Using SILK to Collect and Analyze Useful (and surprisingly interesting) Netflow Data

Learning Objectives

- How to use netflow to analyze the behavior of an IP network
- SILK tool suite and how to use it to analyze flows
- PySILK API essentials
- Basic security analytical workflows
- Data enrichment strategies
- Flow data usage across analytical ecosystems
- More advanced analytics
- Opportunities for further research

About this Presentation

(AKA vital info to plan procrastination, breaks, and/or naps)

(02) About Me & My Work

(05) About Netflow + Use Cases

(10) SILK Architecture + Setup

(10) SILK CLI / PySILK basics

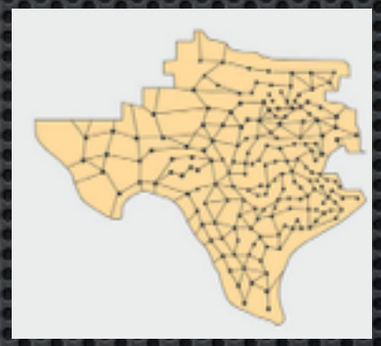
(28-50) All hell breaks loose

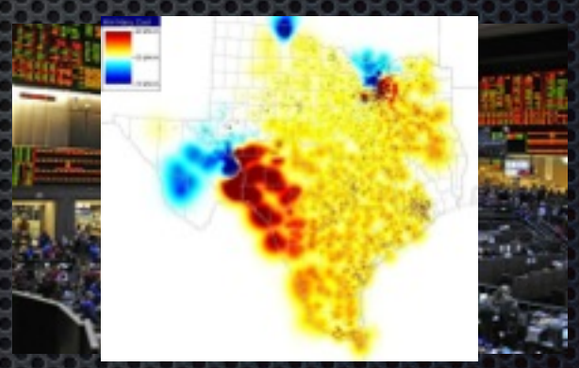
(∞) Questions

About Me

- 17 years in security
- My second time learning python
- I am a Principal at ERCOT
- Pajama Sam
- Ellen Ripley

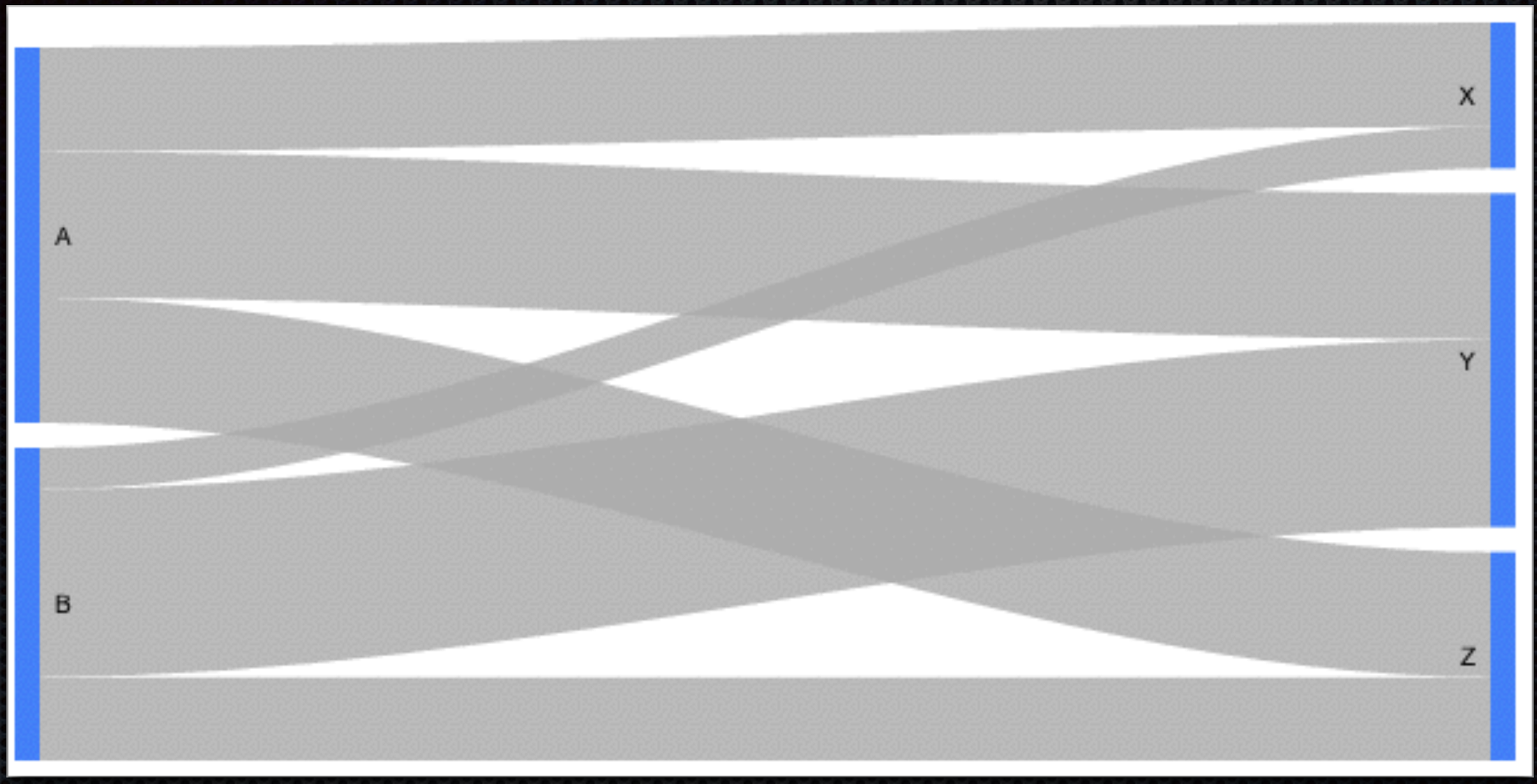
About ERCOT





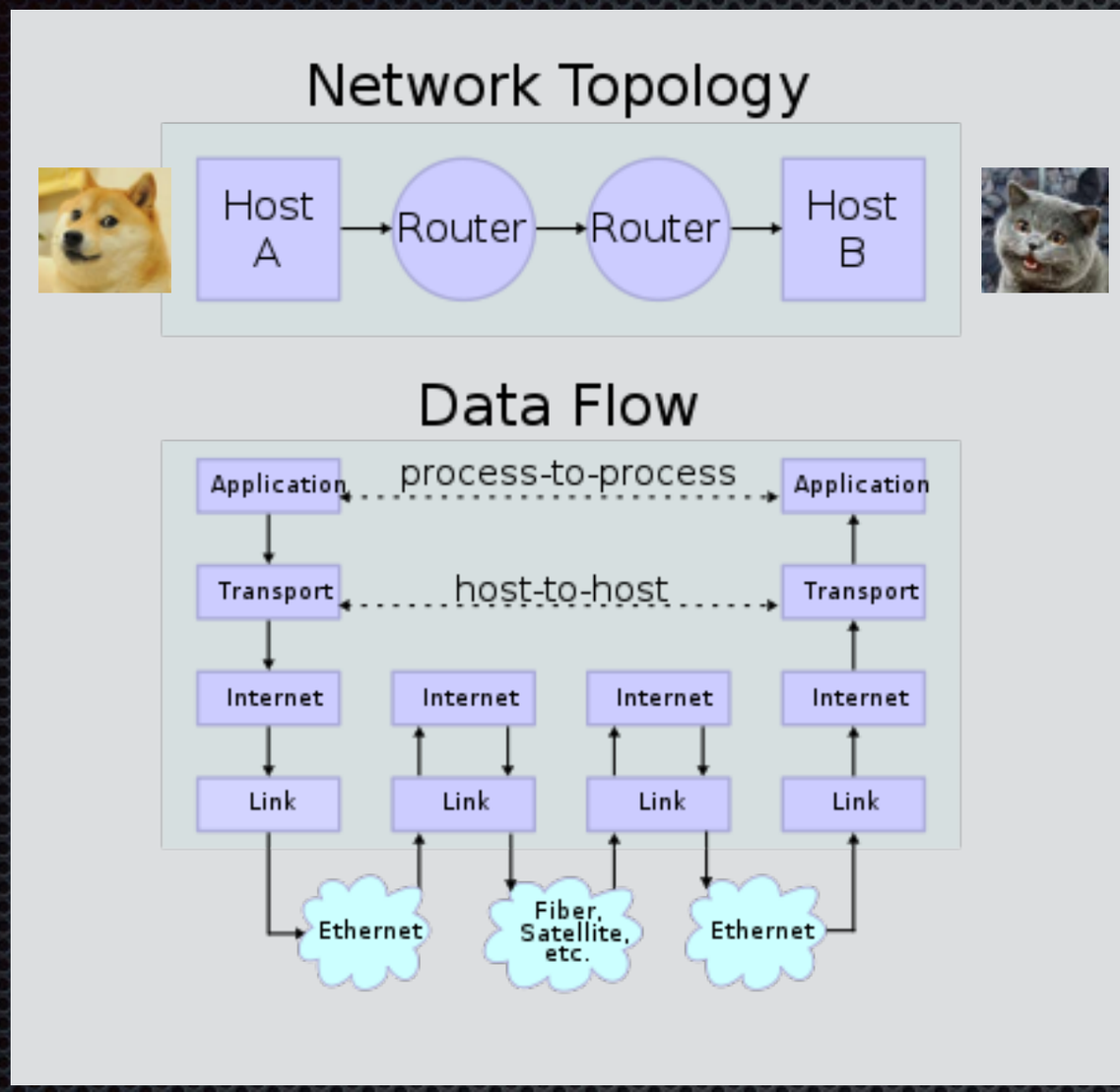
- Independent System Operator
- Retail, Wholesale, and Congestion Market Operator
- 501(c)4 Social Welfare Organization
- Nerd zoo: Engineers, Policy Wonks, Computer Geeks, Economists, Analysts, Operators
- We’re Hiring Smart People
- I’m hiring an intern http://j.mp/cyber-intern

