IP2Vec: Learning Similarities between IP Addresses

Markus Ring, Dieter Landes
Faculty of Electrical Engineering and Informatics
Coburg University of Applied Sciences and Arts,
96450 Coburg, Germany
Email: {markus.ring,dieter.landes}@hs-coburg.de

Alexander Dallmann, Andreas Hotho
Data Mining and Information Retrieval Group
University of Würzburg
97074 Würzburg, Germany
Email: {dallmann,hotho}@informatik.uni-wuerzburg.de

***Abstract*—IP Addresses are a central part of packet- and flow-based network data. However, visualization and similarity computation of IP Addresses are challenging to due the missing natural order. This paper presents a novel similarity measure *IP2Vec* for IP Addresses that builds on ideas from Word2Vec, a popular approach in text mining. The key idea is to learn similarities by extracting available context information from network data. IP Addresses are similar if they appear in similar contexts. Thus, *IP2Vec* is automatically derived from the given network data set. The proposed approach is evaluated experimentally on two public flow-based data sets. In particular, we demonstrate the effectiveness of clustering IP Addresses within a botnet data set. In addition, we use visualization methods to analyse the learned similarities in more detail. These experiments indicate that *IP2Vec* is well suited to capture the similarity of IP Addresses based on their network communications.**

I. Introduction

The idea of detecting novel or obfuscated attacks using data mining algorithms is followed by the community over decades. An overview of the community effort with regard to this issue can be found in [1], [2] and [3]. Yet, most of these approaches do not operate successfully in real operational environments. Sommer and Paxon [4] identified various reasons (e.g. the lack of publicly available training data sets, the variability of input data or the high cost of false positives) for the limited success of anomaly-based intrusion detection systems. In this work, we focus on a specific problem within that setting.

Problem. In particular, we focus on network-based data and tackle the problem of calculating behavioral similarities between *IP Addresses*. *IP Addresses* are typical features in packet- and flow-based network data. Although these features look like numbers at first sight, they need to be treated like categorical features. Distance measures like the *Minkowski distance* cannot be applied easily. Consequently, many standard data mining algorithms cannot be used for network-based data since the data encompass continuous as well as categorical features.

Objective. We endeavor to find continuous vector representations (so called embeddings) for *IP Addresses* that carry information about their behavior within the network. To that end, we propose an approach - which we call *IP2Vec* - that can derive such representations of *IP Addresses* from contextual information given in flow-based data. These real-valued vector representations define meaningful similarities between *IP Addresses* with respect to their behaviour. Within this space, similarity measures like the *Cosine similarity* (or distance measures like the *Minkowski distance*) can be used to calculate behavioral similarities (distances) between *IP Addresses*. As a consequence, the vector representations of *IP Addresses* can be used as features for data mining methods and enable them to exploit the encoded behavioral information. The *IP Address* representations can also be visualized to gain further insights into the collected flow-based data.

Approach and Contributions. Processing of non-continuous features is a well-known problem in text mining. *Word2Vec* [5] is an algorithm which uses a text corpus as input and creates real-valued vector representations of words as output. *Word2Vec* has been proven highly successful in natural language processing in different settings [6]. Since the underlying constellation exhibits some similarities to the domain of network data, we transfer this approach to flow-based network data to learn real-valued vector representations of *IP Addresses*. In our setting, we use the captured flow-based data set as *text corpus* and the *IP Addresses*, *Ports* and *Protocols* as the vocabulary. The main idea is to train a fully connected neural network with a single hidden layer which is much smaller than the input and output layer of the network. The number of neurons in the input layer depends on the size of the vocabulary within the given data set. The size of the output layer is identical to the input layer. After training, the neural network is not actually used. Instead, we use the weights between the input and hidden layer of the neural network as vector representations of the *IP Addresses*.

We experimentally illustrate our approach for clustering hosts with similar behaviour on two public flow-based data sets: *CTU-13* [7] and *CIDDS-001* [8]. Further, we use *t-SNE* [9], a technique for visualizing high dimensional vector spaces in two-dimensional spaces, to illustrate the learnt similarities.

Our main contribution is the presentation of *IP2Vec*, an unsupervised approach to learn vector representations of *IP Addresses* which can be used in order to assess the contextual similarity of these *IP Addresses*. To that end, we build upon ideas of *Word2Vec* and apply an analogous approach to *IP Addresses* by defining a context based on flow-based network data.

Structure. The rest of the paper is organized as follows: Related work on handling *IP Addresses* in anomaly-based network intrusion detection systems is discussed in the next

2375-9259/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDMW.2017.93

section. Since our proposed approach is based on *Word2Vec*, we provide a short description of *Word2Vec* in Section III. Section IV describes the proposed *IP2Vec* approach in detail. An experimental evaluation is given in Section V. Section VI discusses the results as well as advantages, disadvantages and open issues of *IP2Vec*. The last section summarizes the paper.

II. Related Work

This section reviews related work on processing *IP Addresses* within anomaly-based network intrusion detection systems. A comprehensive review of distance measures used for network intrusion anomaly detection is given in [10]. Weller-Fahy et al. [10] discuss various types of distance measures and their theoretical background as well as their use on packet- and flow-based network data. In the following, we concentrate on the different approaches of processing *IP Addresses* within network data. We categorize them into (I) ignoring *IP Addresses* for distance calculation, (II) extracting meaningful features from *IP Addresses*, and (III) defining metrics on *IP Addresses*.

Tran et al. [11] proposed a real-time flow-based intrusion detection system (IDS) which falls within category (I). The authors use a block-based neural network and integrate it within a high-frequency FPGA. They extract only four features (*Packets, Bytes, Duration* and *Flags*) from the available flow-based features and use them as input for their IDS. Najafabadi et al. [12] analyse the detection of *RUDY* attacks based on flow-based network data using classification algorithms. *RUDY* attacks are *application layer DoS attacks* and generate much less network traffic than traditional *DoS* attacks. For the detection of this type of attack, the authors extract several features from flow-based data, but they do not include *IP Addresses* in their feature list. Similarly, Najafabadi et al. [13] analyse the detection of *SSH Brute Force Attacks* on flow-based network data. Different classification algorithms are evaluated with two different subsets of features. *IP Addresses* appear in neither of these subsets. DISCLOSURE is a flow-based approach for botnet command and control server detection [14]. Bilge et al. [14] use *IP Addresses* and *Ports* to distinguish between client and server, but for distance calculation *IP Addresses* are ignored. Beigi et al. [15] evaluate different flow-based features for botnet detection. However, they do not consider *IP Addresses* in their selected feature subsets. Further anomaly-based approaches which do not consider *IP Addresses* are [16], [17], or [18].

