In the IP of the Beholder: Strategies for Active IPv6 Topology Discovery

Robert Beverly[*], Ram Durairajan[†], David Plonka[‡], Justin Rohrer[*]

[*]Naval Postgraduate School
[†]University of Oregon
[‡]Akamai Technologies

October 31, 2018

ACM Internet Measurement Conference 2018

Outline

What We Did

Performed large-scale topological survey of the Internet using IPv6

- Evaluated ability of IPv6 hitlists to produce targets
- Utilized a new traceroute technique
- Analyzed results (1.4M discovered router addresses):
  - IPv6 subnetting
  - Privacy implications

How to map the router-level IPv6 Internet?

What We Did

Performed large-scale topological survey of the Internet using IPv6

- Evaluated ability of IPv6 hitlists to produce targets
- Utilized a new traceroute technique
- Analyzed results (1.4M discovered router addresses):
  - IPv6 subnetting
  - Privacy implications

How to map the router-level IPv6 Internet?

What We Did

Performed large-scale topological survey of the Internet using IPv6

- Evaluated ability of IPv6 hitlists to produce targets
- Utilized a new traceroute technique
- Analyzed results (1.4M discovered router addresses):
  - IPv6 subnetting
  - Privacy implications

How to map the router-level IPv6 Internet?

But wait, *decades* of experience with active topology mapping!

IPv6-Specific Challenges:

1. Massive address space that is sparsely populated
   → *What* to probe?
2. Mandated ICMPv6 rate limiting
   → *How* to send probes?

This work seeks to make progress against both challenges, and increase coverage/fidelity of IPv6 Internet router topologies.

But wait, *decades* of experience with active topology mapping!

IPv6-Specific Challenges:

1. Massive address space that is sparsely populated
   $\rightarrow$ *What* to probe?
2. Mandated ICMPv6 rate limiting
   $\rightarrow$ *How* to send probes?

This work seeks to make progress against both challenges, and increase coverage/fidelity of IPv6 Internet router topologies.

But wait, *decades* of experience with active topology mapping!

IPv6-Specific Challenges:

1. Massive address space that is sparsely populated
   $\rightarrow$ *What* to probe?
2. Mandated ICMPv6 rate limiting
   $\rightarrow$ *How* to send probes?

This work seeks to make progress against both challenges, and increase coverage/fidelity of IPv6 Internet router topologies.

Outline

State-of-the-art:

- CAIDA (Ark) and RIPE (Atlas) continually collect IPv6 topologies via active probing
- Technique and tools of these production systems mirror IPv4
  - For each IPv6 prefix in global BGP table,
  - sequentially traceroute to:
    - `::1` in prefix
    - random address in prefix

Question:

Current production IPv6 active topology mapping systems probe an address in each globally advertised prefix. While this strategy provides breadth, does it miss subnetting and other topological structure?

Hitlists:

- We compare this approach to using existing collections of known IPv6 hosts, or *hitlists* as targets

Question:

Current production IPv6 active topology mapping systems probe an address in each globally advertised prefix. While this strategy provides breadth, does it miss subnetting and other topological structure?

Hitlists:

- We compare this approach to using existing collections of known IPv6 hosts, or *hitlists* as targets

Using Hitlists

Name	Method	Date	Addrs
CAIDA	BGP-derived	2018/05/09	105.2k
DNSDB	Passive DNS	2018/02/15 – 04/28	5.4M
Fiebig		2018/03/27	11.7M
FDNS		2018/04/27	24.8M
CDN Clients		2018/02/18 – 03/03	N/A
6gen		2018/03/13	4.9M
TUM*		varies	5.6M
Random	Random Routed	2018/05/23	26.5M
Combined	Join Sets	varies	50.8M

- Lots of recent work on developing / gathering IPv6 hitlists

Using Hitlists

Name	Method	Date	Addrs
CAIDA	BGP-derived		
DNSDB	Passive DNS		
Fiebig	Reverse DNS		
FDNS	Fwd. DNS		
CDN Clients	*k*IP anonymization		
6gen	Generative		
TUM*	Collection		
Random	Random Routed		
Combined	Join Sets	[illegible]	[illegible]