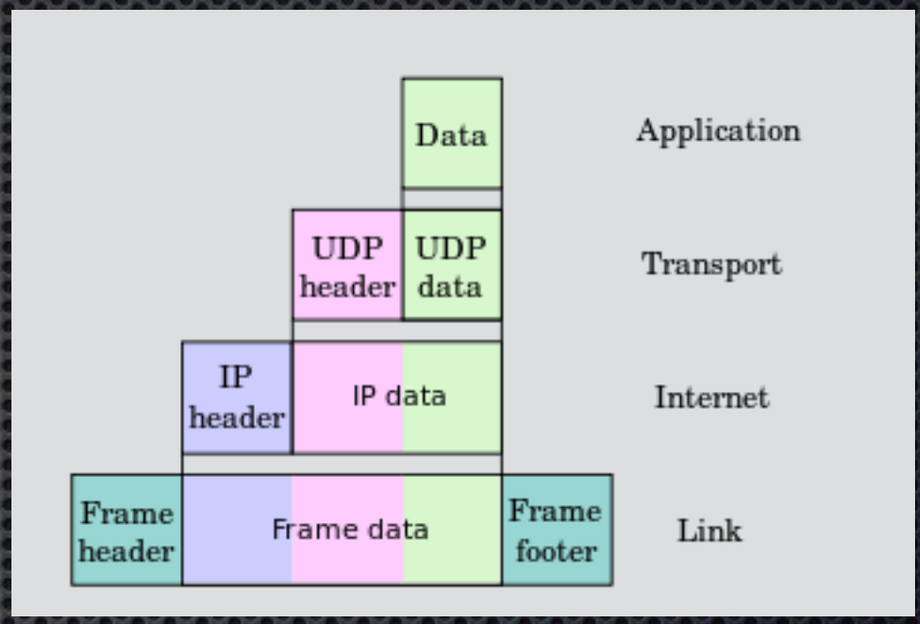
“We serve the public by ensuring a reliable grid, efficient electricity markets, open access and retail choice.”

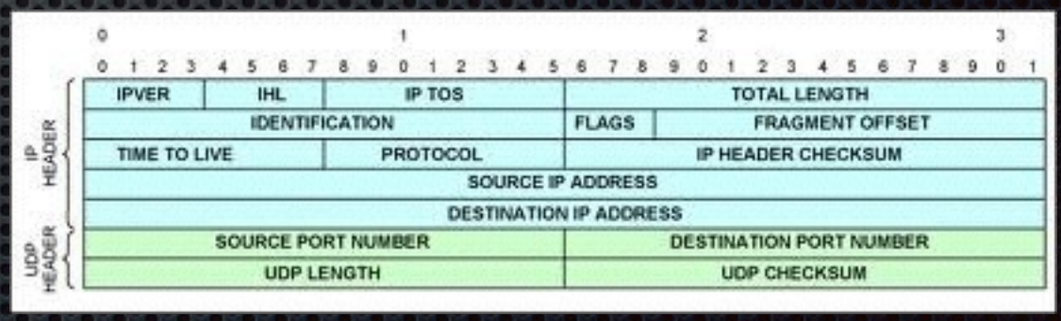


About NetFlow

Network Datagrams





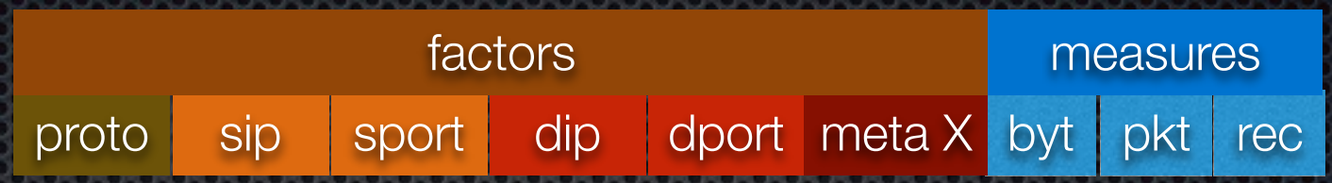


IP Datagram → Flow Record

- Phone Bill, not Phone Tap
- Streaming summary of live traffic
- Many formats: netflow v5-9, sFlow, IPFIX
- L2 Observations: Packets -> Sessionized Reports
- L3 Observations: Transit Sampling & Summarization

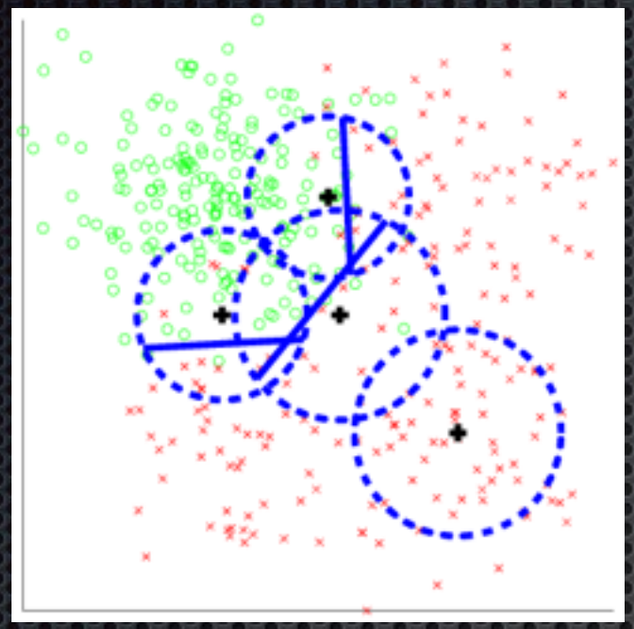
Bits 0..15	Bits 16..31
Version = 0x000a	Message Length = 64
Export Timestamp = 2005-12-31 23:59:60	
Sequence Number = 0	
Observation Domain ID = 12345678	
Set ID = 2 (Template)	Set Length = 20
Template ID = 256	Number of Fields = 3
sourceIPv4Address	Field Length = 4
destinationIPv4Address	Field Length = 4
packetDeltaCount	Field Length = 4
Set ID = 256 (Data Set	Set Length = 28
Record 1, Field 1 = 192.168.0.201	
Record 1, Field 2 = 192.168.0.1	
Record 1, Field 3 = 235 Packets	
Record 2, Field 1 = 192.168.0.202	
Record 2, Field 2 = 192.168.0.1	
Record 2, Field 3 = 42 Packets	

