Framework for comparison of network anomaly detection algorithms

FIT VUT Technical Report

Václav Bartoš, Martin Žádník

Technický report č. FIT-TR-2012-02
Fakulta informačních technologií, Vysoké učení technické v Brně

Last modified: 1.6.2012

Framework for comparison of network anomaly detection algorithms

Václav Bartoš and Martin Žádník

IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, Czech Republic
email: {ibartosv,izadnik}@fit.vutbr.cz

Abstract. This technical report describes a framework for evaluation and comparison of several network anomaly detection (NAD) algorithms. The framework addresses a need for reference implementations of various NAD algorithms as well as a need for publicly available data sets.

1 Introduction

Anomaly detection systems helps to keep track of security violations and disruptions to network and its services. Methods of anomaly detection have received great research attention and large number of methods have been proposed. But there is a lack of comprehensive comparison of existing methods and a lack of annotated data sets for their evaluation.

In this paper we describe design and implementation of a framework for comparison of existing methods. Its goal is to provide a set of tools to help implementation and evaluation of various anomaly detection methods. It also provides reference implementations of several methods. This reduces the effort when comparing detection capabilities in terms of precision and types of detected anomalies.

The framework focuses on network anomalies caused by spreading worms, Distributed Denial of Service attack or network or port scanning. These anomalies exhibit themselves by the increased number of flows which bare only few packets (one in most cases) each. Traffic caused by such anomalies also exhibits increased or decreased variability in packet header fields.

The report is organized as follows. A brief motivation for our work is given in Section 2. The concept of the framework is described in Section 3. Section 4 provides description of selected algorithms. Section 5 describes the utilized data set. The anomaly detection functions and methods are evaluated in Section 6. The report concludes with Section 7.

2 Motivation

There are many methods proposed for anomaly detection systems. But they usually vary in their properties. Some of them are specialized for detection of

particular category of intrusions, others are general. Some work upon short time scales and detect anomalies within seconds, some can detect only a long-term anomalies. Each method generates certain ratio of false alarms and misses certain portion of existing anomalies. There are also great differences in computational complexity.

Although several papers [6,8] categorize various methods according to their properties, there is a lack of experimental evaluation and direct comparison of their detection capabilities. Such evaluation is usually done in papers proposing new methods. Authors usually compare proposed method with one or two previous ones.

Unfortunately, there are no publicly available implementations of AD methods so every researcher who wants to compare his or her new algorithm to others has to implement also all other methods. This is a lot of redundant work. Moreover, many papers do not provide all the details needed for correct reimplementation, so it is possible that comparison of the same methods made by different researches may vary significantly.

Because of these reasons we consider it very helpful to have reference implementations of various anomaly detection methods, or at least a framework containing libraries and tools to simplify writing of these implementations. Such framework could also help a lot during development of new methods.

Another problem of evaluation and comparison of methods is a lack of testing data. There are only few public sources of real network data (MAWI traffic archive [2], CAIDA [1]). Researchers often use their own data (e.g. from a university network) but these data cannot be usually published due to privacy or security issues. Moreover, the data set must be annotated, i.e. all anomalies in the data set must be labeled. Unfortunately available data sets are only a plain data[1], so researchers have to annotate data themselves.

The framework addresses this issue by including several data sets which are used to test implemented methods and may be used for cross-comparison with other or new methods.

3 NADEX Framework

Because of the reasons discussed above, we have implemented a NADEX framework (Network Anomaly Detection EXperiments). This framework contains reference implementations of several well-known anomaly detection methods, but it is designed mainly to support easy implementation and evaluation of others.

With the framework a researcher is abstracted from data loading and preprocessing routines or visualization of the results. She can focus only on the core of an algorithm. It is also possible to easily reuse existing algorithms or their parts (e.g. an algorithm for detection of anomalies in timeseries can be used as a part of some more complex method).

[1] There exist annotated data sets from DARPA Intrusion Detection Evaluation 1998 and 1999, and KDD Cup 1999 data set, evidently these are very outdated now.

To enable objective evaluation and comparison of the methods, there is a large data set included. Currently, the data set is not labeled, but in future versions of the framework, the data will be annotated automatically (by combining data from several detectors) or semi-automatically (with manual checking and labeling of anomalies found by detectors). Of course, the framework is not limited to the specific data set, others data may be used as well.

