New Distribution Scheme

- ***Raw Sockets -> Framework***
  - Much more condensed and managed
  - Transparent
  - Scalable
- ***Flow Generation:***
  - NFStream - automatically parallel
- DATA: 2,830,743 samples (total amongst all CSVs)

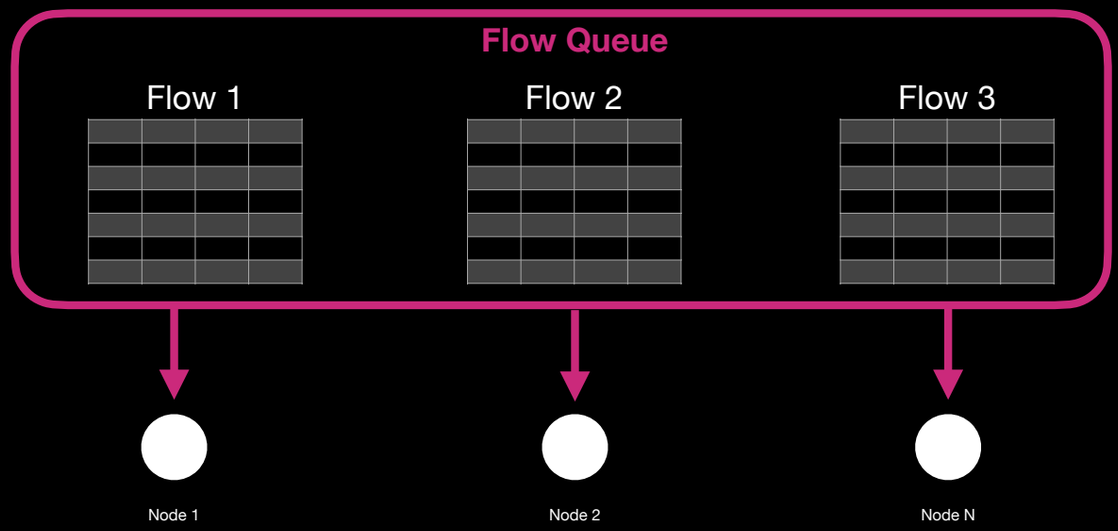
Flows are resource intensive - take time to generate. I got around this by using NFStream only to parse data from a temporary file written to by Scapy. All data written by Scapy is a very small amount of packets that NFStream can easily handle in parallel.

Data Delegation - Options

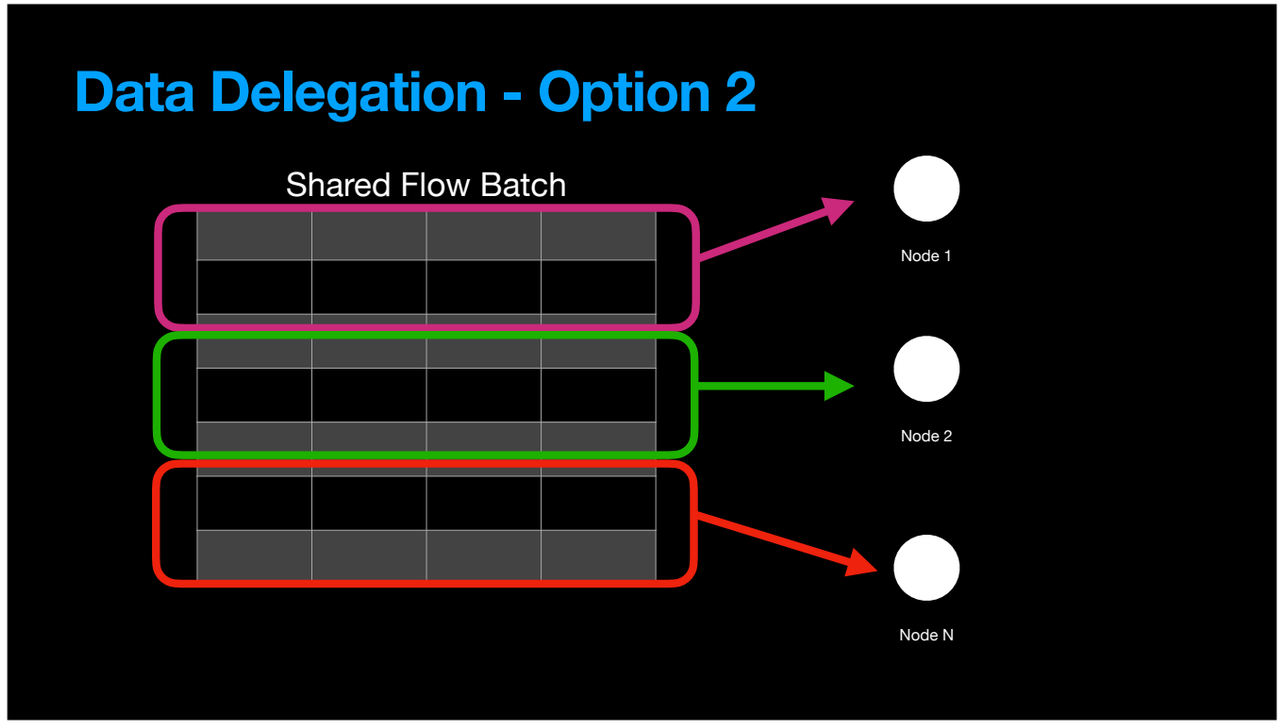
- **Option 1:**
  - Collect flows in increments and each node in the cluster gets handed a batch - our current approach.
- **Option 2:**
  - Collect flows into a larger batch, and have it distributed across the cluster with batch learning (available as an implicitly parallel operation in Ray).