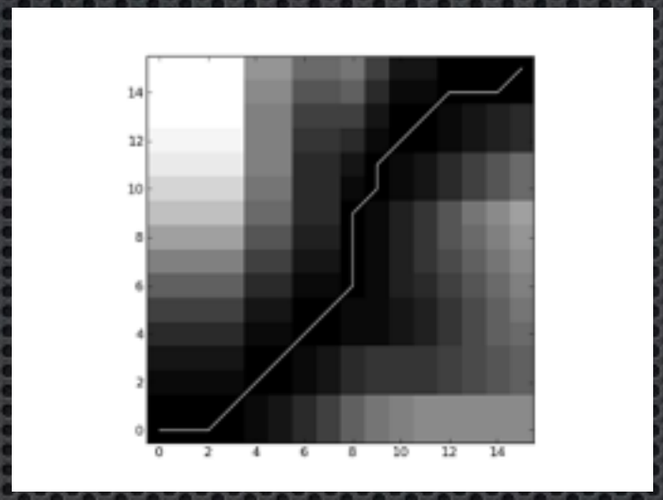
Use Cases

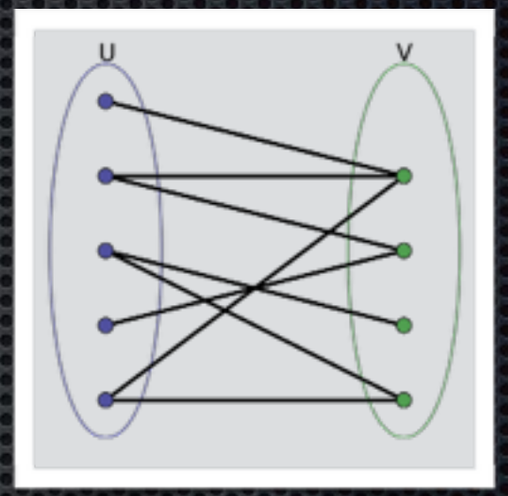


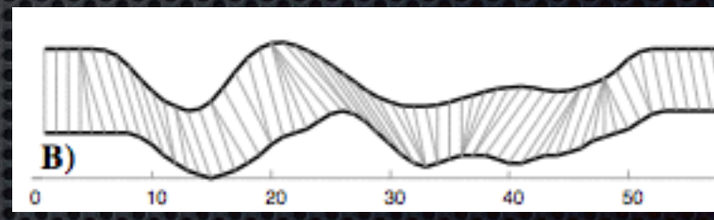
- Core properties:
  - compact representation of network activity
  - can be generated out-of-band, inline, or post-hoc
- Investigate historical traffic flows, generate timelines, corroborate incident data
- Automate basic detective measures
  - presence, absence, threshold
- Compute aggregate & time series statistics
- Characterize endpoints, applications
- Generate and evaluate behavioral models
- Generally not meant to be used for real-time applications