The second category includes anomaly-based approaches that extract features from *IP Addresses*. One approach is to binarize the *IP Address* and extract 32 binary features from each *IP Addresses* [19]. More common approaches aggregate flows over time windows and calculate features out of these aggregations. Garcia et al. [7] presented such an approach for behavioural-based botnet detection. Their method BClus partitions the flow-based data stream in time windows and aggregates the flows with respect to their *Source IP Address* in each time window. Then, new features (e.g. the number of unique *Destination IP Addresses* contacted by this *Source IP Address*) are calculated for each aggregation and machine learning methods are applied on these aggregations for botnet detection. A similar approach is presented by Mathur et al. [21]. The authors aggregate all flows from the same *IP Address* within a time window and calculate features such as the arithmetic mean of *Destination IP Addresses* (treating *IP Addresses* as integers) or the entropy of the distribution of these *Destination IP Addresses*. The same idea is used in the intrusion and insider threat detection toolset CUF [20]. CUF [20] aggregates all flows from the same *Source IP Address* within a time window and calculates new features based on these aggregations. In [22], machine learning approaches are used to classify host roles on flow-based data. Here, the authors also aggregate the flows over time windows and extract features (e.g. sum of first byte of other parties' *IP Addresses*) out of the *IP Addresses*.

Approaches from category (III) define metrics based on *IP Addresses*. A simple metric is used in [23]. The authors interpret *IP Addresses* as 32 bit integers and calculate distances using the *Minkowski distance* on these integers. Another popular approach is the transformation of *IP Addresses* to geographical locations [24] or [25]. One advantage of this transformation is that geographical locations are real numbers and standard distances measures can be applied. A more sophisticated approach is developed by Coull et al. [26]. The authors used domain knowledge to define a hierarchy among *IP Addresses*. The distance between two *IP Addresses* is determined by the level of the hierarchy at which the values differ. Coull et al. use the categorization *Unicast*, *Multicast*, *Broadcast*, *Public* or *Private IP Address* to build their hierarchy. However, such a metric considers only the network structure and not the behaviour of the hosts. Recently, Jakalan et al. [27] proposed another approach that falls within category (III). In their work, the authors divide *IP Addresses* in two categories: *inside* and *outside* the network. Then, the authors define the similarity between *inside IP Addresses* by considering the number of common outside *Destination IP Addresses*.

Another approach of processing *IP Addresses* could be the use of categorical distance measures like ConDist [28], [29], DILCA [30], or the distance measure from Jia and Cheung [31]. These distance measures use correlated context attributes to determine distances between categorical values.

In addition to these approaches, Henry [32] uses *Word2Vec* for calculating similarities between network flows. Our approach is similar in spirit, but we adopt *Word2Vec* to learn similarities between *IP Addresses* based on network flows. This allows us to calculate similarities between *IP Addresses* and not between concrete flows. Further, the size of the input vocabulary is reduced significantly.

Our approach belongs to category (III) and defines a metric for *IP Addresses*. However, we do not use additional domain knowledge like [26]. Instead, we learn the similarity between *IP Addresses* by extracting information from the observed network data like [27]. In contrast to [27], we use the default five tuple of flow-based data and transfer the embedding idea of *Word2Vec* - which works well for word similarities - to *IP Addresses*.

III. Word2Vec

Our approach is based on *Word2Vec* [5], [33]. *Word2Vec* aims at embedding words into a lower-dimensional feature space based on the context in which they frequently occur, such that words with similar contexts are near each other in feature space. Recently, this approach has been successfully applied to derive lower-dimensional feature vectors for nodes in large graphs, by adapting the notion of context [34], [35].

Since we propose a similar adaptation for flow-based data, we provide a short description of the underlying idea of *Word2Vec* based on [5], [33] and [36].

A. Model

The basis of *Word2Vec* is a neural network with a single hidden layer. Since neural networks cannot be fed with *words*, each word is represented as a one-hot vector and the length of this vector is equal to the size of the vocabulary. Let us assume a vocabulary of 20,000 words. Then, the one-hot vector has 20,000 components (one for each word) and only one component is 1, while all the others are 0. The number of input and output neurons of the neural network is equal to the size of the vocabulary. Further, the output layer uses a softmax classifier and indicates the probabilities that a particular word appears in a specific context. The number of neurons in the hidden layer is much smaller than in the input layer. Mikolov et al. [5] used 300 neurons in their hidden layer.

B. Training

The neural network is trained using a large text corpus. In the following, the training process will be explained based on an example. Let us assume that *Computers are subject to security attacks* is a training sentence. Figure 1 illustrates the generation of training samples from this sentence.

Initially, a so-called *input word* is selected from the training sentence. Next, words from the surrounding window (we refer to them as *context words*) of the *input word* are chosen for building training samples. In Figure 1, window size of 2 is used.

In *Word2Vec*, the neural network is fed with the *input word* and tries to predict the probability of the *context word*. For our training samples (see Figure 1), the probability is 1 for the *context word* and 0 for all other words. The output layer of the neural network indicates how likely each word of the vocabulary may be found in the context of the *input word* [36].

C. Negative Sampling

A vocabulary size of 20,000 and 300 hidden neurons would lead to 60 million weights each in the hidden and the output layers in a fully connected neural network. Further, we use a large text corpus for training which leads to a huge number of training samples. As a consequence, the training process of the neuronal network is very time-consuming.

Therefore, Mikolov et al. [5] introduced negative sampling for training. When using negative sampling, only a small part

