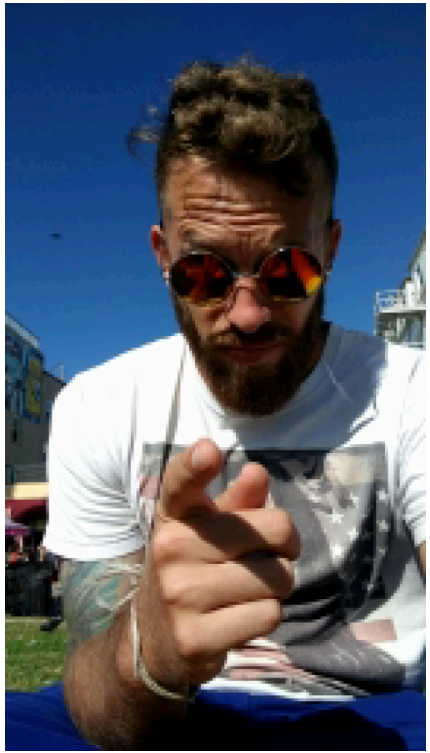
IPv666
Address of the **Beast**

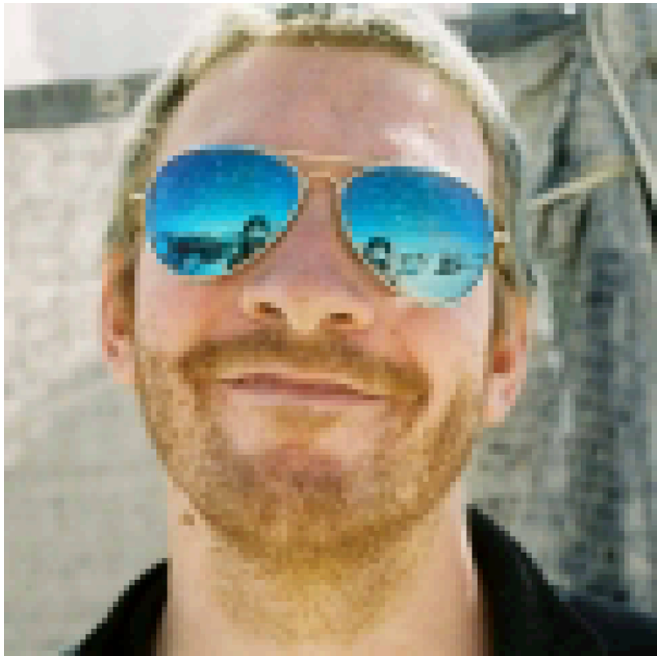
Chris Grayson
@_lavalamp

Marc Newlin
@marcnewlin

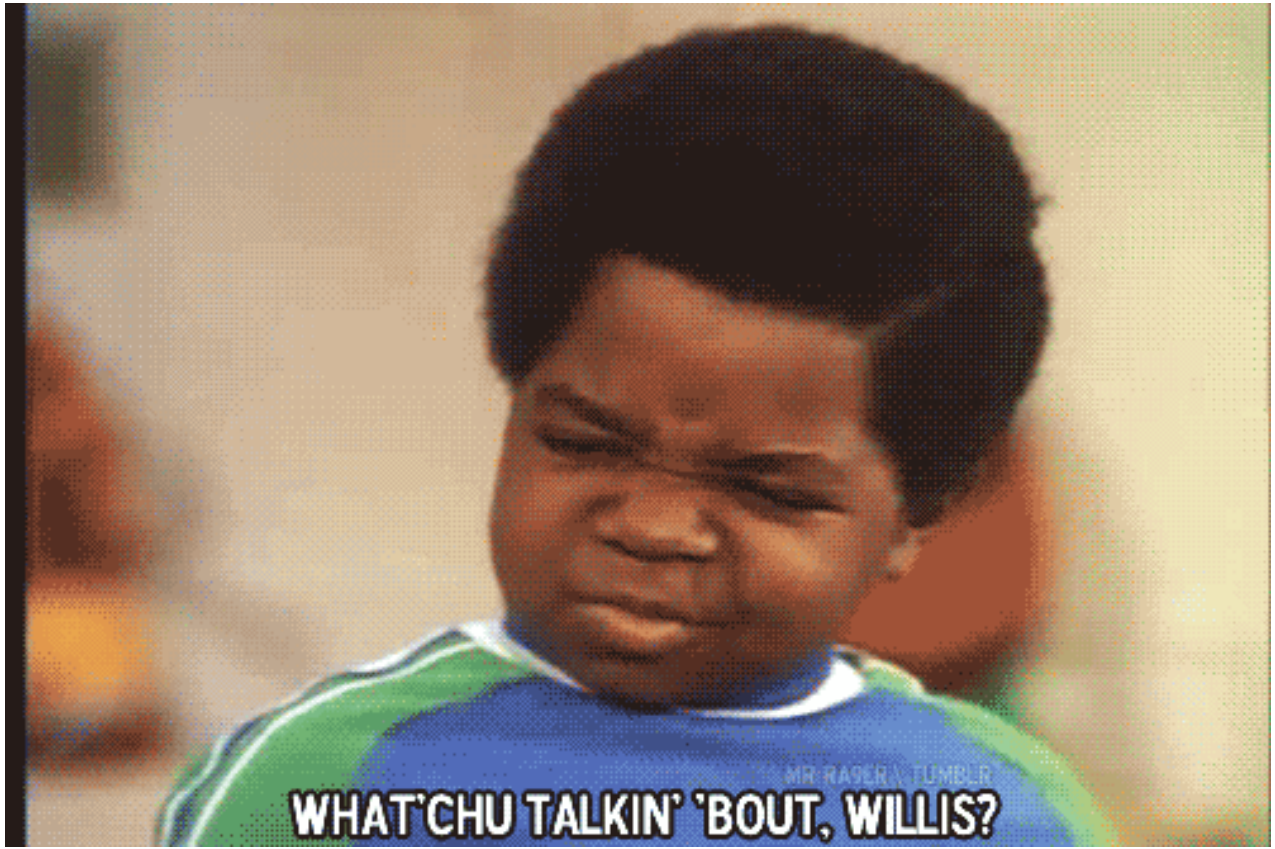
MY NAME IS



CHRIS



MARC


MR RASER | TUMBLR
WHAT'CHU TALKIN' 'BOUT, WILLIS?