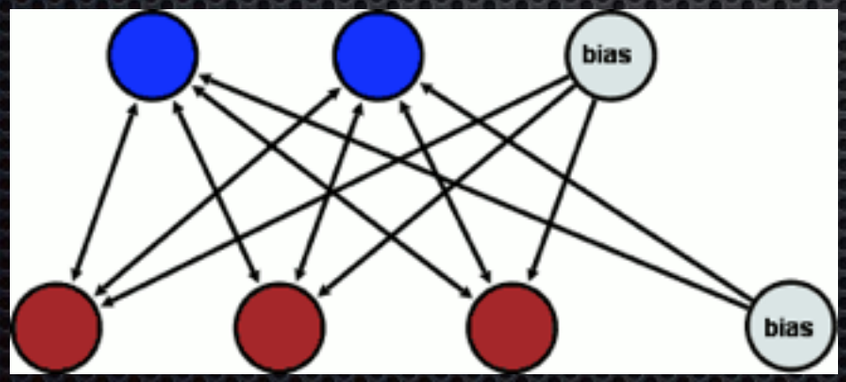
Bizarre Things are Possible

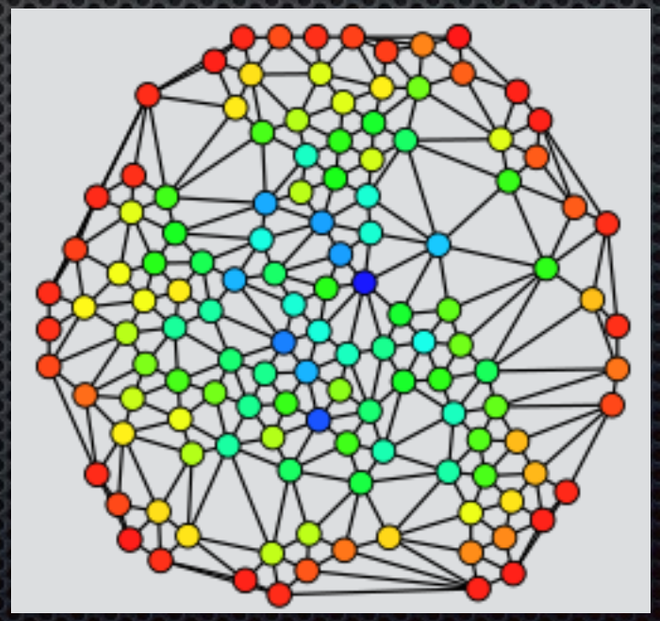


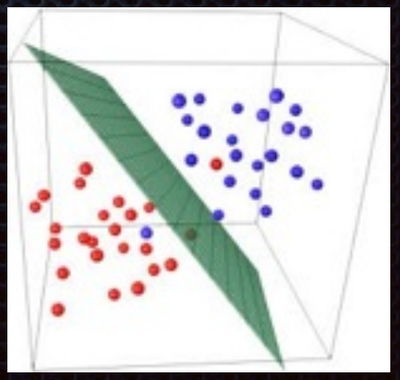


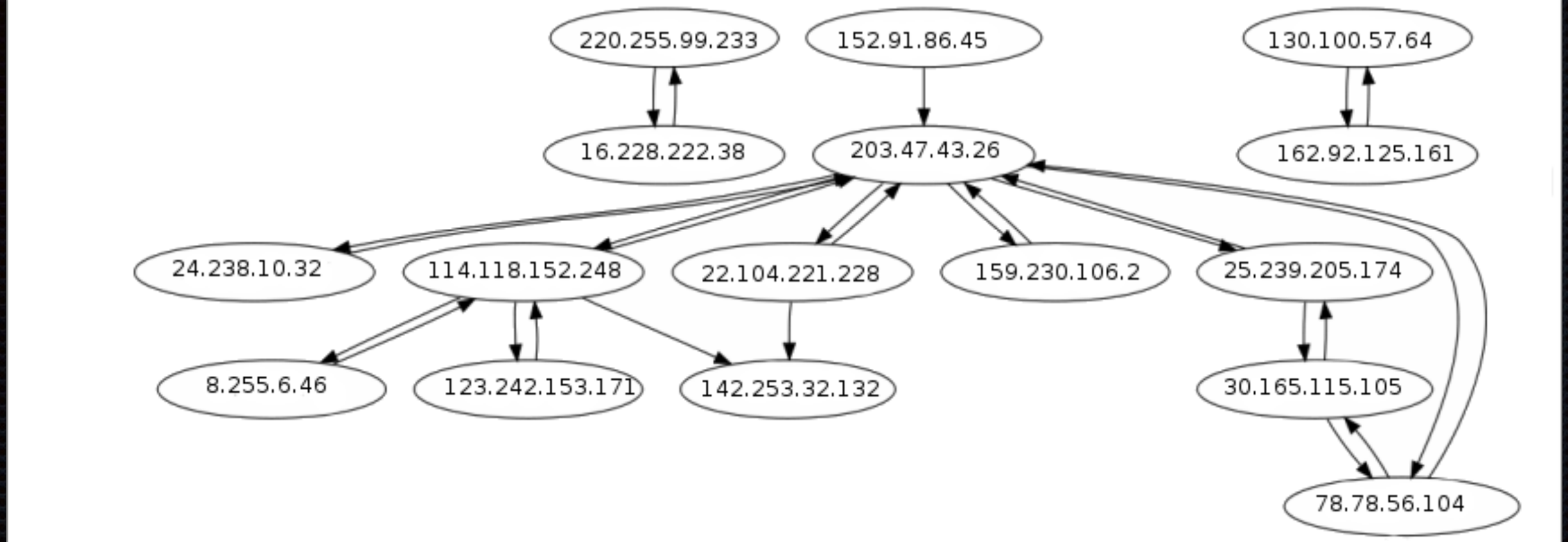








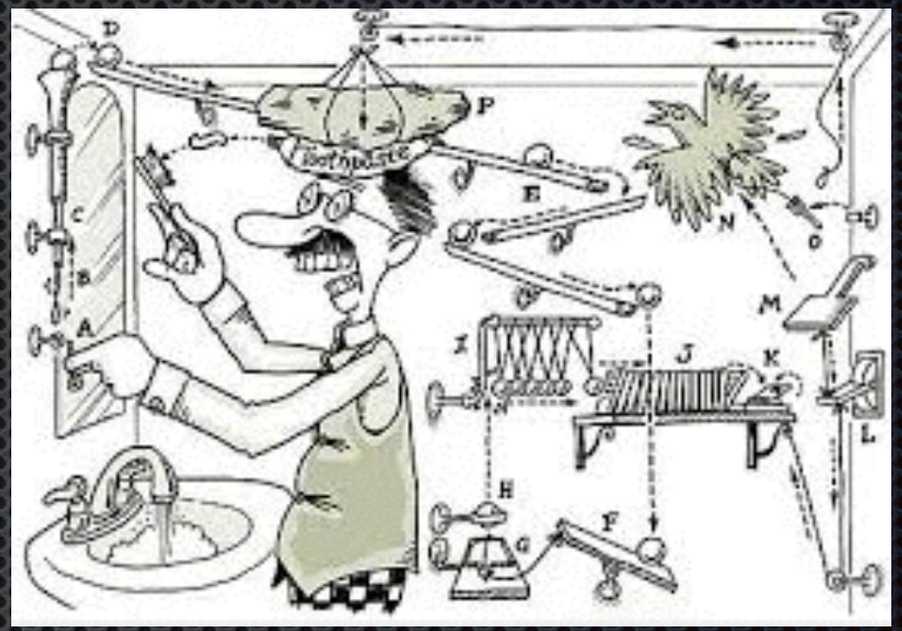



220.255.99.233
152.91.86.45
130.100.57.64
16.228.222.38
203.47.43.26
162.92.125.161
24.238.10.32
114.118.152.248
22.104.221.228
159.230.106.2
25.239.205.174
8.255.6.46
123.242.153.171
142.253.32.132
30.165.115.105
78.78.56.104

About SILK

SILK - Essential Parts

- SILK - System For Internet Level Knowledge
- Analysis suite - CLI (select, sort, group…)
  - **`rwfilter`**/**`rwuniq`**/**`rwstats`**
- Data enrichment - labeling, tagging, etc.
  - **`rwpmapbuild`**



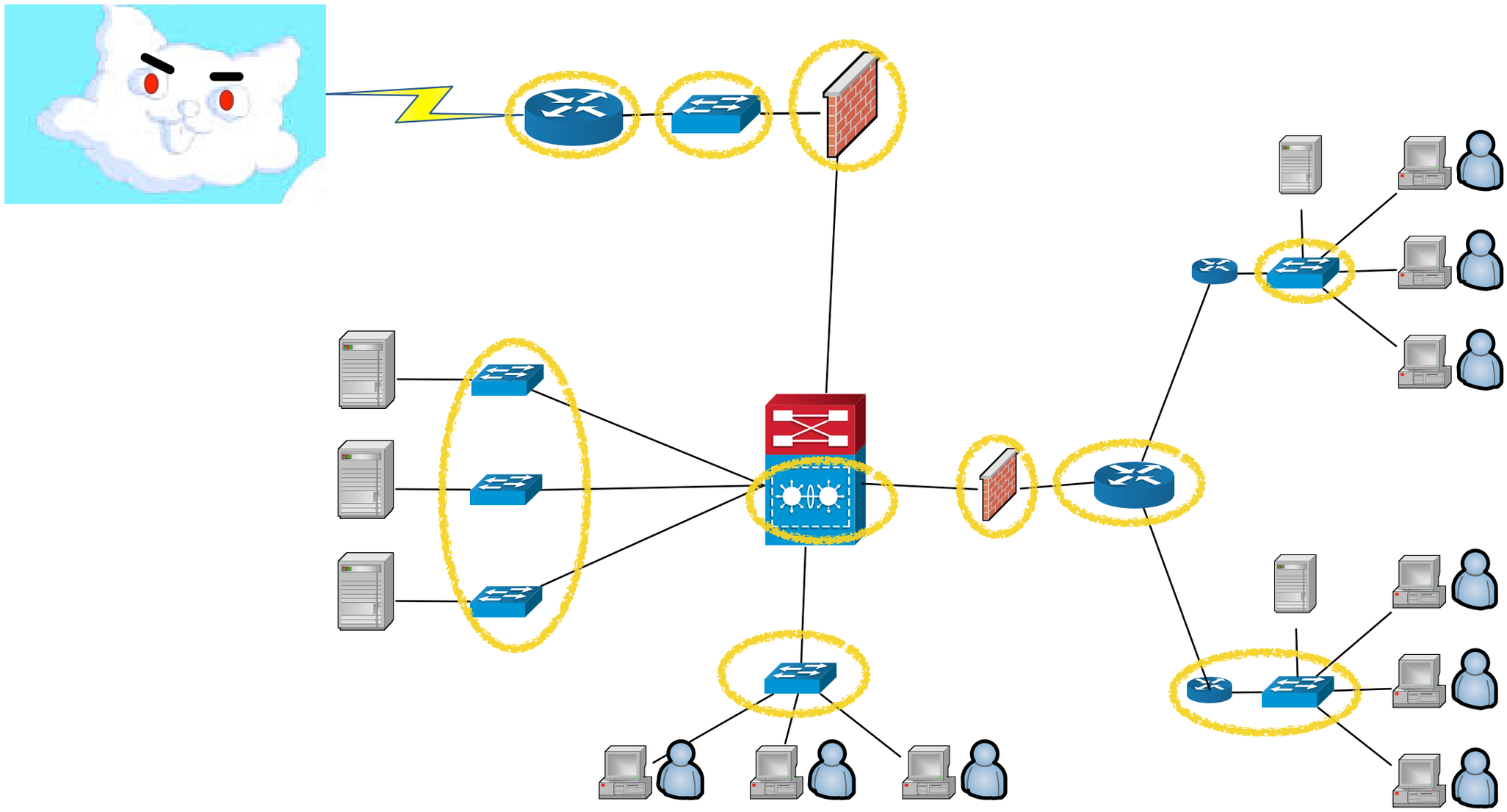
- Flow collection
  - **`yaf`** reads PCAPs or sniffs live traffic, emits IPFIX, tags DPI metadata
  - **`flowcap`** collects ipfix/netflow from flow-enabled devices
- Flow packing, pruning, and transport
  - **`rwflowpack`**/**`rwsender`**/**`rwreceiver`**
- PySILK - Python bindings for most SILK functions

Design Decision Point

Traffic Metering & Flow Collection

- Meter traffic for most diagnostic leverage
- Sample traffic vs process everything
- Collect flows offline or online?
- Intra-network L2 metering?
  - (Hostile Clients? Sensitive Segments?)
- Privacy issues? (http://www.dhs.gov/privacy-compliance)
  - http://phys.org/news/2013-03-easy-identity-cell.html
- Selectable retention? Discovery Requirements?
- n > 1 meters → flow aggregation, post-processing

