TCP Anomaly Detection using Support Vector Machines

Frank Mascarich
Department of Computer Science & Engineering
University of Nevada, Reno
Reno, NV 89557

***Abstract*—In this work, a fast and efficient method is developed for identifying TCP anomalies. The method uses tcp metrics output from a tool called TSTAT and a support vector machine to classify instances of tcp metrics as normal or anomalous. The classifier is also capable of identifying the class of network conditions causing the anomaly, distinguishing between packet loss, packet corruption, packet dropping, and packet delay. Using data collected from tcp flows between two DTN servers, the classifier is trained, and is able to correctly identify anomalous flow instances at a rate of 98%. All datasets and source code, both used to prune the data and to train the classifier, is made available.**

I. Introduction

Traffic monitoring and the detection of unusual network behavior are vital tasks for any network. Passive monitoring is especially appealing as network resources are minimally impacted, and such a monitor can be placed between two actively used networks preventing significant interference. However, TCP anomaly detection can be challenging given the complexities of the TCP protocol, and its own self-correcting behaviors. The objective of this work is to develop a fast, lightweight classifier, capable of detecting anomalous TCP behavior, and of identifying the specific behavior causing the anomaly. For this purpose a Support Vector Machine (SVM) is trained on data transfer datasets collected between two DTN servers under various predetermined conditions. The results of this work show that the classifier is capable of correctly detecting anomalous behavior at a rate of 95%, and is able to correctly identify the type of anomalous behavior at a rate of 80%.

The remainder of the paper is structured as follows: Section III describes the tools used to generate normal and anomalous TCP data, and Section IV describes the usage and configuration of the TSTAT TCP metric tool. Section V gives an overview of the steps taken to prune and clean TCP metric data before SVM training or prediction. Section VI briefly outlines the steps taken to train the SVM, Section VII describes the experimental conditions for each anomalous behavior, and finally Section VIII describes the results of the experiments and discusses the outcome.

II. Related Work

TCP anomaly detection is a well studied problem, and as TCP continues to be the dominant protocol throughout computing networks, the identification of anomalies will continue to be an important area of research. The authors in [1] develop techniques based on loss and delay metrics observed at end hosts to determine if separate flows are congested at the same network resource, and present simulation results for cooperative congestion control. They successfully demonstrate that their delay-based technique can identify congestion on a specific shared resource. The authors in [2] utilize passive observation of end-to-end client-server traffic to identify lossy-links, or particular network resources that are causing loss and reordering. Their method achieves a 5% false positive rate.

The authors in [3] developed a technique for finding the root-cause links of losses or reordering after concluding that a large proportion of delay and losses stem from a single congestion point. Their method in simulation shows a 95% detection rate with a 10% false detection rate. Additionally, [4] uses a heuristic around the sequence number of estimate loss ratios passively and again attempts to detect lossy-links. The authors in [5] developed an optimal loss detection strategy based on a flow's RTT, and then applied the strategy into an analytic performance model. The authors claim to be able to improve TCP throughput by as much as 25% using their model.

In [6], the work presents a heuristic classification technique and are able to specifically identify the class of anomaly the flow is exhibiting. Interestingly, the authors note that the variation of the daily load at various measurement point had little affect on the occurrence of anomalies. The authors [7] use data from a single router to identify the key metrics which correlate to various types of anomaly. Specifically, instead of using the standard PCA process of correlating across spatial measurements, the authors find correlations across data metrics, resulting in improved PCA results which also consider temporal correlation. Finally, the authors in [8] developed a "streaming" PCA algorithm for anomaly detection in a distributed fashion and claim to reduce the typical time complexity of anomaly detection from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$.

This work proposes a method for anomaly detection and classification based on SVMs. This SVM correctly classifies TCP flows as anomalous at a rate of 98% in addition to classifying the type of the anomaly. Furthermore, classifying new TCP flows against the support vectors of the SVM is computationally efficient, requiring few resources in systems which are already of high demand.

III. Data Anomaly Generation Tools

In this work, three commonly used tools were utilized to generate TCP flow data with known parameters. First, "iptables" is a utility program which allows administrators to apply a set of rules to incoming and outgoing connections. It is commonly used as a firewall to prevent or otherwise filter incoming and outgoing connections. However, iptables also provides a statistics module which allows the user to apply a narrow set of conditions on incoming and outgoing packet flows. In this work, iptable's statistics package was used to implement packet loss at random points in the communication flow, effectively dropping a fixed proportion of the packets. In Appendix **??**, the exact rules used for each of the packet loss experiments are given, as well as a shell script which incorporates a series of such commands to facilitating data collection at a number of different packet loss levels.