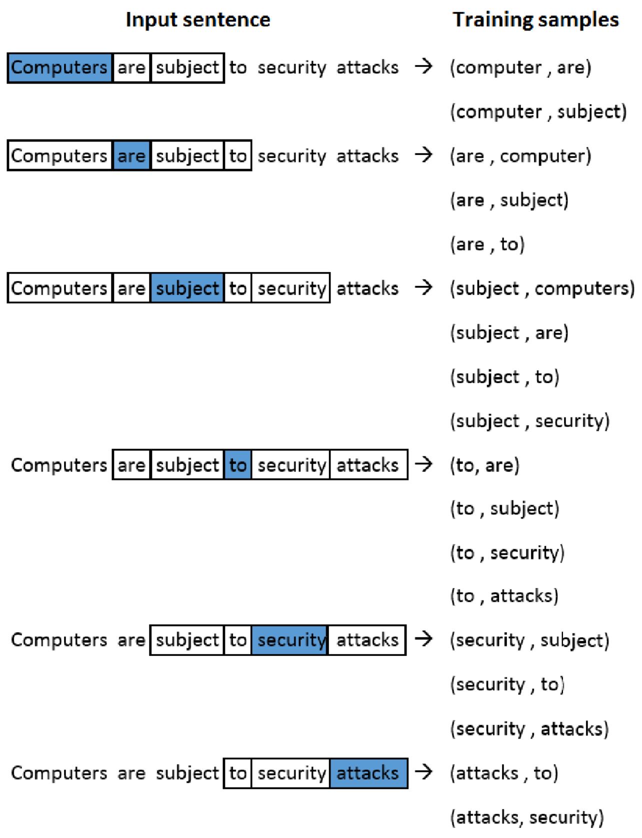
Input sentence
Training samples
Computers are subject to security attacks → (computer , are)
(computer , subject)
Computers are subject to security attacks → (are , computer)
(are , subject)
(are , to)
Computers are subject to security attacks → (subject , computers)
(subject , are)
(subject , to)
(subject , security)
Computers are subject to security attacks → (to, are)
(to , subject)
(to , security)
(to , attacks)
Computers are subject to security attacks → (security , subject)
(security , to)
(security , attacks)
Computers are subject to security attacks → (attacks , to)
(attacks, security)

Fig. 1. Generation of training samples. The *input words* are highlighted in blue colour and the *context words* are highlighted in black frames with white background. The right side of the figure shows the generated training samples for the corresponding combinations of *input word* and *context words*.

of the weights of the neural network is updated for each training sample. Instead of updating all weights of the neural network, only the weights of the correct output and a few randomly selected wrong output neurons are updated.

D. Usage

After training the neural network, the network as such is not used for the task for which it has been trained. Instead, we use only the weights at the hidden layer as feature vectors of the words. Generally, this approach is a well-known trick from unsupervised learning and amongst others it is used by Autoencoders for dimensionality reduction. Through this training, the neural network is able to learn similarities of words.

For two different words (e.g. computer and notebook) with similar contexts, the neural network needs to generate similar output values [36]. As a consequence, the weights of *computer* and *notebook* must be similar.

IV. IP2Vec

This section presents the proposed method *IP2Vec* in more detail. We start with the problem setting and outline the

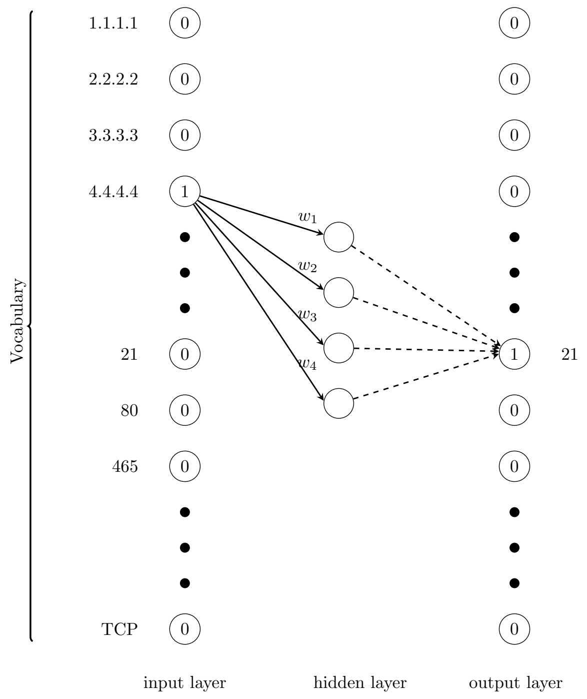

Fig. 2. Architecture of the neural network used by *IP2Vec*. For reasons of clarity, only the weights of one input and one ouput neuron are drawn. In this figure, the neural network is trained with the sample (4.4.4.4 , 21). The vector with the components w_1, w_2, w_3 and w_4 is the vector representation of the *IP Address* 4.4.4.4.

underlying ideas of *IP2Vec*. Then, we describe how we define a context for flow-based data and how training samples are generated.

A. Problem Definition and Approach

Packet- and flow-based network data consist of continuous (e.g. number of transferred *Bytes*) and categorical (e.g. *Source IP Address*) features. This mixture and especially the presence of categorical features complicates the application of many data mining methods. This work transforms *IP Addresses* into vectors from a vector spaces R^d such that we are in a position to calculate meaningful similarities between *IP Addresses*.

The proposed approach follows the idea of *Word2Vec* (see Section III), but adapts it to network data. We extract *IP Address* contexts from flow-based data and build training samples for training the neural network. Similar to *Word2Vec*, the weights of the hidden layer can then be used as a contextual feature representation for the *IP Addresses*. Figure 2 shows the neural network architecture of *IP2Vec*. In the following, the structure and training of the neural network is explained in more detail.

B. Selection of Context

A core issue in the proposed approach is a proper definition of context. This work relies on unidirectional flow-based data. Flows describe meta information about connections between endpoint devices and typically encompass *Source IP Address*, *Source Port*, *Destination IP Address*, *Destination Port*, *Protocol*, *Bytes*, *Packets*, *TCP-Flags*, *Timestamp* and *Duration* as features.

TABLE I
INPUT DATA SET.

#	Source IP Addr.	Dest. IP Addr.	Dest. Port	Proto.
1	192.168.100.5	192.168.220.9	51479	TCP
2	192.168.220.9	192.168.100.5	445	TCP
3	216.58.210.19	192.168.200.8	44444	TCP
4	192.168.200.8	216.58.210.19	80	TCP
5	192.168.220.14	53.53.53.53	53	UDP

Since each flow describes one network connection, the features within a flow are content-related. Therefore, each flow in our data set can be viewed as analog to a *sentence* in *Word2Vec*.

Another issue is the selection of features which are then used in further computations. The choice of features determines how similarities between *IP Addresses* are defined. In our setting, we aim at identifying hosts with similar behaviour. Therefore, we only choose features which describe the type and target of the connections. Specifically, we use *Source IP Address*, *Destination IP Address*, *Destination Port* and *Protocol* as features. Table I shows five flows with these features.

If the goal would be to distinguish real user behaviour from scripted behaviour, it would make sense to include also the *Duration* or the number of transferred *Bytes* in set of selected features. Our experience shows that continuous features of flow-based data (e.g. *Duration*, *Bytes*, or *Packets*) should be discretized before processing in order reduce the possible value range.

C. Generation of Training Samples

Each flow in Table I is considered a ”sentence” in the training process.

