A Hybrid Method for TCP Connection Attack Prediction

Guoyuan An
School of Computing
KAIST
Daejeon, South Korea
anguoyuan@kaist.ac.kr

Fanchen Bu
School of EE
KAIST
Daejeon, South Korea
boqvezen97@kaist.ac.kr

ABSTRACT

What is the omen of a TCP attack? Could we detect potential TCP connection attacks based on the past connection history? In this project, we give an extensive analysis about the occurrence patterns of TCP attacks and propose a novel and effective model to predict the attack types included in a sequence of TCP connections. Experiment result on the valid set shows that our model achieves a amazing weighted F1 score, 0.9663!

1. INTRODUCTION

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite, which, together with the Internet Protocol (IP), plays an extremely significant role in the modern Internet system. However, just because of its importance, TCP connections become a main target of many Internet attackers. As a time-honored protocol, TCP was designed when cyber-security was not paid much attention to, making the design of it fairly vulnerable in the Internet world nowadays. On the other hand, cyber-security is more and more important as immense data including many of our sensitive information are stored on the Internet. Therefore, there have been a lot of researchers working on both finding ways to attack through TCP connections and protecting devices from TCP connection attacks.

In this project, we are given the following three datasets:

- **Training set**: $N_T = 854$ files, each of which contains a list of connections in temporal order, where each connection gives us the following information:
  - Source id (S): id of the source IP address
  - Destination id (D): id of the destination IP address
  - Port (P): port number of the destination
  - Timestamp (T): timestamp indicating the time elapsed from a particular date in seconds
  - Type of connection (C): there are 26 types of TCP connections, which consist of one benign type and 25 attack types.
- **Validation set**: $N_V = 284$ pairs of files, where each pair consists of a query file and an answer file. In each query file, the same information is given as in the training set, except that type of connection is unknown; while in each answer file, there is the list of attack types that the corresponding query file has, or there is nothing if the corresponding query file contains no attacks. Note that we cannot exactly know which connections contribute to the attack types in the answer files.
- **Test set**: $N_E = 284$ query files, where each one has the same form as a query file in the validation set. Answer files are not given.

Our target is to learn from the training set, and figure out a method that is able to predict the attack types contained in a given query file as accurately as possible.

We firstly made an extensive statistical analysis about the appearance pattern of each attack types, based on which we design our prediction method. Our method consists of Stat-Filter and SE-GNN. Stat-Filter makes the inference prediction based on the statistical observations, and SE-GNN further considers the graph properties of the large TCP connection network. Our method achieve a weighted F1 score 0.9963 on valid set!

2. PROPOSED METHOD

Our proposed method consists of Stat-Filter, a statistics-based filter, and SE-GNN, a sequence-based GNN. We first observe and conclude some simple but effective rules in Stat-Filter based on our extensive statistical analysis, which can handle the cases where the ids (source id and destination id) in the connection can be found in the training set. Our experiment results on the validation set using Stat-Filter are listed in Section 3. We also design a GNN model to learn the complex patterns and to tackle the cases where the Stat-Filter is incompetent.

2.1 Stat-Filter

Stat-Filter is a statistics-based filter that can predict the type of a connection if the source id and the destination id of it have been seen before in the training set. Besides, it can also work in some other simple cases.

2.1.1 Statistical Observations

First, we made the following definitions for simplicity.

DEFINITION 2.1. *Each connection in TS is a tuple $D = (s, d, p, t, c)$, where $s, d \in F_{id}$, $p \in F_p$, $t \in \mathbb{N}$, and $c \in F_c$, while each connection in VS or ES is a tuple $\hat{D} = (s, d, p, t)$, where the connection types are not given. The training SDP set T_{SDP} is defined as*

$$\{(s, d, p) : \exists D = (s', d', p', t, c) \in TS \text{ s.t. } (s', d', p') = (s, d, p)\}.$$

Similarly, $T_{SD}, T_{SP}, T_{DP}, T_S, T_D, T_P$ are defined and the counterparts for the validation set and test set are also defined

Notations	Descriptions
TS	The training set
VS	The validation set
ES	The test set
F_{id}	The full set of ids (source ids and destination ids)
F_p	The full set of ports
F_c	The full set of connection types
$T.$	The tuple sets contained in the training set
$V.$	The tuple sets contained in the validation set
$E.$	The tuple sets contained in the test set
$A.$	The type tables for the training set
$\tilde{A}_{SDP}$	The possible answer table for the validation set
$\mathbb{T}(\cdot)$	The time list for an SDP tuple

Table 1: The main notations used in this report

with the names where T for training is replaced by V for validation or E for test, respectively.

For the training set, the type tables where we can find the types that each tuple (or element) corresponds to are defined as follows:

DEFINITION 2.2. *The type table for SDP tuples in T_{SDP} is a function $A_{SDP} : T_{SDP} \to 2^{F_c}$. Given a SDP tuple $x = (s, d, p)$, $A_{SDP}(x)$ is*

$$\{c \in F_c : \exists D = (s, d, p, t, c') \in TS \text{ s.t. } c' = c\}.$$

Similarly, we define $A_{SD}, A_{SP}, A_{DP}, A_S, A_D, A_P$. For simplicity, when the input is not in the domain, we put $\emptyset$ as the default output.

