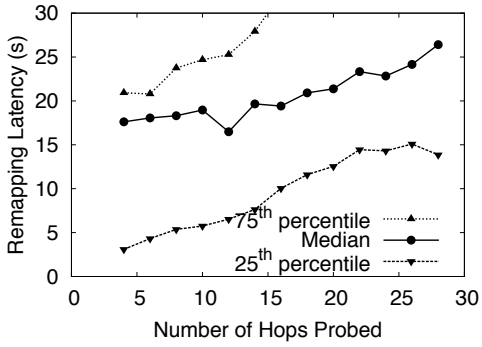

Figure 11: Paris traceroute remap latency in the real deployment.

RemapRoute are equivalent to using Paris traceroute to measure the whole path. For each observed change in the real deployment, we compare hops remapped by RemapRoute with the route measured by Paris traceroute. Only 0,6% of RemapRoute measurements differ from Paris traceroute measurements. We attribute this difference to measurement errors. The identification of routers that perform load balancing in both Paris traceroute and RemapRoute is probabilistic [25]. For example, Paris traceroute's and RemapRoute's default configuration identify all routers performing load balancing in a path with 95% confidence. Measurement errors are inevitable and cause different remaps regardless of the tools used. Another cause for different measurements are path changes that happen while Paris traceroute and RemapRoute are running.

In summary, remapping path changes in the Internet with RemapRoute is as accurate as remapping with Paris traceroute, significantly reduces probing cost, and has little impact on remapping latency.

7. RELATED WORK

Operators can use routing protocol logs (e.g., OSPF and IS-IS) and router configuration files to map the topology of their network [13, 18, 24]. This approach results in accurate and complete topologies, however it is available to network operators only and is restricted to a single network. To map multiple networks, we can use public BGP collectors[3] to build a map of Internet ASes [8, 19]. Unfortunately, BGP does not expose all network links and public BGP collectors have incomplete AS coverage [5, 19]. In this work, we take an orthogonal approach and measure the network topology at the router level using active probes.

Research on topology mapping at the router level using active measurements has three main goals: (i) increase Internet coverage, (ii) increase the accuracy of the measured topology, and (iii) increase measurement frequency.

[3]The Univ. of Oregon Routeviews Project, www.routeviews.org
RIPE RIS, http://www.ripe.net/data-tools/stats/ris

The usual approach to increase network coverage is to monitor a large number of paths. CAIDA's Skitter/Ark platform [4] attempts to cover the whole Internet using monitors to measure paths toward all $/24$ prefixes announced in the Internet. To achieve high coverage, however, Skitter/Ark sacrifices measurement frequency. Skitter/Ark takes between two and three days to collect a topology due to the large number of monitored paths and (unavoidable) bandwidth limitations at monitors. An alternative is to share the probing load among various monitors, like in DIMES [21] and Ono [3]. In this work, we assume that the sets of monitors and destinations are fixed. However, RemapRoute does not impose any restrictions on the set of monitors and destinations. RemapRoute can could be used in these systems to reduce network load or increase measurement frequency.

Techniques to increase the precision of the measured topology try to infer more information about the network than classic traceroute measurements. Paris traceroute, for example, sends additional probes systematically varying values of IP header fields to identify all routers that perform load balancing in a path [1, 25]. RemapRoute also detects routers performing load balancing using the same algorithm as Paris traceroute. DisCarte [22] uses traceroute and probes with the IP Record Route option enabled to collect two sequences of routers in Internet paths. DisCarte post-processes and combines traceroute and Record Route IP address sequences to build a more accurate network topology. These and other techniques send additional probes and thus increase topology mapping cost. RemapRoute's object is complementary: reduce path change remapping costs and increase the amount of probes available to collect more precise measurements.

Our work is more related to techniques that try increase Internet topology mapping measurement frequency. Monitors have limited bandwidth to measure the topology. Reducing the cost of each measurement directly increases the frequency at which measurements can be collected. RocketFuel, for example, reduces the cost to measure a target AS's topology by skipping paths that enter and exit the AS's network through the same border routers as another previously measured path [23]. Another approach is to reduce the cost choosing only a single destination in each subnetwork in an AS [2]. DoubleTree reduces redundant probes to routers close to the monitor (shared by paths starting at that monitor) and routers close to the destinations (shared by paths toward that destination) [11]. RemapRoute was developed to reduce remapping cost in DTRACK our system to track Internet path changes [7]. RemapRoute and DTRACK complement existing techniques that aim at reducing measurement cost. In particular, we note that DTRACK reduces redundant probes to routers close to monitors, like DoubleTree, and is compatible with other techniques above to decide what paths to measure.

