ML* vs Cryptocoin Miners

*Statistical Analysis

About me

- Jonn Callahan
- Principal Sec Consultant @ nVisium (DC-based)
- Python & Golang
- Math is neat, even if I’m terrible at it
  - I’d liken myself to a mathematical skid, but I’m learning!
- Huge metalhead

- Credit @corsoa

Quick Intro

- Cryptocoin mining is a new (ish) path of monetizing popped boxes
  - Monero (XMR) + Verium (VRM)
  - Vast majority of pools utilize the Stratum protocol
    - This is what we'll actually be analyzing
- Network traffic is aggregated and exposed within AWS via VPC Flow Logs
- Mining traffic probably exhibits some patterned behaviors
- So, is it possible to build a mining-specific IDS running against Flow Logs?

Starting Data

- 24 hours of mining traffic from 12 different cryptocoin miners
  - All Monero (XMR)
  - All the same pool and port
  - Varying c4 sizes (had the best $:hashrate at spot pricing)
- 3 weeks worth of non-mining traffic
  - ~82,000 unique-by-IP, one-way streams (pre-transformation)
  - ~38,000 unique-by-IP-pair, bi-directional streams (post-transformation)
  - ~1,000 unique ENIs (virtual NICs)

VPC Flow Logs

- *Aggregate* 5-tuple network logs organized by ENI
- Aggregation occurs over 10-minute windows
- Two log entries for each single TCP stream

- version account-id interface-id srcaddr dstaddr srcport dstport protocol packets bytes start end action log-status
- 2 178069999999 eni-55aa9999 91.189.89.199 172.12.12.12 3333 45113 6 3 760 1520029322 1520029358 ACCEPT OK

VPC Flow Logs Transformation

- Need to re-organization and correlate multiple flow log entries that correspond to TCP streams from two services communicating
- Can filter out REJECT
  - Focusing strictly on clients that are successfully mining
- Organize by IP and protocol (ignore port)
  - Proto 6 == TCP
  - Client ports can vary (ephemeral ports) but a remote mining server is unlikely to receive anything other than mining traffic – safe to aggregate traffic by unique IP pairs
  - Count all unique IP addresses, use to assume which is the source (saves an AWS API call)

Data Extraction

- Eight features available to us
  - Number of bytes sent by src/dst
  - Number of packets sent by src/dst
  - Number of unique src/dst ports
  - Length of src/dst communication

Strategy

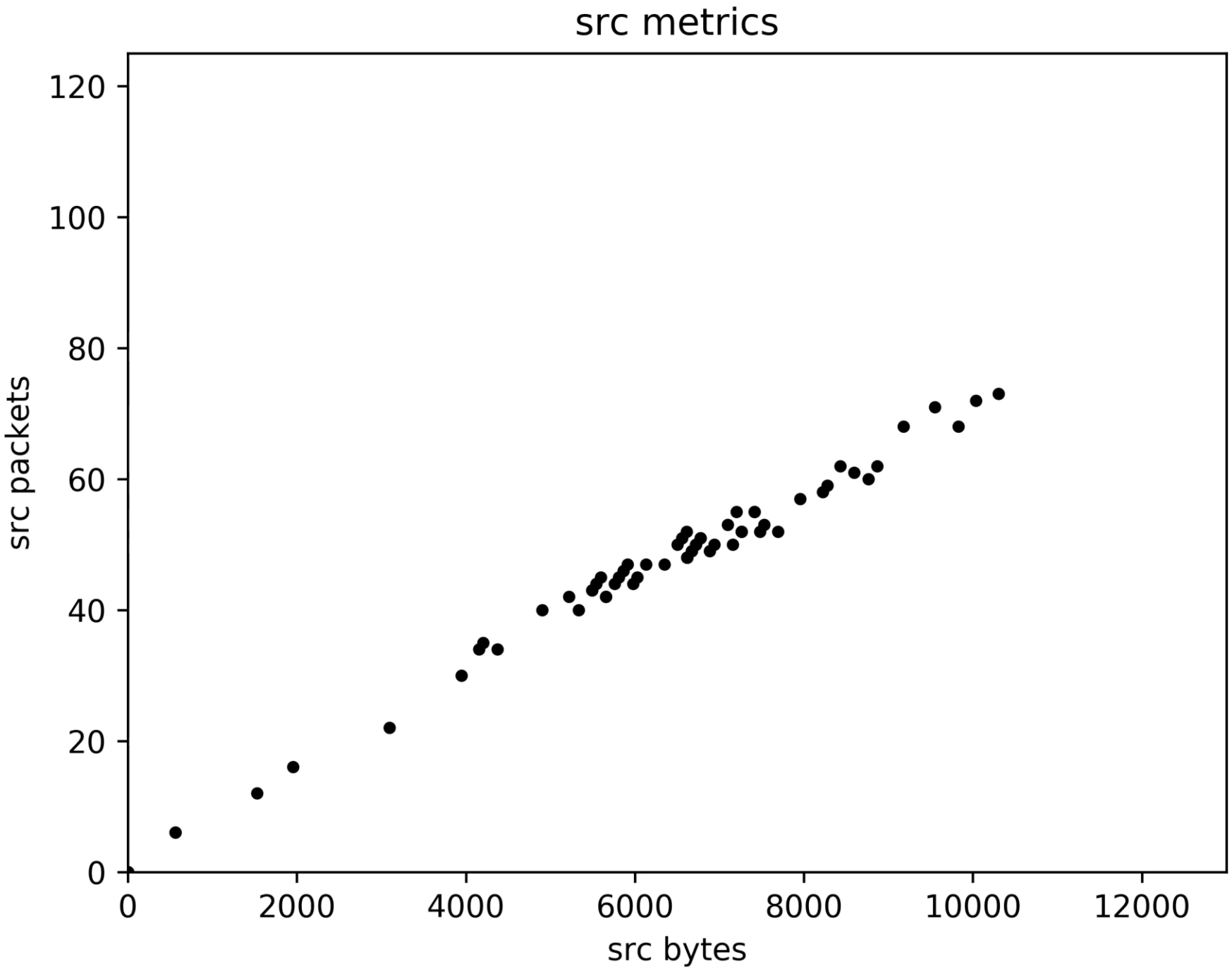
- Graph data (false positives if iteration > 1)
- Eyeball patterns
- Build model
- Filter non-mining traffic
- Repeat until no more false positives