With an assumption to be explained later, we define the table of possible answers for each SDP tuple in V_{SDP} as follows:

DEFINITION 2.3. *The possible answer table for SDP tuples in V_{SDP} is a function $\tilde{A}_{SDP} : V_{SDP} \to 2^{F_c}$. Given a validation query file q, we let its answer $\tilde{A}(q)$ to be the set of type(s) that it contains. For each $x = (s, d, p)$, we say $x \in q$ if it appears in q, i.e., there exists some connection in q whose SDP tuple is exactly x, and we define $\tilde{A}_{SDP}(x)$, the set of its possible type(s) to be*

$$\bigcap_{q \ni x} \tilde{A}(q).$$

By taking all the connections with the same SDP tuple as a single action, we define the time lists of them as follows.

DEFINITION 2.4. *Let $X = T_{SDP} \cup V_{SDP} \cup E_{SDP}$ denote the SDP tuple set for the whole dataset. Given an SDP tuple $x \in X$, the time list of it, $\mathbb{T}(x)$, is the ordered list (duplication is allowed) containing the timestamps of the connections in the whole dataset with x as their SDP tuple.*

With the definitions above and the analysis of the datasets, we have the following observations:

- $|T_{SDP}| = 309073$, where 308996 of them have the corresponding $|A_{SDP}| = 1$, and the remaining 77 tuples have $|A_{SDP}| = 2$.
- $|T_{SD}| = 54944$, where 54422 of them have $|A_{SD}| = 1$, 448 have $|A_{SD}| = 2$, 56 have $|A_{SD}| = 3$, 16 have $|A_{SD}| = 4$, and the remaining 2 have $|A_{SD}| = 6$.
- $|T_{DP}| = 251870$, where 247259 of them have $|A_{DP}| = 1$, 4587 have $|A_{DP}| = 2$, 23 have $|A_{DP}| = 3$, and the remaining 1 has $|A_{DP}| = 4$.
- $|T_{SP}| = 188994$, where 185023 of them have $|A_{SP}| = 1$, and the remaining 3971 have $|A_{SP}| = 2$.
- $|T_S| = 9478$, where 9198 of them have $|A_S| = 1$, 214 have $|A_S| = 2$, 20 have $|A_S| = 3$, 21 have $|A_S| = 4$, 11 have $|A_S| = 5$, 8 have $|A_S| = 6$, 5 $|A_S| = 7$, and the remaining 1 has $|A_S| = 9$.
- $|T_D| = 21348$, where 20982 of them have $|A_D| = 1$, 285 have $|A_D| = 2$, 33 have $|A_D| = 3$, 32 have $|A_D| = 4$, 9 have $|A_D| = 5$, 2 have $|A_D| = 6$, 3 have $|A_D| = 7$, and the remaining 2 have $|A_D|$ as 13 and 18, respectively.
- $|T_P| = 24675$, where 16530 of them have $|A_P| = 1$, 6386 have $|A_P| = 2$, 1257 have $|A_P| = 3$, 378 have $|A_P| = 4$, 105 have $|A_P| = 5$, 18 have $|A_P| = 6$, and the remaining 1 has $|A_P| = 7$.
- The training set, validation set, and test set have some overlap on the tuple sets, the details of which are shown in Table 2. In the table, the first three columns that just indicate the sizes of tuple sets for three parts of the dataset are easy to understand. For the last 2 columns, T∩V means the intersection of training set and validation set and T∩E means the intersection of training set and test set, and the numbers in the parentheses denote the number of SDP tuples in the validation set or test set corresponding to the non-SDP tuples in the intersection. To state it more clear, for example, 9190(78141) in the column T∩V and the row sd means that $|TV_{SD}| = |T_{SD} \cap V_{SD}| = 9190$ and there are 78141 SDP tuples in the validation set such that the corresponding SD tuple contained by each of them is in TV_{SD}, in other words, we can somehow analyze these 78141 SDP tuples with the information of these 9190 SD tuples that can be found in the training set.
- We call these tuples (we take single elements as tuples consisting of one element) whose output of type table has size 1 **unique-type tuples**. These tuples are important because it is possible for us to predict the connections containing them have the unique types they correspond to, which is simple and fast. However, different types of unique-type tuples (SDP, SD, SP, etc.) can have quite different levels of reliability, in the sense that how sure we can be that it has the same type of connection as before when we meet a connection containing a unique-type of a specific type again. Intuitively, unique-type tuples that have less elements are more reliable. Note again that we cannot exactly know the type of connection of the connections in the validation set but can just know the type(s) that each query file contains, which is also the reason why we define the possible answer tables. Therefore, we just check for each SDP tuple in the validation set that contains a unique-type tuple, whether or not the unique type is in the possible answer table's output of it. In detail, if we say a specific type (of unique-type tuples) has the reliability score as m/n if there are totally n SDP tuples in the validation set that contain a unique-type tuple in that type and for m of them, the reliability tests described before of them pass, then the reliability scores of SDP, SD, SP, DP, S, D, P are 530/530, 60702/60799, 709/709, 524/525, 11185/11185, 32801/32801, 1338/1344, respectively. Taking both numbers in the reliability scores into consideration, we can know the unique-type SD,

	T	V	E	T∩V	T∩E
sdp	309073	90058	114048	530	411
sd	54944	16332	28197	9190(78141)	21767(101812)
sp	188994	57708	60273	590(794)	474(646)
dp	251870	76906	98731	543(621)	432(482)
s	9478	2567	7354	2566(90057)	7350(114044)
d	21348	6102	13052	6102(87646)	11954(111807)
p	24675	8301	8326	280(1898)	249(1898)

Table 2: The size of tuple sets

S, and D tuples can give us much help on the prediction.