Most of the framework is implemented in Python[2], because it is a powerful high-level programming language well suitable for fast prototyping of algorithms. It has also a large number of third party packages which can be used to simplify coding. The only disadvantage of Python is its lower computing performance. Because of this reason, some tools in the framework are written in C or C++.

3.1 Concept

The core of NADEX consists of many stand-alone modules. Some of the modules represent an implementation of given anomaly detection algorithm others implement supporting functions. Since various algorithms process various inputs and generate various outputs it is not possible to keep a unified structure of each module. Rather than trying to unify the anomaly detection methods, we suggest to identify those processing tasks that may be shared across the methods. Each method must load its data, it may do some kind of filtering or signal processing and it outputs the results, often in the form of graphs.

In our framework, each module loads input data from a file. Depending on the method the input data may vary greatly. Some methods expect data in the form of a signal, for example a vector representing a number of bytes received each second, others accepts data in the form of packets or flows. In order to simplify loading data the framework provides several functions to read data from text files (vectors of values), pcap files or files containing flows. Loaded data are processed within the module using other modules in the framework or third party libraries.

3.2 Loading data

There are three dedicated modules for loading input data. First module *loader* serves for loading text files containing series of values, for example numbers of bytes, packets and flows per time interval.

Module *pcapfile* allows to load data from pcap files as well as to write down data into pcap files. It may be useful also as a stand-alone module outside the framework, because it provides a way to read and write pcap files independently on the libpcap library which is not multiplatform.

Third module *flowfile* allows to load data from files containing flow records. This file can be in format of either well-known tool NfDump or it can be a NADEX flow file generated by *flowgen* tool. The *flowfile* utilizes *nfreader* to implement reading flowrecords from NfDump files.

[2] The framework is tested under Python 2.6 (but some scripts may work on older versions as well)

The *nfreader* is library providing an interface for reading NfDump files. It is based on an example from source codes of NfDump. It is written in C but there is also a Python wrapper, so it can be used to read NfDump files from both C (or C++) programs and Python scripts.

Flowgen is a tool written in C++ that generates flow records from pcap files. These records are stored in a very flexible format, called *NADEX flow file*. There are always two or three files representing a set of records – *nxfi* (info) file containing information about the contents and structure of records, *nxf* file containing the records and optionally *nxfd* file containing payload data of the flows.

The framework also implements *nf2nxf* tool which converts flow files produced by NfDump to NADEX flow file format.

3.3 Integration of third-party packages

Besides modules themselves, framework integrates also several third-party libraries.

Pandas is a package for working with timeseries. Its classes – Series, TimeSeries and DataFrame – are used by several modules of the framework. Besides effective implementation of arrays with indices, it provides various statistical and signal processing functions such as mean, median, standard deviation, rolling mean or exponentially weighted moving average ewma). It can also handle missing data very well. Such ability may be useful when processing data from a network with outages of measuring equipment. This package works well together with the next one – matplotlib.

Matplotlib is a Python plotting library. It can create various kinds of graphs in a form of an interactive window or variety of image formats. Everything is fully customizable via object-oriented interface or interface similar to Matlab functions. All detection methods in the NADEX framework use this package to visualize its results in the form of nice graphs.

Both these packages needs *numpy* – the fundamental package for scientific computing. It provides mainly an effective representation of arrays and many mathematical functions to work with them.

The last package used in the framework is *dpkt*. It is used for effective parsing of packets.

3.4 Supporting AD functions

The framework also contains specific functions and tools to simplify experimenting with new methods.

Entropy tool *Nfentropy* tool computes sample entropy upon various combinations of fields of flow records. It uses the classical Shanon's definition of entropy:

$$H(X) = -\sum_{i=1}^{N} \frac{n_i}{S} \log_2 \frac{n_i}{S}, \tag{1}$$

where $n_1, n_2, \ldots, n_N$ are number of occurrences of every unique value of the observed feature in the sample X and S is the total size of the sample. Normally, this requires to compute a histogram of values which occurred in the data. In case of large values (e.g. IP addresses) the size of the histogram might not fit in the memory. Therefore *nfentropy* utilizes a modified version which suffices with a relatively small histogram. The trick is to utilize hash of the value as an address in the histogram instead of the value itself. Since the equation 1 does not need original values to compute the entropy the only issues are hash collisions. But these collisions introduce only a small error in comparison with sub-sampling the histogram.