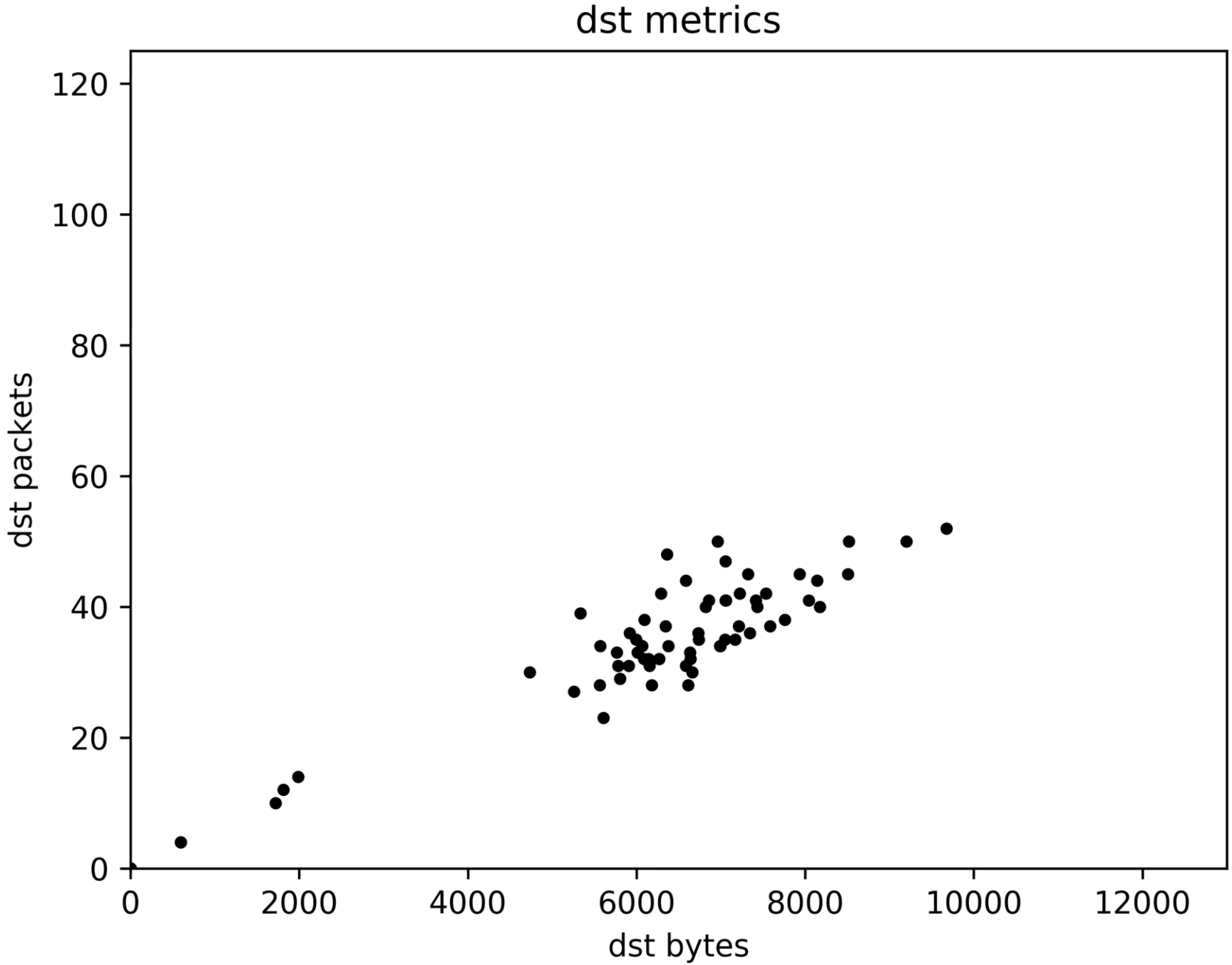
Attempt 0

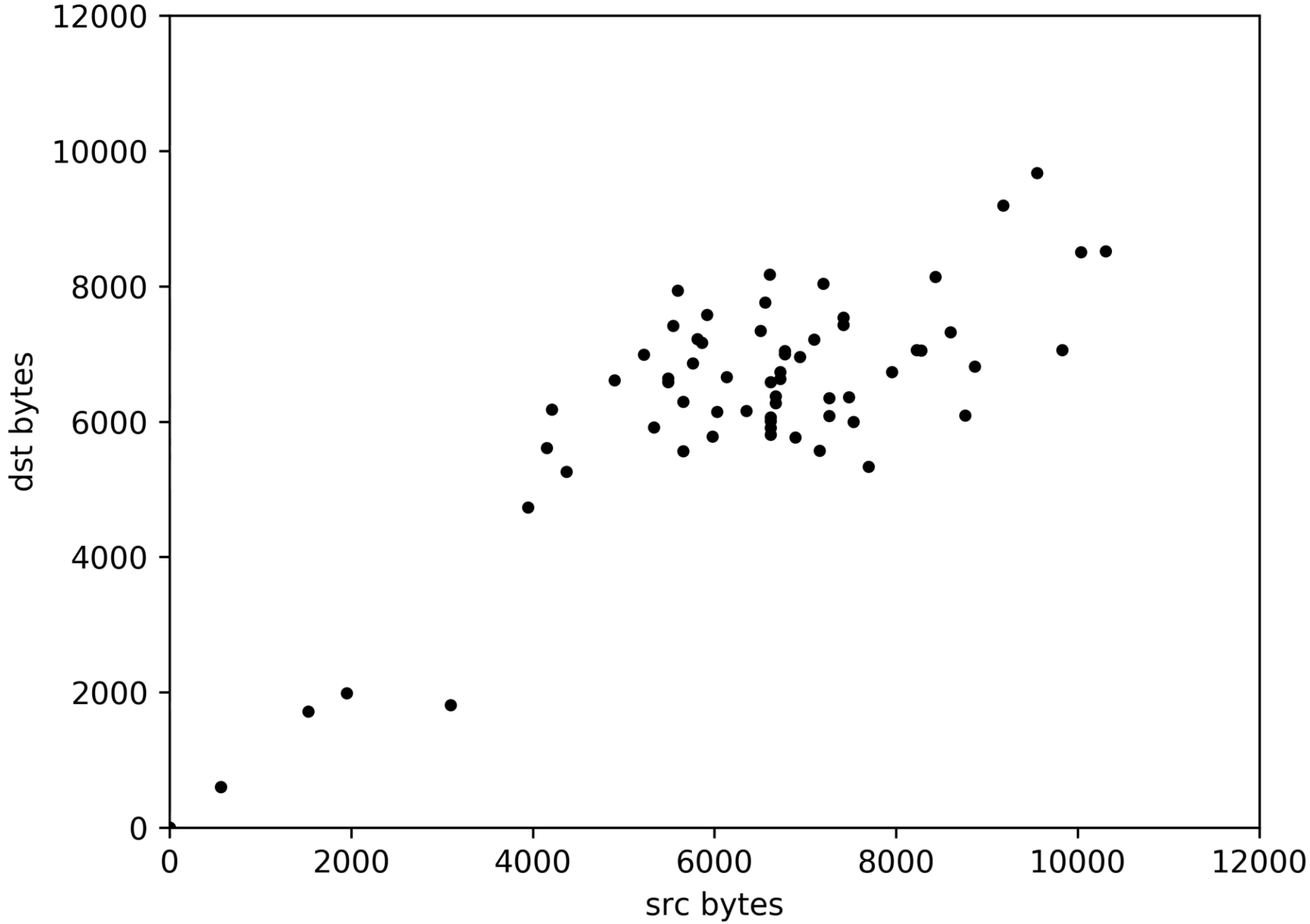
- Quick and dirty check: number of unique dst ports
- Since our infected machine acts as a client communicating with a remote mining pool, it’s safe to assume only a single remote port is used
  - But a variety of ephemeral src ports will likely be used