Optimal Sensor Placement



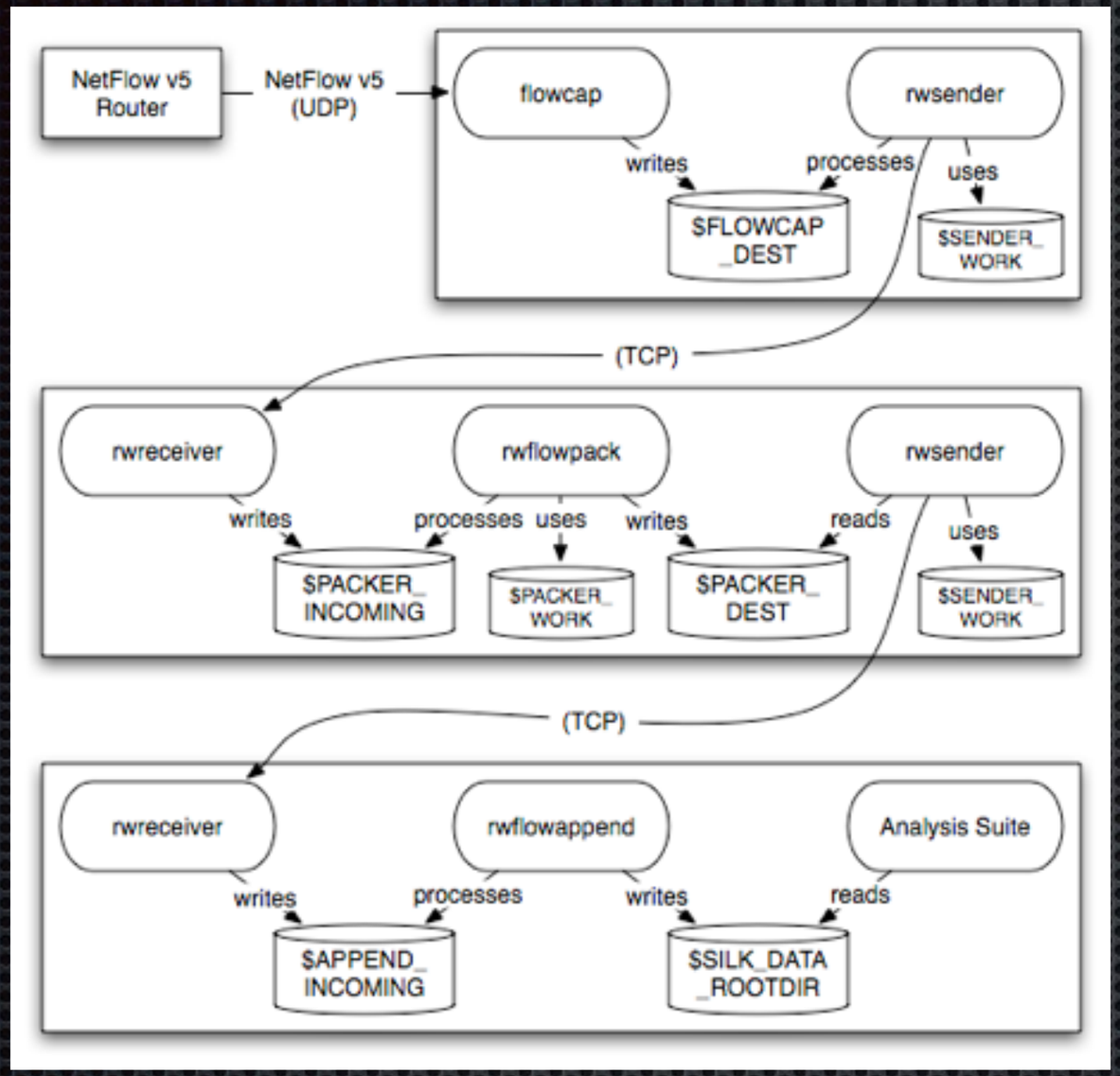
LOL it depends

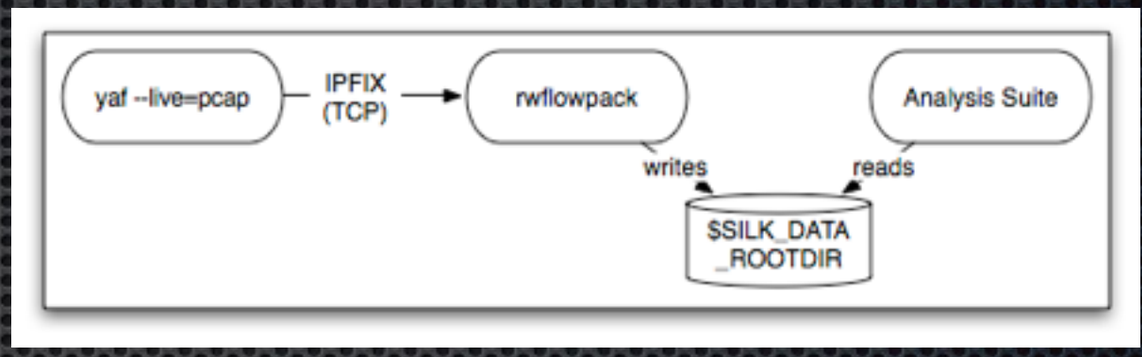
Design Decision Point

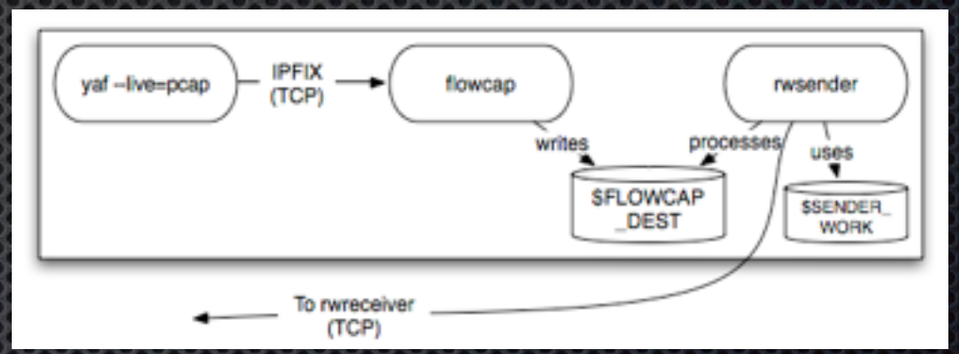
Flow Storage

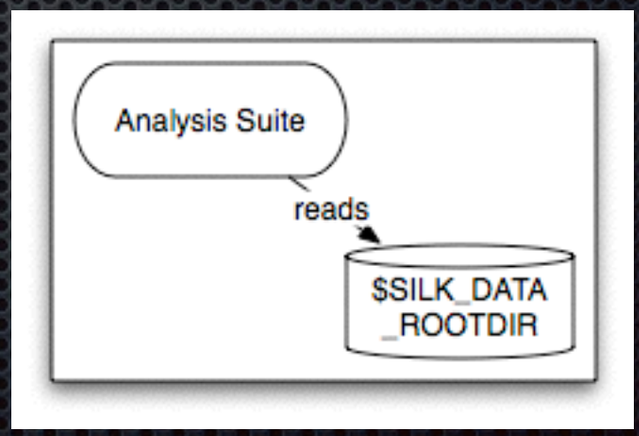
- If disk is cheap and the network is fast:
  - Batch process pcap & centralize
  - `rwflowpack` + NAS
- I have questionable judgement or my network is unreliable:
  - `rwsender`/`rwreceiver` **(LET'S TRY ANYWAY!)**
- I have poor judgement:
  - Store on isolated sensors
- I live in the year 1997, and I am a slave to tools from 1997:
  - RDBMS
- I live in the year 2014, I like to hack data, and I have poor judgement
  - RDBMS **(LET'S TRY ANYWAY!)**
  - Doc store
  - Raw KVS
- I live in the year 2014 and I like to hack data
  - ElasticSearch
  - HDFS + tsv/txt
  - ★Avro
  - ★Parquet
  - ★Tachyon $$$
  - ★Spark $$$ **(LET'S TRY ANYWAY!)**