Nfentropy function computes entropy of:

- source IP address
- destination IP address
- source port
- destination port
- two's logarithm of number of packets in flow
- two's logarithm of number of bytes in flow
- TCP flags
- source and destination address pair
- source and destination port pair
- source address and source port pair
- source address and destination port pair
- destination address and source port pair
- destination address and destination port pair

Computing entropy of logarithm of packet and bytes allows to decrease number of distinct values. At the same time, it allows to distinguish between sizes of small flows such as one-packet flow and two-packet flow. Whereas it gives up on distinguishing between large sizes. Such a bias is deliberate and allows to alleviate the effect of small flows on the entropy.

Moreover, all these values are computed separately for these protocols:

- All protocols
- TCP
- UDP
- ICMP
- Others

Due to performance reasons, the nfentropy was implemented in C rather than in Python.

TCP scan detector *scan_detector* is a rule-based Python script designed to reveal the TCP scanning activities in pcap files. It looks for a lot of connection attempts (SYN packets) with the same source IP address. It reports only those activities where more than 90% of flows have only one packet and the number of responses from a target is less than 25%.

DDoS generator *dos_generator* is a Python script which generates pcap file containing TCP SYN flood with random source addresses and ports.

Other scripts There are also several simple bash scripts to help working with large datasets. These include *volumestats.sh* and *entropy.sh* which can be used to obtain timeseries of the traffic volume or entropy from a set of NetFlow files. It traverses directory structure and calls *nfdump -I* or *nfentropy* to all specified files and transforms output to a format readable by *load_series* function in the framework. These scripts must be modified before use, to customize it to a particular directory structure and file naming conventions.

There is also a similar script *astute.sh* which calls the ASTUTE detection method (see Chap. 4.1) to all files in specified directories.

Next, there is a script *download_mawi.sh* which helps to download data from MAWI traffic archive. Lastly, *pcapcat.sh* can effectively concatenate pcap files.

4 Anomaly Detection Algorithms

The framework contains three AD algorithms proposed by other authors and a prototype of our own algorithm. More methods are being implemented and will be included in future releases.

We try to implement existing algorithms exactly as they are described in the papers. Therefore, the modules are meant not only as an example how to write algorithms in the framework but also as reference implementations of the methods which can be used for comparison.

4.1 ASTUTE

A Short-Timescale Uncorrelated-Traffic Equilibrium (ASTUTE) [7] is a detector based on an assumption that each flow changes independently on others at a sufficiently aggregated and unsaturated link. In case this assumption does not hold then the underlying traffic is considered to be caused by an attack or suspicious activity. The detector operates at fixed time intervals. Volumes of individual flows are measured for each interval. At the end of each interval, the difference to a previous interval is calculated for each flow. The resulting differences form input data for detecting an anomaly. The standard deviation and the confidence interval is computed from the differences. Anomaly is reported if the confidence interval is biased so that it does not contain zero. The detection is done on 6 levels of flow aggregation (5-tuple, IP address pair and source and destination addresses and ports). The fact, that anomaly may occur only on some levels of aggregation, provides additional information about detected anomaly which allows to a certain point automatic backward annotation.

The computational complexity of ASTUTE is quite low. The only issue is a storage of two consecutive values per each active flow. The implementation in the framework utilizes dictionaries to index these values. It takes approx. 1 minute to process a pair of files with a million flows each on a modern PC. Since

this method usually operates on time intervals of 5-minutes the performance of such implementation in Python is sufficient.

4.2 Wavelet

Wavelet method is strongly based on signal analysis of input data. In the paper [3] authors of Wavelet method report good results on detection of four classes of network traffic anomalies: outages, flash crowds, attacks and measurement failures. The wavelet method was proposed to find anomalies in timeseries of traffic volume, but it can be used with series of any numerical values.

The wavelet filters are used to distinguish between ambient and anomalous traffic. As a basis for wavelet authors suggest to use a pseudo-spline filter tuned at specific aggregation levels. First the data are decomposed into different strata. In the second step, the strata are divided into three groups representing different frequencies and these groups are synthesized back to signals. High, mid and low frequency signals are obtained. Then, local variance of low and mid signals is computed. If the weighted sum of these variances (*deviation score*) exceeds the threshold an anomaly is reported. Such technique allows to expose anomalies even in the presence of a large amount of legitimate traffic.