- Therefore, lets strip out streams with multiple dst ports
- Sadly, this strips out very few streams (~200 of our ~38k)

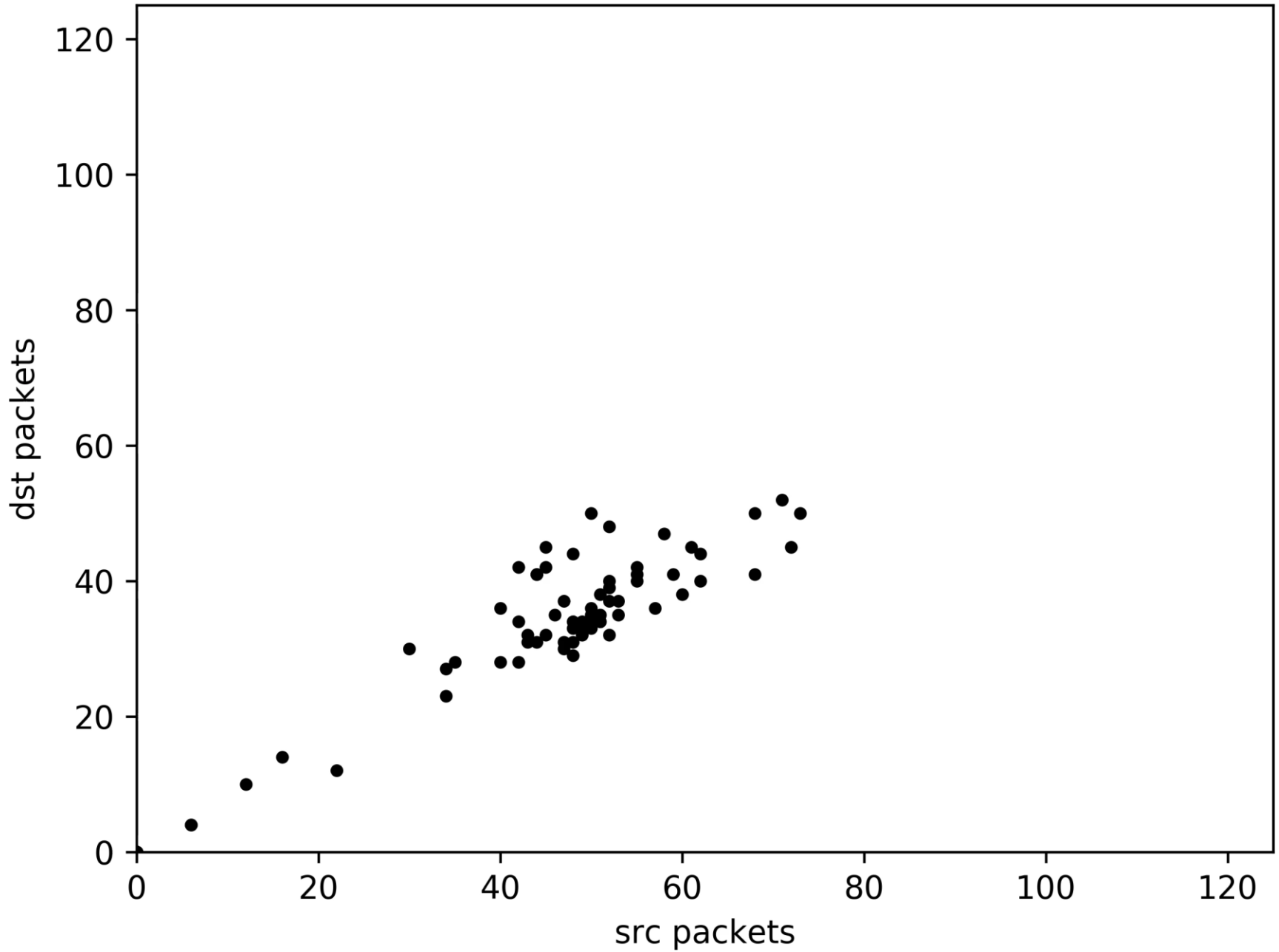
Attempt 1

- Lets graph two dimensions of our data and see how they correlate
- Specifically, lets focus on the number of bytes and packets sent
- Do they correlate?
- Is there clustering?
- Any discernable patterns?


