Interactive traffic classification using semi-supervised graphical approaches

Satyandra Guthula, Vihaan Akshaay

Motivation:

QoS is a metric that is particularly relevant in the case of interactive content, especially for network providers, datacenter administrators and users in general. Hence aggregating existing metrics after identifying whether sessions are interactive could help in identifying the QoS for the users. Although content within the packets is encrypted, the pattern of packets and flags used in the packets can be used to identify whether a service is user-driven interactive session or some set of scripts running in background. Most interactive content such as video streaming, messaging applications and social media applications send data in the form of bursts. So, using the characteristics of the burst would be useful in identifying the type of sessions. We define a burst as a list of packets transmitted in one particular direction within 10 milliseconds. This is because most of the sessions are generated from applications that transmit packets within microseconds, hence accounting for delays in network 10 milliseconds is a sufficient gap between packets to determine whether a burst has ended.

Although there are plenty of methods to generate the labels for network traffic sessions, it would lead to overfitting the data on specific types of network sessions. Hence we are using semi-supervised methods for ML. Most existing works that leverage the unlabelled data use existing models from the NLP community. These models typically use long sequences of network packets and do not embed any useful information related to burst structure, leading to huge models (BERT based model took 168 MB). Hence we want to come up with a model that is smaller in size and requires lower training time using the graphical approaches.

Formally the problem statement is as follows.

- Identify which sessions are user generated interactive sessions given the packet representation and edges between packets.
- Perform multi-class classification to predict the type of service being used by the user.

We have used GCN and DGCNN for extracting the hidden representations using the packet graph. We have used the pseudo-relabelling technique to leverage the unlabelled data.

Very limited work has been done in the space of service classification using graphical approaches. Existing works such as ‘Accurate Decentralized Application Identification via Encrypted Traffic Analysis Using Graph Neural Networks’ and ‘Unveiling the potential of Graph Neural Networks for robust Intrusion Detection’ have achieved state of the art performance by capturing the relationships between packets. Hence we are motivated in trying similar approaches for our problem.

Datasets:

We have anonymized passive packet traces from the UCSB campus border routers. Although there are other datasets with packet traces. The traces were taken almost a decade ago. The changes in the protocol structure and the network infrastructure would not be reflected in the dataset. The paper “AI/ML for Network Security: The Emperor has no Clothes” talks about the shortcomings of various datasets and how the ML models use spurious correlations in the data. Hence there is a need for datasets which are not prone to shortcuts, and generalizable for different use cases. Hence we have used the dataset curated from UCSB campus routers.

The dataset consists of 50,000 sessions which are labelled and 15,000 sessions are unlabelled. The data is raw packet traces which are fed to a tool called Nprint, which extracts packet level features. Out of these features we have chosen 63 features that are relevant and can be generalizable to different protocols.

For the graph structure followed the following algorithm

- Every subsequent packet in a burst is connected bidirectionally
- The first packet of the current burst is connected with the first packet of the next burst unidirectionally
- The last packet of the current burst is connected to the last packet of the next burst unidirectionally
- Nodes have 63 features derived from the Nprint representation.
- The traffic is labelled using Server name Indication values present within TCP headers.

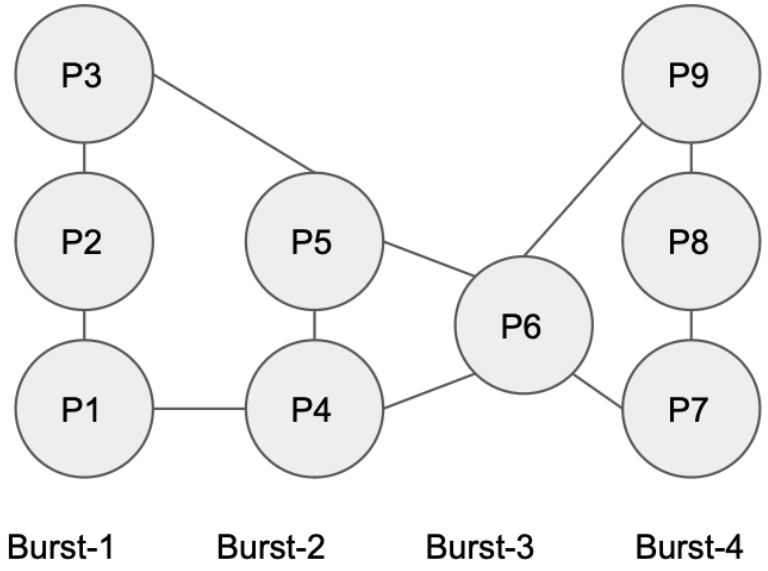

Fig: Example session graph structure

Methods:

Now that we have chosen this particular graph structure for building graphs from the available data, we wish to evaluate the accuracy of the classification task by adopting typical graph classification algorithms. With this motivation, we adopted the typical GCN model as well as Deep Graph CNN module.

