Internet Research Observatory

W205 Spring 2017 - Final Presentation
Authors: Jason Hunsberger, Victoria Baker, Dominic Delmolino

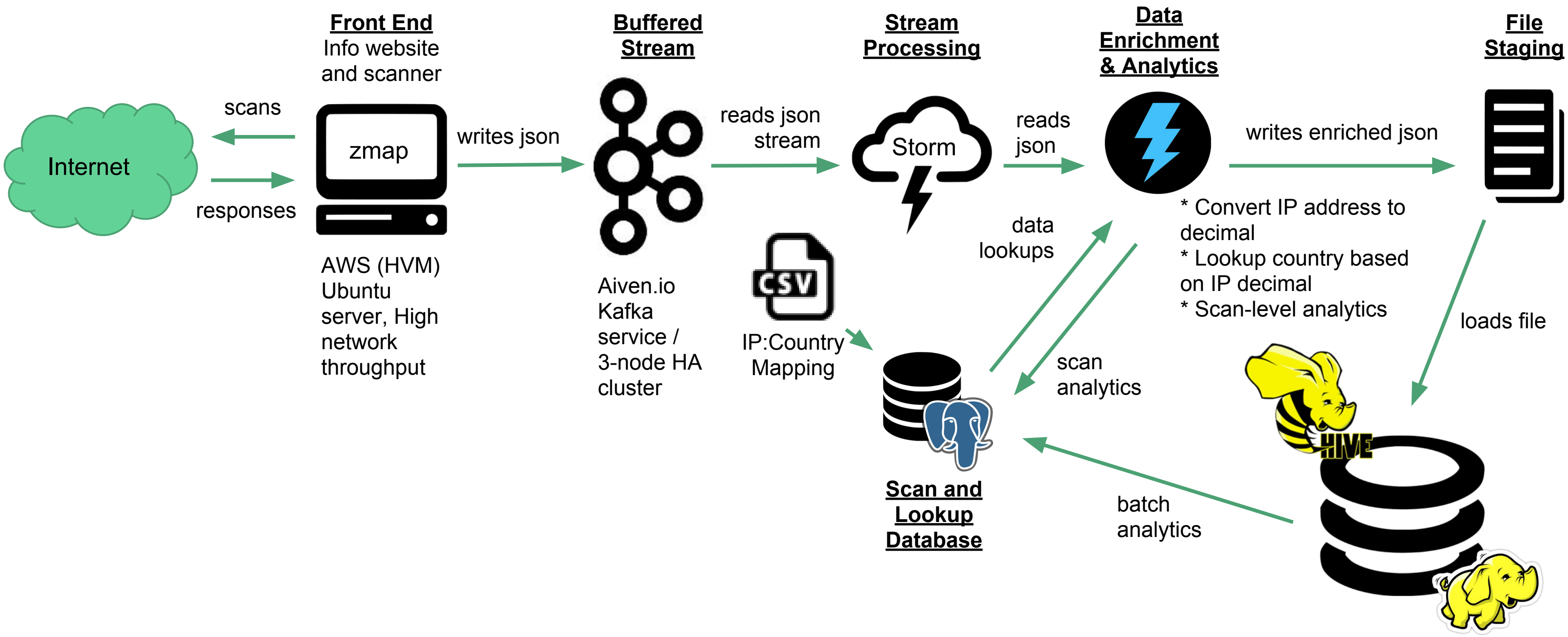
4,294,967,296

IPv4 addresses on the public Internet

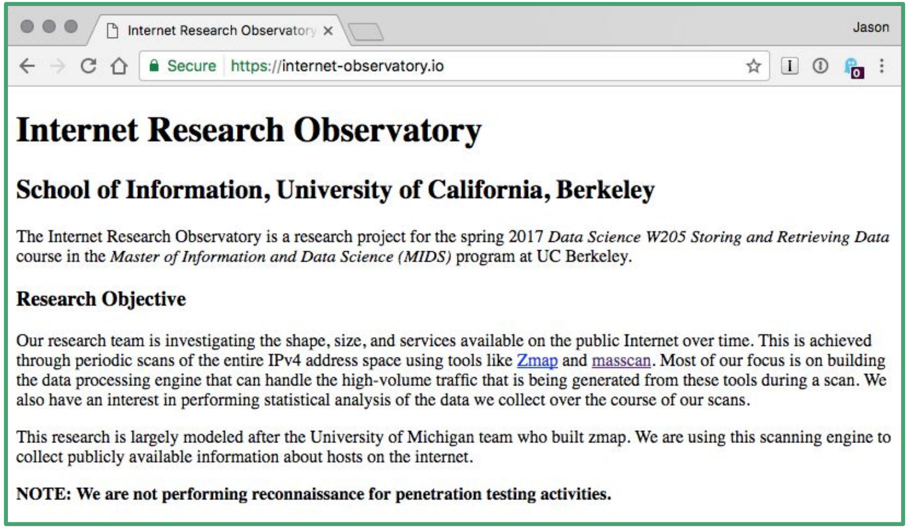
How many of them are actively in use?
What is the geographical distribution of active addresses?
Are there common vulnerabilities across addresses?

Can we scan all of them in a reasonable amount of time, record the results and answer these questions?

Solution Architecture



Scanning the Internet with Zmap


Internet Research Observatory
Jason
Secure
https://internet-observatory.io

Internet Research Observatory

School of Information, University of California, Berkeley

The Internet Research Observatory is a research project for the spring 2017 *Data Science W205 Storing and Retrieving Data* course in the *Master of Information and Data Science (MIDS)* program at UC Berkeley.

Research Objective

Our research team is investigating the shape, size, and services available on the public Internet over time. This is achieved through periodic scans of the entire IPv4 address space using tools like Zmap and masscan. Most of our focus is on building the data processing engine that can handle the high-volume traffic that is being generated from these tools during a scan. We also have an interest in performing statistical analysis of the data we collect over the course of our scans.