Rather than using all features as input words, we only use the *Source IP Address*, *Destination Port* and *Protocol* as *input words*. Further, we use for each *input word* a specific subset of *context words*. For instance, when using the *Source IP Address* as *input word*, the *Destination IP Address*, *Destination Port* and *Proto* are used as *context words*. In contrast to that, when we use the *Destination Port* as *input word*, we only choose the *Destination IP Address* as *context word*. Generally, the complete generation process of training samples is illustrated in Figure 3.

We do not use the *Destination IP Address* as *input word* since we build upon unidirectional flow-based data. Responses from the *Destination IP Address* are captured in a separate flow where the roles of *Source IP Address* and *Destination IP Address* are swapped (see flow #1 and #2 in Table I). Therefore, considering both *IP Addresses* as input words would create duplicate training samples. For the same reason, we use only the *Destination IP Address* as *context word* for the input words *Proto* and *Destination Port*.

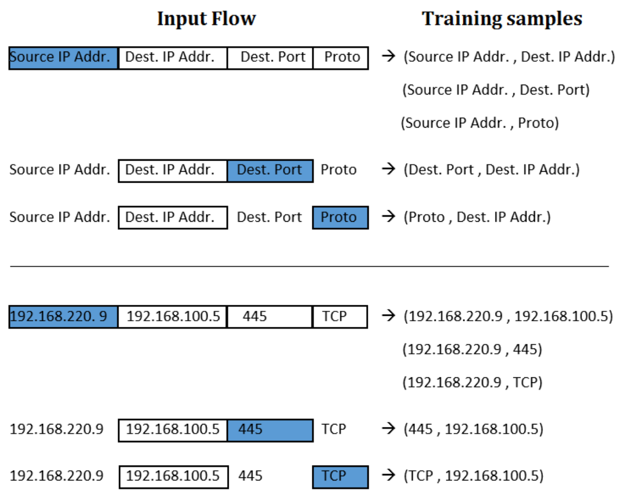

Fig. 3. Generation of training samples. The upper part of the figure shows general process of training sample generation. *Input words* are highlighted in blue colour and *context words* are highlighted in black frames with white background. The right side of the figure shows the generated training samples for the corresponding combinations of *input word* and *context words*. The lower part of the figure provides an example and shows the generated training samples for flow #2 from Table I.

D. Vocabulary

The values of our vocabulary depend on the training data set and the selected features. In our setting, we select *Source IP Address*, *Destination IP Address*, *Destination Port* and *Protocol*. As a consequence, our vocabulary contains not only *IP Addresses*, it also contains *Port* and *Protocol* values (see Figure 2).

Consequently, we can not only calculate similarities between *IP Addresses*, we are also able to calculate similarities between *IP Address* and *Port* or between two *Ports*.

E. Differences to Word2Vec

Word2Vec and *IP2Vec* differ in several aspects. First and foremost, there is a fundamental difference between the underlying data. *Word2Vec* uses a text corpus as input data and extracts relationships through surrounding words while *IP2Vec* relies on data from unidirectional flows. In contrast to *Word2Vec*, *IP2Vec* does not use each selected feature as *input word* and the surrounding features as *context words*. Rather, we use only a subset of features as *input word* and individual adapted subsets of features as *context words*.

Another fundamental difference between *Word2Vec* and *IP2Vec* is that the similarity between *words* is constant over time, while the similarity between *IP Addresses* varies over time. The latter is due to the fact that similarities between *IP Addresses* are computed on the basis of their network connections. Network connections, however, are likely to change, e.g. if a server provides additional services or a client is infected with a virus. In order to exclude this effect, we evaluate *IP2Vec* in the next section only on offline scenarios.

V. Experiments

This section presents an experimental evaluation of *IP2Vec*. *IP2Vec* aims to transform *IP Addresses* to vector representations such that *IP Addresses* with similar behaviour have similar vector representations. For evaluating this transformation, we use the derived vector representations as input for a clustering algorithm and a qualitative analysis based on *t-SNE* visualizations. Evaluation of *IP2Vec* is successful, when *IP Addresses* with similar behaviour are in the same cluster and appear nearby in the visualization method. Further, *IP Addresses* with different behaviour should be assigned to different clusters and be far away in the visualization. We compare *IP2Vec* with a graph-based similarity measure [26] which is explained in Section V-A1.

A. Evaluation Methodology

1) Definition of a Baseline: We use the graph-based metric proposed in [26] as baseline and refer to it as *GRAPH*. Figure 4 gives an overview of this metric.

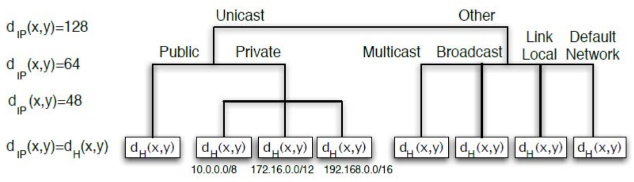

Fig. 4. Graph-based metric for *IP Addresses* [26].

In the first step, *GRAPH* distinguishes the different categories of *IP Addresses*. *IP Addresses* are categorized into *Private*, *Public*, *Multicast*, *Broadcast*, *Link Local*, *Default Network*. Figure 4 indicates that the distance between a *Private IP Address* and a *Multicast IP Address* is $d(Private, Multicast) = 128$. The distance between two *IP Addresses* from the same category is calculated with the *Hamming distance* on the binary representation of the *IP Addresses*.

2) Evaluation Data Sets: For evaluation, we use two flow-based benchmark data sets. The first data set is *CTU-13* [7] which contains normal and malicious network traffic. Different malware scenarios are used to create the malicious network traffic. Overall, the data set is split into 13 scenarios. In this work, we use the split *Scenario-50* because it contains (like *Scenario-51*) the largest number of infected hosts. Host identification is based on the *Source IP Address*. This scenario contains more than 400000 different *IP Addresses*. We train *IP2Vec* and *GRAPH* on the whole data set, but to reduce the number of *IP Addresses*, especially for visualization, we consider only *IP Addresses* from the recorded subnet $(147.34.X.X)$ in following visualization and clustering processes. We label the *IP Addresses* according to *infected* and *normal*.

The second data set in our evaluation is *CIDDS-001* [8] which comes with labelled flow-based data along with a technical report with detailed information about hosts and their activities. From the *CIDDS-001* data set, we use only *week3* from the OpenStack part which is free of attacks. Again, we use all *IP Addresses* for training *IP2Vec* and *GRAPH*, but we consider only the internal hosts for visualization and clustering. We label these *IP Addresses* according to *server* and *client*.