IT’S THE FUTURE BAYBEEEEEE



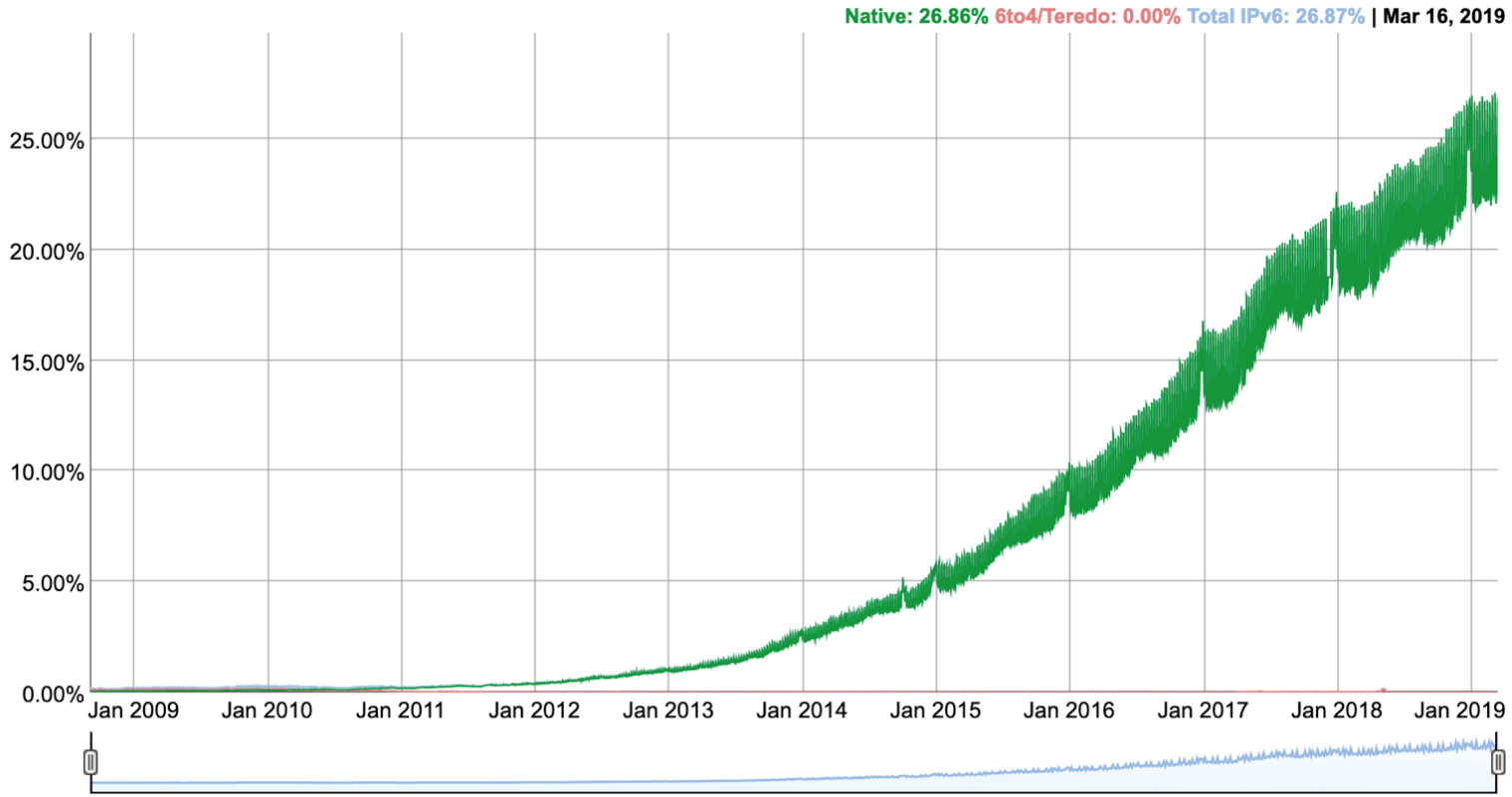
Where to From Here Cap’n?

- Background
- The Scanning Problem
- Honeypotting for PSLAAC
- Modeling for non-PSLAAC
- Being Less Dumb
- To The Cloud
- Results
- IPv666
- Conclusion



A Bit o’ Background

IPv6 - So Hot Right Now



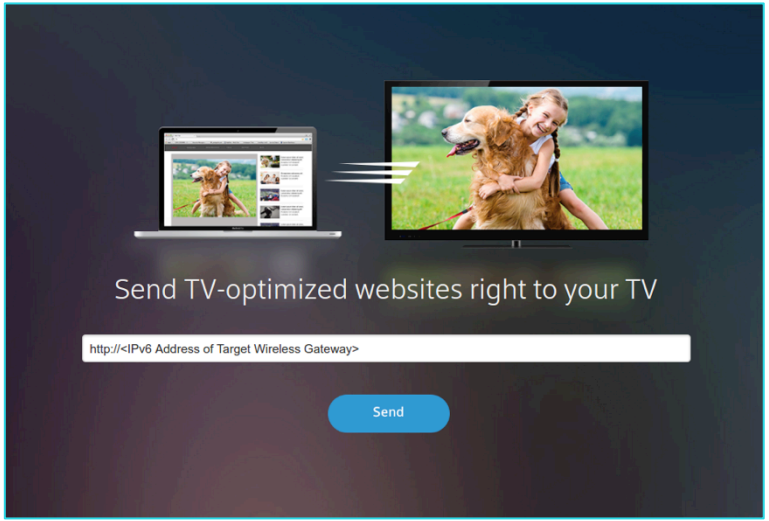
A Tale of Gateways and Set Top Boxes



- DEFCON 25
- 26 CVEs
- All in consumer premise equipment (CPE)
- Ie: Comcast gateways and set top boxes
- IPv6 was a big part
- https://github.com/BastilleResearch/CableTap

Send-to-TV / Remote Web UI

- Gateway web UI accepts remote requests from ISP infrastructure
- IPv6 address of target gateway provides remote web UI access via set-top box



Seems Kinda Spooky

- IPv6 works out of the box without manual configuration
- All your devices and networking equipment prefer it
- There’s no such thing as private address space (for the most part)
- Your IPv4 firewall rules don’t apply
- ICMPv6 is a critical protocol
- Single packets can be relayed to lots and lots of hosts



Let’s Go Hunting

The Problem of Scale

- IPv4
  32 bit addresses
  2^32 possible addresses

  4,294,967,296 addresses



- IPv6
  128 bit
  2^128 possible addresses

  340,282,366,920,938,463,463,374,607,431,768,211,456 addresses

PSLAAC Makes Things Harder

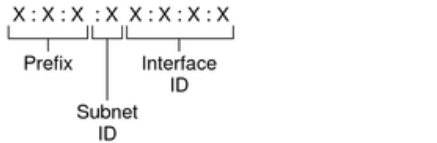
Request for Comments: 4941 — IBM Corporation
Obsoletes: 3041 — R. Draves
Category: Standards Track — Microsoft Research
S. Krishnan
Ericsson Research
September 2007

Privacy Extensions for Stateless Address Autoconfiguration in IPv6

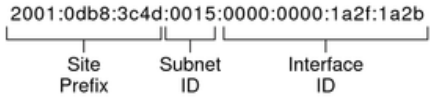
Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Figure 3-2 Basic IPv6 Address Format



Example:



- Addresses have host and network bits
- Network is site prefix and subnet ID
- Host is interface ID
- PSLAAC means interface ID is pseudorandom
- Even “small” networks (/96) have 32 bits of randomness

Breaking Down the Problem

- Modeling for cryptographic entropy is no bueno
- Two independent problems instead
    - Identifying PSLAAC hosts
    - Identifying non-PSLAAC hosts



Honeypotting for PSLAAC

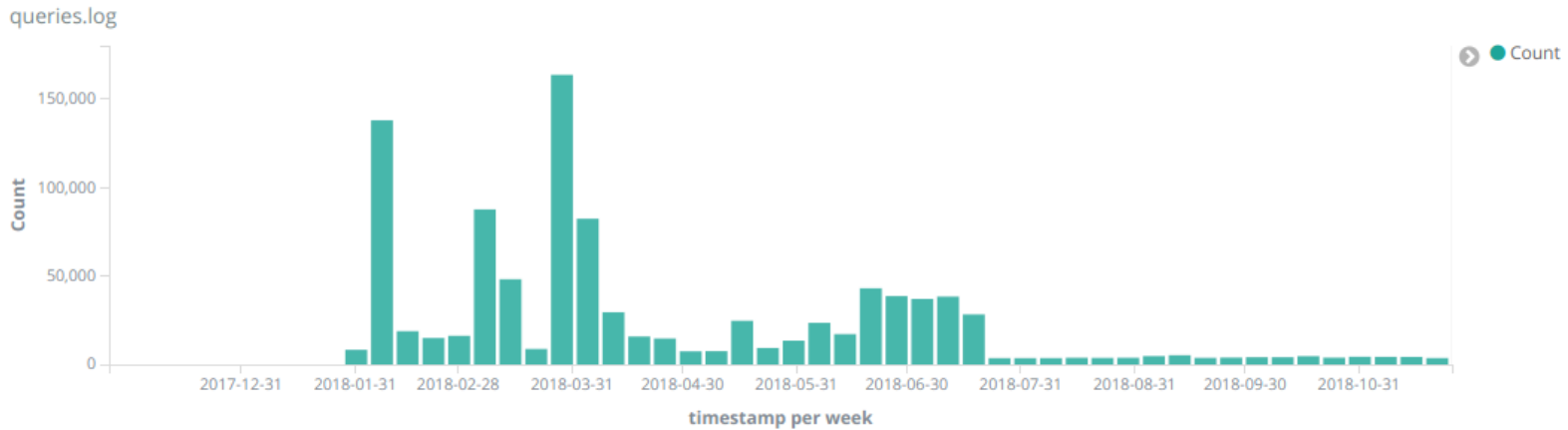
Why Honeypotting?