- Furthermore, it's observed that for each SDP tuple $x = (s, d, p) \in V_{SDP}$ with $(s, d) \in T_{SD}$, as long as none of ipsweep, satan, smurf, smurfttl, warez, and warezclient is in $A_{SD}(s, d)$, $A_{SD}(s, d) \cap \tilde{A}_{SDP} \neq \emptyset$, i.e., we can find the ground truth answer for x in $A_{SD}(s, d)$. We call ipsweep, satan, smurf, smurfttl, warez, and warezclient **vague types**.
- Some types of connection show us very strong temporal patterns. For example, the 'satan' attacks have an average of about 682 total consecutive connection times, which is over 100 times more than the average of non-attack connections; and all the 'smurfttl' attacks make connection exactly once.

Based on the above observations and some experiment results on the validation set, we do the following assumptions to simplify the problem. In the following sections, we use the index of types of connection to denote them, see Table 3 for details.

ASSUMPTION 2.5. *Any two connections in the validation set or test set that have the same SDP tuple should have the same type of connection. In other words, for any SDP tuple $x = (s, d, p)$ in V_{SDP} or E_{SDP}, we assume that it has a unique ground truth type $c(x) \in F_c$.*

ASSUMPTION 2.6. *For any SDP tuple $x = (s, d, p)$ in V_{SDP} or E_{SDP}, if $x \in T_{SDP}$ with $|A_{SDP}(x)| = 1$, then $c(x)$ is the only element in $A_{SDP}(x)$, otherwise $c(x) = 0$; if $(d, p) \in T_{DP}$ with $|A_{DP}(d, p)| = 1$, $c(x)$ is the only element in $A_{DP}(d, p)$ if it's in $A_S(s)$, otherwise, $c(x) = 0$; if $(s, p) \in T_{SP}$ with $|A_{SP}(s, p)| = 1$, $c(x)$ is the only element in $A_{SP}(s, p)$; if $(s, d) \in T_{SD}$ with $A_{SD}(s, d)$ containing no vague types, then the ground truth type of x, $c(x) \in A_{SD}((s, d))$, specifically $c(x)$ is the only element in $A_{SD}((s, d))$ if $|A_{SD}((s, d))| = 1$; if $s \in T_S$, then $c(x) \in A_S$.*

ASSUMPTION 2.7. *For any query file q, if warezclient is in its answer $\tilde{A}(q)$, then warez should not be in $\tilde{A}(q)$, and vice versa. In other words, warez and warezclient cannot appear together in $\tilde{A}(q)$ for any query file q.*

2.1.2 Algorithm

Based on all these statistical analysis and a lot of specific analysis for some types of connection, we build up our Stat-Filter algorithm as shown in Algorithm 1 and Algorithm 2, where we use the indices of connection types to represent them. See Table 3, the index table for connection types, for more details.