3) Visualization: We use *t-SNE* [9] to visualize the similarities between *IP Addresses*. *t-SNE* is a nonlinear dimensionality reduction method for high-dimensional data. It reduces high-dimensional data to two-dimensional spaces in such a way that similar objects have small distances in two-dimensional space.

4) Clustering: After learning similarities between *IP Addresses* within the data sets, we want to cluster the vector representations of *IP Addresses*. The choice of the clustering algorithm depends on the following considerations: Normal user behaviour can be very different (e.g. the behaviour of a system administrator compared to the behaviour of a recruitment consultant). The same applies to servers that offer different services. Since we expect that *IP2Vec* learns also these differences, we do not use a clustering algorithm which groups all *IP Addresses* to a predefined number of clusters. Instead, we require a clustering algorithm which groups only *IP Addresses* with high similarities together.

Therefore, we use *DBScan* [37] in our experimental evaluation. For all experiments, we set the parameter *min_sample* of *DBScan* to 3. The parameter *epsilon* is experimentally determined in each experiment. We evaluate the clustering result with three standard evaluation measures: *accuracy*, *homogeneity* and *completeness*. All three evaluation measures are scaled to the interval $[0, 1]$ with higher values representing better scores.

5) Configuration of IP2Vec: For *IP2Vec*, we use the following parameters. We define the size of the hidden layer to 32 neurons and we train the neural network with 10 epochs.

B. Experiment 1 - Identification of Botnets

In the first experiment, we trained *IP2Vec* on *Scenario 50* of the *CTU-13* [7] data set. After learning the vector representations of the *IP Addresses* we use the *IP Addresses* of the subnet $(147.34.X.X)$ and visualized them with *t-SNE*. Figure 5 illustrates the visualization for *IP2Vec*.

Figure 5 shows that infected hosts can be visually separated from normal clients. Blue crosses represent infected hosts whereas red circles illustrate normal clients. Further, it is observable that the behaviour of normal clients is very different.

Figure 6 illustrates the visualization of the same *IP Addresses* using the *GRAPH* similarity measure. In this visualization, infected hosts can not be visually separated from normal hosts.

Then, we use both similarity measures to cluster the *IP Addresses* with respect to their behaviour in two groups: *infected* and *normal*. The assignment matrix for *IP2Vec* is given in Table II and for *GRAPH* in Table III. Evaluation measures for both are shown in Table IV.

TABLE II
ASSIGNMENT MATRIX FOR THE *CTU-13* DATA SET USING *IP2Vec* AS SIMILARITY MEASURE FOR *DBScan*.

Class	Cluster 1	Cluster 2	Cluster 3	Num. Outliers
normal	1015	0	3	156
infected	0	10	0	0

TABLE III
ASSIGNMENT MATRIX FOR THE *CTU-13* DATA SET USING *GRAPH* AS SIMILARITY MEASURE FOR *DBScan*.

Class	Cluster 1	Cluster 2	Cluster 3	Num. Outliers
normal	1017	3	3	151
infected	10	0	0	0

Table II shows that *DBScan* was able to separate the normal and infected *IP Addresses* in different clusters for *IP2Vec*. Further, *IP2Vec* and *GRAPH* generate a similar number of outliers. Using *GRAPH*, *DBScan* was not able to create homogeneous clusters. Cluster 1 contains normal and infected *IP Addresses* (see Table III). Table IV shows that *DBScan* is in combination with *IP2Vec* able to generate more accurate, homogeneous and complete clusters than with *GRAPH*.

C. Experiment 2 - Server-Client Identification

In the second experiment, we used the above described part of the *CIDDS-001* data set for training *IP2Vec*. After learning the vector representations of all *IP Addresses*, we visualized only the internal *IP Addresses* with *t-SNE*. Figure 7 illustrates the visualization for *IP2Vec*.

Figure 7 shows that clients and servers can be visually separated from each other. Further, it is observable that linux clients are more similar to each other than to windows clients. In contrast to that, similarities between servers are smaller and they have higher distances in the visualization.

Figure 8 illustrates the visualization of the same *IP Addresses* using *GRAPH* similarity measure.

In Figure 8, servers and clients can not be visually separated from each other.

Then, we used *DBScan* to cluster the *IP Addresses*. The assignment matrix for *IP2Vec* is given in Table V and for *GRAPH* in Table VI.

Table V shows that *DBScan* was able to separate the clients and servers in two different cluster. Further, one of the servers is an outlier which could not be assigned to any cluster. In contrast to that, using *GRAPH* as similarity measure for *DBScan*, the resulting clusters are mixed with servers and clients (see Table VI). However, all *IP Addresses* could be

TABLE IV
COMPARISON OF *IP2Vec* AND *GRAPH* FOR CLUSTERING THE *IP Addresses* WITHIN THE *CTU-13* DATA SET.

Similarity Measure	Accuracy	Homogeneity	Completeness	Num. Outliers
IP2Vec	0.8657	1.0	0.1072	156
GRAPH	0.8590	0.0248	0.0029	151

