Local Remapping of Internet Path Changes

Ítalo Cunha
Universidade Federal de Minas Gerais
cunha@dcc.ufmg.br

Renata Teixeira
Inria
renata.teixeira@inria.fr

Darryl Veitch
Dept. Elec. and Elec. Eng.
University of Melbourne
dveitch@unimelb.edu.au

Christophe Diot
Technicolor
christophe.diot@technicolor.com

ABSTRACT

Internet topology maps collected with traceroute may be incomplete or out-of-date because we cannot measure routes frequently enough to detect all routing changes without overloading the network. In this paper, we show that routing changes usually involve few hops. 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

A number of distributed services and applications need to measure Internet paths to maintain an up-to-date view of the underlying infrastructure [3,6,7,9,12,14]. All these systems use some version of traceroute to repeatedly measure a large number of Internet paths. Traceroute sends probes to every interface between a source and a destination, so the measurement of a single path often requires tens of probes and takes few seconds [23]. The number of probes required to measure a path is even larger if one wants to measure all paths between a source and a destination when routers perform load balancing [23]. For example, discovering a (multi)path with Paris traceroute, which is a traceroute version that discovers all paths under load balancing, requires hundreds of probes and takes tens of seconds [23].[1] Because measuring each path takes time, a source cannot measure paths frequently and as a result it may miss path changes. For instance, topology mapping systems can take from several minutes to a few days to measure all required paths [3,4,21]. In between measurements, paths may be outdated or inconsistent.

Our previous work showed that Internet paths being mostly stable, measuring all paths equally wastes probes in paths that are not changing [5]. We developed DTRACK, a system that optimizes probing to track Internet path changes [5]. DTRACK splits the task of tracking paths into two sub-tasks. *Path change detection* sends a single probe per path at any given time, probing more often paths that are more likely to change. *Path remapping* sends probes to discover all interfaces of a path, but DTRACK only remaps the paths that have changed. Although DTRACK detects two times more path changes than traditional probing [5], remapping a path remains costly because DTRACK simply runs Paris traceroute to discover the entire end-to-end path. Remapping end-to-end paths with Paris traceroute guarantees paths are accurate.

In this paper, we show that end-to-end remapping wastes probes because most path changes are localized in a few (consecutive) hops of a path (Sec. 3). We build on this observation to develop a more efficient remapping method for DTRACK (Sec. 4). Given the path before the change and the hop where the change was detected, we first send probes to locate the change and then just remap the (often few) hops that have changed. We call this method *local remapping* as opposed to the end-to-end remapping originally implemented in DTRACK. Local remapping still uses Paris traceroute's multipath detection algorithm [23] for discovering all interfaces at a given hop to guarantee accuracy, but we no longer have to remap all hops of the path. We develop a new version of DTRACK with local remapping. Our evaluation via trace-driven simulations shows that local remapping reduces by half the remapping cost of 88% of path changes in our dataset (Sec. 5). The accuracy of local remapping is almost as good as end-to-end remapping. Local remapping does not work only when we have no measurement of the path before the change or when a path change reorders the interfaces of the path, but these cases are rare (less than 1% of path changes in our dataset). Our evaluation in a real deployment confirms the results from the trace-driven simulation (Sec. 6). If we use the probes we save by using local remapping for detection, our new version of DTRACK can track **[[XXX]]** more path changes than the original DTRACK with end-to-end remapping and **[[XXX]]** more changes than traditional probing.

2. DEFINITIONS AND BACKGROUND

Following Paxson [18], we use *virtual path* to refer to the

[1] Given the prevalence of load balancing [1], it is important to accurately measure paths under load balancing to avoid traceroute errors and misinterpretation of path changes [4].

not defined - rewrite

You can't copy text from the SIGCOMM paper like this. You should at least acknowledge that this background is from Dtrack paper.

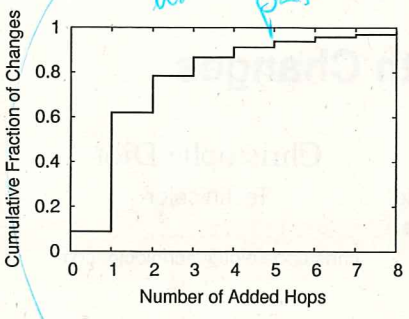

Figure 1: Distribution of the number of hops added in path changes.