This research is largely modeled after the University of Michigan team who built zmap. We are using this scanning engine to collect publicly available information about hosts on the internet.

NOTE: We are not performing reconnaissance for penetration testing activities.

Front End Server

AWS EC2 Ubuntu 16.04 (HVM) w/high-network bandwidth
zmap 2.1.1 (custom build w/ JSON extension)
python 3.5.2
kafka-python 1.3.3
nginx 1.10

Scanning the entire public Internet is a political and technical challenge.

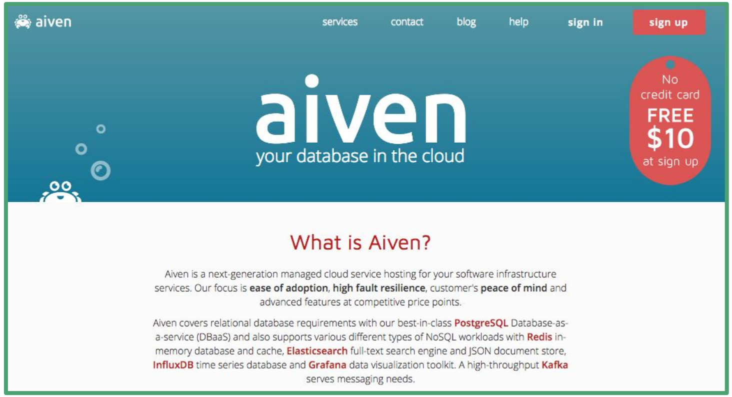
Public Notification

- Custom website w/ SSL cert
- DNS records
- Whois information
- Ability to opt-out

Technical Elements

- High-performance network I/O on AWS
- Zmap network scanner
- Scan management with Python
- Data streaming with Kafka

Kafka-as-a-Service: Aiven


aiven
services
contact
blog
help
sign in
sign up
aiven
your database in the cloud
No credit card FREE $10 at sign up

What is Aiven?

Aiven is a next-generation managed cloud service hosting for your software infrastructure services. Our focus is **ease of adoption**, **high fault resilience**, customer's **peace of mind** and advanced features at competitive price points.