SILK Configuration









http://tools.netsa.cert.org/silk/install-handbook.html

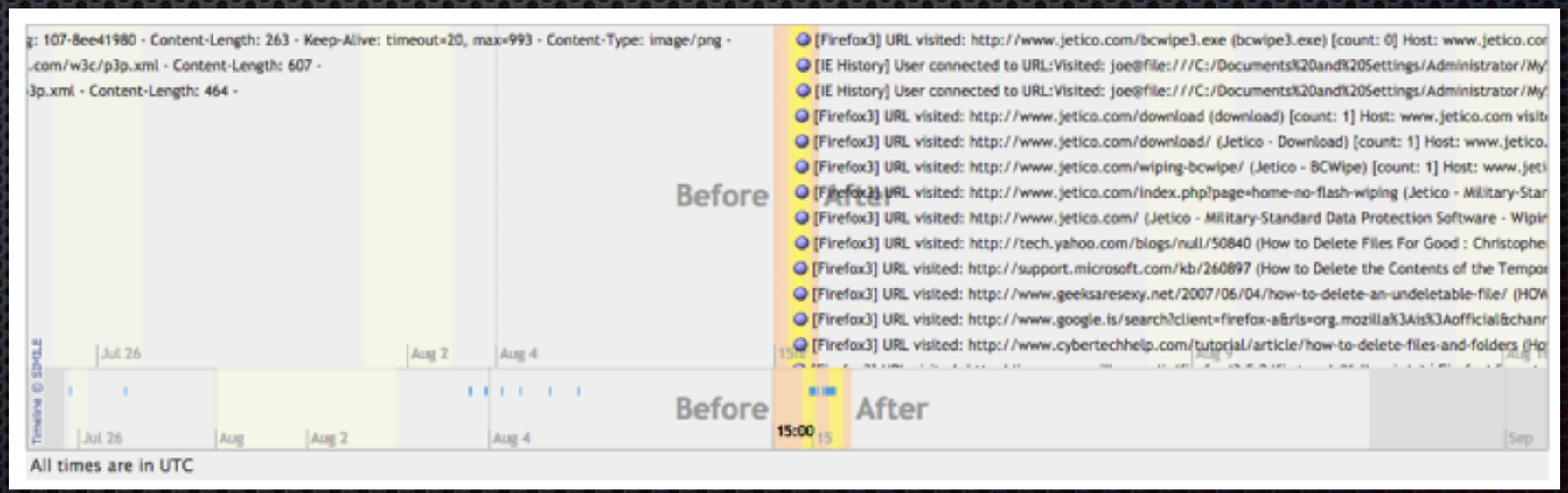
Essential Config Files

- **`silk.conf`** (for analysis suite)
- address_types.pmap (install handbook S 3.2)
- country_codes.pmap (install handbook S 3.3)
- **`sensor.conf`** - probe definitions
- **`rwflowpack.conf`**
- **`rwsender.conf`**/**`rwreceiver.conf`**
- **`yaf`** startup options

http://tools.netsa.cert.org/silk/install-handbook.html

Use Case - Investigations

- What did nodeX talk to before we got that virus alarm?
- Which hosts regularly use that database?
- Have we seen any traffic from that internet host before?
- Have any of our hosts passed traffic to known C2 infrastructure?
- Which ASNs are responsible for the most spam deliveries?



Investigation - CLI example

Basic filtering - what was 192.168.1.0 doing 2-6PM?

```
rwfilter --any-cidr=192.168.1.0/24          --pass=-  \
  --type=in,inweb,out,outweb                          \
  --start=2014/05/03:14 --end=2014/05/03:18           \
  rwcut --fields=stime,sip,dip,application,protocol   \
    --timestamp-format=iso
```

rwfilter(1)

rwcut(1)

```
                sTime|             sIP|             dIP|appli|pro|
2014-05-03 14:03:36.474| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:36.474| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:36.724| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:36.724| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:36.973| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:36.973| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:37.221| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:37.221| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:37.473| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:37.473| 74.125.131.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:42.957| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:42.957| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.205| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.205| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.464| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.464| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.703| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.704| 74.125.137.125|     192.168.1.52|    0|   6|
2014-05-03 14:03:43.964| 74.125.137.125|     192.168.1.52|    0|   6|
```

Investigation - CLI Example

Sets, Grouping, and CSV output

```
jimjam:pytx ryan$ cat <<END > suspicious.txt
> 192.168.1.0/24
> 172.17.4.55
> END
```

```
rwfilter --start=2014/05/03 --end=2014/05/03 \
  --sipset=suspicious.set --type=out,outweb  \
  --pass=- | rwuniq --delimited=,            \
  --fields=dip,dport,protocol,application    \
  --values=bytes,packets,records
```

```
dIP,dPort,protocol,application,Bytes,Packets,Records
108.162.232.202,80,6,80,349,5,1
205.234.175.175,80,6,80,1406,31,1
178.255.83.2,80,6,80,1060,15,3
192.116.242.6,80,6,80,342,5,1
4.59.136.200,80,6,80,56608,484,48
96.16.6.123,80,6,80,2858,29,6
70.167.227.245,80,6,80,341,5,1
108.162.232.200,80,6,80,1035,15,3
23.64.165.163,80,6,80,1502,20,4
96.16.6.129,80,6,80,3724,37,7
204.193.144.15,80,6,80,869,6,1
23.72.136.225,80,6,80,3857,66,1
23.72.136.208,80,6,80,337,5,1
96.16.6.131,80,6,80,496,5,1
72.21.91.29,80,6,80,2542,39,3
```

```
rwsetbuild suspicious.txt suspicious.set
```

rwsetbuild(1)

rwuniq(1)

Investigation - CLI Example

Labeling *(also, dead simple botnet detection)*

```
curl -s https://reputation.alienvault.com/reputation.data |\
  awk -F\# 'BEGIN {OFS=" "} {print $1 "/32",$4}'          |\
  SILK_CLOBBER=true rwpmapbuild --input=- --mode=ipv4      \
    --output="reputation_$( date +%Y%m%d_%H%M%S ).pmap"
```

rwpmapbuild(1)

```
rwfilter --start=2014/05/01 --end=2014/05/01          \
  --type=out,outweb --pmap-file=rep:reputation.pmap   \
  --pmap-dst-rep='UNKNOWN' --fail-destination=-       |\
  rwuniq --delimited=, --time=epoch --sort            \
  --pmap-file=rep:reputation.pmap                      \
  --fields=9,2,4,5,dst-rep --values=bytes,records
```

rwpmapfilter(3)

```
1399006678,144.76.40.132,53,17,C&C,246,1
1399006678,194.85.61.76,53,17,Malware Domain,444,1
1399006681,109.70.26.37,53,17,Malware Domain,255,1
1399006681,194.85.61.76,53,17,Malware Domain,255,1
1399006682,109.70.26.37,53,17,Malware Domain,1020,6
1399006682,194.85.61.76,53,17,Malware Domain,850,4
1399006683,109.70.26.37,53,17,Malware Domain,888,2
1399006683,194.85.61.76,53,17,Malware Domain,1058,4
1399006684,209.99.40.223,53,17,Malware Domain;Malware IP,249,1
1399006699,69.167.161.33,53,17,Scanning Host,246,1
```

Enrichment Fun

- Internal config management systems
  - Enterprise CMDB
  - Chef/Puppet/Ansible
  - Host OS, Application Role, Criticality
- Registrar Data
  - BGP ASN: CERT Routeviews PMAP repo
  - DNS: Passive DNS, WHOIS
- Coincident event data
  - splunk, logstash, etc.
  - DHCP logs
- SIEM Data
  - User identities
  - Browser Agents/Rando HTTP Request headers
- IOCs
  - AlienVault, MalwareDomainList, ZeusTracker, etc
  - CIF, OpenIOC, STIX, CFM, CYBOX
- GeoIP
  - Country Code/Region/Continent
  - http://datacatalog.worldbank.org/
  - http://bgp.potaroo.net/iso3166/v4cc.html
  - http://cpi.transparency.org/cpi2013/results/