src metrics
src packets
src bytes
0
20
40
60
80
100
120
0
2000
4000
6000
8000
10000
12000


dst metrics
dst packets
dst bytes
0
20
40
60
80
100
120
0
2000
4000
6000
8000
10000
12000


12000
10000
8000
6000
4000
2000
0
dst bytes
0
2000
4000
6000
8000
10000
12000
src bytes

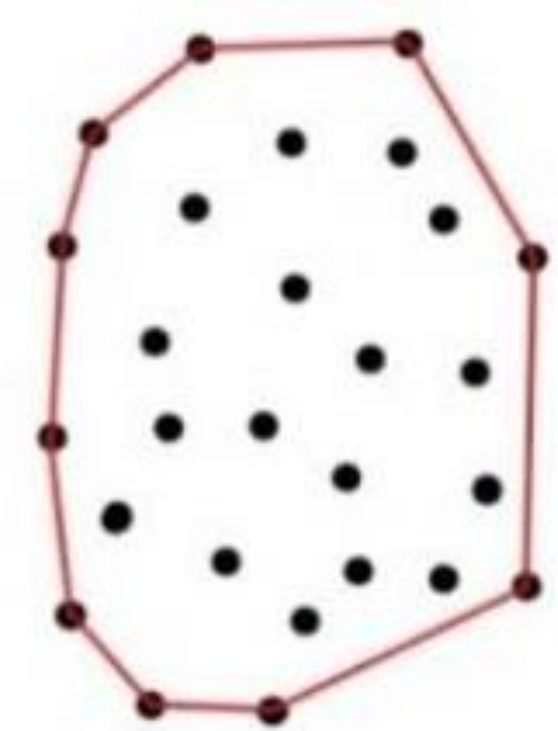

dst packets
src packets
0
20
40
60
80
100
120

Patterns

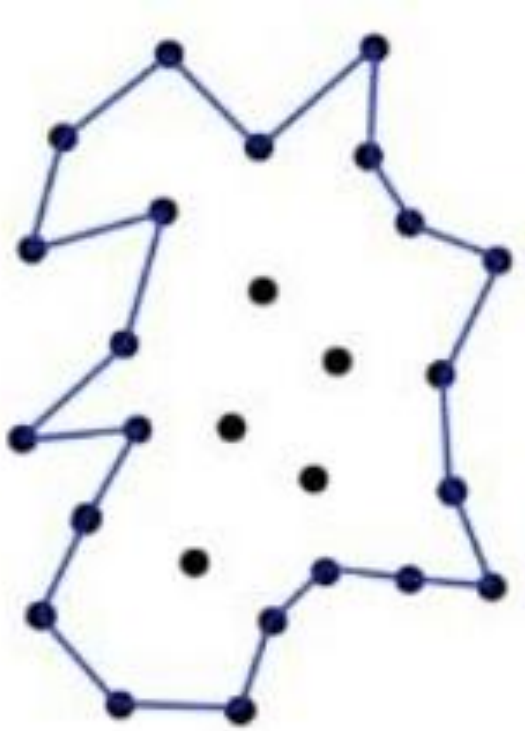
- Fairly strong clustering among most metric-pairs
- src pkts x src bytes has the strongest linear correlation
  - Though all metric pairs look to correlate linearly to some degree
- A striation pattern emerges for src with enough points
  - Also appears with src bytes x dst bytes, but with less linear correlation
  - Not so much for dst, though maybe aggregating all mining data and zooming in a bit will reveal something
- Surprisingly, points at (0, 0)

Build a Model

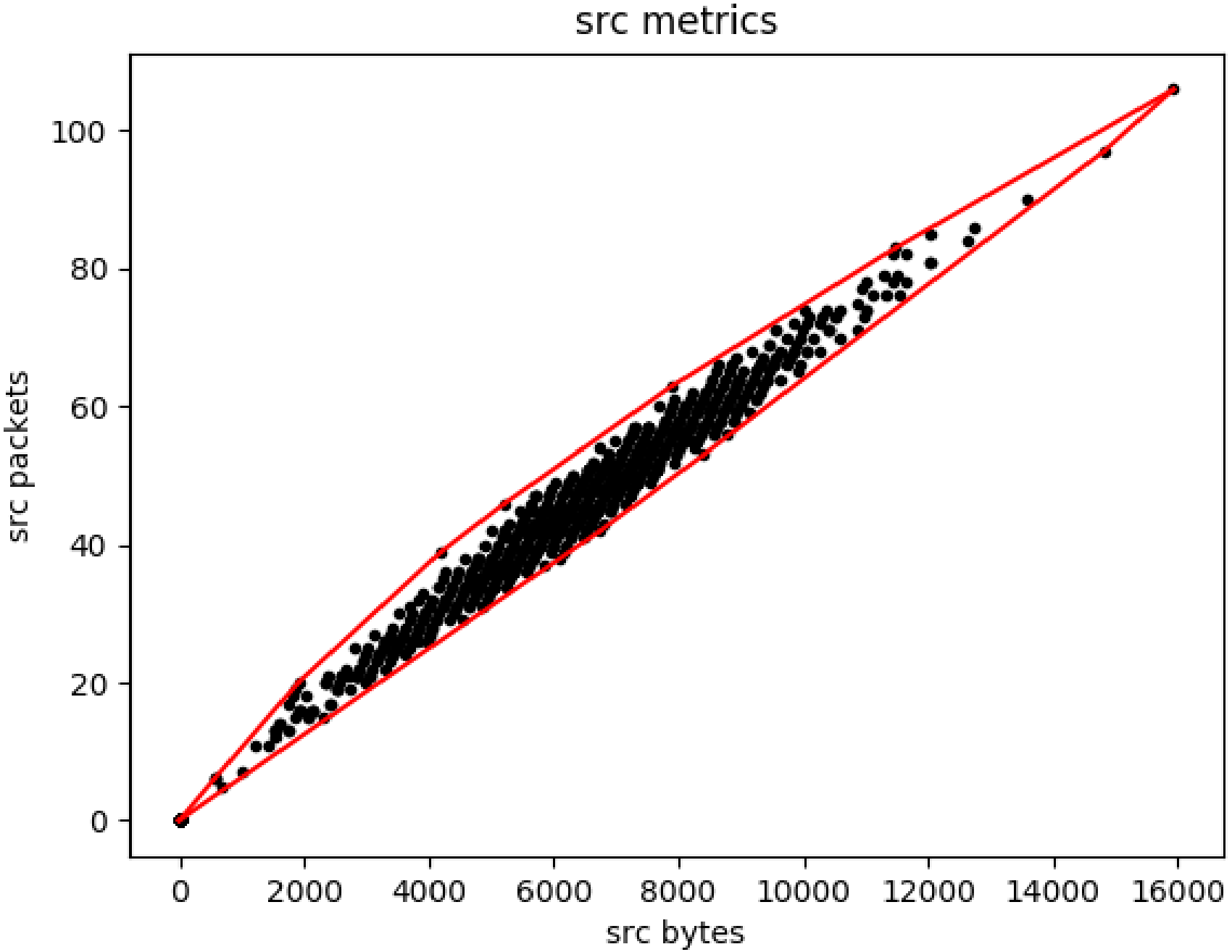
- Lets focus on the clustering
- Since we strong clustering, we can define the area that encompasses all of our samples via a convex hull
  - This methodology is conceptually similar to defining a hyperplane a la SVM, for those familiar