Aiven covers relational database requirements with our best-in-class **PostgreSQL** Database-as-a-service (DBaaS) and also supports various different types of NoSQL workloads with **Redis** in-memory database and cache, **Elasticsearch** full-text search engine and JSON document store, **InfluxDB** time series database and **Grafana** data visualization toolkit. A high-throughput **Kafka** serves messaging needs.

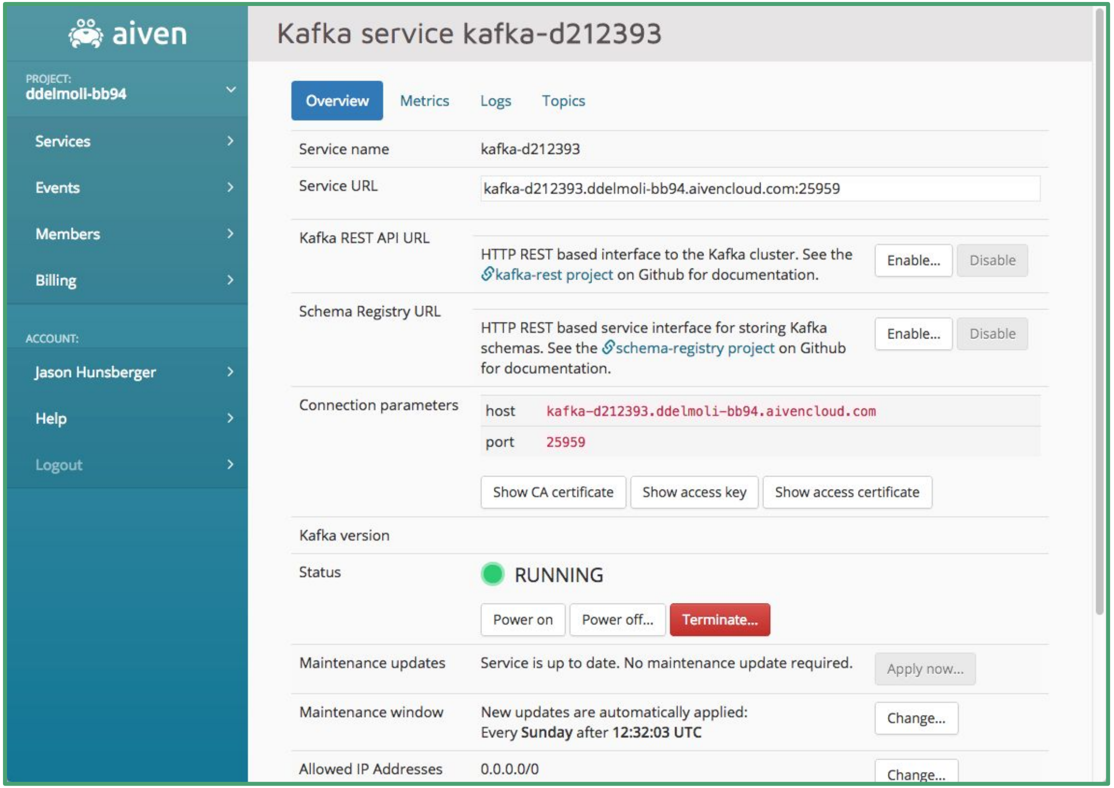
Kafka Cluster

aiven.io Kafka-as-a-Service

kafka 0.10.2

Startup plan 3 node HA set

Secured via client certificates and SSL



Kafka service kafka-d212393

Service name	kafka-d212393
Service URL	kafka-d212393.ddelmoli-bb94.aivencloud.com:25959
Kafka REST API URL	HTTP REST based interface to the Kafka cluster. See the kafka-rest project on Github for documentation.
Schema Registry URL	HTTP REST based service interface for storing Kafka schemas. See the schema-registry project on Github for documentation.
Connection parameters	host kafka-d212393.ddelmoli-bb94.aivencloud.com port 25959
Kafka version	
Status	● RUNNING
Maintenance updates	Service is up to date. No maintenance update required.
Maintenance window	New updates are automatically applied: Every **Sunday** after **12:32:03 UTC**
Allowed IP Addresses	0.0.0.0/0