The second commonly used tool to generate such datasets was "netem" or network emulator. In addition to packet loss features, netem can add delay to packets, with the delay values randomized, but following a normal distribution. Netem can also be used to corrupt packets, in which case a determined proportion of packets are sent with corrupted bits. Finally, netem can be used to duplicate packets. Using a combination of these emulated network behavior settings, sandbox environments can be configured to replicate the behavior of real-world, complex networks. Once again, Appendix **??** provides exact details with respect to the precise commands used to generate the data used in this work.

The first two tools were used to shape traffic and add anomalous behavior to otherwise standard TCP data flows. The final tool, iPerf3 was used to generate the actual TCP traffic between the server and the client. iPerf3 is a commonly used bandwidth measurement tool which generates high bandwidth TCP data transfers between two machines. iPerf3 was used in its default configuration, without any parallelization or bandwidth limiting features being employed.

IV. Data Collection Tools

Given TCP communication flows subject to the anomalous behavior described in Section III, a network packet analyzer named the TCP STatistic and Analysis Tool (TSTAT) was used to populate metrics with respect to each communication flow. TSTAT is a passive network sniffer which collects low-level metrics on TCP flow information for a variety of applications ranging from trouble-shooting and load balancing to usage profiling. TSTAT can run either in real-time, collecting packets from an interface, or can run in post-processing over various TCP packet dump formats. TSTAT maintains data regarding a TCP connection status by analyzing packet headers in both the forward and backward packet flows and allows it to collect data such as the congestion window size, the number of duplicated and/or out-of-sequence segments, and many other metrics at the transport layer critical to analyzing TCP traffic flows. In this work only the completed TCP connection data was captured and analyzed, a configuration file was provided to TSTAT to ensure that only packets between the source and destination servers would be tracked for analysis.

V. TCP Data Pre-Processing

In all, TSTAT collects more than 100 metrics with respect to each TCP interaction. However, in this work, a subset of the metrics were collected and utilized, a brief summary of which is provided in Table A. Following data collection, any client data points measured in terms of packets were normalized to the total number of packets received by the client, and any server data points measured in terms of packets were normalized to the total number of packets received by the server. The same normalization process was repeated with metrics which were measured in bytes, with respect to the total number of bytes received by the server and the client. This normalization process was performed to remove the affects of scale with respect to bandwidth, and render all such scale metrics relative to the total data size. Finally all data metrics were normalized using Scikit Learn's scale function rendering each metric to have a mean of zero and a variance of one.

VI. SVM Training

Using the data gathered from each of the experimental runs detailed in Table I, a multi-class SVM was trained on the normalized, post processed data. Training data consisted of approximately 1200 samples of TSTAT output for each class. Due to small variations in the data collection time, several classes had significantly more training data. Each record is comprised of 91 TSTAT metrics, each of which is detailed in Table A. Scikit-Learn's multiclass SVM was utilized, using a linear kernel and a gamma parameter determined automatically. The SVM was permitted to continue training until the optimizer's default stop condition was reached.

VII. Experiment Description

In this work a number of TCP anomalies were introduced into otherwise standard TCP data flows. Table I describes each of the individual data collection runs. To summarize Table I, packet loss was analyzed at three different levels ranging from 0.1% to 1.0%, packet delay data was collected at four different levels each with 20ms of variability, packet duplication was studied at two levels of 0.1%, 1.0% and 2.0%, and finally, packet corruption was studied at three different levels of 0.1%, 0.5% and 1.0% corruption. These levels were chosen to demonstrate a wide variance of possible networking conditions.

VIII. Experimental Results

From the collected data, 80% of the samples were used to train the SVM, and the remainder was used for testing. The results of the test data classification can be found in Figure 1.

From Figure 1, normal data flows are classified as normal 98% of the time. Figure 1 shows classification results based on not only the type of anomaly, but also the degree of the anomaly. Ignoring the degree of anomaly, and only classifying samples by their anomaly type, the classifier achieves 96%

TABLE I
LIST OF EXPERIMENT COMMANDS

#	Label	Description
1	Normal	Normal TCP Data Flow
2	0.1% Loss	iptables drops 0.1% of packets
3	0.5% Loss	iptables drops 0.5% of packets
4	1.0% Loss	iptables drops 1.0% of packets
5	1ms Delay	netem adds 1ms of delay to packets Gaussian noise centered at 20ms
6	5ms Delay	netem adds 5ms of delay to packets Gaussian noise centered at 20ms
7	10ms Delay	netem adds 10ms of delay to packets Gaussian noise centered at 20ms
8	25ms Delay	netem adds 25ms of delay to packets Gaussian noise centered at 20ms
9	0.1% Duplicate	netem duplicates packets at 1%
10	1.0% Duplicate	netem duplicates packets at 1%
11	2.0% Duplicate	netem duplicates packets at 2%
12	0.1% Corrupt	netem corrupts packets at 0.1%
13	0.5% Corrupt	netem corrupts packets at 0.5%
14	1.0% Corrupt	netem corrupts packets at 1.0%