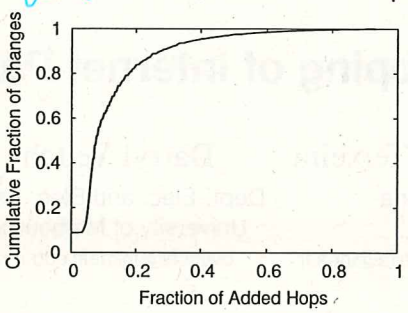

Figure 2: Distribution of the fraction of hops added in path changes.

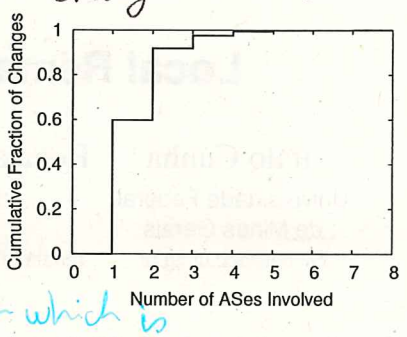

Figure 3: Distribution of the number of ASes involved in path changes.

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 [23]). We call a *hop* the set of interfaces that can be observed on all branches at a given distance h from the source. Finally, we refer to the interfaces at hop h in route $P(t)$ as $P(t)[h]$.

Given two consecutive measurements of a path at times 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 and removing hops) 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 added by the change, respectively. To give an example, consider $P(t_i) = [I_1, I_2, I_3, I_4, I_5, \{I_6, I_7\}, I_8]$ where I_5 is balancing load among interfaces I_6 and I_7; and consider $P(t_{i+1}) = [I_1, I_2, \{I_4, I_9\}, I_5, I_{10}, I_{11}, I_8]$, where I_2 is balancing load among I_4 and I_9. We have a change from $P(t_{i+1})[h_d] = I_2$ to $P(t_{i+1})[h_c] = I_5$ (removal of I_3 and I_4, addition of $\{I_4, I_9\}$), and another change from $P(t_{i+1})[h_d] = I_5$ to $P(t_{i+1})[h_c] = I_8$ (removal of $\{I_6, I_7\}$, addition of I_{10} and I_{11}).

3. 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 [5] 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. We configure DTRACK to track changes using 8 probes per second, similar to the probing rate used by DIMES [20]. We observed 1,202,960 path changes. The observed paths traverse 7,315 ASes and 97% of the ASes with more than 50 customers [15].[2]

Fig. 1 shows the distribution of the number of hops added in path changes in our dataset. Starting from the previous route, the number of hops added in a change is $h_c - h_d - 1$ and the minimum number of hops we need to measure to build the current route. We see that 78% of changes add two or fewer hops, a small number given that the median route length in our dataset is 16 hops. The most common type of path change is when the interfaces in a single hop change, which results in a path change that removes the old hop and adds the new one. We note that 9% of path changes add zero hops. These changes only remove hops from the previous route (the current route is contained in the previous one). Typical causes for path changes that only remove hops are connectivity failures and measurement errors that prevent measurement of the following hops.

Fig. 2 shows the distribution of the fraction of hops added by a change, i.e., the number of added hops divided by the route length, for all changes in our dataset. Again, we observe that 9% of changes add no hops to the path ($x = 0$); the curve starts at $x = 0.033 = 1/30$ as DTRACK only measures up to 30 hops in a route (the default in Paris traceroute [23]). We see that 80% of path changes add less than 18% of new hops in the path. This shows the potential savings from local change remapping, when compared to the current approach of remapping the whole path.

Fig. 3 shows the distribution of the number of ASes involved in path changes. We consider an AS involved in the change if it contains any interface in the convergence and divergence hops, or any interface in any hop added by the change. We translate interface IP addresses measured by DTRACK to AS numbers combining IP-to-AS mapping databases from Team Cymru[3] and iPlane [16]. Each IP address that does not appear in any database is associated with a different AS number. This is a conservative decision that may overestimate the number of ASes involved in a change. 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. The average

[2]Data sets publicly available at www.dcc.ufmg.br/~cunha/datasets.

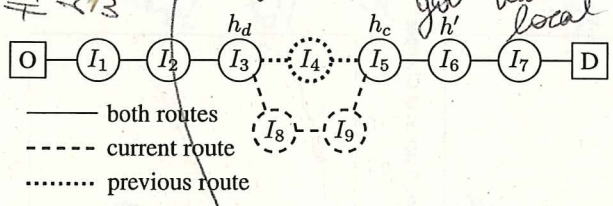

Figure 4: Path change removing I_4 and adding I_8 and I_9.

number of hops inside each AS in a route is 3.04. Combined, these factors further indicate that changes are local and involve few hops.

