Efficient Discovery of Load-Balanced Paths

Alistair King

al@bellstreet.co.nz

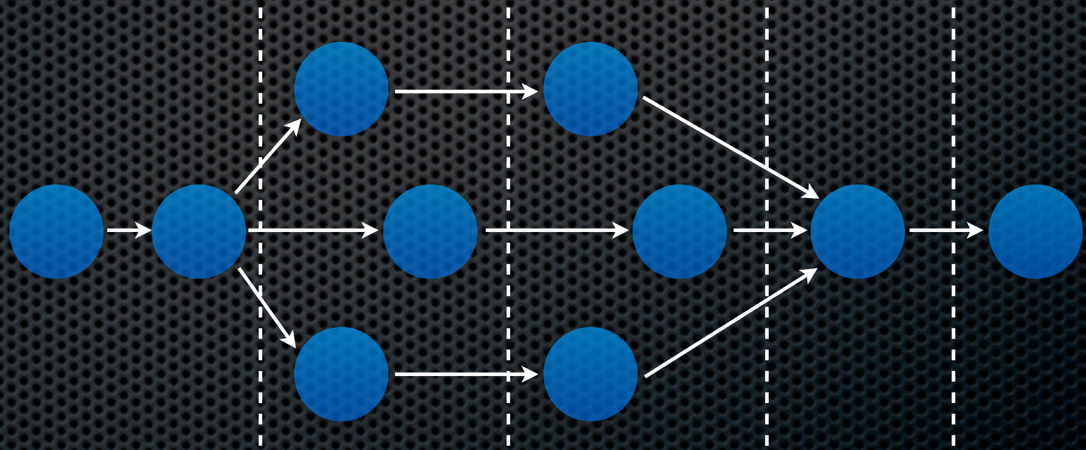
Load-Balancer Traceroute

- Gives confidence that the complete topology has been discovered.
- Probes each TTL repeatedly to discover alternate interfaces.
- Uses a large number of probes (and therefore a long amount of time)

Load-Balancer Traceroute

- E.g. To discover, with 99% confidence, a hypothetical path with 5 hops and 9 interfaces, >= 74 probes would be sent. Compared with >= 5 probes for regular traceroute

TTL: 1 2 3 4 5



Probes: 8 + 29 + 21 + 8 + 8 = 74

Doubletree traceroute

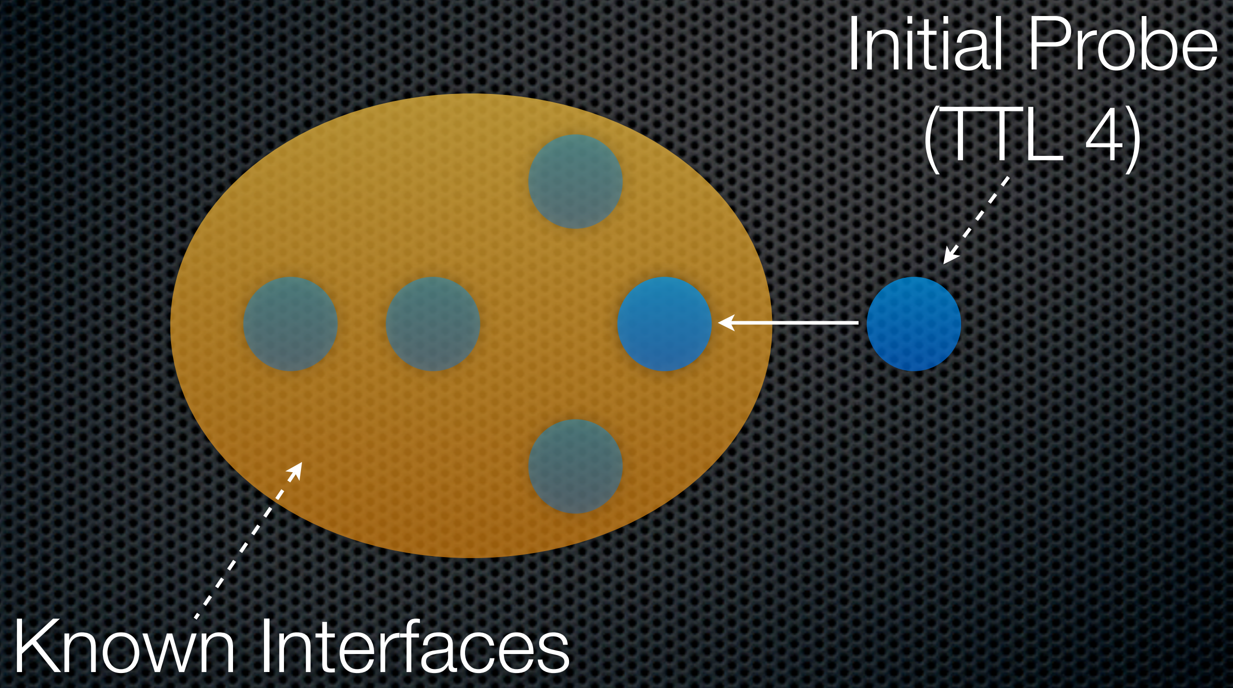
- More efficient version of conventional traceroute algorithm.
- Assumes Internet paths form trees.
- Probes forward from a midpoint in the path until a known interface or the destination is reached.
- Probes backward from the midpoint, again until a known interface or the beginning of the path is reached.
- Empirical testing of scamper doubletree implementation gives 95% coverage whilst sending 77% fewer probes.