In addition, to match the performance of the previous model (BERT) for a similar classification task, we have adopted semi-supervised methods to improve model performance. We've adopted the psuedolabelling strategy. The whole proposed architecture/procedure is illustrated in this picture below.

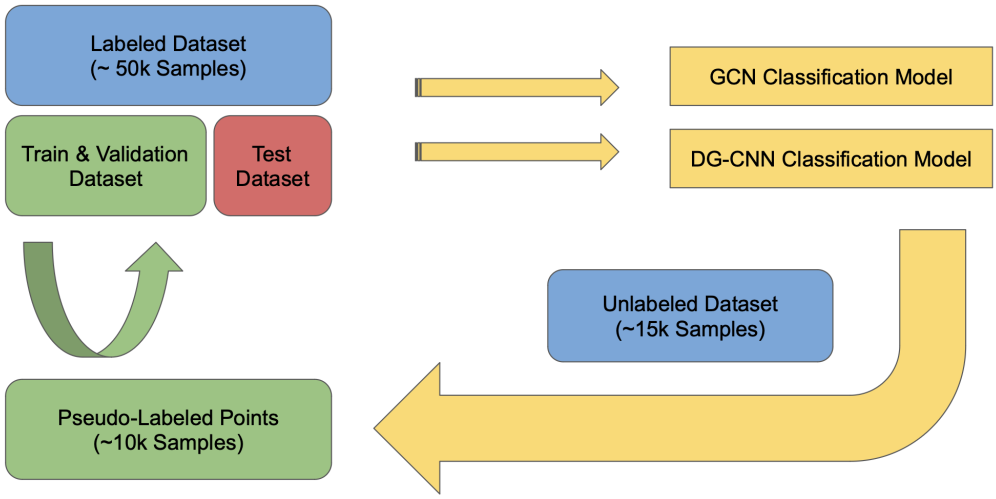

Fig: Procedure Proposal

[Training Phase:] As it can be seen in the illustration above, we first begin with the 'Labeled Dataset'. We have around 50k labeled graphs and 15k unlabeled graphs. We split this 'Labeled Dataset' into three splits. 'Train','Validation' & 'Test' Set. We isolate the 'Test' dataset to only evaluate the models that we train. Now that we have a split, we first train two models: Model A - GCN and Model B - DGCNN using train and validation dataset.

[Psuedo-labeling Phase:] Now that we have the Model A and Model B trained, we obtain unlabeled data and make predictions using these trained models. We assign these predictions as labels for these graphs. We put a threshold on the predictions, that way we only pick graphs that the model is confident in applying labels.

[Re-training Phase:] We use these new data points to retrain Model A and Model B again. We ensure that the model does not get biased on training itself on it's own predictions, we keep the threshold quite high. We reuse these prediction points to retrain the models. To finally evaluate the models, we check the performance of these new retrained models on the Initial Split Test Dataset.

The two models that we were using for the classification are GCN and Deep graph CNNs.

1) Supervised classification with GCN:
   The architecture of the GCN module used is depicted below. We use 64 and 64 GCN layers on the graphs. To obtain a representation, the module has a mean pooling layer. Now we classify this abstraction using a typical feed forward neural network by using fully connected layers to predict the output class.