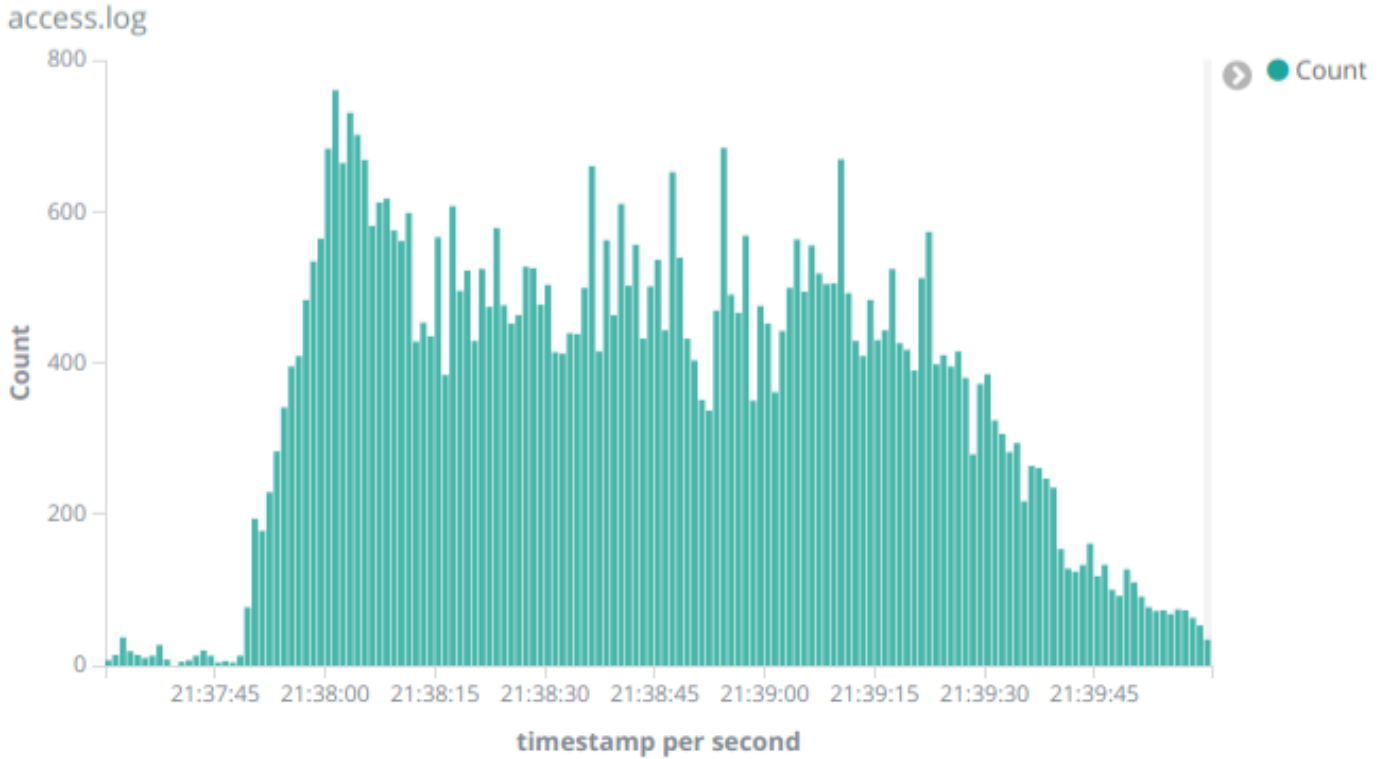
- Search space too massive
- Instead of finding them, have them find us
- Took multiple approaches
    - DNS Server
    - SMTP Server
    - Web Server
- Popads!

Honey DNS Server

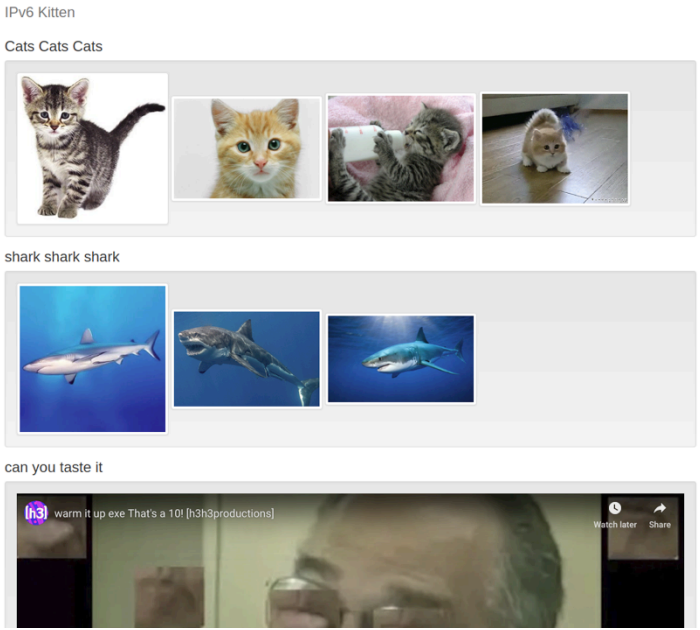
- Set up Bind server
- Glue records point to IPv6
- Zones delegated to IPv4 then IPv6
- Post links all over social media
- Popads!