4. LOCAL REMAP OF PATH CHANGES

DTRACK locally remaps path changes in two phases: first it locates where the change happened (Sec. 4.1) and then remaps it locally (Sec. 4.2). The algorithm receives as input the previous route observed before the path change, $P(t_{i-1})$, and the hop h' where DTRACK detected the change, i.e., $P(t_i)[h'] \neq P(t_{i-1})[h']$. DTRACK measures hops on the current route and compares them with hops on the previous route. DTRACK sends multiple probes to measure a hop, systematically varying the IP header fields to identify all interfaces in branched routes using Paris traceroute's MDA [23].

4.1 Locating path changes

DTRACK 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 hop h' in the current route is not contained in the previous route, i.e., $P(t_i)[h'] \notin P(t_{i-1})$, then h' is 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 hop h', i.e., $P(t_i)[h'] = P(t_{i-1})[h'']$, then we have a path change that added or removed hops to the path upstream of hop h'. Fig. 4 shows an example of one path change where an interface I_4 was removed and interfaces I_8 and I_9 were added. A probe to the sixth hop detects a path change as the answer comes from interface $I_5 \neq P(t_{i-1})[6]$, which is not the expected answer, $I_6 = P(t_{i-1})[6]$.

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

We cannot compare the current route with the previous route if routers at hop h do 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 to 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).

4.2 Remap

Remap starts from a hop h in the current route that does not belong to the previous route, i.e., $P(t_i)[h] \notin P(t_{i-1})$. DTRACK sequentially measures hops in the current route downstream of h until it finds the convergence hop $h_c > h$ 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, DTRACK measures hops until it reaches the destination or up to the last reachable hop. In cases where a route does not reach the destination, DTRACK identifies the last reachable hop after finding three unresponsive hops (just like traceroute).

Similarly, DTRACK sequentially measures hops in the current route upstream of h until it finds the divergence hop $h_d < h$ that belongs to 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 upstream of h_d that added or removed hops to the current route. DTRACK goes back the first phase (Sec. 4.1), making $h' = h_d$, to locate the change and then remap it. This process is repeated recursively until there is no path change to remap.

We need to measure 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 local remap is necessary.

4.3 Local remap example

Consider the path change shown in Fig. 4 is detected with a probe to the sixth hop in the path. We have $P(t_{i-1})[6] = I_6$ and $P(t_i)[6] = I_5$. As $I_5 \in P(t_{i-1})$, a hop was added upstream of the sixth hop and DTRACK starts the binary search to locate the change. DTRACK starts measuring the third hop, where $P(t_{i-1})[3] = P(t_i)[3] = I_3$, indicating that a hop was added between the third and sixth hops. DTRACK then measures the fourth hop, where $P(t_{i-1})[4] = I_4$ and $P(t_i)[4] = I_8$. As $I_8 \notin P(t_{i-1})$, DTRACK starts remapping, measures the fifth hop, and terminates.

5. EVALUATION WITH TRACE-DRIVEN SIMULATIONS

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

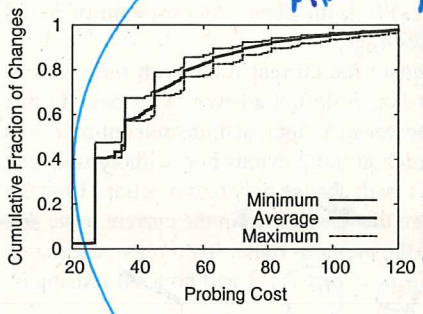

Figure 5: Local remap cost across hops where DTRACK can detect each change.

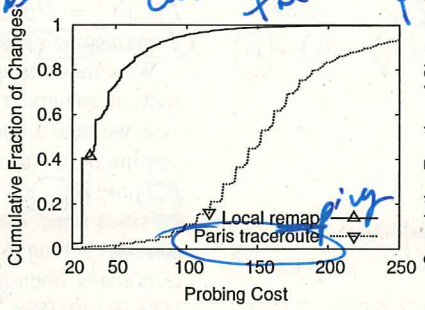

Figure 6: Comparing remap cost between local remap and Paris traceroute.