4.3 EWMA

Next method for detection of anomalies in timeseries is based on computation of *exponentially weighted moving average* (EWMA). This average is defined recursively for each time interval as:

$$E_t = \alpha x_t + (1 - \alpha) E_{t-1} \tag{2}$$

where x_t is input value at time t and α is a coefficient defining weight of the last value.

For anomaly detection purposes, the current EWMA is always used as a prediction of next value. If a difference between predicted and actual value (prediction error) exceeds a threshold, the value is considered anomalous. The threshold is computed using exponentially weighted moving variance of the prediction error, so it is low if the signal is relatively stable, while it is much larger if the signal is noisy.

So, for signal x at time t, the prediction of next value, y_t, is computed as:

$$y_t = \alpha x_t + (1 - \alpha) y_{t-1} \tag{3}$$

A prediction error e_t and its moving variance ϕ_t^2 are given by:

$$e_t = x_t + y_{t-1} \tag{4}$$

$$\sigma_t^2 = \beta e^2 + (1 - \beta) \sigma_{t-1}^2 \tag{5}$$

To be considered normal, a value at time t must lie between *upper and lower control limits*, UCL and LCL.

$$UCL_t = y_{t-1} + T * \sigma_{t-1} \tag{6}$$
$$LCL_t = y_{t-1} - T * \sigma_{t-1} \tag{7}$$

Initialization values y_0 and σ_0 are computed during a short training period in the beginning. For more details see [5].

4.4 Fast Exporter Anomaly Detection

The Fast Exporter Anomaly Detection (further referred to as FAST) is designed with the aim to be as simple as possible so it can be installed directly in the probes on the network. Its idea is to observe behavior of the probe's flow cache. Namely the number of new flows per short time interval. An alarm is triggered, if the measured number of new flows differs significantly from a predicted value. The predicted value is an output of specific predictor which must be simple enough to run on the probe yet robust to withstand certain unpredictability of network traffic.

The predictor is based on EWMA and Holt-Winters forecasting methods. Predicted value is a sum of two components – *base* and *seasonal*. Base component is exponentially weighted moving average (EWMA) of previous values, seasonal component is included to cancel out periodic daily deviations. For each time in a day it is computed as EWMA of values at the same time of previous days. Prediction y_{t+1} computed at time t is given as follows:

$$b_{t+1} = \alpha(x_t - s_{t-L+1}) + (1-\alpha)b_t \tag{8}$$
$$s_{t+1} = \gamma(x_t - b_t) + (1-\gamma)s_{t-L+1} \tag{9}$$
$$y_{t+1} = b_{t+1} + s_{t+1} \tag{10}$$

where x_t is an input value at time t, L is length of a season and α and γ are parameters of the method.

Normally, it is necessary to store last L values of s_t, but as we are using short time intervals (5 to 30 seconds) and long seasons (day or week), L may be very large. In order to save memory and also to smooth noise in the data, not all these values are stored. Instead, only an average of N samples (e.g. for N corresponding to half an hour) is stored and s_t is linearly interpolated from these values.

Initialization values b_0 and $\{s_0 \dots s_L\}$ are computed during a training phase of length of one season (L samples).

In normal conditions, difference of prediction and actual value (prediction error) should be small. Large error signifies an unexpected change in the time series, i.e. an anomaly. Prediction error e_t is given by:

$$e_t = y_t - x_t \tag{11}$$

Last Q values of e_t are stored and 0.05 and 0.95 quantiles of these values are used as lower and upper limits, (UCL, LCL) respectively.

When the input value exceeds the limit, an anomaly is not reported immediately. Instead, a CUSUM method is used. Exceeding values are summed up and the anomaly is reported only when the sum exceeds a threshold. The threshold T is computed as:

$$T = (UCL - LCL)S \tag{12}$$

where S is a sensitivity parameter. When error drops back below limit, the sum is set to zero.

5 Data sets

Besides the functions and methods, the framework includes also its own data set. The data set is provided separately from the framework package due to its large volume. The data is obtained from two ten gigabit access points. These access points serves as a connection of Brno University of Technology into CESNET backbone network. The traffic mix is generated by more than 15000 users from academic and campus network. The source of data are INVEA-TECH FlowMon probes. These probes are capable of generating NetFlow about passing traffic (both directions of the link are monitored without sampling or packet loss). The provided raw data constitutes of anonymized NetFlow records in the form of compressed nfdump files. The anonymization procedure is the default prefix-preserving method implemented in nfdump.