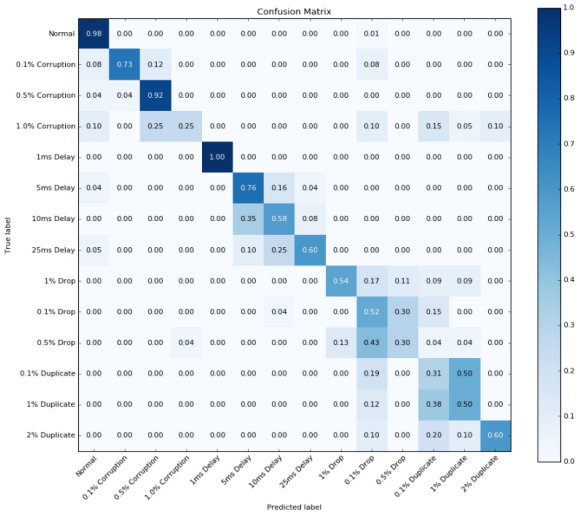

Fig. 1. Confusion matrix showing classification result of normal data against all types of anomalous data.

accuracy across all anomaly classes. However for the purposes of anomaly detection, Figure 2 presents the classification of normal TCP flow data against all others. As is visible from Figure 2, the classifier results in a 2% false positive rate, and a 2% false negative rate, meaning that the classifier detected anomalous behavior 98% of the time. Section IX will discuss in detail how future work may improve this detection rate to better detect true anomalies.

SVMs not only classify samples into their respective classes, but also allow researchers to explore the support vectors used in the classification task and give insight into which data fields of the samples are the most significant towards explaining variance between classes.

From Figure 3, the normal class is most dependent on the serv_last, s_cwin_max, and the c_pkt_unk data fields to identify normal data from anomalous data. serv_last describes the amount of milliseconds between the last segment to arrive at the server with a payload and the first flow segment. As

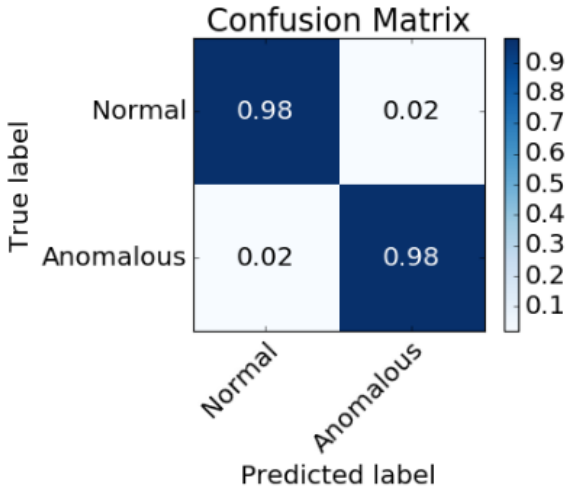

Fig. 2. Confusion matrix showing classification result of normal data against anomalous data.

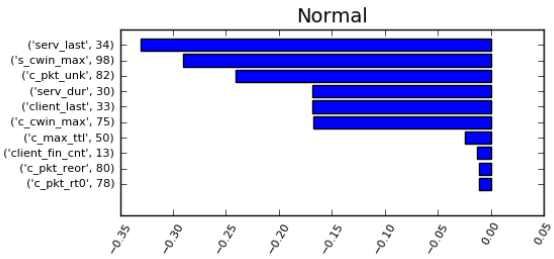

Fig. 3. Confusion matrix showing classification result of normal data against anomalous data.

stated above in Section V, this value is normalized the the total length of the TCP flow. This result indicates that normal TCP flows tend to be shorter, given that the weight of this field coefficient is negative. s_cwin_max indicates the largest congestion window size, and c_pkt_unk indicates the number of packets which are not in sequence or are duplicates. These metrics indicate that normal TCP flows have larger congestion windows and small number of out-of-sequence or duplicate packets.

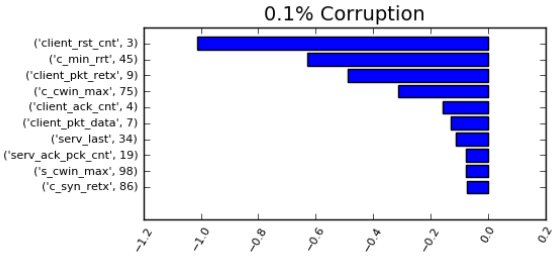

Fig. 4. Confusion matrix showing classification result of normal data against anomalous data.