Doubletree Load-Balancer

- Developed and tested last year as part of my honors research.
- Goal is to reduce the number of probes that load-balancer traceroute uses.
- Uses a doubletree-like algorithm to determine an initial TTL for the load-balancer traceroute algorithm to probe from.

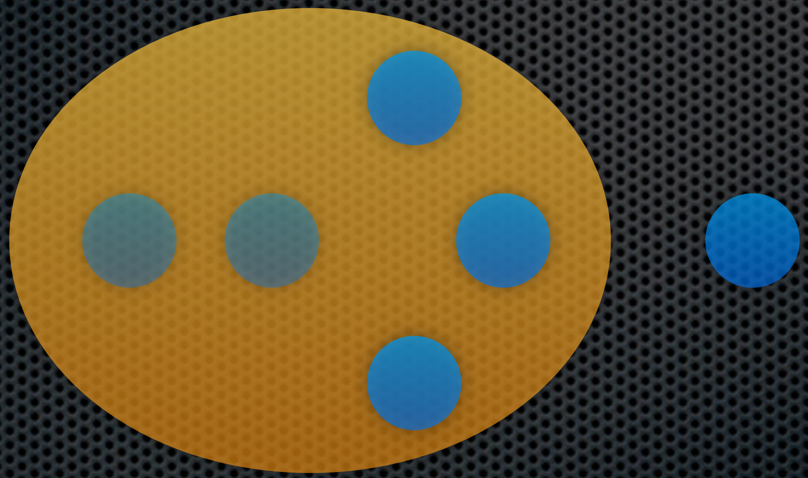
Doubletree Load-Balancer

- Probes backward from a midpoint until a known hop is discovered.



Doubletree Load-Balancer

- Probes known hop multiple times to discover alternate interfaces.