For the head node on a cluster: > ray start --head —port=6379
For any nodes manually managed: > ray start —address='127.0.0.1:6379'
To run a script on the cluster (must be done on the cluster itself) either use:
ray.init(address="auto") or ray.init(address=…)
Then use the submission on the cluster as such:
ray submit <script work>
To resolve any dependencies on the cluster use:
ray rsync-up

Data Delegation - Option 1



Data Delegation - Option 2



Data Preprocessing

- CIC Dataset ~ 79 features
  - Our data is 46 **direct** features from NFStream. We reduce both to match.
  - Roughly half of the features are dropped, but we can always attempt to calculate them ourselves if need be.
- CIC Dataset is modified where all attacks are now marked “Malicious” instead of specific attack.
- In training, we change the Benign/Malicious to 0/1 respectively.

Flow Processing

- Usually this will always be a bottleneck (aggregation time of statistics for each bidirectional communication of packets).
  - With NFStream, it is designed to be fast but it is still a slow process even if asynchronous to the server on realtime traffic.
  - Solution:
    - We capture packets into a temporary file (disk overhead), and still use NFStream to read and parse it due to its efficiency - use a large packet file to demonstrate its effectiveness.
    - Alternatively, if we had enough RAM, we could segment a small amount as RamDisk for the temporary file to reduce disk overhead.

Flow Benchmark - Offline

- Large Fuzzing capture file (493 MB - 2,244,139 total packets) for benchmarking.
  - Using NFStream to parse the pcap and our code to turn into a dataframe, it takes between 6 to 9 seconds - depending on the system.

```
Starting stream read...
File "Fuzzing_pcap.pcapng" size: 470.12377548217773 MB
Time to read & convert to dataframe using NFStream: 6.700629949569702 seconds

      Destination Port  Flow Duration  Total Fwd Packets  ...  URG Flag Count  CWE Flag Count  ECE Flag Count
0               1900.0          189.0                6.0  ...             0.0             0.0             0.0
1               1900.0          207.0                6.0  ...             0.0             0.0             0.0
2               1900.0          206.0                6.0  ...             0.0             0.0             0.0
3               1900.0          202.0                6.0  ...             0.0             0.0             0.0
4              51002.0            0.0                1.0  ...             0.0             0.0             0.0
...                ...            ...                ...  ...             ...             ...             ...
1802           51002.0            0.0                1.0  ...             0.0             0.0             0.0
1803           51002.0            0.0                1.0  ...             0.0             0.0             0.0
1804           51002.0            0.0                1.0  ...             0.0             0.0             0.0
1805           51002.0            0.0                1.0  ...             0.0             0.0             0.0
1806           51002.0            0.0                1.0  ...             0.0             0.0             0.0

[1807 rows x 46 columns]
```

Flows can reach many thousands if not careful, even for relatively small files. We have to limit it if it goes beyond a certain point.