A Quick Note on PopAds

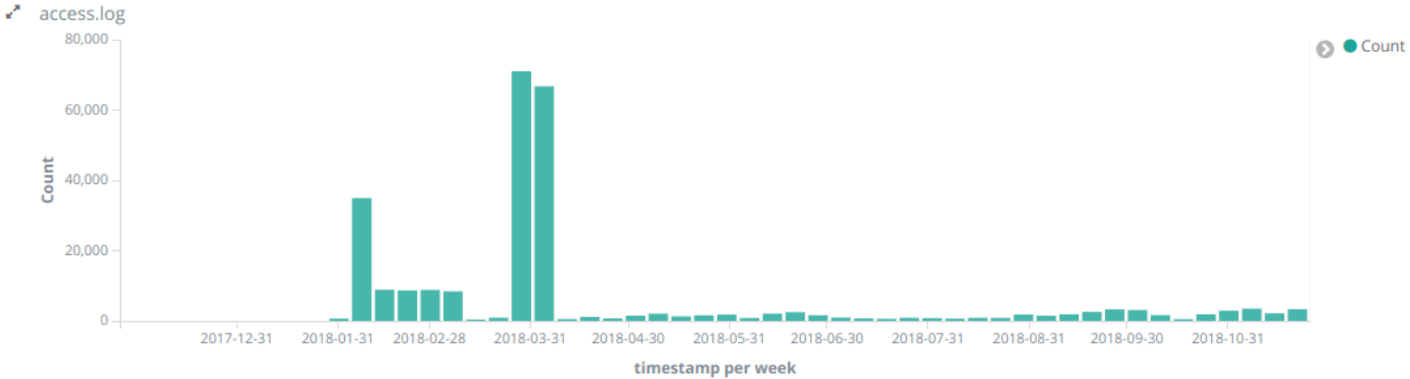


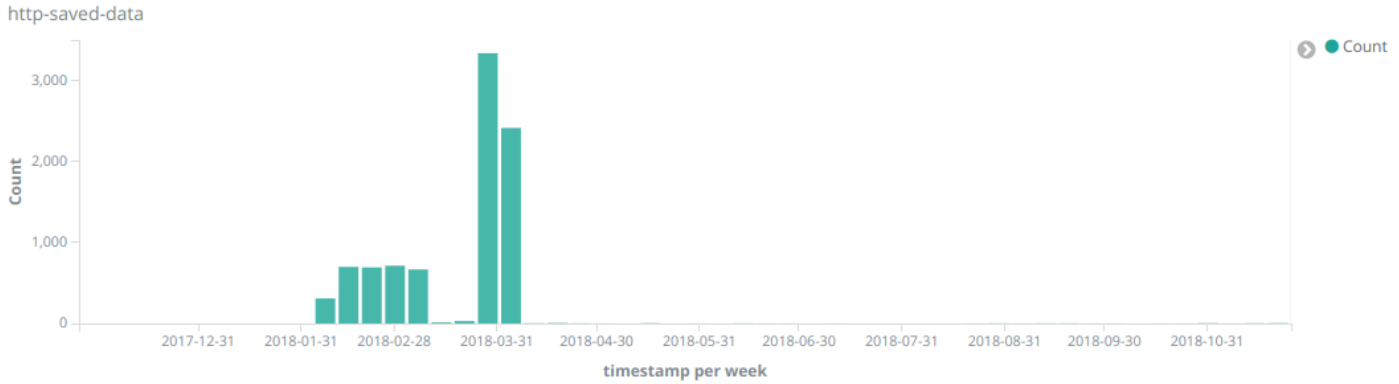
Honey Web Server



- Set up best site ever at http://ipv6.exposed/
- Available over IPv6, some other shenanigans for forcing IPv6
- WebRTC for IP address enumeration
- Post all over social media
- Popads!

Honey Web Server (cont'd)





Honey SMTP Server

- Set up SMTP server
- Have our DNS honeypot point to it
- Sign up for ALL THE THINGS
- Use MailBait to sign up for spam
- Mega womp womp



Sweet(?) Honey Results

- 92,609 unique IP addresses over ~10 months
- Cost $500+
- Lost focus, but still suboptimal



AND THEN...

Lol...

Internet Draft
<draft-kitamura-ipv6-ephemeral-address-00.txt>

H. Kitamura
NEC Corporation
S. Ata
Osaka City University
M. Murata
Osaka University

Expires June 2009

October 20, 2008

IPv6 Ephemeral Addresses
<draft-kitamura-ipv6-ephemeral-address-00.txt>

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Modeling for Non-PSLAAC

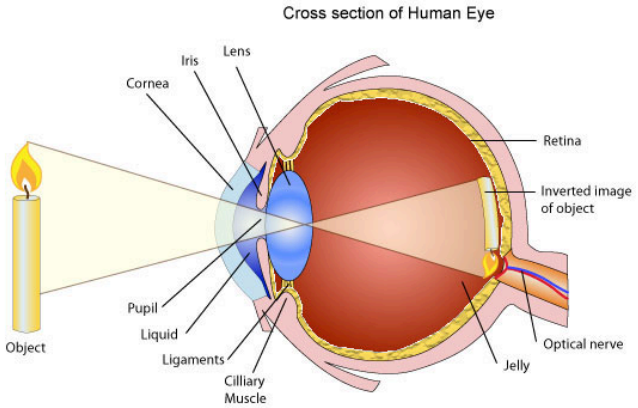
Structure in IPv6 Addresses

2001:1284:f01c:2c0a:8238:bcff:fed3:4e03
2001:1498:1::32:48
2001:16b8:101:3175:a96:d7ff:fe7e:c4aa
2001:16b8:6100:0:3631:c4ff:fe14:3d72
2001:1890:1f8:46::1:6
2001:1bc8:1004::2:0:99
2001:260:450:7b::4
2001:41d0:2:275b::182
2001:41d0:2:3d77::
2001:4998:44:6027::2005
2001:4b78:2::f06b
2001:558:370:ba::1
2001:579:6f03:600:c0b3:b230:5c3:df35
2001:610:1a0:30::2
2001:638:708:30da:ea11:32ff:fe70:8ae6
2001:8d8:921:c500::70:18e1
2001:980:6972::1
2001:bc8:2800:36dd:87f9:1ef0:8a7a:c21f
2001:e42:102:1819:160:16:234:111
2001:ee0:4041:37a5:8acf:98ff:fee7:fec
2001:ee0:4140::1230:2502:7004
2001:ee0:4501:5062:1894:5a03:9cc3:216f
2001:ee0:5500:8b5a:585d:c41a:7a09:f73a
2003:0:2e02:1050::1
2003:5a:4049::1
2400:6180:100:d0::34:8001

2400:8500:1302:803:133:130:127:180
2400:b800:1:1::18
2400:cb00:2048:1::6810:7166
2400:cb00:2048:1::6818:d3a
2404:6800:4007:801::2003
2406:da00:ff00::1715:7892
2406:e00:120:391c:0:30:ab4:4f40
2407:500::2:5a9:b7bf
2600:0:2:1239:144:232:2:85
2600:3c00::f03c:91ff:feae:ee1c
2600:3c00::f03c:91ff:fedf:426
2600:3c03::f03c:91ff:fe79:e1a4
2600:3c03::f03c:91ff:fea1:4761
2600:3c03::f03c:91ff:fea2:42c7
2603:3006:103c:b000::17f6
2604:2d80:4030:0:91c2:c016:2329:7219
2605:de00:1:1:4a:32:0:23
2606:b400:8808:f000::a022:fa6d
2620:11:0:c2b4:749a:46bf:291:38cf
2620:8d:0:7f47::d827:7f47
2800:370:2:418a:b8d3:ddc8:13a8:8768
2800:370:2:972d:dd4a:a203:9d92:3353
2800:370:44:256d:5950:afb3:7627:10f4
2800:370:55:3c60:404a:d212:49dd:4477
2800:370:55:b9c6:70b1:9c61:69e:2b02
2800:370:61:11dc:e57f:1804:a456:eddc

2800:370:84:0:d0bd:6968:d1ae:d26f
2800:370:84:bba1:857c:f02d:cd6c:cd51
2800:370:a:ae43:79b9:348b:3c0:8c7b
2800:4f0:1:ecd5:dd66:8006:1203:be04
2800:4f0:62:850c:92d:85a1:3431:a7b8
2803:c300::2
2804:14d:1a87:0:7815:7414:e01:dd02
2804:14d:8e8c:1000:b42f:9577:7fbb:778b
2804:292c:1::5
2804:a8:2:c8::12a
2806:102e:9:5055:272:63ff:fe83:e620
2806:108e:c:2e3:7279:90ff:fe9c:2a07
2a00:d0c0:200:0:b9:1a:9c36:20c
2a01:488:42:1000:50ed:8479:33:339a
2a01:488:42:1000:50ed:84f5:1c:ff1f
2a01:5a60:3::92
2a01:7c8:d002:1c::1
2a01:a8:dc0:330:1::1c5d
2a02:2028:80c:e900::1
2a02:26f0:df:202:e3cc:80db:ebaa:3e93
2a02:6b8:0:161b:ec4:7aff:fe18:c48
2a02:6b8:b000:63a:96de:80ff:fe81:1258
2a02:6b8:b000:6509:215:b2ff:fea9:66fa
2a02:8108:0:12:587b:b48e:e629:436b
2a02:8108:8000:21:2864:5009:d4d2:36f0
2a02:810d:8000:29:e8b0:6937:7ddd:e597

MACHINE LEARNING BAYBEEEE



- Model is a compact representation of data set
- Projection through model creates new data set with error %
- Errors are representative of structure in IPv6 addresses
- Hopefully find new addresses

lol jk

- All attempts resulted in over-fitting
- Projected addresses were the same as our input addresses
- We’re not ML experts sooooo….