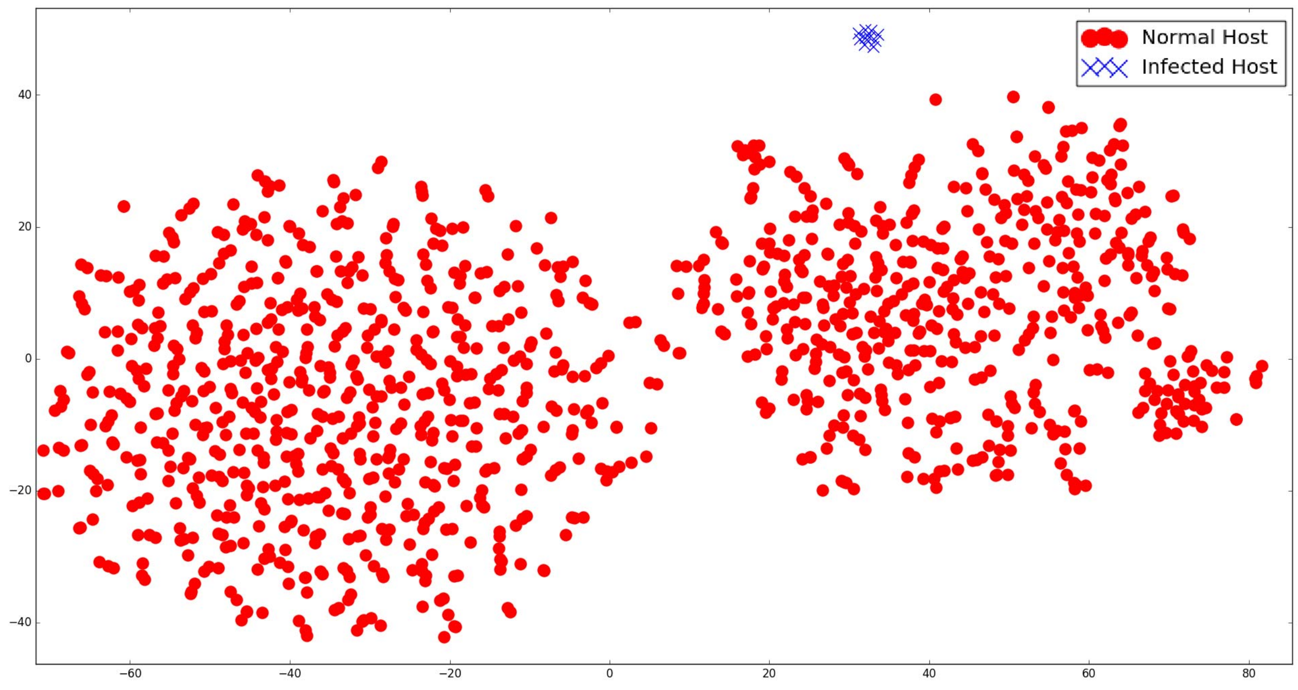

Fig. 5. *t-SNE* visualization of the *IP Addresses* using the similarity measure *IP2Vec* for Scenario 50 of the *CTU-13* data set.

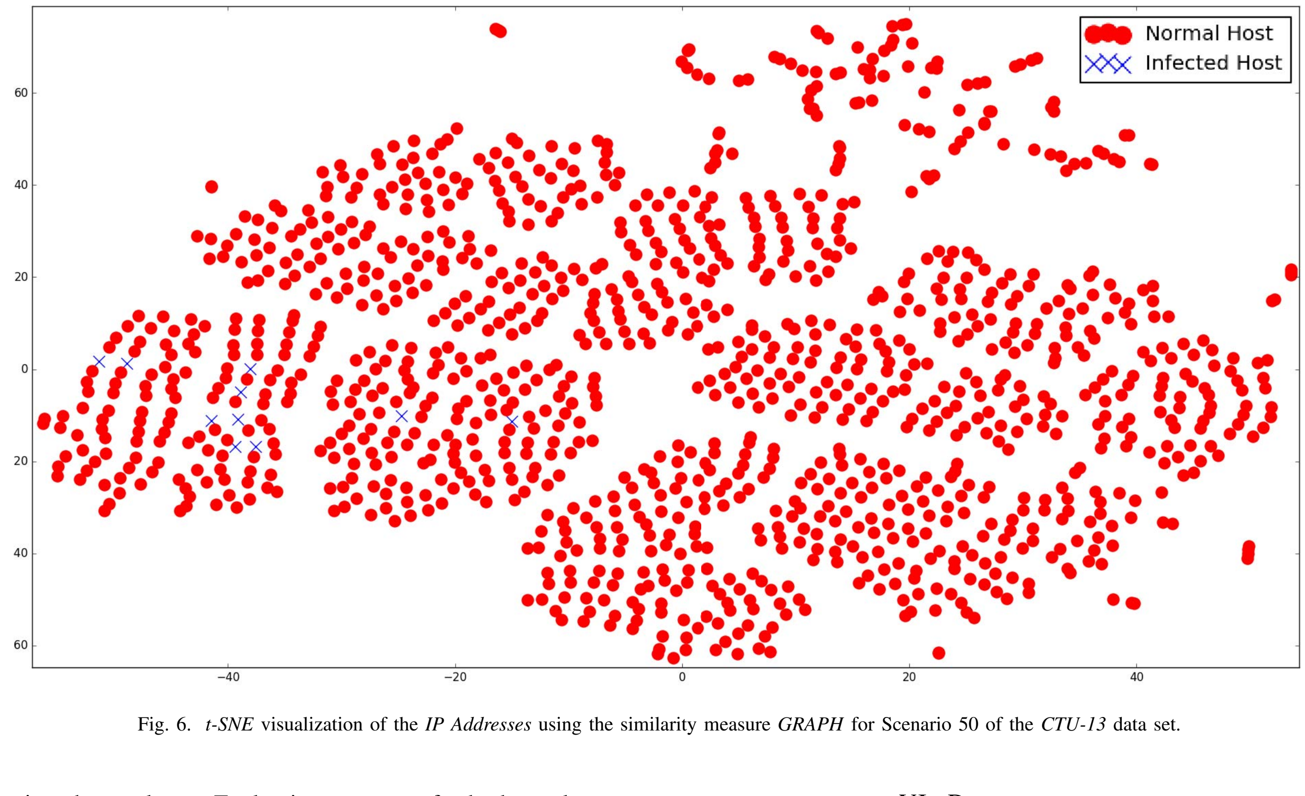

Fig. 6. *t-SNE* visualization of the *IP Addresses* using the similarity measure *GRAPH* for Scenario 50 of the *CTU-13* data set.

assigned to a cluster. Evaluation measures for both results are shown in Table VII. It shows that *DBScan* is in combination with *IP2Vec* able to generate more accurate, homogeneous and complete clusters than with *GRAPH*.

VI. Discussion

A. Experiment 1 - Identification of Botnets

In the first experiment, we evaluated *IP2Vec* and *GRAPH* for the detection of infected *IP Addresses* within a botnet data

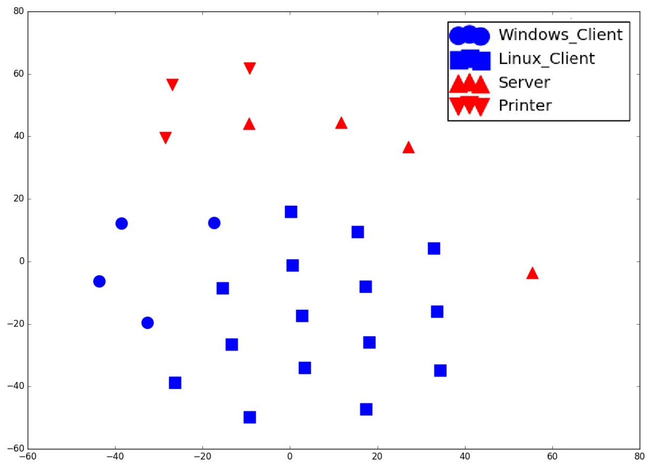

Fig. 7. *t-SNE* visualization of the *IP Addresses* using the similarity measure *IP2Vec* for the *CIDDS-001* data set.