Model Architectures

- **Intermediate Complexity Binary Classifier NN:** ~93% Testing accuracy

How much flow is necessary to determine whether this traffic is benign or malicious? Remember that we are taking flows in segments, and we don't have the complete picture.

- Each flow coming in to the nodes is a temporal capture of a brief set of traffic. The inferences made are not definitive. We need some kind of confidence or evidence building on the local levels as well to send back to the master node.

Each entry in the data frame represents a *flow*, which can either indicate we have an attack taking place or not for that set of communications

All regression models are currently using SGD optimizer and CrossEntropyLoss

Learning Benchmark - Offline

- Largely accurate on a new data capture (totally benign) - SUPER important

source	likely benign flows	likely malicious flows	final judgment
10.10.0.100	18	1	likely benign node
10.10.0.139	1	1	likely benign node
10.10.0.226	1	0	likely benign node
10.10.0.33	1	0	likely benign node
fe80::10d4:b134:d4f0:2b79	1	0	likely benign node
fe80::fad0:27ff:fe8a:d27b	1	0	likely benign node
10.10.0.131	1	0	likely benign node

- Also pretty good with our first attack captures (ssh brute force traffic flows)

source	likely benign flows	likely malicious flows	final judgment
192.168.0.122	184	1805	likely malicious node
192.168.0.104	288	673	likely malicious node
192.168.0.41	0	6	likely malicious node
192.168.0.1	10	1	likely benign node
fe80::ba27:ebff:fe2e:597d	8	5	likely benign node
fe80::dea6:32ff:fe6e:363	7	6	likely benign node
fe80::dea6:32ff:fe8a:e7fe	3	3	likely benign node
fe80::e65f:1ff:fe85:5d50	3	4	likely malicious node
192.168.0.176	10	1	likely benign node
192.168.0.22	0	28	likely malicious node
192.168.0.21	1	6	likely malicious node
192.168.0.214	1	0	likely benign node
192.168.0.109	10	1	likely benign node
fe80::562a:1bff:fecc:583a	1	0	likely benign node
fe80::4aa6:b8ff:feff:a63a	1	0	likely benign node

The images here are based on a ratio-threshold (i.e. if more than half of the flows are inferenced to be malicious). Not evidence based, so this is a single collection of flows over time, with a single inference on our offline benchmarks.

From Zero (offline) Deployment Pipeline

- Step 1: Train all known nodes in a **federated** manner.
  - 1.5: Consider “best” model; e.g., accuracy, performance.
- Step 2: Begin IDS System
  - Consider data delegation strategy (collaboration).
- Step 3: Scaling
  - New nodes can join asynchronously and be served the model - already trained in step 1.
  - Now these nodes obtain their own flow batches and can predict using the model.
  - Many nodes can cause many aggregation requests (master node needs to be reasonably resource-capable).

Federated learning: Each node trains locally its own dataset, and contributes to global accuracy.
Dual data delegation strategy - either way each node predicts on it’s own “batch” of flows
- Question may not necessarily be which is better, or easiest, but rather most efficient.

Master node - responsible for a lot; training, and IDS management.
We can build our own router linux.

Ray - it is able to make the models served into a web service and also automatically select the resources to be used on each node. Need to explore its applicability on lower-end devices.

Additional thought - Distributed **model,** but how can it be made parallel?
Profile different models: Inference times, [CPU/Memory/Disk] footprints, training times (federated). ** IMPACT on real time performance.

Online machine learning (is there a possibility of concept drift - maybe with new attacks, but they are all generally the same pattern)? We can use evidence building to correlate with inference strength - confidences can be taken into account from the individual nodes **as the inferences are made and then push the result to the master when ready.**

Why is federated training a benefit? Consider full training on the complete dataset on a single node versus training all of them separately in federated mode - perhaps we can integrate it as a framework.