The Entropy/IP Paper

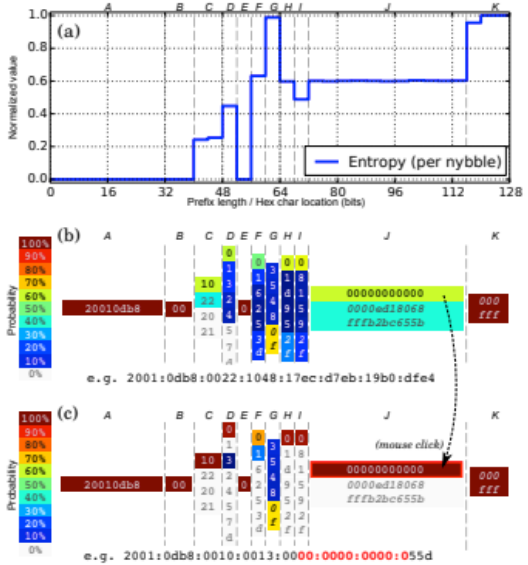
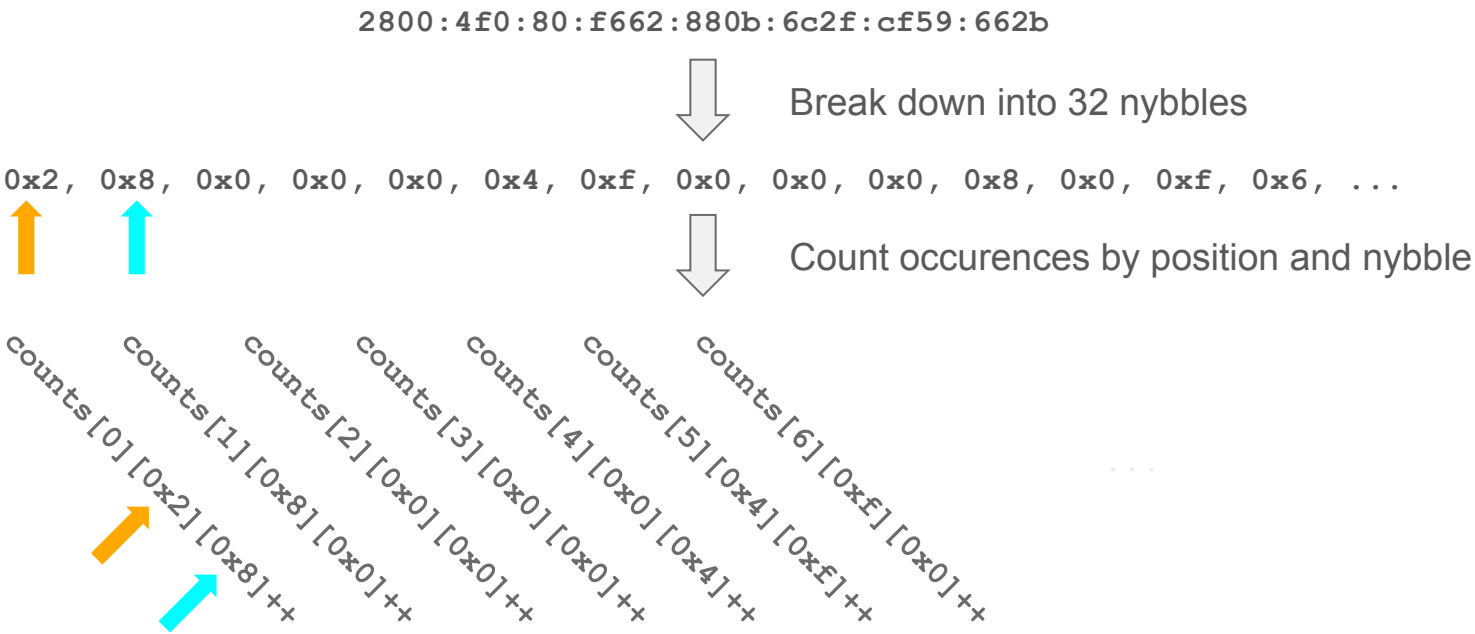


Figure 1: Entropy/IP's user interface displaying an analysis of a Japanese telco prefix with 24K active client IPs. Entropy by nybble plotted in (a). In (b), we select the 00000... value (60%) for segment J by mouse click, resulting in updated probabilities in (c) (*e.g.*, 100%).

- http://www.entropy-ip.com/
- Really interesting paper from Akamai
- Maps entropy of different segments of IPv6 addresses
- Big takeaways:
  - Not THAT much entropy in non-PSLAAC IPs
  - Simpler modeling might work better

Dumbing Things Down (Modeling)



Dumbing Things Down (Prediction)

0x2
Position 0

probabilities[0][0x2]

```
p(0x0) => 0.05
p(0x1) => 0.05
p(0x2) => 0.01
p(0x3) => 0.09
p(0x4) => 0.00
p(0x5) => 0.15
p(0x6) => 0.05
p(0x7) => 0.09
p(0x8) => 0.00
p(0x9) => 0.06
p(0xa) => 0.15
p(0xb) => 0.09
p(0xc) => 0.06
p(0xd) => 0.10
p(0xe) => 0.03
p(0xf) => 0.02
```



0xa
Position 1

Looks Promising… TOO PROMISING

- Generated 10mm addresses
- After scanning, over 50k responded to ICMP probes
- WOW WE TOTES SOLVED THE PROBLEM!!! ZOMG CELEBRATION TIME!!!!