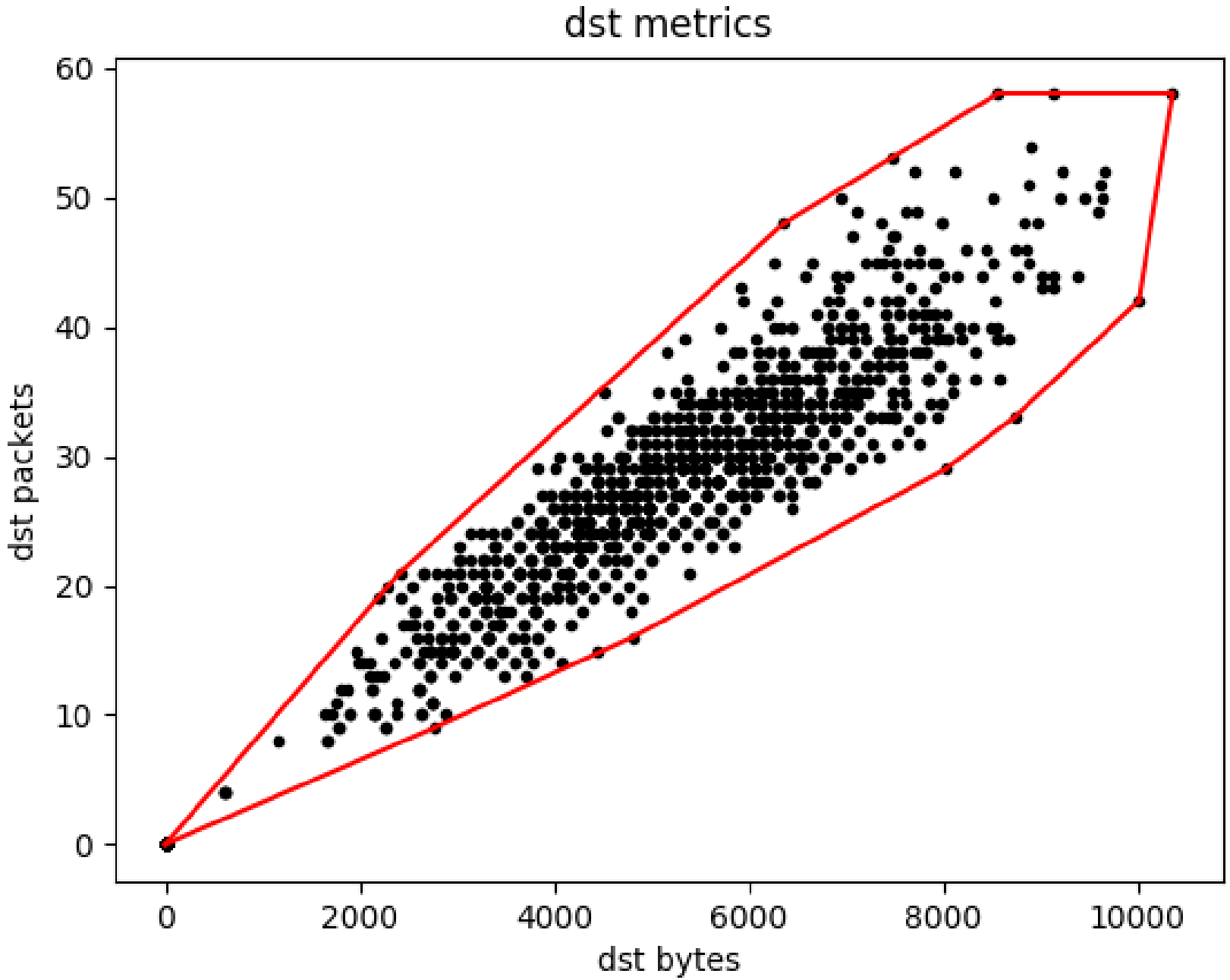


(a) Convex hull



(b) Concave hull


src metrics
src packets
src bytes
0
20
40
60
80
100
0
2000
4000
6000
8000
10000
12000
14000
16000


dst metrics
dst packets
dst bytes
0
10
20
30
40
50
60
0
2000
4000
6000
8000
10000

Let’s filter!