2.1.3 Shortcomings

Algorithm 1: Stat-Filter

input : An SDP tuple $x = (s, d, p)$ in V_{SDP} or E_{SDP}
output: Prediction of its type $c(x)$
$CT \leftarrow |\mathbb{T}(x)|$
$SP \leftarrow \max \mathbb{T}(x) - \min \mathbb{T}(x)$
if $s \notin T_S$ **then**
  return *0*
else if $x \in T_{SDP}$ **then**
  if $|A_{SDP}(x)| = 1$ **then**
    return *the only element in* $A_{SDP}(x)$
  else
    return *0*
  end
else if $|A_S(s)| = 1$ **then**
  return *the only element in* $A_S(s)$
else if $A_S(s) = \{18, 19\}$ **then**
  if $19 \in A_{DP}(d, p)$ **then**
    return *19*
  else
    return *18*
  end
else if $|A_D(d)| = 1$ **then**
  return *the only element in* $A_D(d)$
else if $A_D(d) = \{7, 16\}$ **then**
  if $16 \in A_S(s)$ **then**
    return *0*
  else
    return *7*
  end
else if $|A_P(p)| = 1$ **then**
  return *the only element in* $A_P(p)$
else if $17 \in A_{SD}(s, d)$ **then**
  if $CT > 1000$ *and* $SP \leq CT/10$ **then**
    return *17*
  else
    return *0*
  end
else if $|A_{DP}(d, p)| = 1$ **then**
  return *the only element in* $|A_{DP}(d, p)$
else if $|A_{SP}(s, p)| = 1$ **then**
  return *the only element in* $|A_{SP}(s, p)$
else if $A_{SD} \subsetneq \{23, 24\}$ **then**
  if $CT = 1$ *or* $CT \geq 5$ **then**
    return *0*
  else if $SP \geq 1$ **then**
    return *24*
  else
    return *23*
  end
else if $|A_{SD}(s, d)| = 1$ *and* A_{SD} *contains no vague types* **then**
  return *the only element in* $A_{SD}(s, d)$
else if $A_{SD}(s, d) = \{12, 13\}$ **then**
  if $2 \leq CT \leq 3$ *and* $35 \leq SP \leq 50$ *OR* $27 \leq CT \leq 30$ *and* $56 \leq SP \leq 59$ *OR* $(CT, SP) = (2, 20)$ **then**
    return *13*
  else if $CT \geq 10$ *and* $SP \leq 1$ **then**
    return *12*
  else
    return *0*
  end
else if $13 \in A_{SD}(s, d)$ **then**
  if $2 \leq CT \leq 3$ *and* $35 \leq SP \leq 50$ *OR* $27 \leq CT \leq 30$ *and* $56 \leq SP \leq 59$ **then**
    return *13*
  else
    return *0*
  end

```
else if A_SD(s,d) = {3,18} then
    if 18 ∈ A_DP(d,p) then
        return 0
    else
        return 3
else if 3 ∈ A_SD(s,d) then
    if 51 ≤ SP ≤ 59 and 11 ≤ CT ≤ 14 then
        return 3
    else
        return 0
else if 12 ∈ A_SD(s,d) then
    if CT ≥ 10 and SP ≤ 1 then
        return 12
    else
        return 0
else if A_SD(s,d) = {20,21} then
    if CT = 1 then
        return 0
    else if CT ≤ 13 and SP ≥ 20 then
        return 20
    else if CT ≥ 37 and 49 ≤ SP ≤ 59 then
        return 21
    else
        return 0
else if 6 ∈ A_SD(s,d) then
    if CT = 2 and 38 ≤ SP ≤ 42 then
        return 6
    else
        return 0
else if 16 ∈ A_SD(s,d) then
    if (CT,SP) = (2,10) OR CT = 3 and
      (𝕋(x)[2] − 𝕋(x)[1], 𝕋(x)[1] − 𝕋(x)[0]) ∈
      {(10,10),(9,10),(9,9)} then
        return 16
    else
        return 0
else if 1 ∈ A_SD(s,d) then
    if CT ≥ 100 and SP ≤ CT/5 then
        return 1
    else
        return 0
else if 2 ∈ A_S(s) then
    if 60 ≤ 100 and 58 ≤ SP ≤ 59 then
        return 2
    else
        return 0
else if 5 ∈ A_SD(s,d) then
    if (CT,SP) = (2,30) then
        return 5
    else
        return 0
else if 8 ∈ A_SD(s,d) then
    if 47 ≤ CT ≤ 49 and 57 ≤ CT ≤ 59 then
        return 8
    else
        return 0
else if 14 ∈ A_SD(s,d) then
    if CT = 15 and 56 ≤ CT ≤ 57 then
        return 14
    else
        return 0
else
    return 0
```

Algorithm 2: Prediction using Stat-Filter

```
input : A query file g
output: The prediction of the types of attacks
        contained in the given file PRED
Initialization: PRED ← ∅
for each connection D = (s,d,p,t) in g do
    x ← (s,d,p)
    PRED ← PRED ∪ Stat-Filter(x)
end
if 23 ∈ PRED and 24 ∈ PRED then
    PRED ← PRED \ {23}
if 0 ∈ PRED then
    PRED ← PRED \ {0}
return PRED
```

Stat-Filter basically only can analyze connections whose corresponding SD-tuples can be found in the training set. Besides, for these types of connection that usually only connect once, the temporal patterns are indistinguishable by Stat-Filter because we only consider the timestamps of each single SDP tuple in the prediction. Therefore, we need to generalize the algorithm and find out some other techniques to analyze the temporal information and the interaction between different SDP tuples.

2.2 SE-GNN

2.2.1 Motivation

Although Stat-Filter can handle most cases in the validation set and the test set, it faces two challenges. First, there are some ids never seen in the training set, so Stat-Filter cannot get their corresponding information and give prediction. Second, it is difficult to answer some complex questions about the sequence of the connections, such as whether the previous connection type is influential to the next connection type? To tackle these challenges, we propose SE-GNN, a GNN-based model.

2.2.2 Aggregation Module

We first use a GNN process, called Aggregation Module, to deal with SD pairs that never appear in the training set. Aggregation Module updates every node by aggregating its neighbors' information in every training epoch. Even if one TCP id never appears in the training set, information could still transfer to it through its connections with other nodes. Therefore, we are able to do the prediction based on a node's aggregated embedding vector even when we meet it for the first time in the test set.

More specifically, we represent every id as a node in our graph, and represent each connection as an edge between the corresponding nodes of its source id and destination id. Some SD tuples appear for multiple times, and we treat such repeated patterns as a sequential issue and design a novel aggregation method to tackle it later. The edges are unweighted when building the graph for SE-GNN. We visualize our built graph in Figure 2.