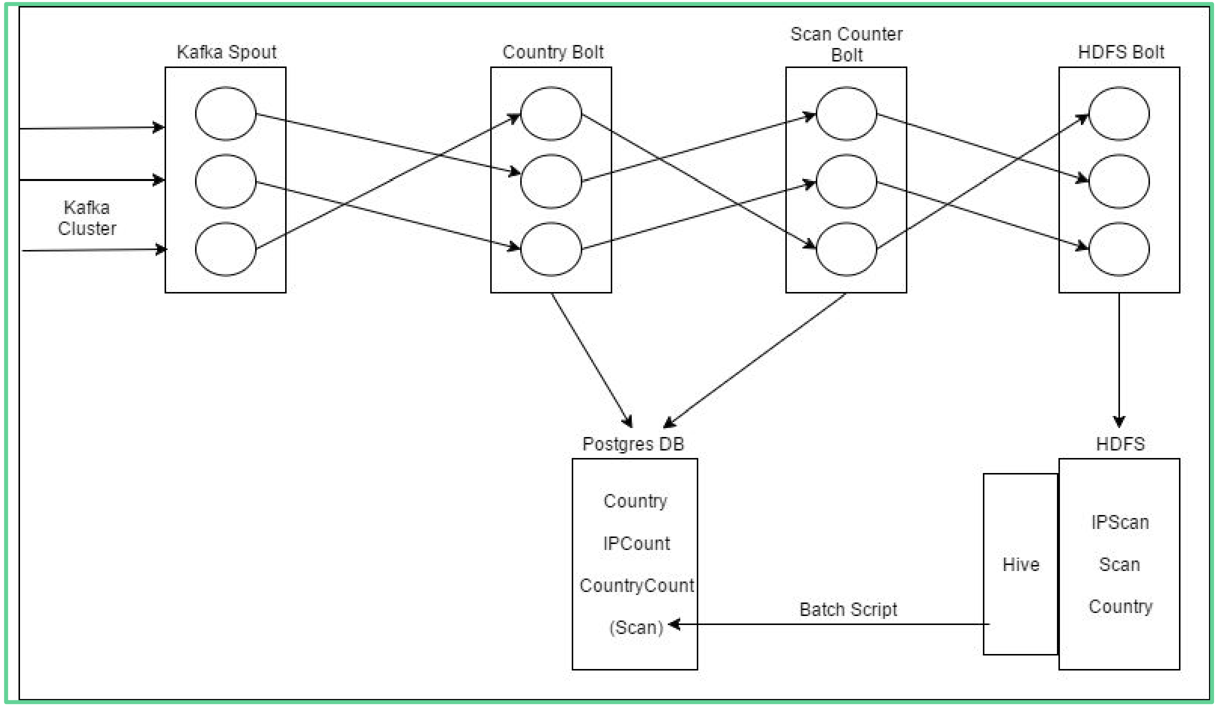
Apache Storm Processing Implementation details

Storm DAG

- Receive Kafka stream
- Convert IP to decimal
- Enrich with Country codes
- Scan-level analytics

Storm Processor

AWS EC2 Amazon Linux
python 2.7.12 (compiled with SSL)
kafka-python 1.3.3
java 1.7
apache storm 0.9.3
streamparse 2.1.4