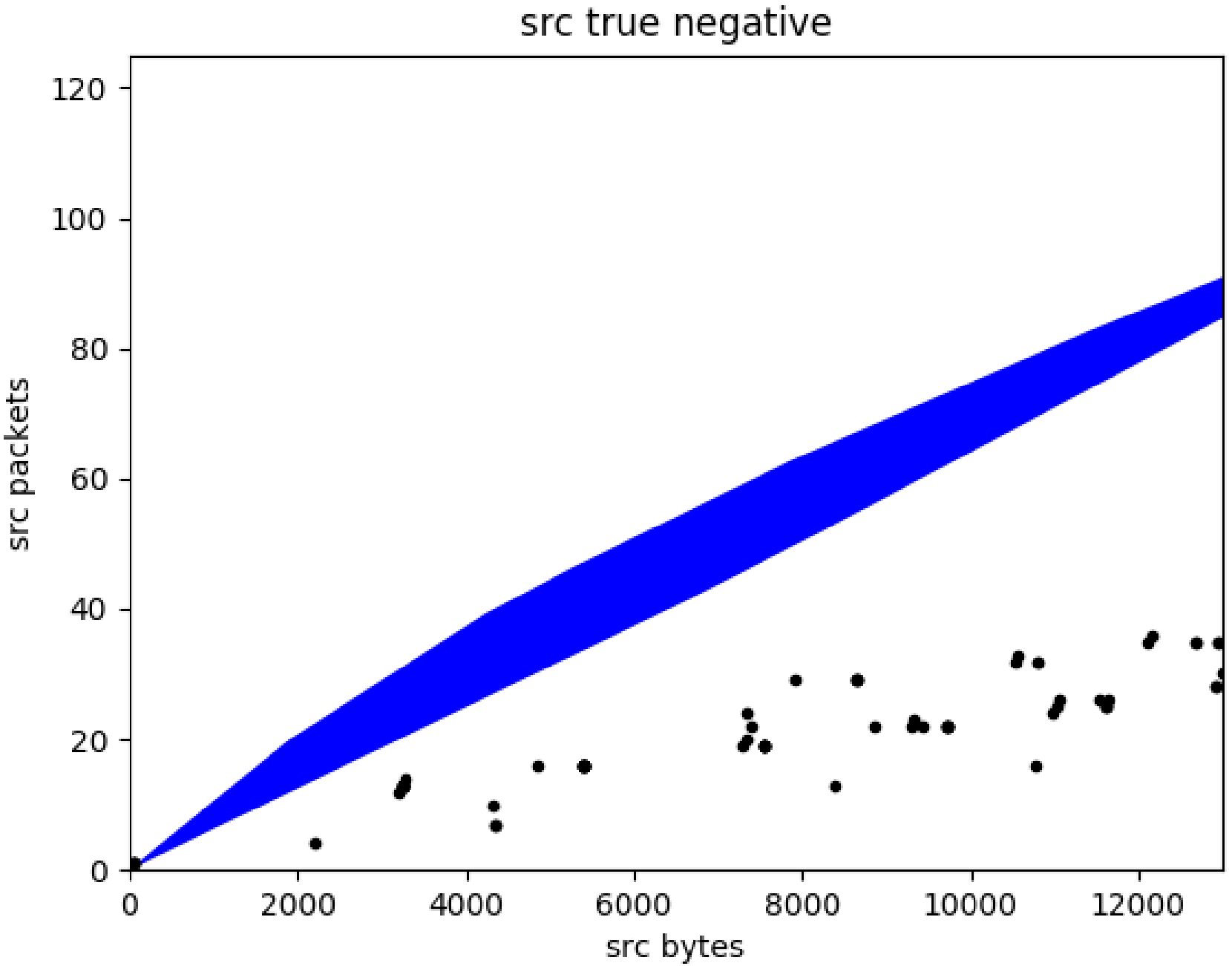
- With a defined hull, we can now pipe the samples through
- If a set of traffic stream’s membership rate within our hull is under our limit, strip it out
- For now, let’s pick an arbitrary rate of 98%

n-Dimensional Euclidean Magic

- The previous hull graphs aren't really representative of what we're *actually* doing
- Since computing the convex hull of a point cloud relies on Euclidean distances, we don't need to limit ourselves to 2-dimensional hulls (and 5 different sets of hulls, one for each metric pairing)
- Instead, let's define a 4-dimensional hull encompassing:
  - src bytes sent
  - src packets sent
  - dst bytes sent
  - dst packets sent