Many IPv6 Hitlists

- "CAIDA" (BGP) is baseline for today's systems
- "Random" is baseline for unguided probing
- Wide variety of methods

Using Hitlists

Name	Method	Date	Addrs
CAIDA	BGP-deri		105.2k
DNSDB	Passive D		5.4M
Fiebig	Reverse D		11.7M
FDNS	Fwd. DN		24.8M
CDN Clients	*k*IP anonymi		N/A
6gen	Generati		4.9M
TUM*	Collectio		5.6M
Random	Random R		26.5M
Combined	Join Se		50.8M

Many IPv6 Hitlists

- Composition varies widely
- Primarily focused on end hosts
- → Targets in some hitlists concentrated in small number of prefixes / ASes

Using Hitlists

Name	Method	Date	Addrs
CAIDA	BGP-deri		105.2k
DNSDB	Passive D		5.4M
Fiebig	Reverse D		11.7M
FDNS	Fwd. DN		24.8M
CDN Clients	*k*IP anonymi		N/A
6gen	Generati		4.9M
TUM*	Collectio		5.6M
Random	Random Ro		26.5M
Combined	Join Se		50.8M

Many IPv6 Hitlists

- Composition varies widely
- Primarily focused on end hosts
- → Targets in some hitlists concentrated in small number of prefixes / ASes

How can hitlists inform active IPv6 topology mapping?

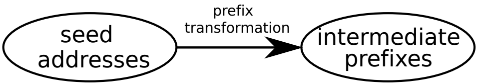
We develop a generalized method for *generating* targets from “seeds”

Target Generation


```
2607:5300::1029
2607:5300::109f
2607:5300::102a
2a07:18e8:4005:80b:e3ae::200e
2a07:18e8:4005:80b:87e8::400a
```

1. Begin with seeds: hitlist addresses

Target Generation



`2607:5300::1029`
`2607:5300::109f`
`2607:5300::102a`
		`2607:5300::/64`

`2a07:18e8:4005:80b:e3ae::200e` `2a07:18e8:4005:80b:87e8::400a`	$\xrightarrow{z64}$	`2a07:18e8:4005:80b::/64`

1. Begin with seeds: hitlist addresses
2. *zn* aggregation: Group addresses into prefixes of length *n*

Q: What aggregation granularity?

zn	Packets	Other ICMPv6	Router Addrs
/40	1.4M	17.5k	27.0k
/48	3.6M	105.8k	45.5k
/56	6.1M	194.8k	60.5k
/64	11.8M	486.8k	85.5k

Evaluate parameter impact:

- Packets (cost)
- Router addresses discovered (benefit)
- Collateral impact as non-TTL exceeded responses (cost)

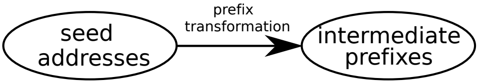
Q: What aggregation granularity?

zn	Packets	Other ICMPv6	Router Addrs
/40	1.4M	17.5k	27.0k
/48	3.6M	105.8k	45.5k
/56	6.1M	194.8k	60.5k
/64	11.8M	486.8k	85.5k

Evaluate parameter impact:

- /64 has highest cost, but most benefit
- /48 strikes a balance
- We perform full probing with both z64 and z48

Target Generation




```
2607:5300::1029                          2607:5300::/64
2607:5300::109f
2607:5300::102a
2a07:18e8:4005:80b:e3ae::200e    z64 →   2a07:18e8:4005:80b::/64
2a07:18e8:4005:80b:87e8::400a
```

1. Begin with seeds: hitlist addresses
2. *zn* aggregation: Group addresses into prefixes of length *n*

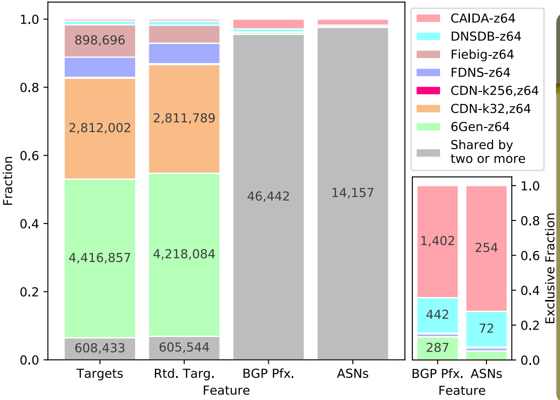
Target Generation

2607:5300::1029	2607:5300::/64		2607:5300:::1234:5678
2607:5300::109f			
2607:5300::102a			
2a07:18e8:4005:80b:e3ae::200e	2a07:18e8:4005:80b::/64	→	2a07:18e8:4005:80b::1234:5678
2a07:18e8:4005:80b:87e8::400a			