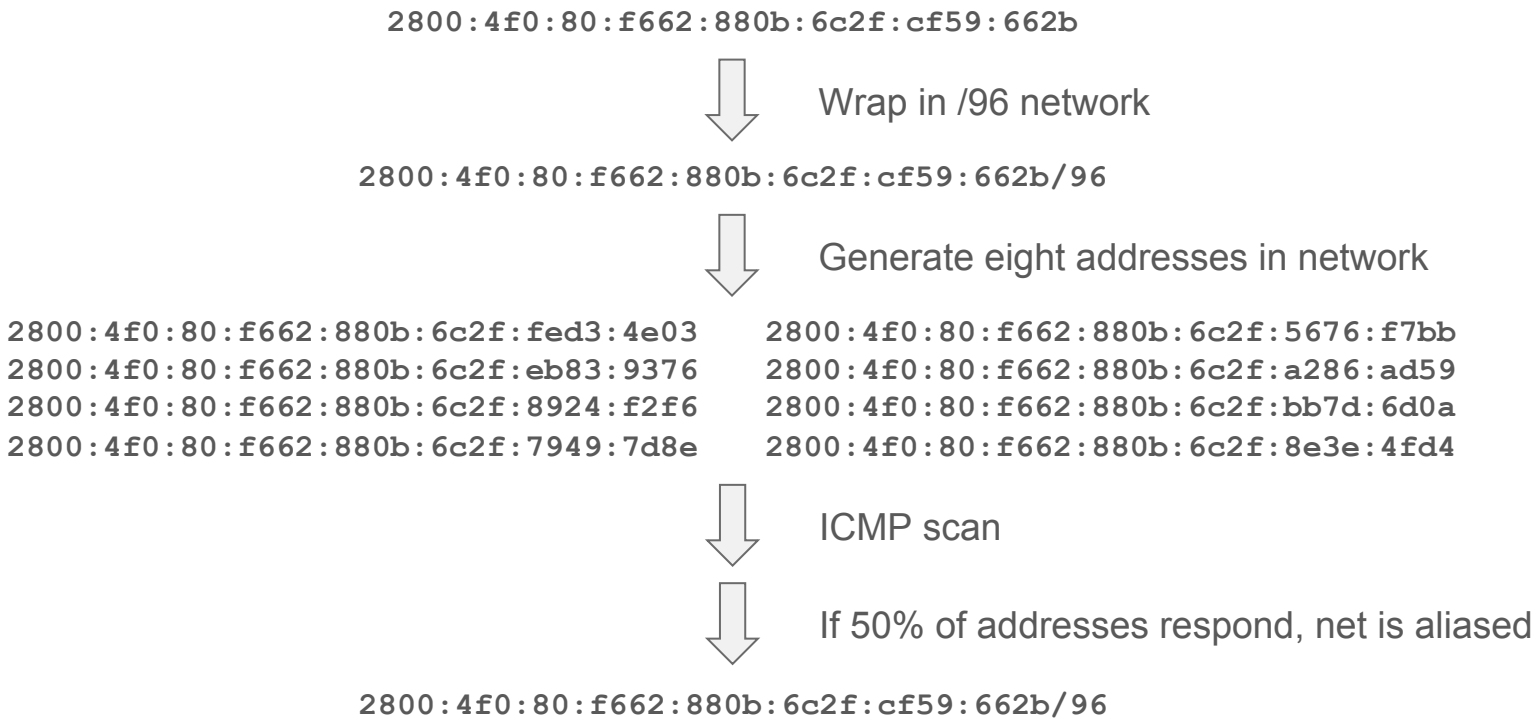


Enter Aliased (ie: Jerk) Networks

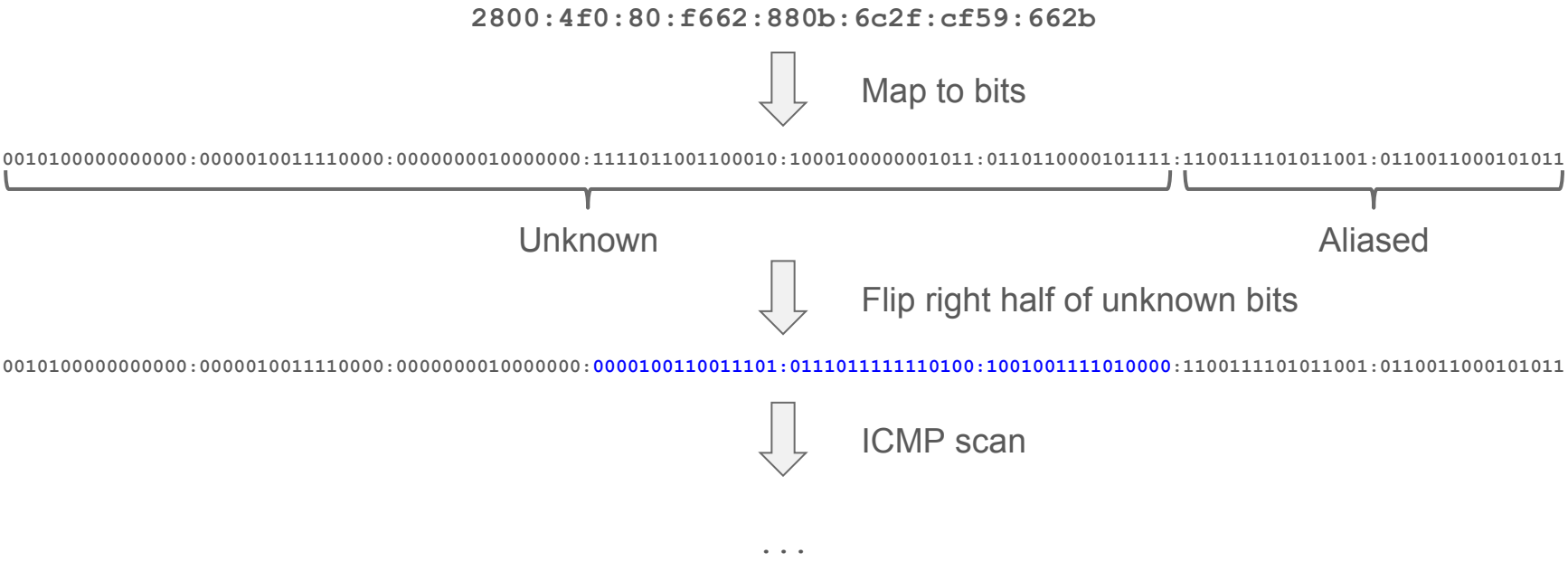


- Network ranges where every IP address responds to ICMP pings
- Why?? Because they’re jerks
- Not great for scanning
- Even worse for statistical modeling

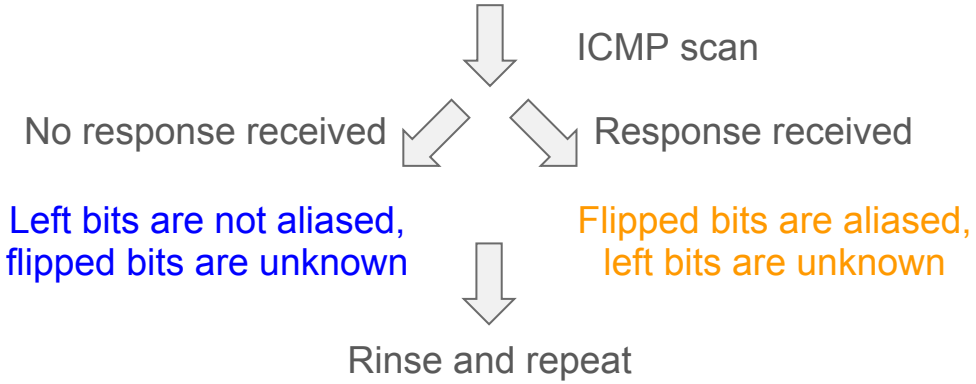
Identifying Aliased Networks (Initial)

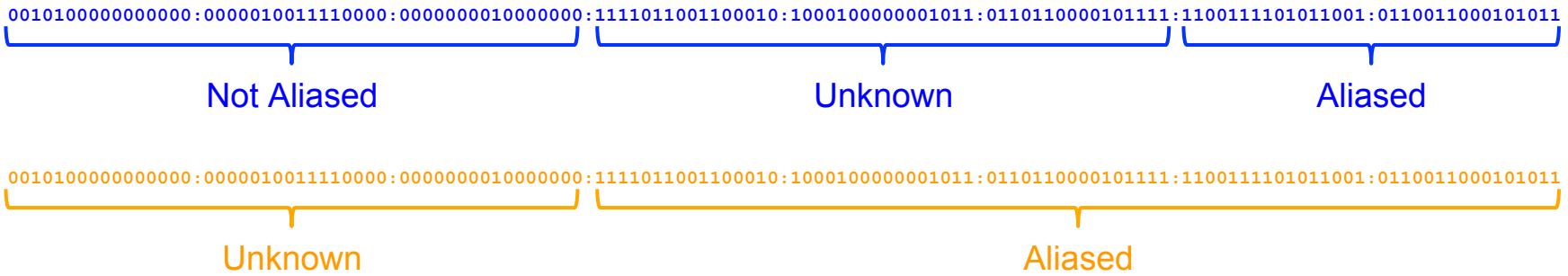


Identifying Aliased Networks (Network Size)



Identifying Aliased Networks (Network Size cont'd)





Getting Less Dumber

6gen

Target Generation for Internet-wide IPv6 Scanning

Austin Murdock[1,2], Frank Li[1,2], Paul Bramsen[1], Zakir Durumeric[2], Vern Paxson[1,2]

{austinmurdock, frankli, paulbramsen, vern}@berkeley.edu, zakir@icsi.berkeley.edu

[1] University of California, Berkeley [2] International Computer Science Institute

ABSTRACT

Fast IPv4 scanning has enabled researchers to answer a wealth of new security and measurement questions. However, while increased network speeds and computational power have enabled comprehensive scans of the IPv4 address space, a brute-force approach does not scale to IPv6. Systems are limited to scanning a small fraction of the IPv6 address space and require an algorithmic approach to determine a small set of candidate addresses to probe. In this paper, we first explore the considerations that guide designing such algorithms. We introduce a new approach that identifies dense address space regions from a set of known "seed" addresses and generates a set of candidates to scan. We compare our algorithm 6Gen against Entropy/IP—the current state of the art—finding that we can recover between 1–8 times as many addresses for the five candidate datasets considered in the prior work. However, during our analysis, we uncover widespread IP aliasing in IPv6 networks. We discuss its effect on target generation and explore preliminary approaches for detecting aliased regions.

and Masscan [18] have fundamentally enhanced the ability of researchers to conduct wide-ranging assessments of Internet services, including the use of cryptography in practice [4], uncovering network administrator behaviors [21], and tracking vulnerability remediation [11].

These tools leverage the density and limited size of the IPv4 address space: today's scanning speeds are such that it is feasible to exhaustively enumerate all possible IPv4 addresses in order to conduct comprehensive scans. However, as has long been recognized [3], IPv6's much larger address space renders exhaustive probing completely infeasible. This then raises the question for measurement researchers of how to obtain at least a degree of global IPv6 address visibility somewhat comparable to the comprehensive IPv4 visibility provided by tools such as ZMap.