The provided data set was collected from 01.09.2011 till 29.02.2012 continuously. Each day consumes 10 to 30 GB of disk space. Due to the size we make publicly available only one day out of the whole data set. The rest may be obtained individually (please send email to ibartosv@fit.vutbr.cz).

Further, precomputed time series are provided from the whole measurement period. This includes volumes of flows, packets and bytes per 5 minutes interval differentiated by a protocol (TCP, UDP, ICMP). Additionally, the data set is extended with entropy series computed upon various dimensions as listed in Section 3.4 Entropy tool.

Figures 1, 2 display the number of bytes and flows, respectively, in the presented sample. The graph follows daily as well as weekly pattern. During the night the activity (number of users and volume of traffic) is low as well as during weekends whereas during working hours the activity is at its peak.

6 Evaluation

Presented functions and anomaly detection methods were applied on a sample of data set provided with the framework. We select one week long sample (from 13.2.2012 to 19.2.2012) which is sufficiently long for demonstration of behavior of all the functions and methods. In general, the evaluation shows various behavior of signals produced by the functions and gives a notion how these signals may

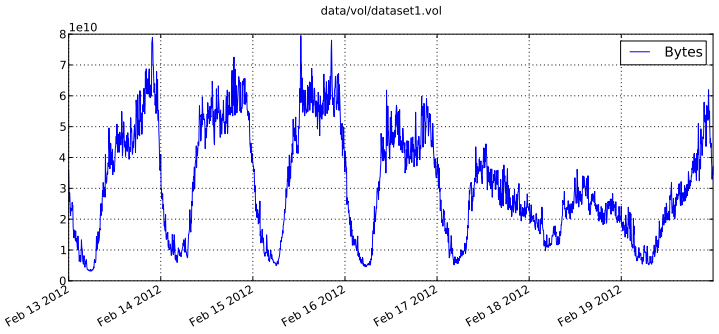

Fig. 1. Number of bytes per five minute in the a week sample of the data set (interval from 13.2.2012 till 19.2.2012).

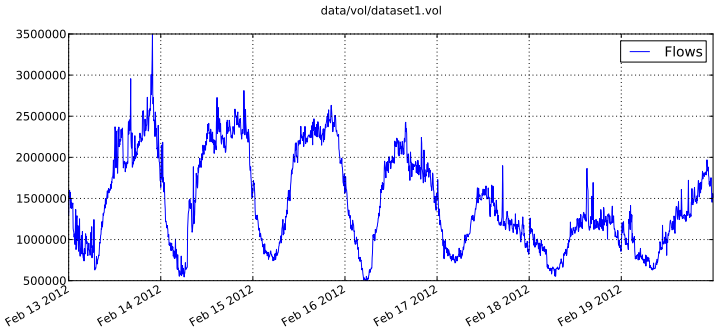

Fig. 2. Number of flows per five minute in the a week sample of the data set (interval from 13.2.2012 till 19.2.2012).

be used for detection of anomalies. Figures are presented in the same order in which the functions and methods are described in Section 3.4. First, the output of nfentropy tool is plotted in Figure 3.

The nfentropy produces several signals, one for each dimension. Only some of them are plotted here. It is clearly visible that there are groups of signals which are strongly correlated. One group is formed by signals of IP addresses and ports, another is formed by size of flows in terms of their volume measured in packets and bytes, and a special category is formed by a signal representing entropy of accumulated TCP flags per flow. The anomaly detection based on the entropies may be based on Principal Component Analysis as suggested by Lakhina [4] or on searching of large deviance from the base trend of the signal. If the deviation is detected, the decrease or increase of various entropy signals may provide additional information necessary for recognition of an anomaly type.