1. Begin with seeds: hitlist addresses
2. *zn* aggregation: Group addresses into prefixes of length *n*
3. Targets are synthesized with interface identifier

In this example, 5 seed addresses are used to generate 2 targets

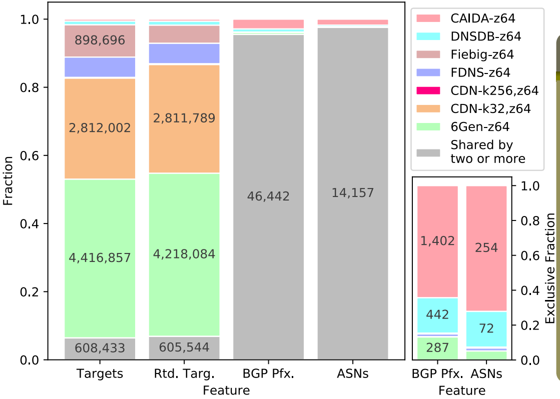
Q: How do Target Sets Compare?



Portion in Each Target Set

- Color: unique, Gray: shared
- "Rtd Targ": Not all targets routed
- While many targets are unique, significant prefix/AS overlap

Q: How do Target Sets Compare?



Coverage

- Inset: Non-trivial numbers of prefixes / ASes that exist in only one target set
- Intuition: increasing coverage in targets increases coverage in topology results

Outline

Strategies for increasing coverage

- Select better destinations (hitlists)
- Probe more destinations → probe faster

Probing faster:

- RFC4443, §2.1.1: "*an IPv6 node MUST limit the rate of ICMPv6 error messages it originates*"
- Implemented with a token bucket

State-of-the-art

- Production: e.g., CAIDA and RIPE
  - "Sequential" (i.e. TTL=1,2,. . .)
  - Limited parallelism (i.e. waiting for responses, window of destinations)
  - Probing faster can be self-defeating: triggers more rate-limiting

Question:

How to probe in IPv6 to minimize effect of rate-limiting, while maintaining complete probing?

State-of-the-art

- Production: e.g., CAIDA and RIPE
  - “Sequential” (i.e. TTL=1,2,. . .)
  - Limited parallelism (i.e. waiting for responses, window of destinations)
  - Probing faster can be self-defeating: triggers more rate-limiting

Question:

How to probe in IPv6 to minimize effect of rate-limiting, while maintaining complete probing?

Yarrp: "Yelling at Random Routers Progressively" (IMC2016)

- Uses a block cipher to **randomly permute** the $\langle IP, TTL \rangle$ domain
- Is **stateless**, recovering necessary information from replies
- By randomly spreading probes in time/space, permits **fast** Internet-scale active topology probing

Yarrp6

- We extend Yarrp to support IPv6
- And add IPv6-specific enhancements
- Hypothesis: Yarrp-mapping of the IPv6 Internet will suffer less rate-limiting, even at higher probing rates

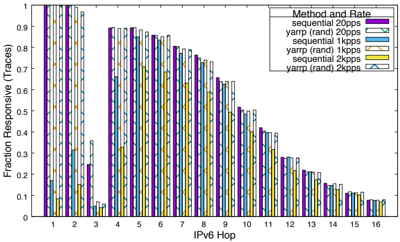
Yarrp: "Yelling at Random Routers Progressively" (IMC2016)

- Uses a block cipher to **randomly permute** the $\langle IP, TTL \rangle$ domain
- Is **stateless**, recovering necessary information from replies
- By randomly spreading probes in time/space, permits **fast** Internet-scale active topology probing