In every training epoch, Aggregation Module first updates the embedding vector of each node by combining the information of its first order neighbors. Many GNN models have been proposed to efficiently aggregate a node's neighbor messages. We use GraphSage [2], one of the most popular

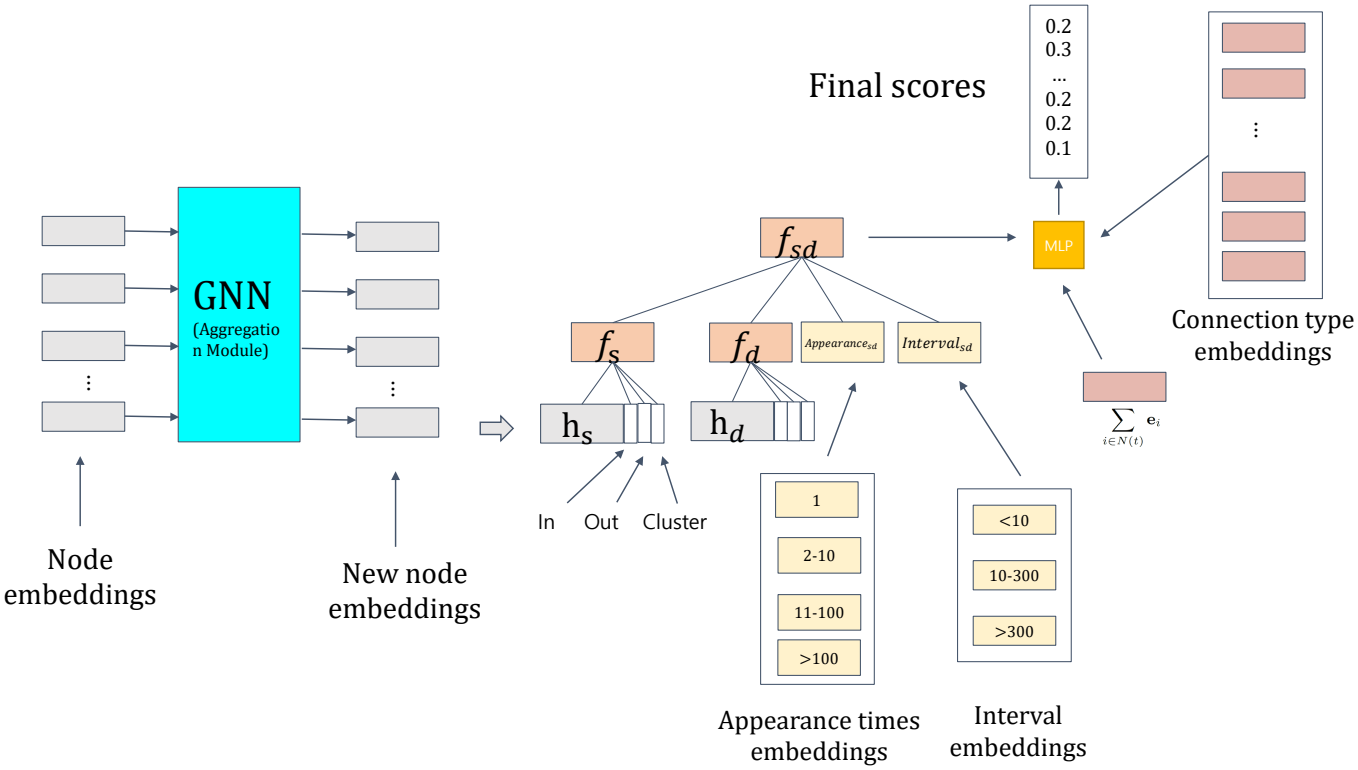

Figure 1: Overview of our proposed SE-GNN. The example is based on our assumption about the sequence of destination id.

generalized neighborhood aggregation methods, in our Aggregation Module. And we choose the mean method when dealing with the neighborhood sequence problem. The update process is:

$$\mathbf{h}_v^k = \sigma\left(\left[W_k \cdot \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right) \quad (1)$$

,

where $\mathbf{h}_v^k$ is the hidden vector of node v in step k, $\mathbf{W}_k$ and $\mathbf{B}_k$ are the GraphSage weights and $|N(v)|$ are the number of neignbors of node v.

2.2.3 *Spatial-Temporal Information*

The second challenge is how to detect the complex patterns that are not easy to observe. We divide the possible patterns into to categories: graph spatial pattern and temporal pattern. Graph spatial pattern refers to the structure of the network, such as degree, clustering coefficient and connected component. Temporal pattern refers to the sequence pattern, such as the connection frequency. Note that there are numerous possible influential spatial and temporal patterns. In this project, we only choose and evaluate some most important spatial and temporal information.