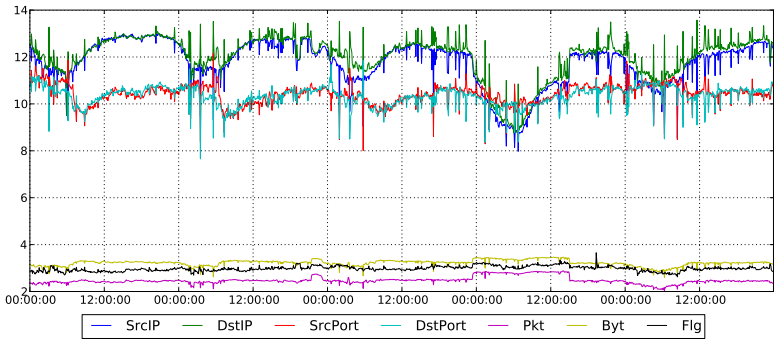

Fig. 3. Example of plotted output of nfentropy function.

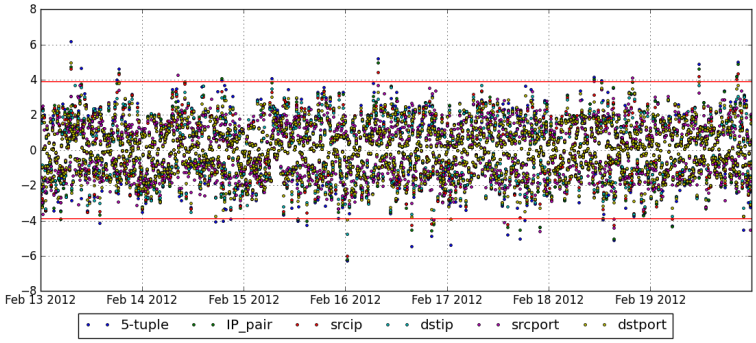

Fig. 4. Example of output from ASTUTE method.

Figure 4 displays the output of ASTUTE method. The output is formed by points as there is no relation among subsequent values. These points represent the degree of mutual correlation between flows when considering various aggregation schemes. The further the point is from zero on Y-axis the more flows are correlated and the more likely it is an anomaly. Setting up the threshold values is subject of balance between reported false negatives and false positives. The threshold denoted in the figure by red line corresponds to theoretical false positive rate of 0.01%.

Figure 5 displays the output of Wavelet method applied on the number flows per 5 minute interval. The red signal represents the deviation score which is a subject for thresholding when the anomaly should be detected. If a specific type of traffic is selected and analyzed then the detection might be more detailed and precise. Figure 6 depicts analysis of ICMP traffic only. It is clearly visible sudden change of the deviation score for ICMP anomaly at the time where no anomaly is detected in case of an analysis of all traffic, depicted in previous Figure 5.

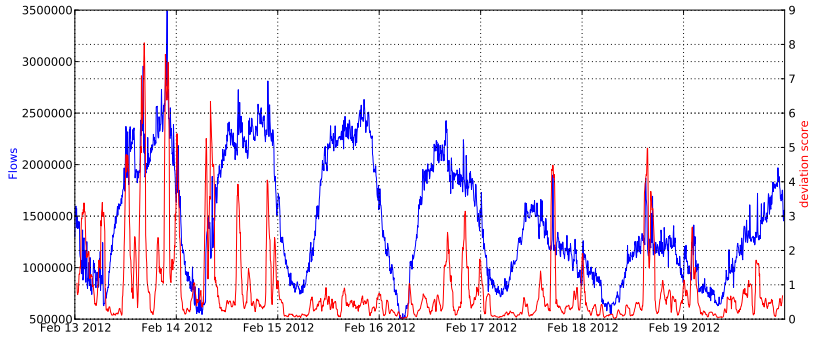

Fig. 5. Example of Wavelet deviation score computed upon all flows.

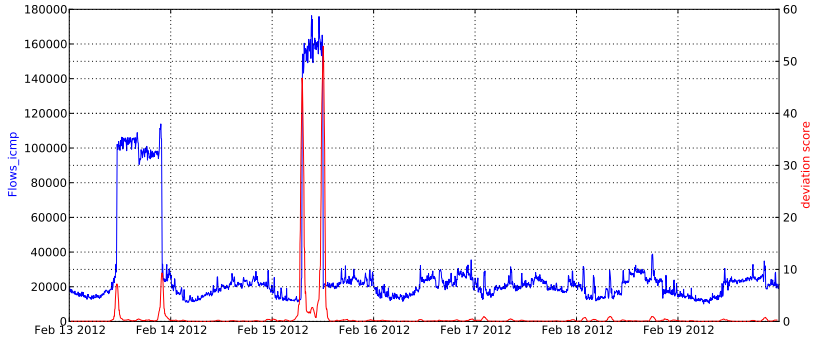

Fig. 6. Example of Wavelet deviation score computed upon ICMP only flows.