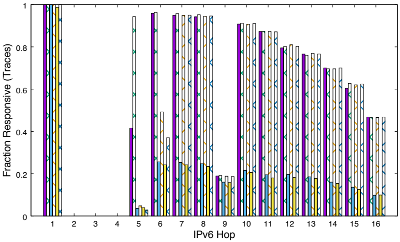
Yarrp6

- We extend Yarrp to support IPv6
- And add IPv6-specific enhancements
- Hypothesis: Yarrp-mapping of the IPv6 Internet will suffer less rate-limiting, even at higher probing rates

Comparison of Sequential vs. Yarrp Probing



US-EDU-3



US-EDU-2

- Same targets, same vantage point
- Varied probing rate (20-2kpps)
- Yarrp outperforms sequential, especially near source and as rate increases
- Some hops exhibit different rate-limiting behavior

What about techniques to avoid re-probing initial hops?

- e.g., DoubleTree, also designed for Internet-scale topology probing:
    - Probes backward until it receives a response from a known hop
    - Does not probe complete path, infers missing hops (can be wrong)
- We find that DoubleTree performs better than sequential
- But, rate-limiting (missed responses) causes DoubleTree to continue to probe backward (feedback loop)

Fill Mode

Yarrp is stateless

- Must select TTL range (*maxTTL*) (potentially missing hops)
- Don't know when to stop probing (potentially wasting probes)

Fill mode:

For response to probe with TTL=h, immediately probe w/ TTL=$h+1$ if $h \geq maxTTL$.

- Not random, but uncommon and at path tail
- Win/win efficiency gain: Allows us to lower the *maxTTL* (less wasted probing), without missing hops.

Fill Mode

Yarrp is stateless

- Must select TTL range (*maxTTL*) (potentially missing hops)
- Don't know when to stop probing (potentially wasting probes)

Fill mode:

For response to probe with TTL=h, immediately probe w/ TTL=$h+1$ if $h \geq maxTTL$.

- Not random, but uncommon and at path tail
- Win/win efficiency gain: Allows us to lower the *maxTTL* (less wasted probing), without missing hops.

Outline

Probing

- Single runs: May 14, 2018
- 3 vantage points: 2 US Universities; 1 EU Network
- 18 different target sets
- Yarrp6 w/ TTL=16 and fillmode
- ICMPv6 probes
- 2kpps

Ethical Considerations

- Followed good "Internet citizenship" guidelines
- Received two-opt outs (someone's actually monitoring IPv6!)

Probing

- Single runs: May 14, 2018
- 3 vantage points: 2 US Universities; 1 EU Network
- 18 different target sets
- Yarrp6 w/ TTL=16 and fillmode
- ICMPv6 probes
- 2kpps

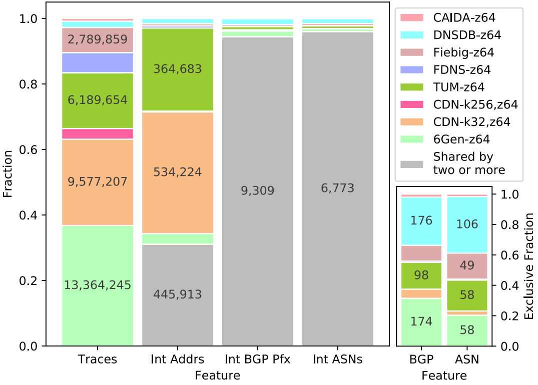
Ethical Considerations

- Followed good “Internet citizenship” guidelines
- Received two-opt outs (someone’s actually monitoring IPv6!)

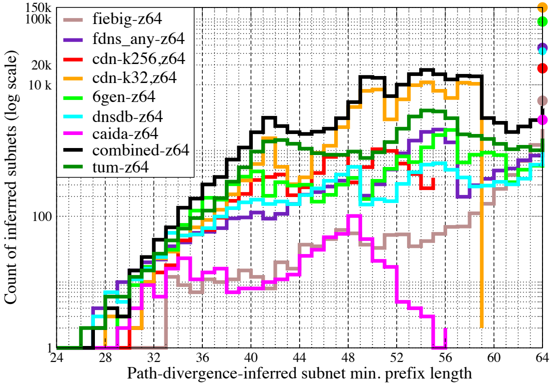
Macro Results

- 45.8M traces to 12.5M destinations (in less than a day)
- Discover 1.4M IPv6 router addresses
- Order of magnitude more than prior efforts
- Including ∼0.6M EUI64 addresses (45%!)