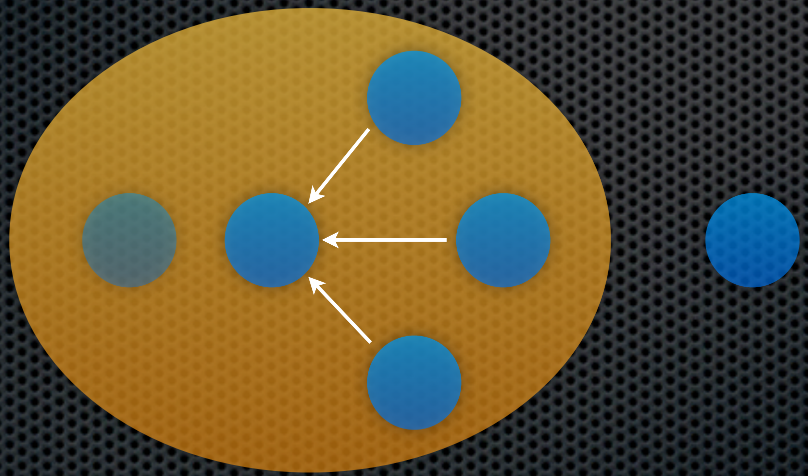


Sends 21 probes to TTL 3

Discovers 2 alternate interfaces

Doubletree Load-Balancer

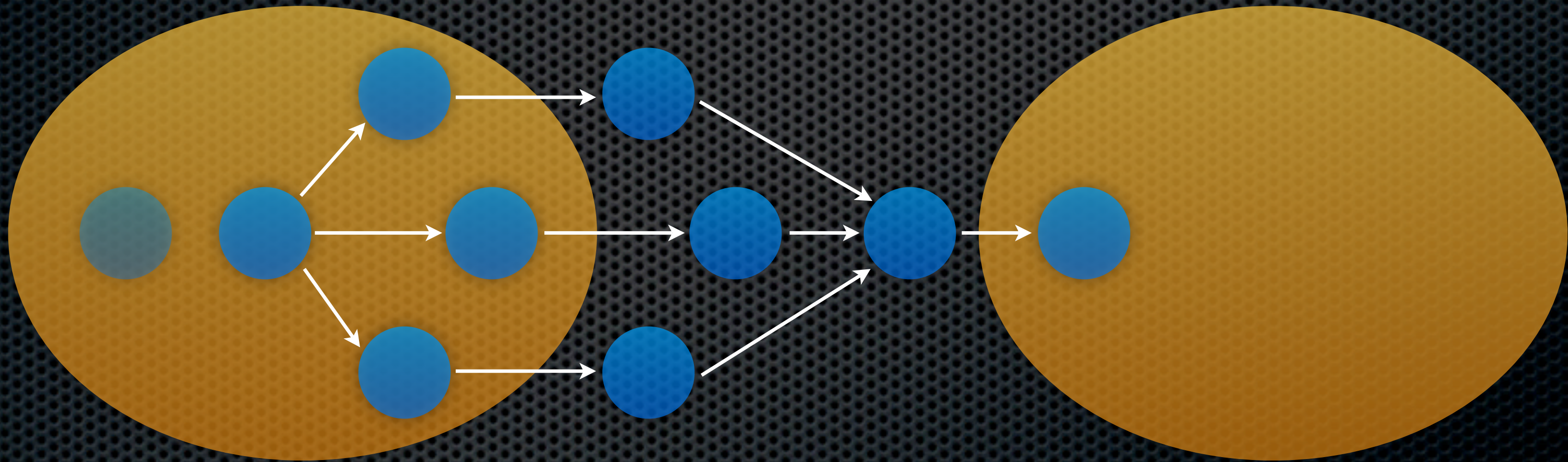
- Follows the paths from each alternate interface back to their convergence point.



- This ensures that no alternate paths are missed

Doubletree Load-Balancer

- Probes forward using load-balancer traceroute until a known interface is discovered.