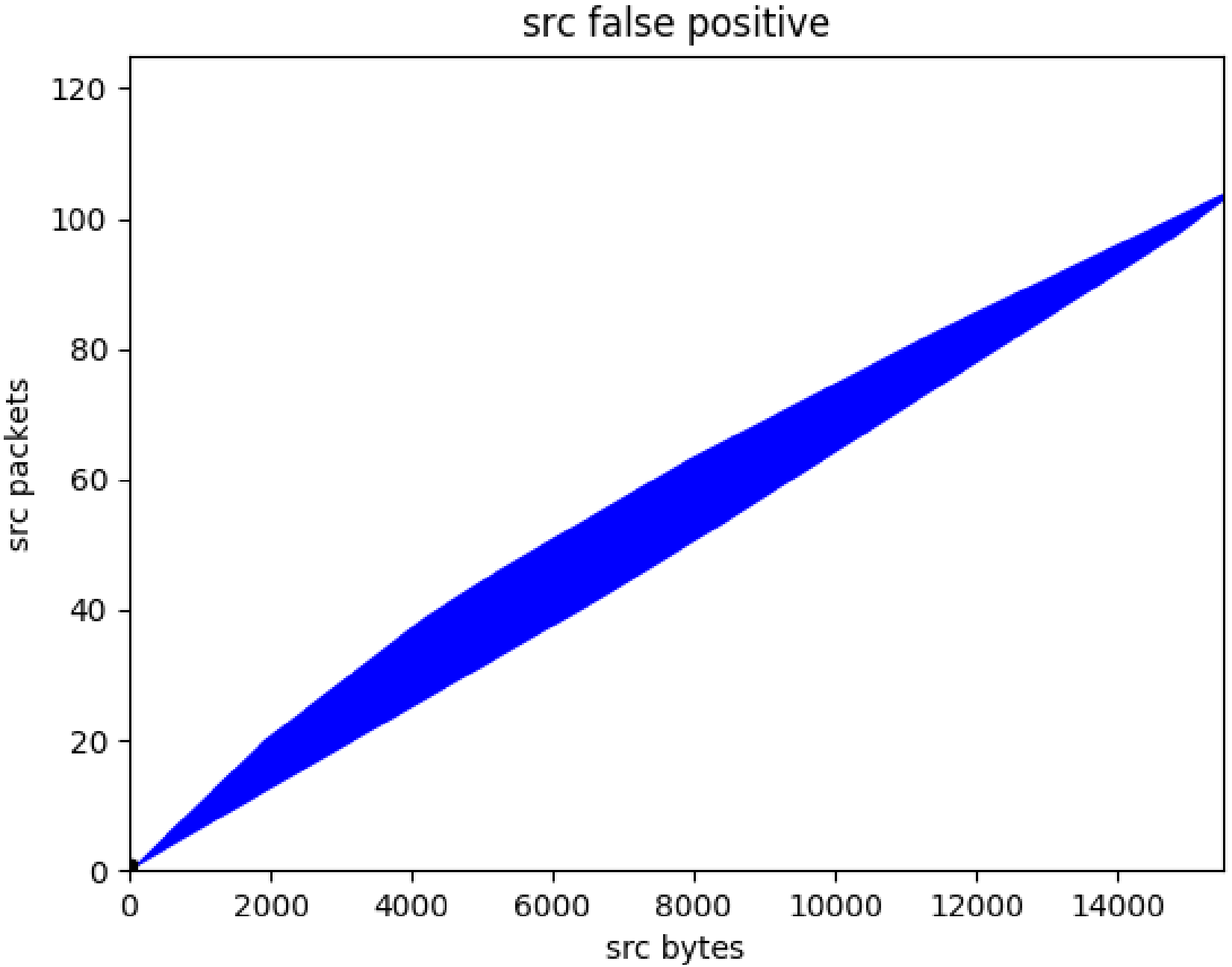
Curse of Dimensionality

- As the number of dimensions increases, volume increases at such a rate that points become sparse
- Restated: as dimensions increases linearly, amount of data needed increases exponentially
- Typically only starts to occur with hundreds of dimensions, so irrelevant to us
  - But an important note nonetheless!


src true negative
src packets
src bytes
0
20
40
60
80
100
120
0
2000
4000
6000
8000
10000
12000

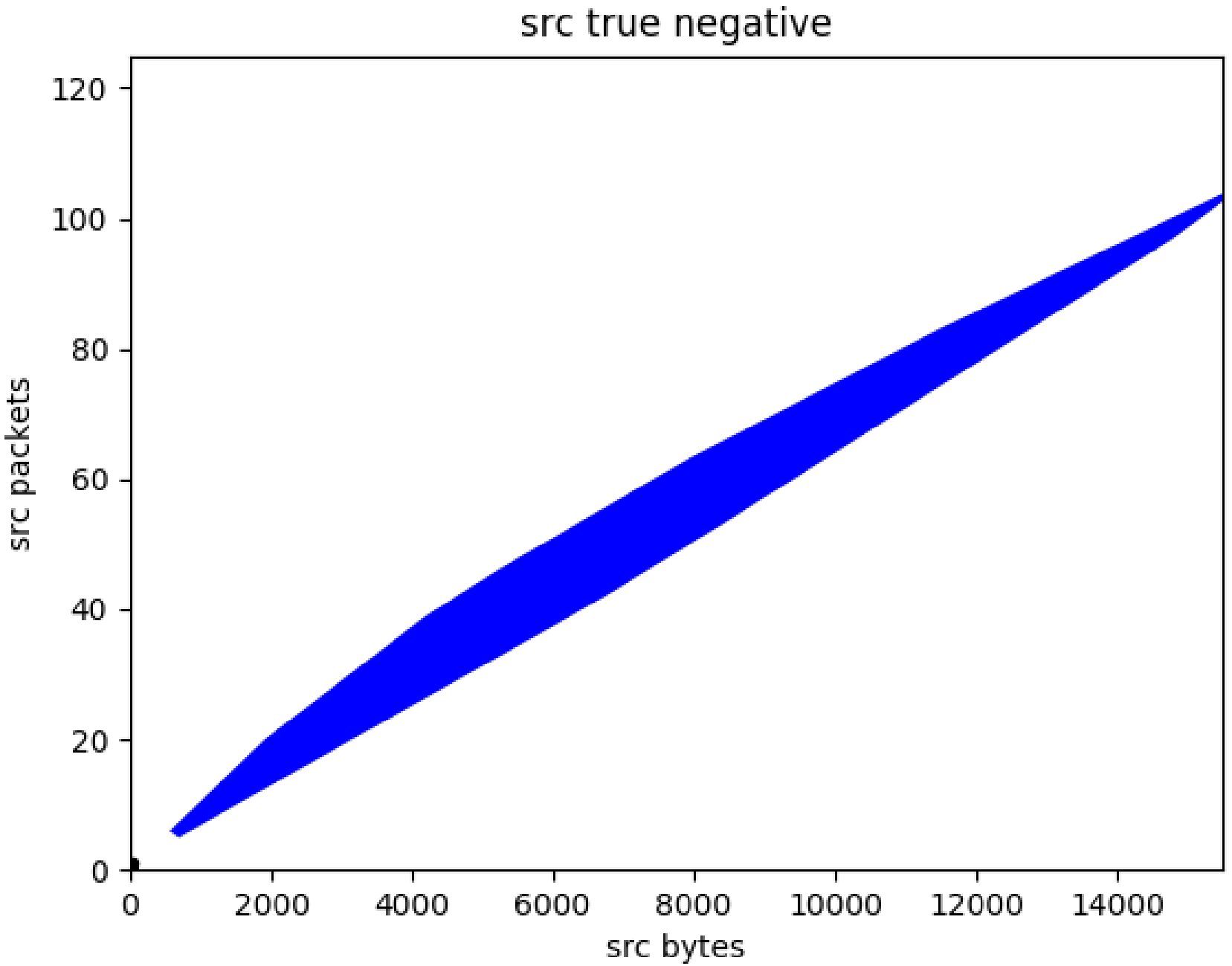
Results

- ~38k streams -> ~19k streams
- Good first attempt, but lets take a look at what got through


src false positive
120
100
80
60
40
20
0
src packets
0
2000
4000
6000
8000
10000
12000
14000
src bytes