Figure 7 displays the output of EWMA method applied on the number flows per 5 minutes interval. The output constitutes of several signals. The signals represent the actual (blue), predicted (gray) and threshold values (yellow). If the threshold is exceeded by the actual an anomaly is detected. The anomalies are depicted by a star in the graph.

Again, ICMP traffic is analyzed separately. Figure 8 depicts the outcome. Similarly to Wavelet, the method detects sudden increase or decrease in the number of flows or other dimensions. An anomaly is reported several times until the limits adapts on the on-going traffic. The anomaly traffic is then considered as a normal one. Therefore when the anomaly ends an anomalous event is reported again. This redundant reporting could be mitigated by further post-processing. Minor anomalies (which might not be anomalies at all, i.e., false positives) might be ignored in dependence on the limits.

Further, we present the results of our own method designed specifically for fast exporter anomaly detection. Figure 9 displays the output of FAST method

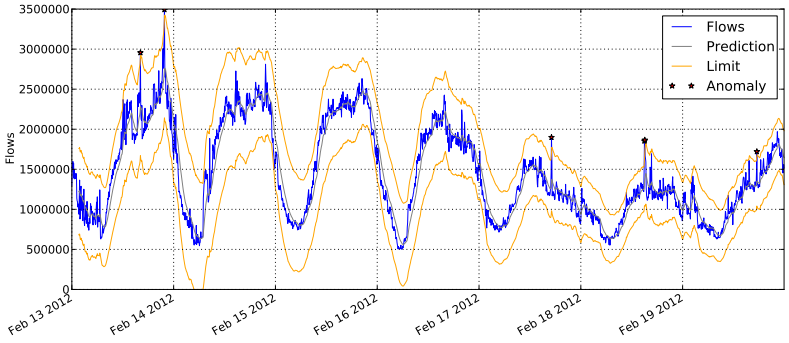

Fig. 7. Example of EWMA output for all flows.

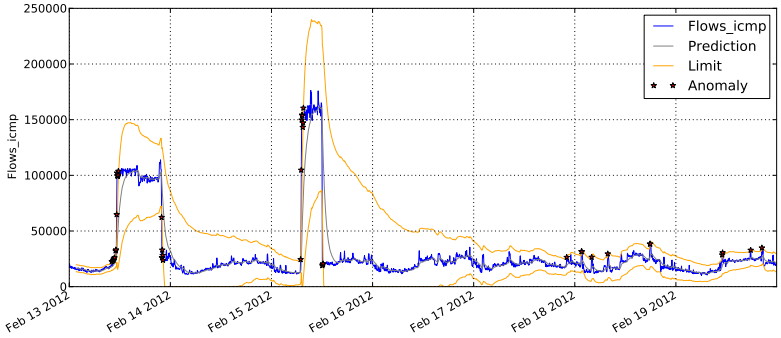

Fig. 8. Example of EWMA output for ICMP flows only.

applied on the number of flows per 5 second interval. Due to the analysis of such a short interval it is possible to observe anomalies which would otherwise vanish if longer interval is used. The outputs plotted on the Figure 9 depicts various signals such as base (black), seasonal (gray), limits (green). An anomaly is reported (red star) when the CUSUM (orange) exceeds the CUSUM threshold (dashed orange).

It is interesting to observe behavior of the base signal. If the input signal was perfectly periodic then the base would be zero as the seasonal signal would perfectly predict the next input value. Since the input signal is not periodic the base compensates for the error of the seasonal.