We consider in-degree, out-degree and clustering coefficient as the spatial information of each node. In-degree reflects how often is a node be connected by other nodes, out-degree reflects how often a node actively connect other nodes, and clustering coefficient shows what portion of a node's neighbors are connected. We think the combination of these three characters could give important information to predict the attack types. For example, if an id has high out-degree but low clustering-coefficient, this means that it randomly connects many nodes and might have high possibility to be a attacker. As show in Figure 3, the distribution of in-degree and out-degree have a long-tail shape, where the distribution of clustering coefficient is centred at 0, 0.5 and 1. More than 99% nodes' in-degree and out-degree are below 10, but few nodes have very high in-degree or out-degree. The highest in-degree is 8061 and the highest out-degree is 7294. We pre-process the in-degree and out-degree using min max normalization before input them into SE-GNN.

As for the temporal information, we mainly consider the appearance times and the largest interval of a SD-pair in a single file. We have 854 training files, 284 validation files and 284 test files. Each file contains many connections in a temporal window. Given a file, we record how many times a SD-pair appears in this temporal window as appearance times, and record the time interval of the first appearance and the last appearance of a SD-pair as the largest interval. If a SD-pair only appear once in a file, then its largest interval is 0. We think the combination of the appearance times and the largest interval could reflect the frequency of each SD-pair. To better cope with the temporal information, we divide the appearance times as 1 time, 2 - 10 times, 11 - 100 times and more than 100 times. We divide the largest time intervals as less than 10 seconds, 10 - 300 seconds and more than 300 seconds. And we do the embedding for every group of appearance times and largest intervals.

In the training and test steps, we combine the spatial temporal information and the hidden state of the source node and destination node using two MLPs, as following:

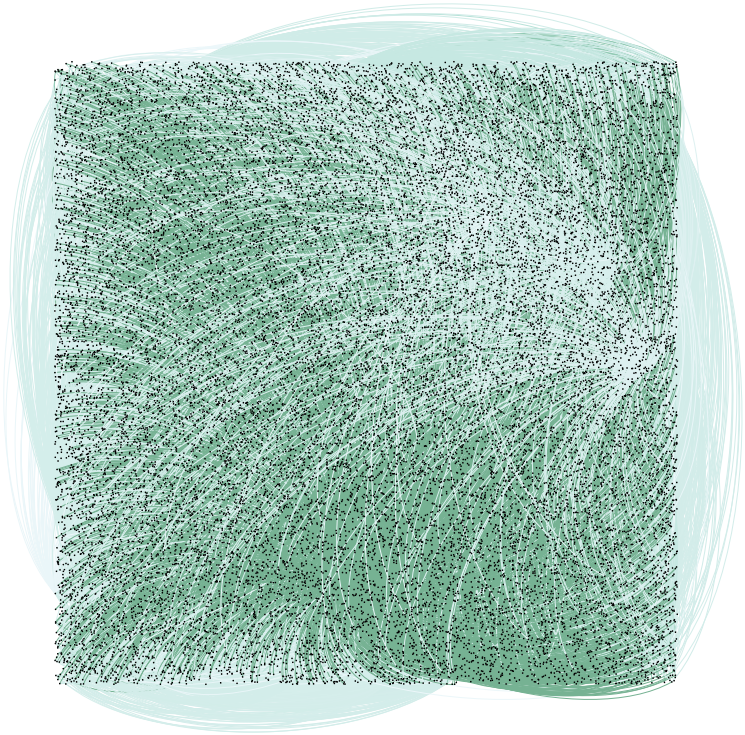

Figure 2: Visualization of TCP Graph

$$\mathbf{f}_v = MLP_1\left([\mathbf{h}_v, In_v, Out_v, Cluster_v]\right), \tag{2}$$

$$\mathbf{f}_{sd} = MLP_2\left([\mathbf{f}_s, \mathbf{f}_d, \mathbf{Appearance}_{sd}, \mathbf{Interval}_{sd}]\right). \tag{3}$$

$\mathbf{h}_v$ and $\mathbf{f}_v$ are the embedding vector (hidden state vector) and the feature vector of node v respectively, $\mathbf{f}_{sd}$ is the feature vector of connection between source node s and destination node d, In_v, Out_v and $Cluster_v$ are the scalar values of the in-degree, out-degree and clustering coefficient of node v, and $\mathbf{Appearance}_{sd}$ and $\mathbf{Interval}_{sd}$ are the embedding vectors of appearance times and largest interval of a SD-pair.

2.2.4 *Embedding the Attack Types*

As discussed in Section 2.1, attack types don't appear independently. For example, attacks 'warez' and 'warezclient' often appear in a same file together. This phenomenon inspires us to further study the correlations among the 25 connection types (including the benign connection). To achieve this target, instead of considering the TCP attack prediction as a multi-classes classification problem, we do the embeddings for all the attack types and treat the attack prediction as 25 binary classification problems. In this way, our SE-GNN could better deal with the relationship between different types based on their embedding vectors.

More specifically, for each SD-pair, SE-GNN predicts its connection type based on not only its feature vector $\mathbf{f}_{sd}$, but also the other connection types appeared in the same file.