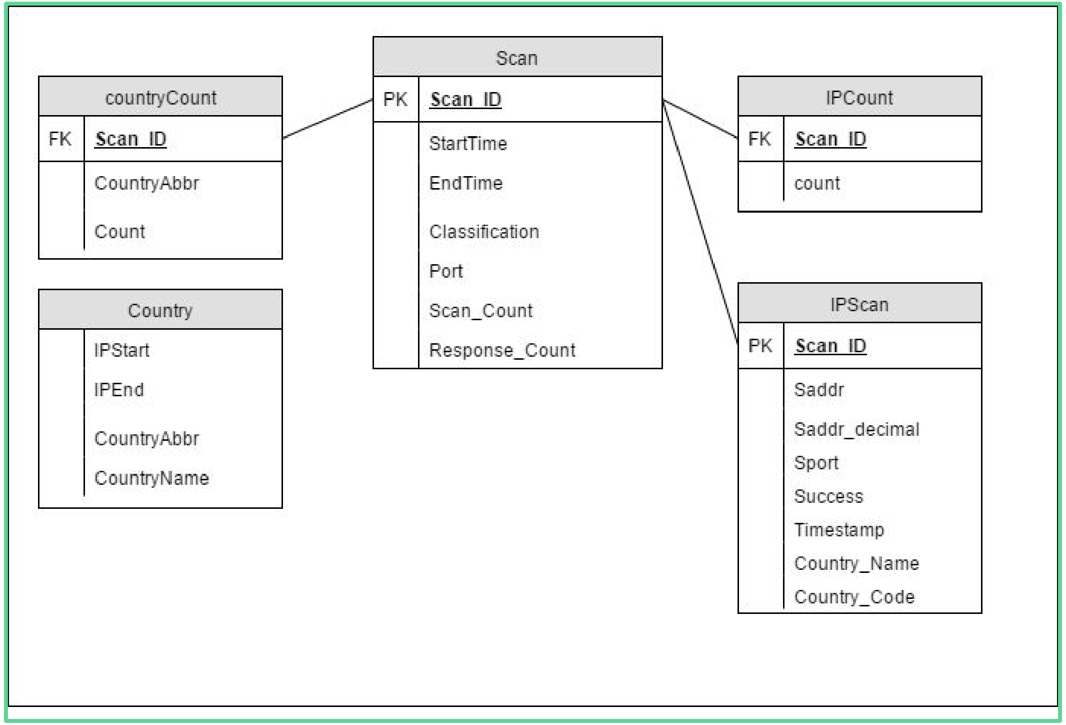


Postgres Database details

Database

Amazon AWS RDS PostgreSQL

1 TB of storage



	countryCount
FK	**Scan_ID**
	CountryAbbr
	Count

	Country
	IPStart
	IPEnd
	CountryAbbr
	CountryName

	Scan
PK	**Scan_ID**
	StartTime
	EndTime
	Classification
	Port
	Scan_Count
	Response_Count

	IPCount
FK	**Scan_ID**
	count

	IPScan
PK	**Scan_ID**
	Saddr
	Saddr_decimal
	Sport
	Success
	Timestamp
	Country_Name
	Country_Code

Technical Challenges

Front End

Challenge: needed access to high-performance hardware
Solution: use AWS HVM

Challenge: needed to manage long-running tools and create repeatable, dependable output
Solution: Python wrapper classes

Challenge: needed to supplement Zmap-generated output
Solution: use JSON output, purchase Country to IP mapping DB, and augment JSON records per scan.

Storm Processing

Challenge: Kafka cluster connections require SSL and python >= 2.7.9
W205 AMIs use python 2.7.3 and lack SSL
Unable to add SSL / upgrade python on W205 AMIs
Solution: build new AMI with python 2.7.12 and SSL

Challenge: latest versions of apache storm (1.1.0) & streamparse (3.4.0) are incompatible; latest version of streamparse (3.4.0) uses non-Clojure topology definitions
Solution: Install apache storm 0.9.3 and streamparse 2.1.4 to mimic successful W205 AMI installation