*More is possible when you have indexed pcap

Use Case - Aggregates

- Aggregate different facets of flows
- Projection onto a lower dimensional space
- Feature elimination

- Optional transformations between factors & measures via summing/ binning
- Compute top(N) by facet
- Labeled flows enable many viz options

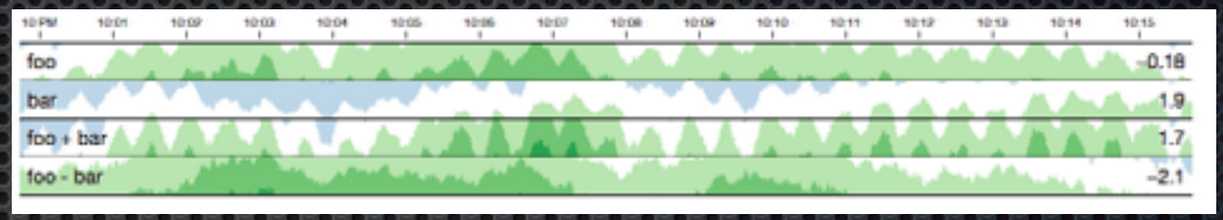
PySILK Overview

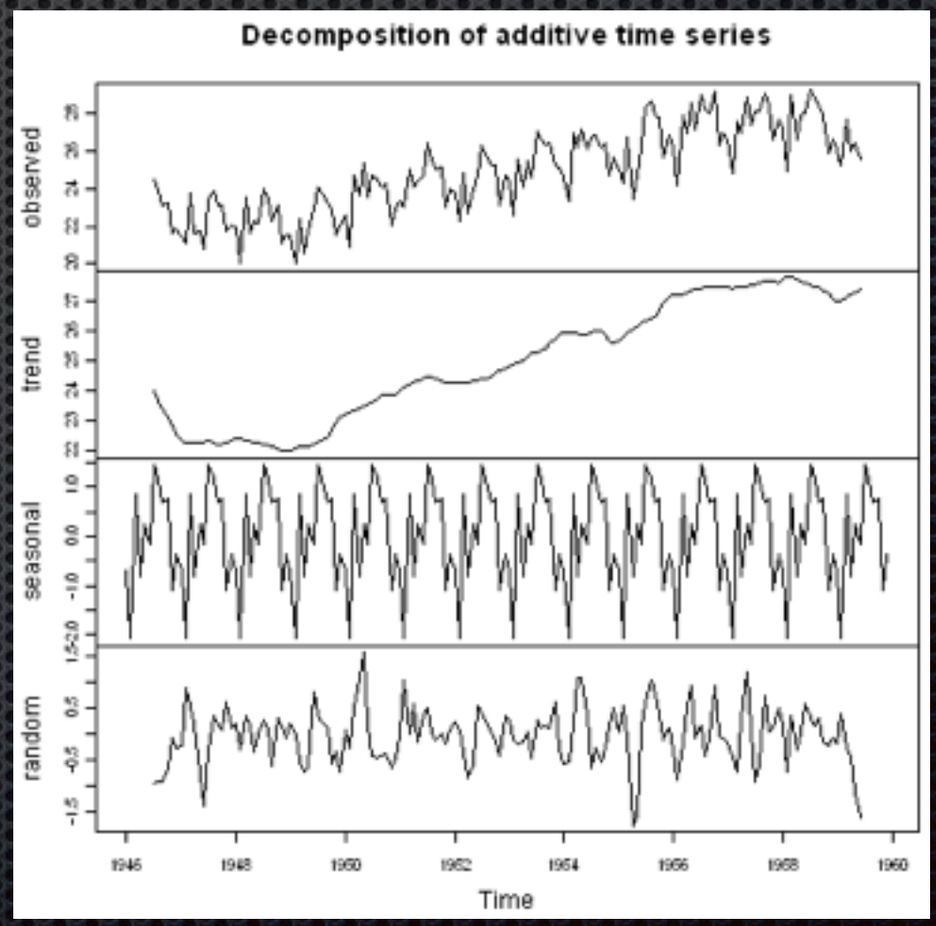
- https://tools.netsa.cert.org/silk/pysilk.html
- silk.site.init_site() - set config, base dir
- silk.site.repository_iter() - iterate over filenames
  - apply conditions for time, traffic type, sensor id
- silk.silkfile_open() - return iterable collection of records
- silk.RWRec - flow record
  - cheat: rec.as_dict()
  - warning: some fields not pickle-able by default serializer
- others: silk.PrefixMap, silk.IPSet, etc…

https://github.com/ryanbreed/pysilk-notebooks_pytx2014/pysilk_pandas_aggregates.ipynb

Use Case - Time Series

- Faceted aggregates over regular time intervals
- Useful for historical comparison
- Can be decomposed via standard transforms
- Useful for forecasting
- Many viz options





TimeSeries in Pandas

https://github.com/ryanbreed/pysilk-notebooks_pytx2014/pysilk_pandas-timeseries.ipynb

Time Series - Shortcut for Simple Aggregates

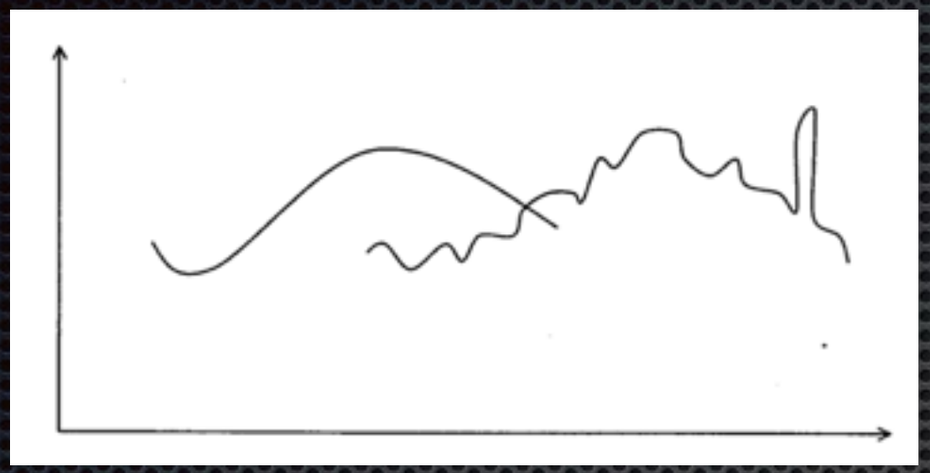
- Simple theory about clustering client network access patterns
- We *could* pull all data straight into DataFrames, filter, sort and then aggregate to make the series…
- Or we could start with preprocessed data…
- **`rwuniq(1) --bin-time`** to the rescue!
- PDQ!