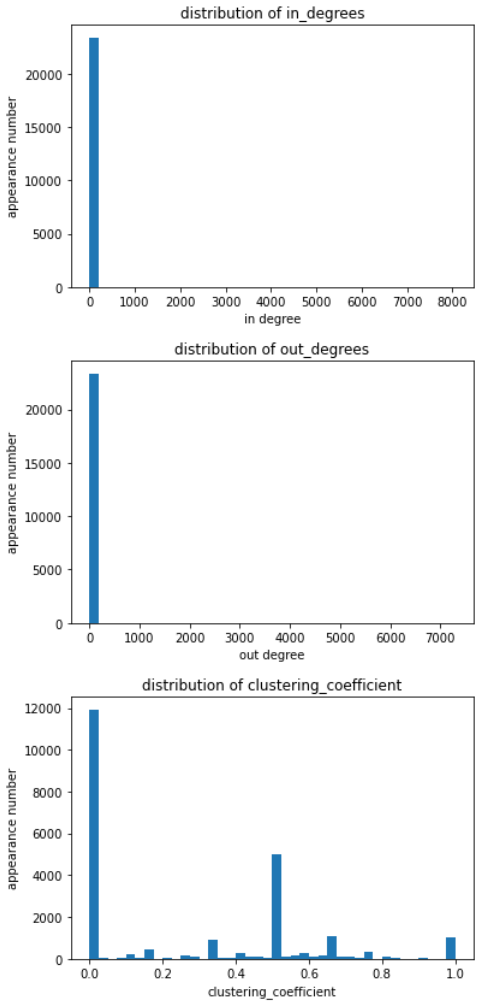

Figure 3: The distribution of in-degree, out-degree and clustering coefficient

The final prediction score for a SD-pair and connection type t is:

$$\hat{p}_{sd,t} = MLP_3\left(\left[\mathbf{f}_{sd}, \sum_{i \in N(t)} \mathbf{e}_i, \mathbf{e}_t\right]\right), \tag{4}$$

where $\mathbf{e}_t$ is the embedding vector of connection type t and $N(t)$ is the set of other connection types appeared in the same file. $\hat{p}_{sd,t}$ is the prediction of the possibility that SD-pair sd have the connection type t. The set of other connection types appeared in the same file is known in training set. And we use the prediction result of Stat-Filter to find the set of other connection types in the same file in the valid and test sets.

Although embedding every connection type gives us flexibility to consider the relations among them, it brings about the difficulty of sampling in the same time. The appearance frequency of different connection types are very different. Benign connections dominate the instances in the training set. So the training result will tremendously increase the importance of the embedding vector of benign connection. In the test time, SE-GNN might rank the possibility of benign connection first place for all the SD-pairs. Researches in nature language process usually down sampling the high frequency words to avoid this problem. In this project, however, down sampling the connection types might lose the chance to train some SD-pairs. We thus up sampling the low frequency connection types in every epoch, so that the training times of each connection type be similar.

For every SD-pair, we also do the negative sampling. Specificly, we sample 5 negative connection types for each SD-pair in the training step.

3. EXPERIMENTS

In this section, we list and analyze our experimental results of our method.

3.1 Implementation

We use python to implement all the algorithms. In SE-GNN, we use deep graph library to implement the Aggregation Module and use networkx to find the graph attributes of each node. The visualization of the TCP graph is drawn using Gephi.

3.2 Evaluation criterion

We evaluate the performance by the weighted F1-score F_1^w on the validation set or the test set, which is a weighted average of F1-score computed as follows:

$$F_1^w := \frac{1}{2}\left[\frac{1}{|B|}\sum_{g \in B} F_1(g) + \frac{1}{|A|}\sum_{g \in A} F_1(g)\right],$$

where B represents the set of networks without any attacks, and A represents these containing at least one type of attacks. And the standard F1-score is

$$F_1(g) = 2 \times \frac{prec(g) \cdot recall(g)}{prec(g) + recall(g)},$$

where

$$prec(g) = \frac{1 + |G(g) \cap P(g)|}{1 + |P(g)|}$$

and

$$recall(g) = \frac{1 + |G(g) \cap P(g)|}{1 + |G(g)|},$$

with $G(g)$ denoting the set of attack types actually included in g and $P(g)$ denoting the set of attack types predicted to be included in g.

3.3 Mid-term results

Up to the progress report deadline, we have implemented Stat-Filter, which gives us a weighted F1 score 0.9122 on the validation set.

To analyze the performance, we also summarize the mismatching cases of the prediction of Stat-Filter in Table 4. Generally, false positive cases are much more than false negative cases. For example, attack types 'warez', 'warezclient', and 'nmap' tend to be wrongly regarded as positive by Stat-Filter, while 'portsweep' and 'dict' tend to be missed by Stat-Filter.