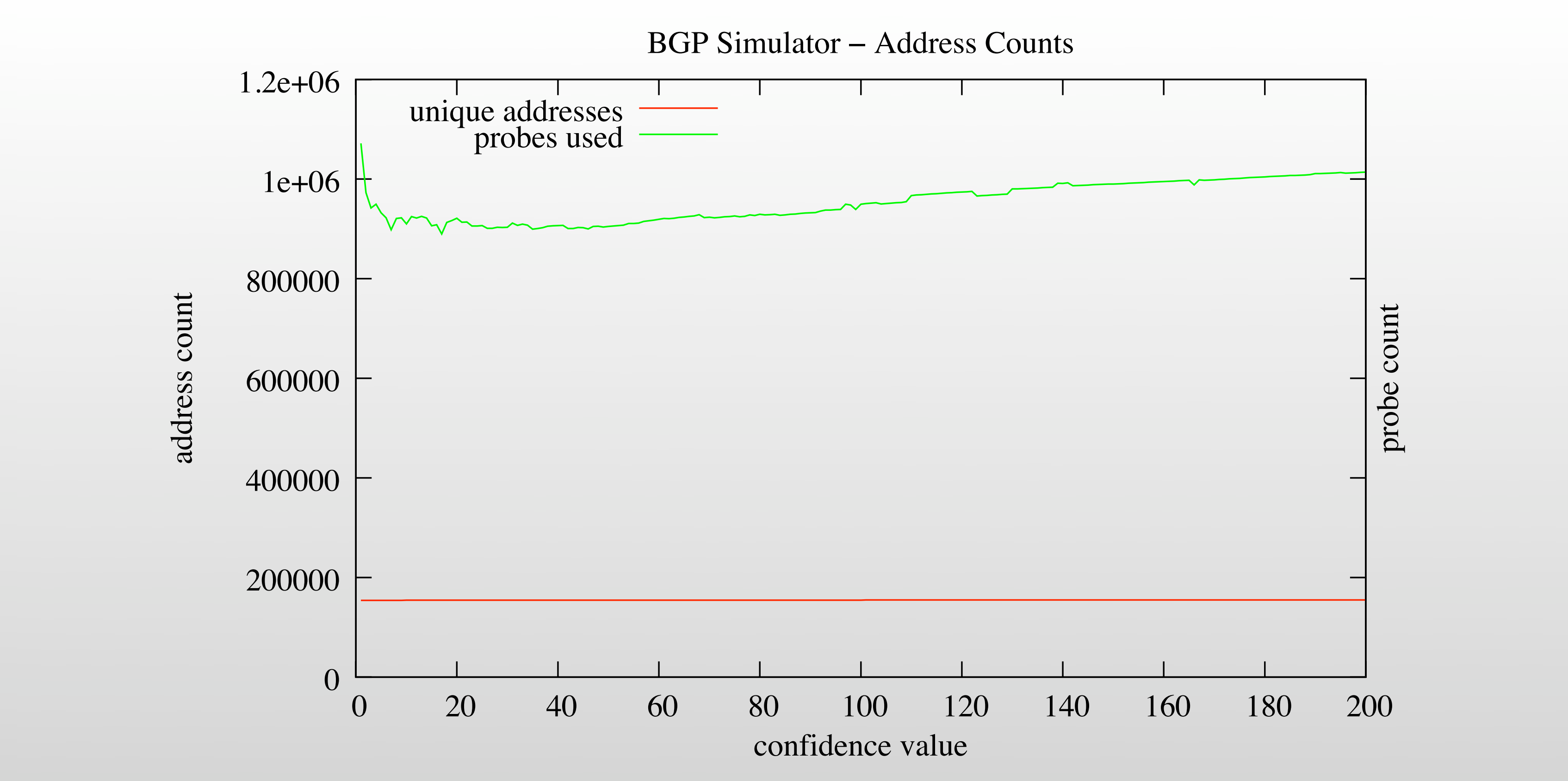
- Testing shows that dtlb maintains 96% coverage compared to tracelb, whilst sending 31% fewer probes

Doubletree Load-Balancer

- Analysis shows that most missed links are probably due to per-destination load-balancing, causing alternate paths to be missed.
- Idea is to keep state about how many paths an interface is seen in when probing different destinations.
- Only stop probing at an interface if it has been seen in the path to x (8?) destinations.

BGP Influenced Probing

- Goal is to use knowledge of BGP routes to infer which segments of a path have not been probed before.
- Used BGP tables to map interfaces to AS number.
- Whilst probing, keep track of the average size of each AS.
- Once an AS has been transited enough times to have an accurate average width, use that width to allow future traces to skip over the AS.
- Used existing Ark data to run simulations.



- What is the optimal number of times to transit an AS in order to minimize probe numbers and maximize address count?
- Makes little difference to the number of addresses discovered
- ~20 times appears to be optimal in terms of probes used

Doubletree and Ark

- Developing a set of scripts which will allow large-scale doubletree measurements to be made using the Ark infrastructure.
- Doubletree is implemented in scamper trunk
- Use Marinda to drive multiple doubletree monitors and allow them to share their stop set data with each other

Algorithm Simulator

- Concept: A tracing simulator which allows new algorithms to be quickly and easily tested.
- Written in Ruby to allow a quick development cycle and dynamic addition of new algorithms.
- Uses Ark trace data to drive the simulator.
- Currently have trace and doubletree algorithm plugins
- Simulated performance is very close to real-world performance

Where to from here?

- Investigate dtlb per-destination improvements mentioned earlier.
- Re-investigate doubletree load-balancer to determine where other improvements can be made.
- Continue investigations into using BGP to direct probing.
- Complete implementation of doubletree ark coordination scripts.
- Conduct large-scale measurements using doubletree

Questions, Suggestions?