Features of discovered Interface Addresses (all VPs, *z64*)

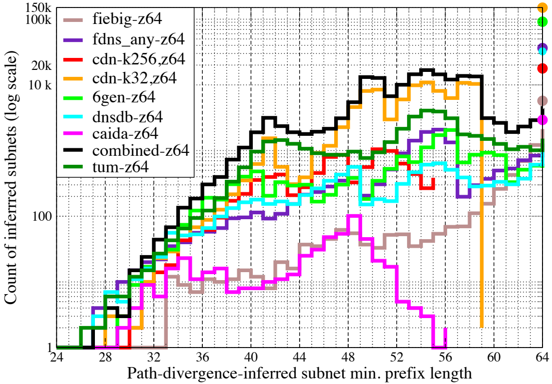


- $\sim$ 70% of interface addresses discovered only via single target set
- 100's of prefixes and ASes only discovered via single target set
- Thus, target sets are *complementary*



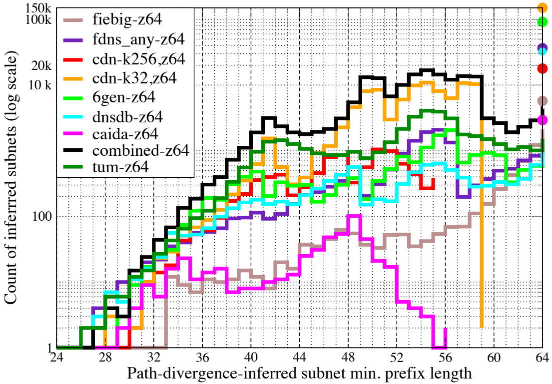
Subnet Discovery

- Anecdotal evidence: wide variety of production IPv6 subnetting practices
- Subnets important to how IPv6 is being used, geolocation, reputation, etc.
- Inspired by Lee et al. , developed a method using traces to find subnetting



Subnet Discovery

- Peaks at /40, /48
- CAIDA has fewest subnet and largest subnets
- Many more subnets, and more granular subnets discovered using CDN, TUM targets



Subnet Discovery

- Seeds with high-clustering (e.g. Fiebig) discover primarily small subnets
- Ability to discover subnets constrained by target sets' DPL (see paper for details)

EUI64

Unanticipated Result

- EUI64 embeds a device's H/W MAC into its IPv6 address
- For privacy reasons, most OSes use ephemeral random addresses instead
- Surprisingly, across 45.8M traces, discover 651.4k EUI64 addresses (45% of all addresses!)

EUI64

Unanticipated Result

- EUI64 embeds a device's H/W MAC into its IPv6 address
- For privacy reasons, most OSes use ephemeral random addresses instead
- Surprisingly, across 45.8M traces, discover 651.4k EUI64 addresses (45% of all addresses!)

Implications to Security and Privacy (RFC7721)

- Primarily at the end of the path (CPE!)
- Concentrated among providers and manufacturers
- Working with community to address
- (E.g., next week at IETF maprg WG)

Summary

- Studied *where* and *how* to send IPv6 topology probes
  - Using hitlists to generate targets
  - Yarrp6 to probe
- Inferred IPv6 subnetting and structure
- Step toward more complete IPv6-level router topologies
- Working within IETF to address privacy aspects of EUI64 infrastructure addresses
- Working toward production deployment within CAIDA

Thanks! – Questions?

https://www.cmand.org/yarrp

Summary

- Studied *where* and *how* to send IPv6 topology probes
  - Using hitlists to generate targets
  - Yarrp6 to probe
- Inferred IPv6 subnetting and structure
- Step toward more complete IPv6-level router topologies
- Working within IETF to address privacy aspects of EUI64 infrastructure addresses
- Working toward production deployment within CAIDA

Thanks! – Questions?

`https://www.cmand.org/yarrp`