While prior work has developed sophisticated techniques for inferring the underlying structure of how IPv6 network operators assign addresses in their networks [14], and, separately, for how to leverage IPv6 address assignment policies to abet network reconnaissance [17], the question of how to employ these insights to

- Published in 2017 out of UC Berkeley
- Improves upon the Entropy/IP paper with a new clustering algorithm

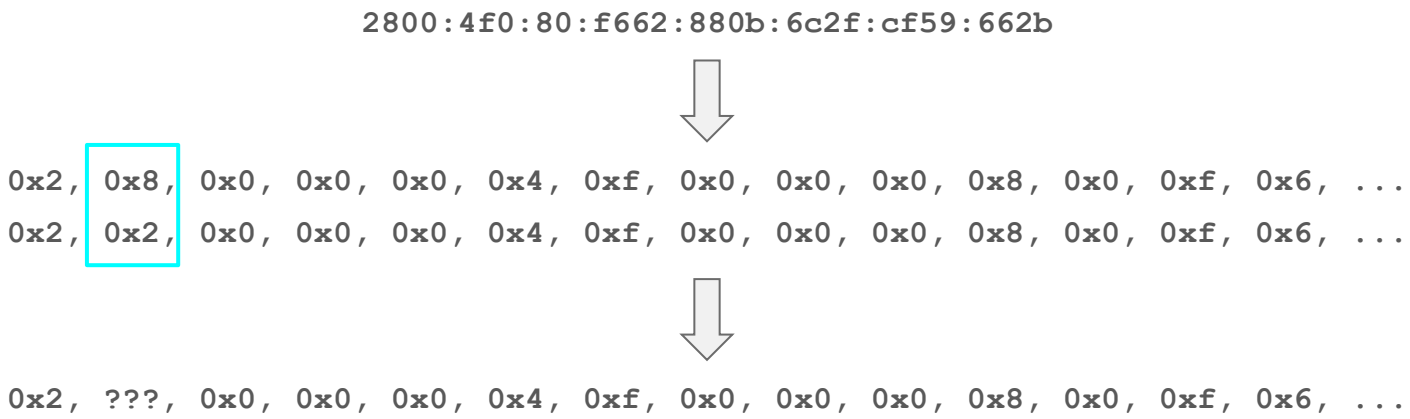
IPv6 Address Clusters

- An IPv6 address cluster is defined as an IP address and a set of wild card nybble indices

- A cluster's utility is based on its…

    - **Capacity** - How many possible addresses are in the cluster?
    - **Density** - How many of the cluster's IPs are in the input data set?

IPv6 Address Clusters (Example)



Capacity	Captured	Density
16	**2**	**12.5%**

IPv6 Address Clusters (Example)

```
0x2, ???, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
```



```
0x2, 0x0, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x1, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x2, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x3, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x4, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x5, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x6, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x7, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x8, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0x9, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0xa, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0xb, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0xc, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0xd, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0xe, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
0x2, 0xf, 0x0, 0x0, 0x0, 0x4, 0xf, 0x0, 0x0, 0x0, 0x8, 0x0, 0xf, 0x6, …
```

Original Algorithm

- For every IPv6 address
  - Create a cluster of size 1
  - For every IPv6 address
    - Evaluate density of cluster with adding address
  - Pick the best upgrade candidate
- Sort all upgrades by density
- While scanning budget not exhausted
  - Pick the best density upgrade
  - Evaluate best upgrade candidates from it
  - Add upgraded cluster to cluster set

15mm * 15mm

EXPENSIVE!

Scanning budget not great for continuous scanning

EXPENSIVE!

Not intended to identify new addresses

666gen

- For every IPv6 address
  - Evaluate best cluster upgrades with adding only one wildcard
- Remove all clusters that had no good upgrades (*stardust*)
- Sort upgrades by density
- While cluster score still improving
  - Take the next best upgrade
  - Evaluate best cluster upgrades with adding only one wildcard
  - Add the upgraded cluster to the cluster set
  - De-duplicate

15mm

significant size reduction

no scanning budget

expensive!

666gen Generation

- Select a cluster from cluster set at random
- For i in 0..32
  - Roll a weighted die to determine nybble source
    - Generate from *stardust* probability distribution
    - Generate from selected cluster

Fanning Out

- Fan-out to discover addresses **similar** to those generated with modeling
- Addresses with a 1-nybble difference from a newly discovered address
- Sequentially-neighboring /64 networks and hosts



Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  `2000::/4`
- Starting IP address
  `2001:0558:0110:0000:ffff:ffff:0000:0021`

Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  `2000::/4`
- Starting IP address
  `2001:0558:0110:0000:ffff:ffff:0000:0021`

`2001:0558:0110:0000:ffff:ffff:0000:0020`

Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  `2000::/4`
- Starting IP address
  `2001:0558:0110:0000:ffff:ffff:0000:0021`

```
2001:0558:0110:0000:ffff:ffff:0000:0020
2001:0558:0110:0000:ffff:ffff:0000:0022
```

Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  `2000::/4`
- Starting IP address
  `2001:0558:0110:0000:ffff:ffff:0000:0021`

```
2001:0558:0110:0000:ffff:ffff:0000:0020
2001:0558:0110:0000:ffff:ffff:0000:0022
2001:0558:0110:0000:ffff:ffff:0000:0023
```

Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  `2000::/4`
- Starting IP address
  `2001:0558:0110:0000:ffff:ffff:0000:0021`

```
2001:0558:0110:0000:ffff:ffff:0000:0020
2001:0558:0110:0000:ffff:ffff:0000:0022
2001:0558:0110:0000:ffff:ffff:0000:0023
...
2001:0558:0110:0000:ffff:ffff:0000:002f
```

Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  `2000::/4`
- Starting IP address
  `2001:0558:0110:0000:ffff:ffff:0000:0021`

```
2001:0558:0110:0000:ffff:ffff:0000:0020
2001:0558:0110:0000:ffff:ffff:0000:0022
2001:0558:0110:0000:ffff:ffff:0000:0023
...
2001:0558:0110:0000:ffff:ffff:0000:002f

...
```

Nybble-Adjacent Fanout

- For each discovered address, generate up to 465 candidate addresses, each differing by one nybble
- 15-465 addresses are generated depending on the target network mask
- Target network
  2000::/4
- Starting IP address
  2001:0558:0110:0000:ffff:ffff:0000:0021

```
2001:0558:0110:0000:ffff:ffff:0000:0020
2001:0558:0110:0000:ffff:ffff:0000:0022
2001:0558:0110:0000:ffff:ffff:0000:0023
...
2001:0558:0110:0000:ffff:ffff:0000:002f

...

2101:0558:0110:0000:ffff:ffff:0000:0021
2201:0558:0110:0000:ffff:ffff:0000:0021
2301:0558:0110:0000:ffff:ffff:0000:0021
...
2f01:0558:0110:0000:ffff:ffff:0000:0021
```

/64 Sequential Fanout

- Sequentially fan out from each discovered ::1/64 address
- /64 networks
    - Neighboring ::1/64 addresses (sequentially increasing and decreasing)
- Hosts within a /64
    - Neighboring ::X/64 addresses (sequentially increasing)
- Target IP Address
  `2001:0558:0110:2fab::1`

/64 Sequential Fanout

- Sequentially fan out from each discovered ::1/64 address
- /64 networks
  - Neighboring ::1/64 addresses (sequentially increasing and decreasing)
- Hosts within a /64
  - Neighboring ::X/64 addresses (sequentially increasing)
- Target IP Address
  `2001:0558:0110:2fab::1`

Neighboring /64 networks:

```
2001:0558:0110:4fa9::1
2001:0558:0110:4faa::1
2001:0558:0110:4fac::1
2001:0558:0110:4fad::1
...
```

/64 Sequential Fanout

- Sequentially fan out from each discovered ::1/64 address
- /64 networks
  - Neighboring ::1/64 addresses (sequentially increasing and decreasing)
- Hosts within a /64
  - Neighboring ::X/64 addresses (sequentially increasing)
- Target IP Address
  2001:0558:0110:4fab::1

Neighboring /64 networks:

2001:0558:0110:4fa**9**::1
2001:0558:0110:4fa**a**::1
2001:0558:0110:4fa**c**::1
2001:0558:0110:4fa**d**::1
...