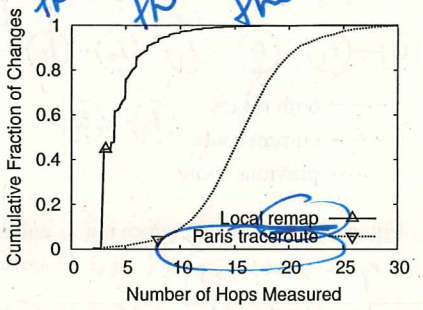

Figure 7: Comparing number of hops measured during remapping.

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

5.1 Remapping cost

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

Fig. 6 compares probing costs when using local remapping and Paris traceroute. Despite the binary search to locate changes and the need to measure both convergence and divergence hops, local remapping has significantly lower costs than Paris traceroute. Fig. 7 compares the number of hops measured by local remapping and Paris traceroute. Comparing with Fig. 1, we see that local remapping frequently measures more hops than the number of added hops, but still only a small fraction of hops in a path (given by the "Paris traceroute" line in Fig. 7). Note that local remapping rarely requires measuring less than three hops, even when 9% of path changes only remove hops from the path (Fig. 1). Local remapping needs to locate where hops were removed and the binary search requires measuring at least three hops unless DTRACK detected the change in the first three hops of the path, which happens only 0.4% of the time in our data.

Fig. 8 shows the distribution of probing cost savings when using local remapping instead of Paris traceroute. We compute probing cost savings as $(C_{\text{Paris}} - C_{\text{local}})/C_{\text{Paris}}$, where C_{Paris} and C_{local} are Paris traceroute and local remapping probing costs, respectively. The solid line, computed for all path changes in our data set, shows that cost savings are significant. Local remapping reduces probing cost by more than half for 88% of path changes. The dashed lines in Fig. 8

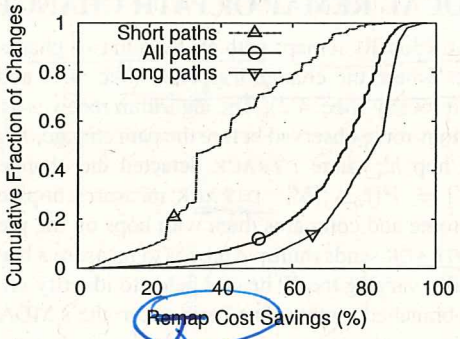

Figure 8: Probing cost savings of local remapping over Paris traceroute.

show probing cost savings for routes shorter than 10 hops (labelled "short") and routes longer than 20 hops (labelled "long"). Local remapping probing cost savings is higher for long routes, where Paris traceroute wastes probes in many hops that are not involved in the change. Also, local remapping brings cost savings even for changes on short routes.

5.2 Remapping errors

DTRACK start remapping from the hop h' where it detected the change. This may lead to inconsistencies if a path undergoes two disjoint changes before we detect the first one. For example, a path $P(t_{i-1}) = [I_1, I_2, I_3, I_4, I_5, I_6, I_7]$ may change into $P(t) = [I_1, I_8, I_3, I_4, I_9, I_6, I_7]$ before we detect a change. In this case, we can detect changes in hops $P(t)[h'] = I_8$ and $P(t)[h''] = I_9$. Unfortunately, given h' or h'', DTRACK will remap only one change.

To evaluate the prevalence of this problem, Fig. 9 shows the distribution of the number of disjoint changes for consecutive path measurements in our data set.[4] We have that 79% of consecutive measurements remap only a single disjoint change, which DTRACK 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 DTRACK will correctly remap the current route

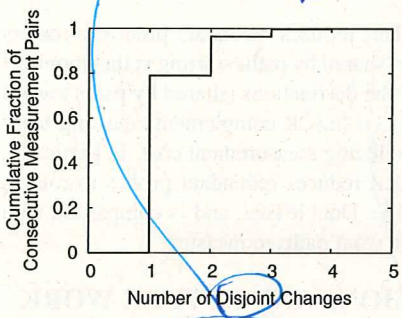

Figure 9: Disjoint path changes between consecutive measurements.

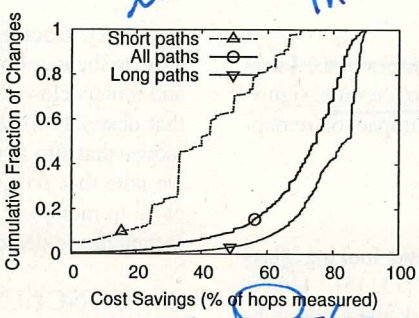

Figure 10: Probing cost savings with local remapping in a real deployment.