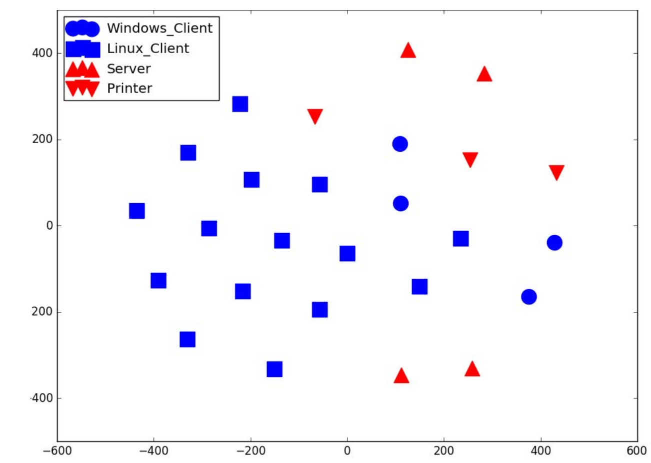

Fig. 8. *t-SNE* visualization of the *IP Addresses* using the similarity measure *GRAPH* for the *CIDDS-001* data set.

set. Infected hosts are characterized by the same activities (e.g. try to infect further hosts or communicate with their botnet master). Therefore, we assume that infected hosts are more similar to each other than normal hosts. This assumption is confirmed in Figure 5. Figure 5 shows that the infected hosts represent a group with very high similarities to each other and that they have smaller similarities to normal hosts. Using the similarities of *IP2Vec*, we are able to separate the infected hosts from normal hosts in the visualization as well as in the

TABLE V
ASSIGNMENT MATRIX FOR THE *CIDDS-001* DATA SET USING *IP2Vec* AS SIMILARITY MEASURE FOR *DBScan*.

Class	Cluster 1	Cluster 2	Num. Outliers
server	0	6	1
client	19	0	0

TABLE VI
ASSIGNMENT MATRIX FOR THE *CIDDS-001* DATA SET USING *GRAPH* AS SIMILARITY MEASURE FOR *DBScan*.

Class	Cluster 1	Cluster 2	Cluster 3
server	2	1	4
client	17	2	0

TABLE VII
COMPARISON OF *IP2Vec* AND *GRAPH* FOR CLUSTERING THE *IP Addresses* WITHIN THE *CIDDS-001* DATA SET.

Similarity Measure	Accuracy	Homogeneity	Completeness	Num. Outliers
IP2Vec	0.9615	1.0	0.8406	1
GRAPH	0.6923	0.4518	0.3434	0

clustering scenario. Since the normal behaviour of the hosts differs more, *DBScan* was not able to group all normal hosts to one cluster. In contrast to that, the infected hosts have a very high similarity and are already integrated into the same cluster at small *epsilon* values for *DBScan*.

If we use *GRAPH* as similarity measure, we are not able to separate the *IP Addresses* into normal and infected. This is observable in Figure 6. Normal and infected hosts are nearly evenly distributed in the two-dimensional representation. The reason for that is the similarity measure *GRAPH* which only considers the given network information and does not consider the behaviour of the hosts.

B. Experiment 2 - Server Client Identification

In the second experiment, we used *IP2Vec* and *GRAPH* to learn similarities between *IP Addresses* within week 3 of the *CIDDS-001* data set. Then, we evaluated similarities between *IP Addresses* through visualization and clustering.

Using the similarities of *IP2Vec*, we are able to separate clients from servers in the visualization as well as in the clustering scenario. The *CIDDS-001* data set contains seven servers: *file*, *http*, *e-mail*, *backup* and three *printers*. Since these servers offer different services, the similarities between them are smaller compared to the similarities between the clients. Consequently, *DBScan* grouped not all servers to the same cluster. The outlier in the clustering and visualization (see Figure 7) is the *file* server.

In contrast to that, using *GRAPH* as similarity measure, we are not able to separate the *IP Addresses* into two classes (server and client). The reason is that *GRAPH* uses only the given structural information and has no additional information about the behaviour of the hosts. To fulfil this task with *GRAPH* correctly, we would have to add a layer with information about clients and servers in the hierarchy of the similarity measure.

C. Advantages and Disadvantages

IP2Vec comes with advantages and disadvantages compared to previous work. The disadvantages are primarily that the behaviour of *IP Addresses* can change over time. This could lead to problems in classification settings. Assume host A is a normal client in the training data set and host A is infected with a virus in the test data set. If we include the vector representations of *IP Addresses* from *IP2Vec* in the list of classification features, then this feature would reinforce the classification of flows from host A in the test data set as normal and not infected. Therefore, vector representations of *IP Addresses* should not be used as features in classification tasks when behavioural changes of *IP Addresses* are expected. However, there are two obvious solutions to that problem. First, we could update the vector representations (embedding) using new incoming flows. Second, we could calculate new features based on the vector representations (e.g. the similarity between *Source IP Address* and *Destination IP Address* or the similarity between *Source IP Address* and *Destination Port*) instead of using them directly. If the *IP Address* changes its behaviour over time, then the *IP Address* establishes more flows with smaller similarities in above mentioned features. Especially this idea can be used to apply *IP2Vec* for change detection.

Another challenge may be the occurrence of new *IP Addresses*. For *IP Addresses* which do not occur in the training data set, no vector representations are available. A solution to that problem could be to update and enlarge the neuronal network with incoming flows or to relearn the vector representations. Another approach might be to learn a default vector representation for unknown *IP Addresses* which is calculated by means of the known *IP Addresses*.

The biggest advantage of *IP2Vec* is the transformation of *IP Addresses* to continuous vectors. These vectors can be used as input values for data mining methods and visualization techniques. In our setting, we used the features *Source IP Address*, *Destination IP Address*, *Destination Port* and *Protocol* from the given flow-based data set. This has the additional benefit that we learn not only vector representations for *IP Addresses* but also for *Ports* and *Protocols*. Thus allows us to calculate similarities between *Ports* and *IP Addresses* and so on. Consequently, we learn vector representations for all categorical features within a flow.

Another advantage of *IP2Vec* is that the selected context features can be customized according to the desired target. If someone wants to group *IP Addresses* with respect to their traffic volume (e.g. for estimating bandwidth requirements), the feature *Bytes* could be included in the set of context features.