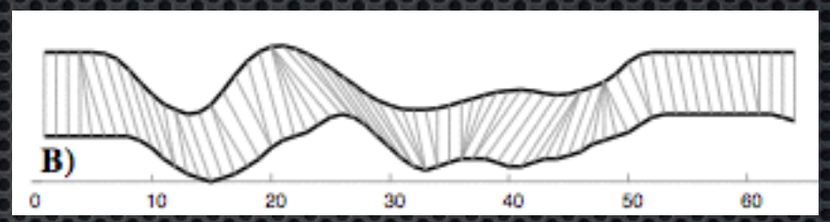
```
real    4m29.218s
user    3m33.256s
sys     0m14.328s
```

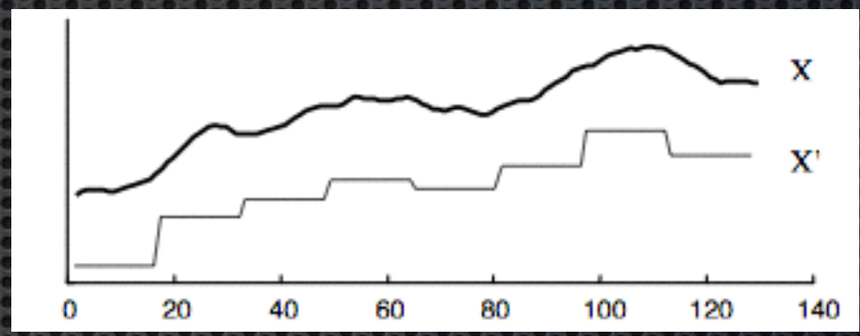
```
rwfilter --sipset=sets/users.set                \
  --start=2014/05/01 --end=2014/05/31           \
  --type=out,outweb                    --pass=- |\
  rwuniq --fields=stime,sip --sort              \
    --values=distinct:dip   --time=iso          \
    --delimited="     "  --bin-time=3600 >\
      clients_dip_distinct.tsv
```

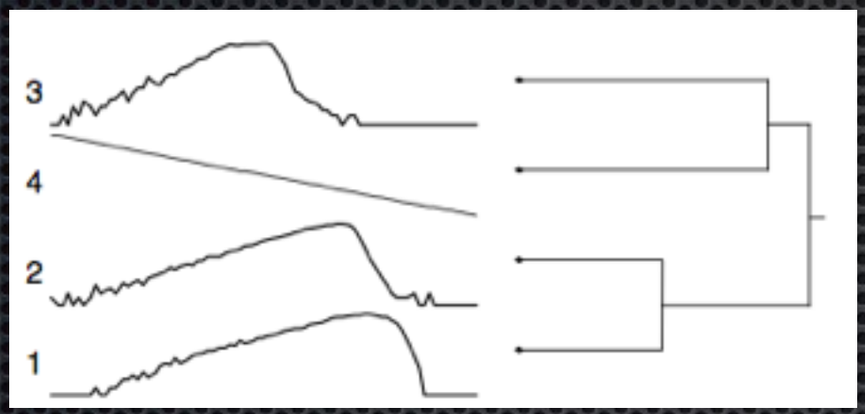
```
sTime    sIP      dIP-Distinct
2014-05-01 00:00:00      192.168.1.46    1
2014-05-01 00:00:00      192.168.1.47    4
2014-05-01 00:00:00      192.168.1.49    9
2014-05-01 00:00:00      192.168.1.51    1
2014-05-01 00:00:00      192.168.1.55    2
2014-05-01 00:00:00      192.168.1.56    5
2014-05-01 00:00:00      192.168.1.61    1
2014-05-01 00:00:00      192.168.1.64    1
2014-05-01 00:00:00      192.168.1.68    1
```

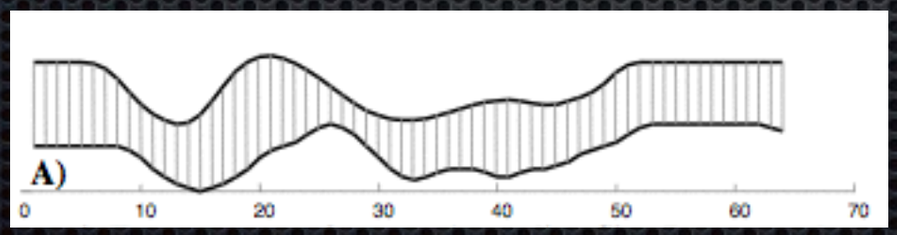
Complexity of Time Series Similarity

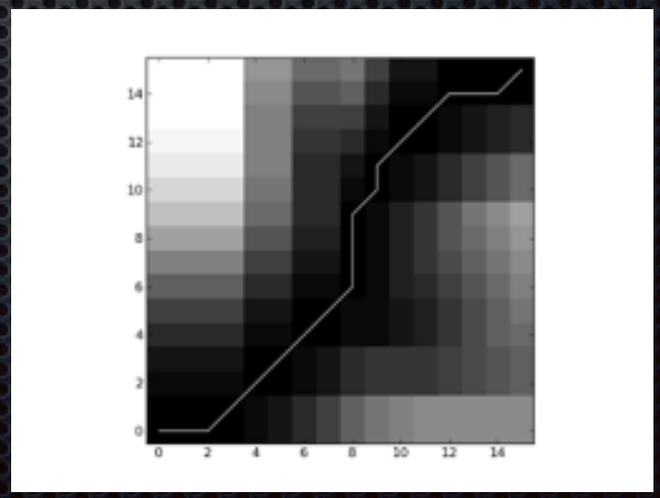










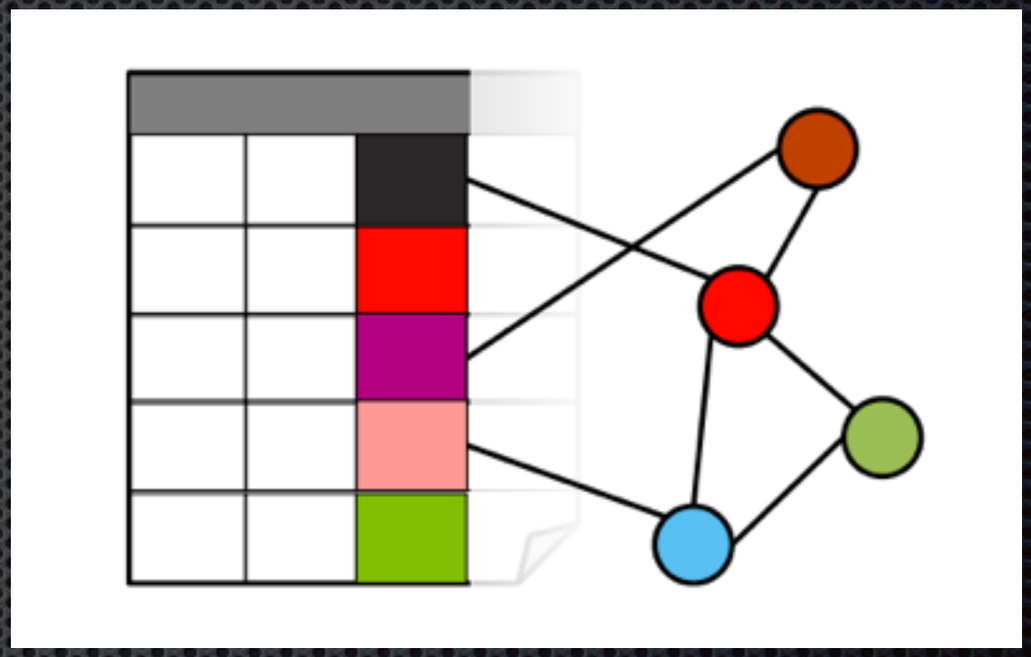


Netflow TimeSeries Similarity Clustering via Dynamic Time Warping

https://github.com/ryanbreed/pysilk-notebooks_pytx2014/pysilk_dtw_hclust.ipynb

Use Case - Network Characterization

- You could use a filter pipeline
  - what server nodes are accessed that aren't dns, dhcp, file, print, backup, logging, mail, or other infrastructure services
  - (ノ °□°) ノ ⁀┻━┻



- Load into graph
  - *G=(V,E)* lol your network is a facebook
  - nodes attached to subnet S accessed by "Internet" and also "users"
  - more measures - pagerank, betweenness, centralities, coloring, etc.
  - great excuse to drop "sparse matrices", "law of triadic closure", and "eigenvector centrality" in a meeting
  - ┬──┬ノ(° _ °ノ)

https://github.com/ryanbreed/pysilk-notebooks_pytx2014/pysilk_graphlab.ipynb

Silk + Apache Spark

- Don’t use PySILK to export flows
- If you’re doing anything interesting, PySpark blows up
- Or the data overflows JVM heap
- (**`rwuniq`**, **`rwcut`** > tsv)

https://github.com/ryanbreed/pysilk-notebooks_pytx2014/pysilk_pyspark_sql.ipynb


Regards.