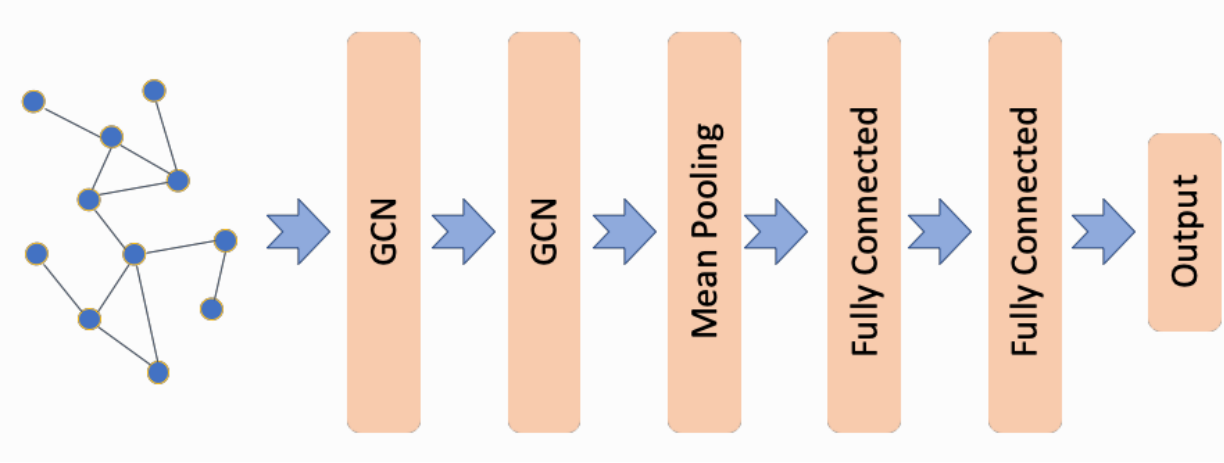

Fig: GCN Architecture

2) Deep graph CNN:
This module uses a few graph convolutional layers to begin with. Now from each of these convlutions, WL colors are obtained and stacked. Now these nodes are prioritized based on the WL colors of the latest convolution. Later we take only a few of those nodes (top k nodes) and form an image (like in the illustration below). Once we obtain this image like representation for the graph, we classify it further by a traditional CNN, given its superior performance.

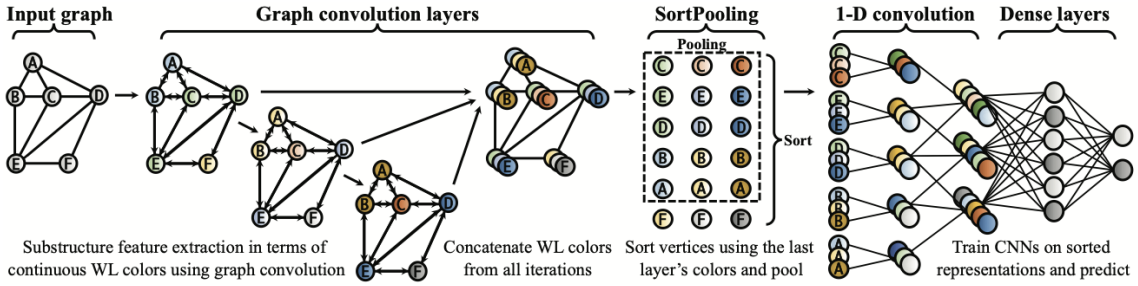

Fig: DGCNN Model Architecture

Results:

We evaluate these graph classification tasks based on accuracy metric (Since we have accuracies from previous experiments from other relevant methods applied for similar classification problems but not as graphs). We generated the datasets in two forms - binary classification task and multi-class classification task. In addition, we run these experiments on batches of 30 on 10-fold cross-validation with 5 Repeats and on 100 epochs with 90% Train and 10% Validation.

Model	Accuracy
DGCNN	92.6%
GCN	95.6%
GCN + Pseudo-labeling	96.2%
DGCNN + Pseudo-labeling	94.3%
DGCNN multi-class	90.3%
GCN multi-class	95.2%
GCN + Pseudo-labeling multi-class	95.8%
DGCNN + Pseudo-labeling multi-class	97%

From the table above, we can make the following observations.

1) While comparing Binary vs Multi-class classifications, we can see that the accuracy if good for binary classification and it slightly drops while moving from the binary setting to a multiclass setting.
2) While comparing the classification models used here, we notice that GCN can perform better than DGCNN in most of the different experiment settings here. It is possible that the current hyper parameter setting for DGCNN might be pulling the performance back as we did not perform much of explicit hyper parameter tuning.
3) One success of our experiments was how Psuedolabeling helped the model improve performance in comparison with the counterpart without psuedolabeling and retraining.

One consequence of this retraining is that we were able to match the performance of our previous benchmark on this dataset (without rephrasing as a graph classification problem). While we compare our performance using the dataset as a graph and using graph ml methods, we were able to attain good performance but with way lesser parameters. This shows the power of representation of data and how simple graph machine learning methods have enabled ideas to use simple data modelled as graph to attain top-notch performance.

Model	Memory	Multi-class Accuracy
Graph GCN	**351KB**	95.2%
DGCNN	**403KB**	97%
ET-BERT	168MB	**97.7%**

Conclusions:

1. Using the graphical approaches, we can exploit the dependencies between bursts and provide near State-of-Art performance with much simple models (0.2% size of the BERT based model).
2. Using pseudo-labelling approach has improved accuracy of our model in all variants and problem formulations.

Future work:

1. Hyper parameter tuning needs more analysis from both the computer networks theory and theory.
2. More comparative studies of Classification Models in this pipeline.
3. Explore more representations of networks as graphs like exploiting the network structure as a graph instead of modelling the sessions.
4. Bottlenecks exist due to the large size of graphs. Some form of hierarchical learning (subgraph representations instead of nodes) may reduce complexity.