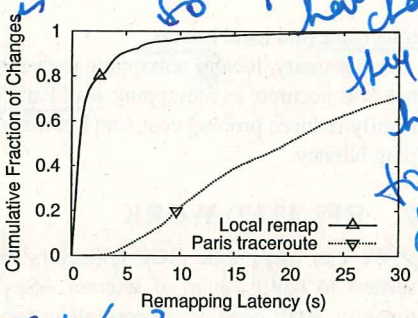

Figure 11: Comparing remap latency latency in the real deployment.

in most cases.

Three other factors contribute to minimize the impact of disjoint changes. First, DTRACK may remap disjoint changes before the detection hop h' if DTRACK detects the change during the remap process. The probability that DTRACK will remap a disjoint change before the detection hop h' in our data set is 43% (not shown). Second, a disjoint change which is not remapped when we run DTRACK will be detected and remapped in the future (assuming the change lasts long enough). Any disjoint change that is not remapped causes only a temporary inconsistency. Third, probing cost savings obtained with local remap can be used to increase probing frequency and reduce the chance that two path changes occur before we detect the first.

Another limitation of local remapping is that the binary search mechanism may fail when the relative order of two hops changes between the previous and current routes. An extreme, but illustrative, example is a change from $P(t_{i-1}) = [I_1, I_2, I_3, I_4, I_5, I_6]$ to $P(t) = [I_1, I_5, I_4, I_3, I_2, I_6]$. Only 0.9% of path changes in our data set 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 local remapping probing cost savings.

6. EVALUATION IN PLANETLAB

In this section we evaluate DTRACK in PlanetLab. We deployed a modified version of DTRACK in 140 PlanetLab nodes and collected measurements for one week starting April 19th, 2014. Our modified DTRACK runs local remapping and Paris traceroute whenever it detects a 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. We observed XXX path changes. The observed paths traverse XXXX ASes and XXX% of the ASes with more than 50 customers [15].

Fig. 10 shows probing cost savings when using local remapping instead of Paris traceroute in the real deployment. Comparing with Fig. 8, the average cost savings in the real deployment is quantitatively similar to the results we obtained via simulation (solid lines). For example, local remapping reduces probing cost by more than half for 85% of path changes in the real deployment. Remap cost savings for short and long routes are also similar to those obtained in the trace-driven simulations.

Fig. 11 shows the distribution of remapping latency for local remapping and Paris traceroute for all changes in our data sets. We study remap latency because local remap measures hops sequentially: the next hop to measure is computed based on the observation at the last measured hop. Traceroute, however, could parallelize probing of different hops. As most local remaps require probing only a few hops (Fig. 6), remapping latency is generally less than 5 seconds. Paris traceroute's remapping latency is significantly higher, as the classic implementation measures hops sequentially. Our objective is not to compare local remap and Paris traceroute remapping latency, as latency is heavily affected by implementation choices. Our objective is to show that local remap latency is acceptable for use in real systems. We also note that DTRACK can execute local remap simultaneously on different paths if more than one path change is detected in a short time interval.

Finally, we also evaluate whether local remaps performed by DTRACK are equivalent to using Paris traceroute to measure the whole path. For each observed change in the real deployment, we compare hops remapped with local remap and the route measured by Paris traceroute. Only 0,6% of local remap measurements differ from Paris traceroute measurements. We attribute this difference to measurement errors. The identification of routers that perform load balancing is probabilistic [23]. For example, Paris traceroute's and DTRACK's default configuration identify all routers performing load balancing in a path with 95% confidence. Estimation errors are inevitable and cause different remaps regardless of the tools used. Another cause for different measurements are path changes that happen while running Paris

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

traceroute and local remap.

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

7. RELATED WORK

We can use public BGP collectors[5] and looking glass servers to build a map of Internet ASes [13, 15]. Unfortunately, BGP does not expose all network links and public BGP collectors have incomplete AS coverage [17]. AS-level topologies provide limited insight about an AS's internal topology. In this work, we take an orthogonal approach and measure the network topology at the interface level using active probes.

Research on topology mapping at the interface 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.