8. CONCLUSIONS AND FUTURE WORK

9. REFERENCES

[1] B. Augustin, T. Friedman, and R. Teixeira. Measuring Load-balanced Paths in the Internet. In *Proc. IMC*, 2007.

[2] R. Beverly, A. Berger, and G. Xie. Primitives for Active Internet Topology Mapping: Toward High-Frequency Characterization. In *Proc. IMC*, 2010.

[3] D. R. Choffnes, F. E. Bustamante, and Z. Ge. Crowdsourcing Service-level Network Event Monitoring. In *Proc. ACM SIGCOMM*, 2010.

[4] K. Claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov. Internet Mapping: from Art to Science. In *Proc. IEEE CATCH*, 2009.

[5] R. Cohen and D. Raz. The Internet Dark Matter - on the Missing Links in the AS Connectivity Map. In *Proc. IEEE INFOCOM*, 2006.

[6] Í. Cunha, R. Teixeira, and C. Diot. Measuring and Characterizing End-to-End Route Dynamics in the Presence of Load Balancing. In *Proc. PAM*, 2011.

[7] Í. Cunha, R. Teixeira, D. Veitch, and C. Diot. Predicting and Tracking Internet Path Changes. In *Proc. ACM SIGCOMM*, 2011.

[8] A. Dhamdhere and C. Dovrolis. Ten Years in the Evolution of the Internet Ecosystem. In *Proc. IMC*, 2008.

[9] A. Dhamdhere, R. Teixeira, C. Drovolis, and C. Diot. NetDiagnoser: Troubleshooting Network Unreachabilities Usind End-to-end Probes and Routing Data. In *Proc. ACM CoNEXT*, 2007.

[10] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[11] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient Algorithms for Large-scale Topology Discovery. In *Proc. ACM SIGMETRICS*, 2005.

[12] N. Duffield. Network Tomography of Binary Network Performance Characteristics. *IEEE Trans. on Inf. Theory*, 52(12):5373–5388, 2006.

[13] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. USENIX NSDI*, 2005.

[14] V. Jacobson. traceroute, Feb 1989. Available at ftp://ftp.ee.lbl.gov/traceroute.tar.gz.

[15] E. Katz-Bassett, C. Scott, D. R. Choffnes, I. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy. LIFEGUARD: Practical Repair of Persistent Route Failures. In *Proc. ACM SIGCOMM*, 2012.

[16] R. Kompella, J. Yates, A. Greenberg, and A. Snoeren. Detection and Localization of Network Blackholes. In *Proc. IEEE INFOCOM*, 2007.

[17] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: an Information Plane for

Distributed Services. In *Proc. USENIX OSDI*, 2006.
[18] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. N. Chuah, Y. Ganjali, and C. Diot. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Trans. Netw.*, 16(4):749–762, 2008.
[19] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. Quantifying the Completeness of the Observed Internet AS-level Structure. *IEEE/ACM Trans. Netw.*, 18(1):109–122, 2010.
[20] V. Paxson. End-to-end Routing Behavior in the Internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.
[21] Y. Shavitt and U. Weinsberg. Quantifying the Importance of Vantage Points Distribution in Internet Topology Measurements. In *Proc. IEEE INFOCOM*, 2009.
[22] R. Sherwood, A. Bender, and N. Spring. DisCarte: a Disjunctive Internet Cartographer. In *Proc. ACM SIGCOMM*, 2008.
[23] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, 2002.
[24] D. Turner, K. Levchenko, A. Snoeren, and S. Savage. California Fault Lines: Understanding the Causes and Impact of Network Failures. In *Proc. ACM SIGCOMM*, 2010.
[25] D. Veitch, B. Augustin, T. Friedman, and R. Teixeira. Failure Control in Multipath Route Tracing. In *Proc. IEEE INFOCOM*, 2009.