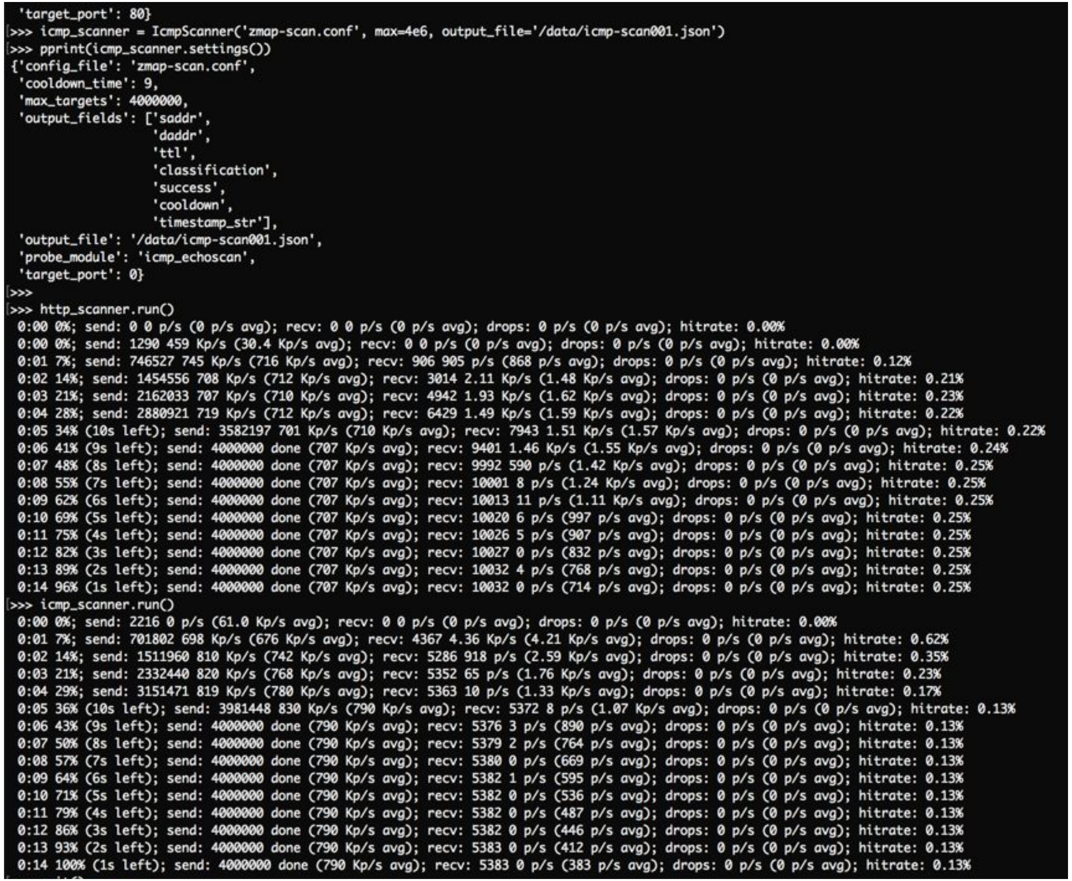
Project Results

As of today, we are able to:

- Scan the Internet with both TCP and ICMP scans
- Send data into our Kafka-as-a-Service cluster
- Store/lookup data in Postgres

What we have left:

- Confirm our analytics are correct in Storm
- Scale the infrastructure to meet our processing needs



```
 'target_port': 80}
>>> icmp_scanner = IcmpScanner('zmap-scan.conf', max=4e6, output_file='/data/icmp-scan001.json')
>>> pprint(icmp_scanner.settings())
{'config_file': 'zmap-scan.conf',
 'cooldown_time': 9,
 'max_targets': 4000000,
 'output_fields': ['saddr',
                   'daddr',
                   'ttl',
                   'classification',
                   'success',
                   'cooldown',
                   'timestamp_str'],
 'output_file': '/data/icmp-scan001.json',
 'probe_module': 'icmp_echoscan',
 'target_port': 0}
>>>
>>> http_scanner.run()
 0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:00 0%; send: 1290 459 Kp/s (30.4 Kp/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:01 7%; send: 746527 745 Kp/s (716 Kp/s avg); recv: 906 905 p/s (868 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.12%
 0:02 14%; send: 1454556 708 Kp/s (712 Kp/s avg); recv: 3014 2.11 Kp/s (1.48 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.21%
 0:03 21%; send: 2162033 707 Kp/s (710 Kp/s avg); recv: 4942 1.93 Kp/s (1.62 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.23%
 0:04 28%; send: 2880921 719 Kp/s (712 Kp/s avg); recv: 6429 1.49 Kp/s (1.59 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.22%
 0:05 34% (10s left); send: 3582197 701 Kp/s (710 Kp/s avg); recv: 7943 1.51 Kp/s (1.57 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.22%
 0:06 41% (9s left); send: 4000000 done (707 Kp/s avg); recv: 9401 1.46 Kp/s (1.55 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.24%
 0:07 48% (8s left); send: 4000000 done (707 Kp/s avg); recv: 9992 590 p/s (1.42 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:08 55% (7s left); send: 4000000 done (707 Kp/s avg); recv: 10001 8 p/s (1.24 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:09 62% (6s left); send: 4000000 done (707 Kp/s avg); recv: 10013 11 p/s (1.11 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:10 69% (5s left); send: 4000000 done (707 Kp/s avg); recv: 10020 6 p/s (997 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:11 75% (4s left); send: 4000000 done (707 Kp/s avg); recv: 10026 5 p/s (907 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:12 82% (3s left); send: 4000000 done (707 Kp/s avg); recv: 10027 0 p/s (832 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:13 89% (2s left); send: 4000000 done (707 Kp/s avg); recv: 10032 4 p/s (768 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
 0:14 96% (1s left); send: 4000000 done (707 Kp/s avg); recv: 10032 0 p/s (714 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.25%
>>> icmp_scanner.run()
 0:00 0%; send: 2216 0 p/s (61.0 Kp/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:01 7%; send: 701802 698 Kp/s (676 Kp/s avg); recv: 4367 4.36 Kp/s (4.21 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.62%
 0:02 14%; send: 1511960 810 Kp/s (742 Kp/s avg); recv: 5286 918 p/s (2.59 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.35%
 0:03 21%; send: 2332440 820 Kp/s (768 Kp/s avg); recv: 5352 65 p/s (1.76 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.23%
 0:04 29%; send: 3151471 819 Kp/s (780 Kp/s avg); recv: 5363 10 p/s (1.33 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.17%
 0:05 36% (10s left); send: 3981448 830 Kp/s (790 Kp/s avg); recv: 5372 8 p/s (1.07 Kp/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:06 43% (9s left); send: 4000000 done (790 Kp/s avg); recv: 5376 3 p/s (890 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:07 50% (8s left); send: 4000000 done (790 Kp/s avg); recv: 5379 2 p/s (764 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:08 57% (7s left); send: 4000000 done (790 Kp/s avg); recv: 5380 0 p/s (669 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:09 64% (6s left); send: 4000000 done (790 Kp/s avg); recv: 5382 1 p/s (595 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:10 71% (5s left); send: 4000000 done (790 Kp/s avg); recv: 5382 0 p/s (536 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:11 79% (4s left); send: 4000000 done (790 Kp/s avg); recv: 5382 0 p/s (487 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:12 86% (3s left); send: 4000000 done (790 Kp/s avg); recv: 5382 0 p/s (446 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:13 93% (2s left); send: 4000000 done (790 Kp/s avg); recv: 5383 0 p/s (412 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
 0:14 100% (1s left); send: 4000000 done (790 Kp/s avg); recv: 5383 0 p/s (383 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.13%
```

Limitations and Future Extensions

- Limitations
  - The cost of the infrastructure was beginning to grow by the end of the project
  - The front end would scan even faster using PF_RING network drivers.
  - Later versions of Storm & Streamparse may be more efficient

- Future Extensions / Additions
  - Expand beyond TCP and ICMP scans
  - Enable different JSON record formats depending upon the type of scan
  - Provide not only Country-level data, but also city-level and ISP-level lookups
  - Sharded or more scalable database design for analysis (i.e., AWS Aurora PostgreSQL or AWS RedShift)
  - Hosted query option for data
  - Geographic distribution of scanning servers
  - HTML visualizations for our streaming and batch analytics

Thank you

Appendix