The usual approach to increase network coverage is to monitor a large number of paths. CAIDA's Ark platform [3] attempts to cover the whole Internet using monitors to measure paths toward all /24 prefixes announced in the Internet. Unfortunately, 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 [20] and Dasu [19]. DTRACK can 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 than classic traceroute about the network. 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 [23]. DTRACK also detects routers performing load balancing using the same algorithm as Paris traceroute. Other tools combine traceroute with IP aliasing [11], IP Record Route probes [21], or DNS records [10] to build router-level topologies. These and other techniques send additional probes and thus increase topology mapping cost. DTRACK'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. 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 [22]. Another approach is to reduce the number of destinations and probe only a single 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) [8]. DTRACK complements 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 techniques to decide what paths to measure.

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

8. CONCLUSIONS AND FUTURE WORK

shouldn't these algorithms be discussed explicitly in section 4?

Algorithm 1: Local remap ~~startup~~ ping

Input: $P(t_{i-1})$ ← previous route
Input: h' ← detection hop, $P(t_i)[h'] \neq P(t_{i-1})[h']$
if $P(t_i)[h'] \in P(t_{i-1})$ **then**
  binary(0, h')
else
  remap(h')
end

Algorithm 2: Binary search algorithm (§ 4.1)

Input: h_{left} ← left-end of binary search, $P(t_i)[h_{\text{left}}] = P(t_{i-1})[h_{\text{left}}]$
Input: h_{right} ← right-end of binary search, $P(t_i)[h_{\text{right}}] \in P(t_{i-1})$
if $h_{\text{left}} \geq h_{\text{right}} - 1$ **then**
  return
end
$h_{\text{center}} \leftarrow (h_{\text{left}} + h_{\text{right}})/2$
measure(h_{center})
if $P(t_i)[h_{\text{center}}] \notin P(t_{i-1})$ **then**
  remap(h_{center})
else
  if $P(t_i)[h_{\text{center}}] = P(t_{i-1})[h_{\text{center}}]$ **then**
    binary(h_{center}, h_{right})
  else
    binary(h_{left}, h_{center})
  end
end

Algorithm 3: Remap algorithm (§ 4.2)

Input: h' ← remap location, $P(t_i)[h'] \notin P(t_{i-1})$
$h_c \leftarrow h'$ *// convergence hop*
$h_d \leftarrow h'$ *// divergence hop*
while $P(t_i)[h_c] \notin P(t_{i-1})$ **do**
  $h_c \leftarrow h_c + 1$
  measure(h_c)
end
while $P(t_i)[h_d] \notin P(t_{i-1})$ **do**
  $h_d \leftarrow h_d - 1$
  measure(h_d)
end
if $P(t_i)[h_d] \neq P(t_{i-1})[h_d]$ **then**
  binary(0, h_d)
end

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

[4] 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.

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

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

[7] 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.

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

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

[10] A. D. Ferguson, J. Place, and R. Fonseca. Growth Analysis of a Large ISP. In *Proc. IMC*, 2013.

[11] S. Garcia-Jimenez, E. Magana, D. Morato, and M. Izal. Pamplona-traceroute: Topology Discovery and Alias Resolution to Build Router Level Internet Maps. In *Global Information Infrastructure Symposium*, 2013.

[12] 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.

[13] A. Khan, T. Kwon, H. Kim, and Y. Choi. AS-level Topology Collection Through Looking Glass Servers. In *Proc. IMC*, 2013.

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

[15] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and K. Claffy. AS Relationships, Customers Cones, and Validations. In *Proc. IMC*, 2013.

[16] 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.

[17] 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.

Overall, I have a problem with the style. You use passive voice a lot and sentences are often in negative form. You should read the Elements of Style again.

[18] V. Paxson. End-to-end Routing Behavior in the Internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.

[19] M. Sánchez, J. Otto, Z. Bischof, D. Choffnes, F. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing Experiments to the Internet's Edge. In *Proc. USENIX NSDI*, 2013.

[20] Y. Shavitt and U. Weinsberg. Quantifying the Importance of Vantage Points Distribution in Internet Topology Measurements. In *Proc. IEEE INFOCOM*, 2009.

[21] R. Sherwood, A. Bender, and N. Spring. DisCarte: a Disjunctive Internet Cartographer. In *Proc. ACM SIGCOMM*, 2008.

[22] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, 2002.

[23] D. Veitch, B. Augustin, T. Friedman, and R. Teixeira. Failure Control in Multipath Route Tracing. In *Proc. IEEE INFOCOM*, 2009.