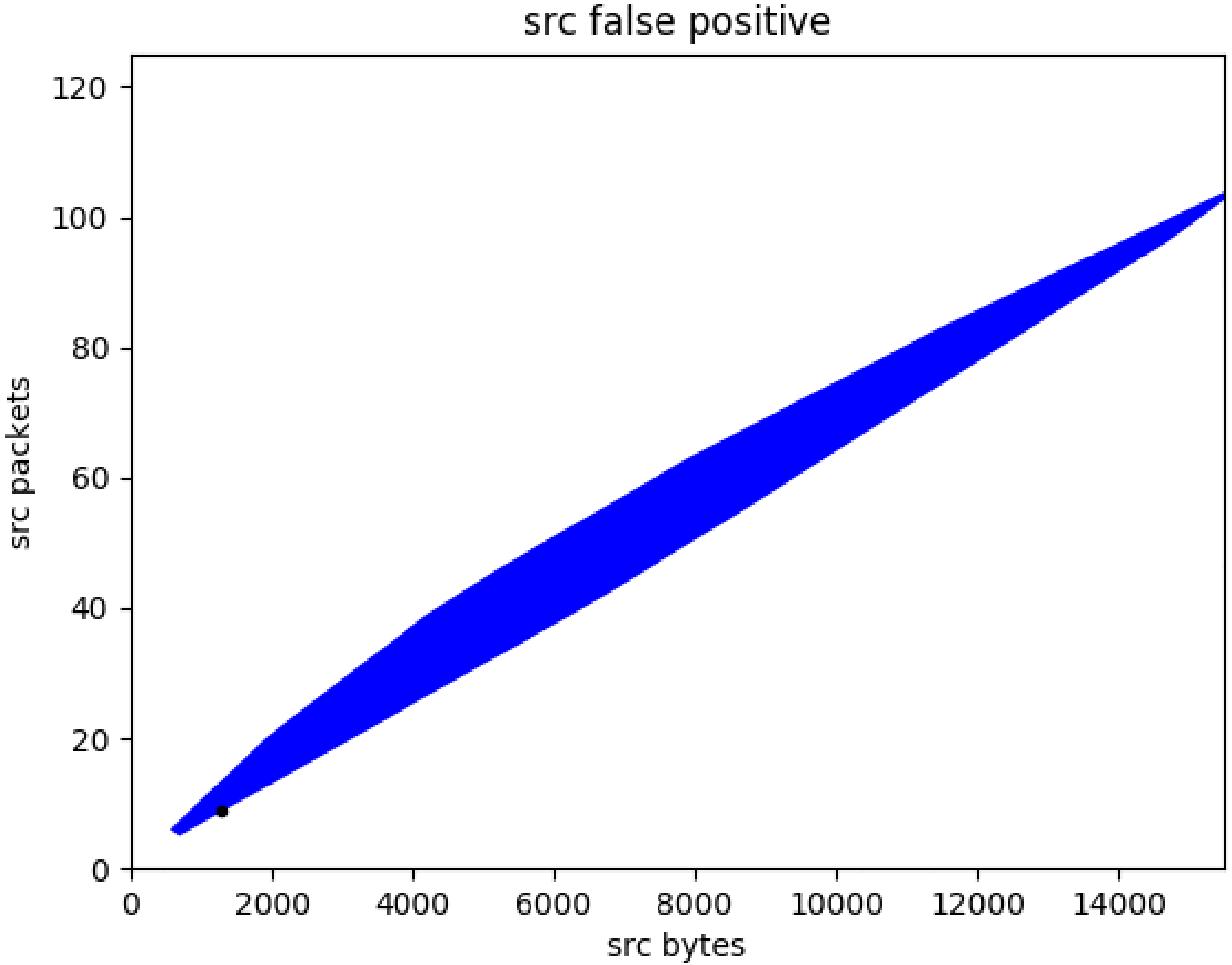
Attempt 2

- Many of the FPs looked like the previous graph
  - Lots of points at (0,0,0,0)
- Since our test data populates far more than (0,0,0,0), lets strip those out -> build a new hull -> re-filter


src true negative
120
100
80
60
40
20
0
src packets
0
2000
4000
6000
8000
10000
12000
14000
src bytes

Et voilà!

- ~19k -> 4
  - No, that’s not a typo. Four streams, not 4k.


src false positive
120
100
80
60
40
20
0
src packets
0
2000
4000
6000
8000
10000
12000
14000
src bytes

So what next?

- Lots of other options for fingerprinting cluster shape
  - Size of the convex hull interior
  - Mean/median nearest-neighbor
  - Spatial homogeneity within the convex hull
    - Ripley K and L functions (expensive and accurate)
    - Number of members within a set of concentric circles centered on the center point of the convex hull (cheap and inaccurate)
  - Quantifying the striations found in src pkts x src bytes pairing
  - Timestamp analysis (bursts of traffic vs continuous)
    - This would easily strip out our last four false positives
  - Linear regression line
    - Angle
    - Mean/median pt distance to regression line

So what next?

- Optimization
- A perfect model is useless if it takes too long to run
- Compare time-to-execute of other strategies

So what next?

- Collect more mining data
  - Pools with different Stratum configurations
    - Share difficulty
    - Reward systems (PPLNS, PPS, etc)
  - Solo mining
  - Unsuccessful/blocked miners
    - REJECT traffic – miners that can’t reach their target pool/wallet
  - Different coins
- Collect more non-mining data
  - Now taking VPC Flow Log donations….

Tooling + Learning

- Python!
- numpy + scipy for all the mathematical black magic
- matplotlib for making the pretty charts (and some 2D topological stuff)
- Algorithms of the Intelligent Web by Douglas McIlwraith et. al.
  - (second edition)

If you’re feeling generous….

- I need lots and lots of VPC Flow Logs
- Looking for people to donate
- I plan to release OSS tooling – would be happy to let those who help with this preview tooling in a sort-of closed beta

Thanks for listening!

- jonn.callahan@nvisium.com
- @jonn.callahan

- https://www.slideshare.net/JonnCallahan/owasp-m-lvscryptocoins-128298635

nVISIUM