VII. Summary

IP Addresses appear in packet- and flow-based network data. The fact that *IP Addresses* are categorical attributes without natural order complicates their treatment in data mining methods as well as their visualization.

We proposed *IP2Vec*, an unsupervised method to learn similarities between *IP Addresses*. *IP2Vec* aims to compensate the lack of inherent order within *IP Addresses* by learning similarities between them using available context information from flow-based data. Therefore, like *Word2Vec*, the proposed approach uses a skip-gram neural network architecture to train a fully connected neuronal network with a single hidden layer. After training the neuronal network, the weights from input to the hidden layer can be used as vector representations of *IP Addresses*.

Preliminary results indicate the suitability of *IP2Vec*. For the *CTU-13* data set, *IP2Vec* was able to identify infected *IP Addresses* by assigning very high similarities to them. After learning similarities within the *CIDDS-001* data set, the clustering algorithm *DBScan* was able to assign all clients to the same cluster.

In the future, we want to spend more effort in the evaluation of *IP2Vec* in order to strengthen its general suitability. However, design of appropriate evaluation scenarios is challenging. Further, we want to study the above mentioned open issues of *IP2Vec* such that we are able to use this approach for classification tasks and streaming scenarios.

References

[1] A. L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cyber security intrusion detection,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[2] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *Computers & Security*, vol. 28, no. 1, pp. 18–28, 2009.

[3] A. Patcha and J.-M. Park, “An overview of anomaly detection techniques: Existing solutions and latest technological trends,” *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[4] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 305–316.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Proceedings of Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[6] M. Baroni, G. Dinu, and G. Kruszewski, “Don’t count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors.” in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, pp. 238–247.

[7] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” *Computers & Security*, vol. 45, pp. 100–123, 2014.

[8] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, “Flow-based benchmark data sets for intrusion detection,” in *Proceedings of the 16th European Conference on Cyber Warfare and Security*. ACPI, 2017, pp. 361–369.

[9] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[10] D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, “A survey of distance and similarity measures used within network intrusion anomaly detection,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 70–91, 2015.

[11] Q. A. Tran, F. Jiang, and J. Hu, “A real-time netflow-based intrusion detection system with improved BBNN and high-frequency field programmable gate arrays,” in *Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2012, pp. 201–208.

[12] M. M. Najafabadi, T. M. Khoshgoftaar, A. Napolitano, and C. Wheelus, “Rudy attack: Detection at the network level and its important features.” in *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, 2016, pp. 288–293.

[13] M. M. Najafabadi, T. M. Khoshgoftaar, C. Calvert, and C. Kemp, “Detection of SSH brute force attacks using aggregated netflow data,” in *Proceedings of the 14th International Conference on Machine Learning and Applications*. IEEE, 2015, pp. 283–288.

[14] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, “Disclosure: Detecting botnet command and control servers through large-scale netflow analysis,” in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 129–138.

[15] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Proceedings of the 2014 IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 247–255.

[16] P. Winter, E. Hermann, and M. Zeilinger, "Inductive intrusion detection in flow-based network data using one-class support vector machines," in *Proceedings of the 4th International Conference on New Technologies, Mobility and Security*. IEEE, 2011, pp. 1–5.

[17] M. Idhammad, K. Afdel, and M. Belouch, "Dos detection method based on artificial neural networks," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 4, 2017.

[18] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning," in *IEEE International Conference on Computing, Networking and Communications*. IEEE, 2014, pp. 797–801.

[19] K. Valentín and M. Malỳ, "Network firewall using artificial neural networks," *Computing & Informatics*, vol. 32, no. 6, pp. 1312–1327, 2014.

[20] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "A toolset for intrusion and insider threat detection," in *Data Analytics and Decision Support for Cybersecurity: Trends, Methodologies and Applications*, I. Palomares, H. Kalutarage, and Y. Huang, Eds. Springer, 2017, pp. 3–31.

[21] S. Mathur, B. Coskun, and S. Balakrishnan, "Detecting hidden enemy lines in ip address space," in *Proceedings of the 2013 Workshop on New Security Paradigms*. ACM, 2013, pp. 19–30.

[22] B. Li, M. H. Gunes, G. Bebis, and J. Springer, "A supervised machine learning approach to classify host roles on line using sflow," in *Proceedings of the first edition workshop on High Performance and Programmable Networking*. ACM, 2013, pp. 53–60.

[23] J. Wang and I. C. Paschalidis, "Botnet detection based on anomaly and community detection," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 392–404, 2017.

[24] Z. Hu, J. Heidemann, and Y. Pradkin, "Towards geolocation of millions of ip addresses," in *Proceedings of the 2012 ACM Conference on Internet measurement*. ACM, 2012, pp. 123–130.

[25] H. Jiang, Y. Liu, and J. N. Matthews, "Ip geolocation estimation using neural networks with stable landmarks," in *Proceedings of the 2016 IEEE Conference on Computer Communications Workshops*. IEEE, 2016, pp. 170–175.

[26] S. E. Coull, F. Monrose, and M. Bailey, "On measuring the similarity of network hosts: Pitfalls, new metrics, and empirical analyses." in *Proceedings of the Network and Distributed System Security Symposium*, 2011, pp. 1–16.

[27] A. Jakalan, J. Gong, Q. Su, X. Hu, and A. M. Abdelgder, "Social relationship discovery of ip addresses in the managed ip networks by observing traffic at network boundary," *Computer Networks*, vol. 100, pp. 12–27, 2016.

[28] M. Ring, F. Otto, M. Becker, T. Niebler, D. Landes, and A. Hotho, "Condist: A context-driven categorical distance measure," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 251–266.

[29] M. Ring, D. Landes, and A. Hotho, "Automatic threshold calculation for the categorical distance measure condist," in *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB*, 2015, pp. 52–63.

[30] D. Ienco, R. G. Pensa, and R. Meo, "Context-based distance learning for categorical data clustering," in *Proceedings of the 8th International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 83–94.

[31] H. Jia, Y.-m. Cheung, and J. Liu, "A new distance metric for unsupervised learning of categorical data," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 5, pp. 1065–1079, 2016.

[32] E. M. Henry, "Netflow and word2vec - flow2vec," Dec 2016, (Date last accessed 01-August-2017). [Online]. Available: https://edhenry.github.io/2016/12/21/Netflow-flow2vec/

[33] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[34] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[35] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[36] C. McCormick, "Word2vec tutorial - the skip-gram model," Dec 2016, (Date last accessed 01-August-2017). [Online]. Available: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[37] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. AAAI, 1996, pp. 226–231.