Figure 10 shows a detailed view of an anomaly. When value exceeds upper limit, CUSUM (orange dots) is increased but anomaly is not reported yet. It is reported at the time when CUSUM exceeds the threshold (dashed orange line). Start of the anomaly is then determined as a time when value exceeded the limit for the first time. Yellow and red stars label anomalous values. Red star labels

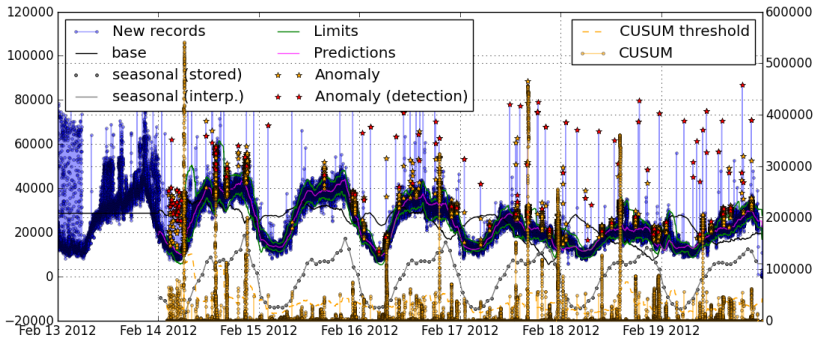

Fig. 9. Example of FAST output for all flows.

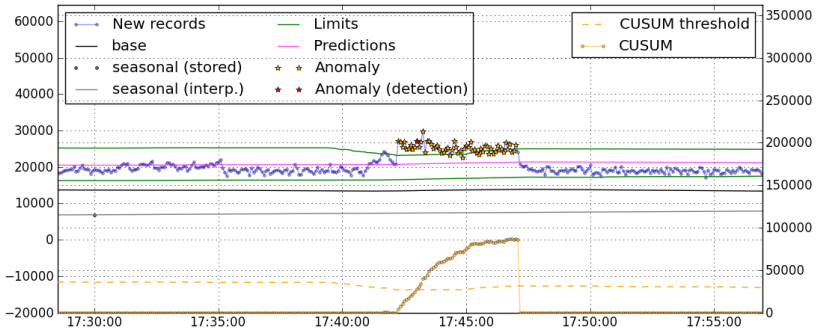

Fig. 10. Detailed view of one anomaly detected by FAST method.

the time when the anomaly is reported for the first time while yellow star labels detection of an anomaly.

To sum up, each method provides different output with varying behavior. For example, while ASTUTE reports several anomalies between 16^{th} and 17^{th} February other methods working upon 5 minute intervals does not detect any anomaly during this period. FAST detects large amount of anomalies in comparison to other methods. We have manually checked some of them and confirmed that most of them are true positives. Other methods do not detect these anomalies as they work upon a longer interval.

7 Conclusions

This paper described the framework for experimenting with network anomaly detection methods. Its aim was to address several issues related to experimenting with current methods, designing new ones and comparison to others. To this end, the framework consists of various functions and methods used for anomaly detection. A sample of traffic from BUT network was utilized to compare outputs of various methods. The utilized traffic sample may be obtained upon notice.

Our future plan is to further extend this framework with new methods and build a system for automated or semi-automated annotation.

Acknowledgments

This work was supported by the research programme MSM 0021630528 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070, by the Research Plan No. MSM, 0021630528 - Security-Oriented Research in Information Technology, the grant BUT FIT-S-11-1 and TeamIT project CZ.1.07/2.3.0067.

References

1. The CAIDA web page.
   URL `http://www.caida.org`
2. MAWI Traffic Archive.
   URL `http://mawi.wide.ad.jp/mawi/`
3. Barford, P.; Kline, J.; Plonka, D.; aj.: A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, New York, NY, USA: ACM, 2002, ISBN 1-58113-603-X, s. 71–82.
4. Lakhina, A.; Crovella, M.; Diot, C.: Mining Anomalies Using Traffic Feature Distributions. *ACM SIGCOMM Comp. Com. Rev.*, ročník 35, č. 4, Oct. 2005.
5. Nong Ye, Y. Z., Connie Borror: EWMA techniques for computer intrusion detection through anomalous changes in event intensity. *Qual. Reliab. Engng. Int.*, ročník 18, 2002: s. 443–451.
6. Patcha, A.; Park, J.-M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, ročník 51, August 2007, ISSN 1389-1286.
7. Silveira, F.; Diot, C.; Taft, N.; aj.: ASTUTE: detecting a different class of traffic anomalies. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, New York, NY, USA: ACM, 2010, ISBN 978-1-4503-0201-2, s. 267–278.
8. Zhang, W.; Yang, Q.; Geng, Y.: A Survey of Anomaly Detection Methods in Networks. In *International Symposium on Computer Network and Multimedia Technology, CNMT 2009.*, January 2009, ISBN 978-1-4244-5272-9, doi:10.1109/CNMT.2009.5374676.