3.4 Final results

After the modification on the algorithm of Stat-Filter, and the combination of Stat-Filter and SE-GNN, we finally

Index	Connection Type	Index	Connection Type
00	-	15	rootkit
01	apache2	16	saint
02	back	17	satan
03	dict	18	smurf
04	guest	19	smurfttl
05	httptunnel-e	20	snmpgetattack
06	ignore	21	snmpguess
07	ipsweep	22	teardrop
08	mailbomb	23	warez
09	mscan	24	warezclient
10	neptune		
11	nmap		
12	pod		
13	portsweep		
14	processtable		

Table 3: The index table for connection types

Index	Attack type	# false positive
23	warez	29
24	warezclient	20
11	nmap	6
7	ipsweep	5
13	portsweep	5
20	snmpgetattck	5
18	smurf	4
17	satan	2
9	mscan	1
19	smurfttl	1

(a) The false positive cases

Index	Attack type	# false negative
13	portsweep	17
3	dict	6
7	ipsweep	4
4	guest	3
24	warezclient	3
12	pod	4
17	satan	2
5	httptunnel-e	2
2	back	1
6	ignore	1
10	neptune	1
15	rootkit	1
16	saint	1
23	warez	1

(b) The false negative cases

Table 4: The mismatching cases in mid-term results

Mismatching type	Count
$23 \rightarrow 24$	8
$24 \rightarrow 23$	3

(a) Two special mismatching cases

Index	Attack type	# false positive
13	portsweep	3
18	smurf	1
24	wareclient	1

(b) The false positive cases

Index	Attack type	# false negative
3	dict	6
4	guest	3
17	satan	2
2	back	1
12	pod	1
13	portsweep	1
15	rootkit	1
16	saint	1
23	warez	1
24	warezclient	1

(c) The false negative cases

Table 5: The mismatching cases in final results

achieve a weighted F1 score 0.9663 on the validation set. In detail, we use SE-GNN to find the most possible connection type candidates for those SD tuples whose destination id cannot be found in the training set. Therefore, we can extend our prediction framework of Stat-Filter for those SD tuples.

To analyze our results more clearly, again, we summarize the mismatching cases in Table 5, where we list two specific cases aside, which are the case when ground truth contains 23 but it is predicted to contain 24 (represented as $23 \rightarrow 24$) instead of it and the opposite case (represented as $24 \rightarrow 23$). We list them separately because they are the most common mismatching cases.

3.5 Prediction results on the test set

As the ground truth of the test set is unknown, we list our prediction results on the test set in Table 6. Those test query files whose corresponding indices don't appear in the table are predicted to contain no attacks. And we use indices of connection types to represent the results.

4. CONCLUSIONS

Our hybrid method consisting of Stat-Filter and SE-GNN can do fast and accurate prediction for SD tuples seen before in the given history as well as have the capacity of dealing with general cases. On the given dataset, our method achieves quite high performance on the validation set.

5. REFERENCES

[1] Debdeep Dey, Archisman Dinda, Poornima Panduranga Kundapur, and R Smitha. Warezmaster and warezclient: An implementation of ftp based r2l attacks. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE, 2017.

File index	Prediction result
19	[10]
39	[7]
58, 61	[13]
64	[2]
76	[11]
77	[5, 11]
78 - 79	[11]
81	[10]
88	[7, 13]
93 - 97	[24]
98 - 99	[23]
104	[24]
105	[23]
106	[24]
107	[13, 23]
108 - 110	[13, 24]
111	[13, 23]
112 - 113	[13, 24]
114	[23]
115 - 116	[24]
124	[7]
132	[6]
136	[3, 10, 13, 18]
137	[24]
139	[13]
140	[10, 13]
141	[13]
147	[16]
152	[10]
154 - 160	[13]
163	[10]
172	[12]
173	[10, 13, 22]
174	[10, 13]
175	[7, 13]
176	[10]
185 - 188	[13]
190	[7]
194	[13]
204	[9]
206	[10, 13]
207, 222	[10]
225 - 232	[11]
233	[21]
234	[11, 21]
235 - 236	[11]
237 - 241	[20]
242	[13, 20]
243	[12, 20]
244 - 246	[20]
247 - 251	[23]
252 - 253	[9, 23]
254	[9, 24]
255	[9]
256	[9, 24]
257 - 260	[9, 23]
261 - 262	[11, 20, 23]
263	[11, 20, 24]
264 - 265	[11, 20, 23]
266	[20, 23]
267	[5, 10, 11, 13, 20, 24]
268 - 272	[11, 13, 20, 23]
273	[11, 13, 20, 24]
274	[6, 11, 13, 20, 23]
275	[3, 6, 11, 13, 20, 23]
276	[3, 6, 13, 20, 23]
277	[5, 6, 20, 23]
278	[3, 20, 23]
279	[3, 20, 24]
280	[20, 23]
281	[5, 20, 23]
282 - 283	[20, 23]

Table 6: The prediction results on the test set

[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[3] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[4] Robbie Myers. Attacks on tcp/ip protocols. *Last accessed Jan*, 4, 2016.

[5] Peter W Singer and Allan Friedman. *Cybersecurity: What everyone needs to know*. oup usa, 2014.

[6] Prem Uppuluri and R Sekar. Experiences with specification-based intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 172–189. Springer, 2001.

[7] Mark Wolfgang. Host discovery with nmap. *Exploring nmap's default behavior*, 1:16, 2002.

APPENDIX

A. APPENDIX

A.1 Labor Division

The team performed the following tasks

- Statistical Analysis [Fanchen]
- Implementation of Stat-Filter [Fanchen]
- Survey of GNN methods [Guoyuan]
- Implementation of SE-GNN [Guoyuan]