Neighboring /64 hosts:

2001:0558:0110:4fab::**2**
2001:0558:0110:4fab::**3**
2001:0558:0110:4fab::**4**
...

Is Us Smarter?

Improved Discovery Rate

v0.2

- 58,838 addresses found in eight days, ~80% not found in public data sets
- 0.068 new IPs / second

v0.3

- **1.57M** addresses found in **one hour**, ~78% not found in public data sets
- 342.19 new IPs / second

503,234% Improvement

MOAR RESULTS

- Port scanned ~100K addresses from the newly discovered set, testing some common ports
- Lots of network equipment (both infrastructure and CPE)
- No-auth Mongo instances
- Lots of ancient SSH and telnet servers

Port	Count	Port	Count	Port	Count
21	2,593	3000	111	8443	755
22	17,979	3306	643	8888	93
23	2,620	5000	58	27017	7
80	9,604	5432	159	49152	77
443	8,491	8000	155	49153	81
445	173	8080	687	49154	133

To The Cloud

Crowdsourced Scan Data

- How can we make this data more accessible?
- Users of IPv666 optionally push scan results to the cloud
- Scan results are aggregated into a queryable datastore



IPv666 Online Portal

- Online portal where users can query the crowdsourced scan result set
- <placeholder for link to online portal>



IPv666

Official v0.4 Release!

- Install with go get:

```
go get github.com/lavalamp-/ipv666/ipv666
```

- Push crowdsourced scan results to the cloud

Was Ist Das?

`ipv666` is a set of tools that enables the discovery of IPv6 addresses both in the global IPv6 address space and in more narrow IPv6 network ranges. These tools are designed to work out of the box with minimal knowledge of their workings.

The tools included in this codebase are as follows:

- `scan discover` - Locates live hosts over IPv6 using statistical modeling and ICMP ping scans
- `scan alias` - Tests a single IPv6 network range to see if the network range is aliased
- `generate addresses` - Generate IPv6 addresses based on the content of a probabilistic clustering model
- `generate model` - Generate a probabilistic clustering model based off an input set of IPv6 addresses
- `generate blacklist` - Adds the contents of a file containing IPv6 network ranges to the aliased network blacklist
- `clean` - Cleans the contents of a file containing IPv6 addresses based on an aliased network blacklist
- `convert` - Converts the contents of a file containing IPv6 addresses to another IP address representation

scan discover

```
This utility scans for live hosts over IPv6 based on the network range you specify. If no range is
specified, then this utility scans the global IPv6 address space (e.g. 2000::/4). The scanning process
generates candidate addresses, scans for them, tests the network ranges where live addresses are found
for aliased conditions, and adds legitimate discovered IPv6 addresses to an output list.

Usage:
  ipv666 scan discover [flags]

Flags:
  -h, --help                  help for discover
  -o, --output string         The path to the file where discovered addresses should be written.
  -t, --output-type string    The type of output to write to the output file (txt or bin).

Global Flags:
  -b, --bandwidth string    The maximum bandwidth to use for ping scanning
  -f, --force               Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string          The log level to emit logs at (one of debug, info, success, warn, error).
  -n, --network string      The IPv6 CIDR range to scan.
```

scan alias

```
A utility for testing whether or not a network range exhibits traits of an aliased network range.
Aliased network ranges are ranges in which every host responds to a ping request, thereby making it
look like the range is full of IPv6 hosts. Pointing this utility at a network range will let tell you
whether or not that network range is aliased and, if it is, the boundary of the network range that is
aliased.

Usage:
  ipv666 scan alias [flags]

Flags:
  -h, --help   help for alias

Global Flags:
  -b, --bandwidth string   The maximum bandwidth to use for ping scanning
  -f, --force              Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string         The log level to emit logs at (one of debug, info, success, warn, error).
  -n, --network string     The IPv6 CIDR range to scan.
```

generate addresses

```
This utility will generate IPv6 addresses in target network range (or in the global address
space) based on the default included cluster model or a cluster model that you specify.

Usage:
  ipv666 generate addresses [flags]

Flags:
  -c, --count int         The number of IP addresses to generate. (default 1000000)
  -h, --help              help for addresses
  -m, --model string      Local file path to the model to generate addresses from (if empty,
                          uses the default model packaged with ipv666).
  -n, --network string    The address range to generate addresses within (if empty, generates
                          addresses in the global address space of ::/0).
  -o, --out string        File path to where the generated IP addresses should be written.

Global Flags:
  -f, --force        Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string   The log level to emit logs at (one of debug, info, success, warn, error).
```

generate model

```
This utility will generate a predictive clustering model based on the contents of
an IPv6 address file.

Usage:
  ipv666 generate model [flags]

Flags:
  -h, --help           help for model
  -i, --input string   An input file containing IPv6 addresses to use for the model.
  -o, --out string     The file path to write the resulting model to.

Global Flags:
  -f, --force        Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string   The log level to emit logs at (one of debug, info, success, warn, error).
```

generate blacklist

```
This utility takes a list of IPv6 CIDR ranges from a text file (new-line delimited),
adds them to the current network blacklist, and sets the new blacklist as the one to use
for the 'scan' command.

Usage:
  ipv666 generate blacklist [flags]

Flags:
  -h, --help           help for blacklist
  -i, --input string   An input file containing IPv6 network ranges to build a blacklist from.

Global Flags:
  -f, --force        Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string   The log level to emit logs at (one of debug, info, success, warn, error).
```

clean

```
This utility will clean the contents of an IPv6 address file (new-line delimited, standard ASCII hex
representation) based on the contents of an IPv6 network blacklist file. If no blacklist path is
supplied then the utility will use the default blacklist. The cleaned results will then be written to
an output file.

Usage:
  ipv666 clean [flags]

Flags:
  -b, --blacklist string   The local file path to the blacklist to use. If not specified, defaults to
                           the most recent blacklist in the configured blacklist directory.
  -h, --help               help for clean
  -i, --input string       An input file containing IPv6 addresses to clean via a blacklist.
  -o, --out string         The file path where the cleaned results should be written to.

Global Flags:
  -f, --force        Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string   The log level to emit logs at (one of debug, info, success, warn, error).
```

convert

```
This utility will process the contents of a file as containing IPv6 addresses, convert those addresses
to another format, and then write a new file with the same addresses in the new format. This
functionality is (hopefully) intelligent enough to determine how the addresses are stored in the file
without having to specify an input type.

Usage:
  ipv666 convert [flags]

Flags:
  -h, --help           help for convert
  -i, --input string   The file to process IPv6 addresses out of.
  -o, --out string     The file path to write the converted file to.
  -t, --type string    The format to write the IPv6 addresses in (one of 'txt', 'bin', 'hex').

Global Flags:
  -f, --force        Whether or not to force accept all prompts (useful for daemonized scanning).
  -l, --log string   The log level to emit logs at (one of debug, info, success, warn, error).
```

Linky Links

- IPv666 Blog Post
  https://l.avala.mp/?p=285
- IPv666 GitHub Repository
  https://github.com/lavalamp-/ipv666



Conclusion

Recap



- Background
- The Scanning Problem
- Honeypotting for PSLAAC
- Modeling for non-PSLAAC
- Being Less Dumb
- Results
- To The Cloud
- IPv666
- Conclusion

Moar Links

- Entropy/IP
  http://www.entropy-ip.com/
- 6gen
  https://zakird.com/papers/imc17-6gen.pdf
- Clustering of IPv6 address structure
  https://arxiv.org/pdf/1806.01633.pdf
- IPv6 hitlist
  https://ipv6hitlist.github.io/



Q&A

THANKS!

<3

Chris Grayson

@_lavalamp

Marc Newlin

@marcnewlin