Network Device Fingerprinting

Siddharth Shastry

Context

We define a device fingerprint as a particular network behavior of a device that responds to a probe. In the context of this project, the probe being used is nmap. Nmap sends specialized IP packages to hosts on the network and records the manner in which they respond. The tool also has some inherent fingerprinting capabilities itself. For the dataset used in this project, labels were collected via SSH and Telnet banner grabs for routers and Shodan for other IoT devices. The dataset contains 15 classes of devices and can be found here (https://drive.google.com/file/d/1vd38hHMB77Qk7V7Q3mXy4ucvMq0KC7Pu/view). The dataset needs to be extracted and placed in the same directory as the notebook. It must be named "traffic.pcapng".

Code Snippets in this report document are not commented, however, comments exist in the notebook.

Goal

To train a machine learning model to learn the device fingerprints for each of the 15 classes in the provided dataset. The benchmark presented is ~94% so an achieved accuracy around that number would be seen as a success.

Pipeline

Data Processing:

The first step in the pipeline was to extract the dataset from the archive. Once this was done, I was left with a file "traffic.pcapng". Each entry in this file is a specific sample interaction between a network device and the nmap probe. The entry contains a metadata section which labels the device as well as several embedded packets which define the full sequence of the interaction. In the pipeline, we first iterate through this larger file that contains the samples, and save each individual interaction (associated with a particular device and containing several embedded packets) as its own pcap file.

```python
with open(LABELS_CSV, "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Item", "Label"])

    for sample in tqdm(pcapml_fe.sampler(PCAPML_PATH), desc="Samples"):
```

```
        sid = str(sample.sid)
        label = extract_label(sample)
        pcap_name = f"{sid}.pcap"
        pcap_path = os.path.join(PCAP_DIR, pcap_name)

        pkts = sample.packets

        # Find earliest timestamp in this sample to normalize
        ts0 = float("inf")
        for p in pkts:
            ts0 = min(ts0, p.ts)

        with open(pcap_path, "wb") as pcap_f:
            # write pcap file per sample
            w = dpkt.pcap.Writer(pcap_f)
            for pkt in pkts:
                # dpkt requires positive timestamp so we normalize with ts0
                norm_ts = max(0,pkt.ts - ts0)
                w.writepkt(pkt.raw_bytes, ts=norm_ts)

        writer.writerow([pcap_name, label])
```

There are a couple more complexities to this process. First, we ensure that the timestamps are positive because the package used to write the pcaps (dpkt) accepts only positive values. We achieve this by normalizing relative to the earliest packet in the sample. Apart from writing the actual pcap, we also write to the labels file with the device we extracted from the metadata associated with this sample.

Constructing the features with nprint:
I tried manually extracting features for this task but got very low accuracy. Instead, I chose to follow the approach shown in the benchmark and used nprint. I used nprint version 1.2.1. The command I used to generate the nprint csv is.

```
Shell
nprint -4 -t -c 7 -P pcaps/{id}.pcap
```

-4: Include IPv4
-t: Include TCP

-c 7: Truncate at 7 packets per sample; I chose a limit here because the length of each sample varied between 9-14 packets and for training purposes this needed to be fixed.
-P pcaps/{id}.pcap: The sample to target

Given this command, I cycle through all the saved sample pcaps and create the nprint representations. This is accelerated via multithreading.

```python
with ThreadPoolExecutor(max_workers=8) as ex:
        for vec, label_str in tqdm(ex.map(_worker, jobs), total=len(jobs)):
            if vec is None:
                continue

            if label_str not in labels_seen:
                labels_seen[label_str] = len(labels_seen)

            X_list.append(vec)
            y_list.append(labels_seen[label_str])

    X = np.stack(X_list)
    y = np.array(y_list, dtype=np.int64)
```

The nprint representations for all 7 packets in each sample are flattened into one array. I then stack each sample's corresponding packet representation into one large matrix, which is our dataset for the ML models. Similarly I create a second array of labels per each sample.

```python
np.save(OUT_X, X)
np.save(OUT_Y, y)
```

These are saved to disk so that the time consuming data processing steps don’t have to be run repeatedly.

Training/Testing the Models:
There are two classes of models that I trained in this project. The first is a random forest classifier. I chose the number of estimators to be 500 to limit overfitting.

```python
clf = RandomForestClassifier(
```

```
    n_estimators=500,
    max_depth=None,
    class_weight="balanced",
    n_jobs=-1,
    random_state=42,
)
```

The dataset is loaded into a numpy array and a train/test split is created (80/20).

```Python
X = np.load("X_nprint.npy")
y = np.load("y_nprint.npy")

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

The model is trained on the training set and evaluated on the test set

```Python
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
acc = balanced_accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
print(classification_report(y_test, y_pred))
```

Results

```Shell
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       430
           1       0.80      0.75      0.78       282
           2       0.85      0.94      0.89       481
           3       0.85      0.95      0.90       531
           4       0.88      0.81      0.84       276
           5       0.94      0.92      0.93       285
           6       0.81      0.67      0.73       272
```

```
           7       0.85      0.87      0.86       290
           8       0.96      0.92      0.94       289
           9       0.97      0.94      0.95       290
          10       0.96      0.94      0.95       285
          11       0.98      0.99      0.98       290
          12       0.98      0.98      0.98       290
          13       0.98      0.93      0.95       295
          14       0.99      0.98      0.99       574

    accuracy                           0.90      5160
   macro avg       0.91      0.89      0.90      5160
weighted avg       0.90      0.90      0.90      5160
```

Next, a MLP was trained with the same dataset. It was defined as such:

Python
```python
clf = Pipeline([
    ("scaler", StandardScaler(with_mean=False)),
    ("mlp", MLPClassifier(
        hidden_layer_sizes=(256,128),
        activation="relu",
        solver="adam",
        max_iter=30,
        random_state=42,
        early_stopping=True,
        verbose=True,
        batch_size=256,
    )),
])
```

Results

Shell
```
    accuracy                           0.88      5160
   macro avg       0.88      0.87      0.87      5160
weighted avg       0.88      0.88      0.88      5160
```

Conclusion

Utilizing nprint to create a standardized feature map of the packet sequences for each sample was effective in learning device fingerprints over a class size of 15. Although the benchmark value of ~94% accuracy was not hit, the models presented in this report approached that value closely at a peak accuracy with the random forest classifier at 90%.