Figure 4 depicts the support vector coefficients for the 0.1% packet corruption class. As is visible from the figure, this class most heavily depends on client_rst_cnt, c_min_rrt, and client_pkt_retx data fields. client_rst_cnt and client_pkt_retx both are sensible data fields for corrupt data flows, both of which should increase in the presence of corrupted packets.

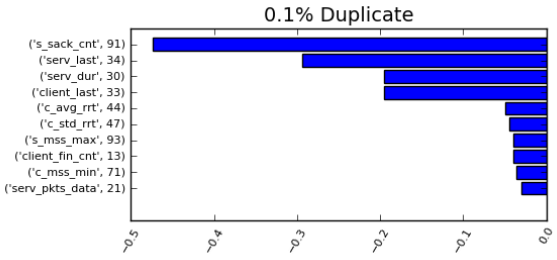

Fig. 5. Confusion matrix showing classification result of normal data against anomalous data.

From Figure 5, s_sack_cnt is the dominating coefficient for identifying samples from the duplicated packet class. The strength of this data field is no immediately intuitive, there should not be any reason for increased numbers of selective acknowledgement packets in this class relative to others.

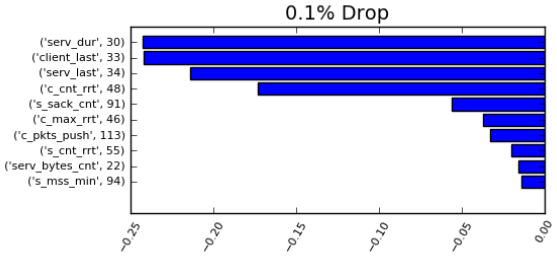

Fig. 6. Confusion matrix showing classification result of normal data against anomalous data.

Finally, from Figure 6, the most significant data fields are concerned with the duration that the flow took, and the ̈last ̈fields, which indicate the time difference between the start of the flow and the last payload packet. This result is intuitive, flows with dropped packets should dramatically increase the duration of the flow.

IX. Conclusion

This work presented a method for detecting anomalous TCP behavior, using a support vector machine trained with TCP metrics recorded using TSTAT. The results demonstrate that anomalous behavior can be quickly and reliably detected using the limited number of metrics passively collected by TSTAT. Furthermore, the results demonstrate that not only can anomalous behavior be detected with a support vector machine, but, if trained with the appropriately labeled data, can also identify the type of anomalous behavior. These results could be further improved in two ways. First, evaluating the collected data and the trained system on a variety of networks would demonstrate the robustness and generalizability of the solution to a variety of network conditions. Second, combining the SVM output with a time-based filter would greatly improve the accuracy of detection, and likely filter out the remaining false positives and false negatives. In the interest of furthering research on this topic, all datasets are made available for public use and may be found at: [https://drive.google.com/drive/folders/1BUhfbkFcREfAkWwAQQMUbxN8c1fqv5vq?usp=sharing].

References

[1] D. Rubenstein, J.F. Kurose, D.F. Towsley *Detecting shared congestion of flows via end-to-end measurement*. IEEE/ACM Transactions on Networking, 10 (3) (2002), p. 381395

[2] V. Padmanabhan, L. Qiu, H. Wang *Server-based inference of Internet performance*. IEEE INFOCOM03, San Francisco, CA, USA, April 2003.

[3] E. Brosh, G. Lubetzky-Sharon, Y. Shavitt *Spatialtemporal analysis of passive TCP measurements*. IEEE INFOCOM05, Miami, FL, USA, March 2005.

[4] P. Benko, A. Veres *A passive method for estimating end-to-end TCP packet loss*. IEEE GLOBECOM02, Taipei, TW, November 2002.

[5] N. Fonseca, M. Crovella *Bayesian packet loss detection for TCP*. IEEE INFOCOM05, Miami, FL, USA, March 2005.

[6] M. Mellia, M. Meo, L. Muscariello *TCP anomalies: identification and analysis*. Tyrrenian International Workshop on Digital Communication TIWDC, Sorrento, Italy, July, 46, 2005.

[7] Daniela Brauckhoff, Kav Salamatian, Martin May *Applying PCA for Traffic Anomaly Detection: Problems and Solutions*. Proceeding of IEEE INFOCOM 2009, Apr 2009, Rio de Janeiro, Brazil. pp.2866-2870, ff10.1109/INFCOM.2009.5062248

[8] Yang Liu, Linfeng Zhang and Yong Guan *Sketch-based Streaming PCA Algorithm for Network-wide Traffic Anomaly Detection*. 2010 International Conference on Distributed Computing Systems

Appendix

TABLE II
LIST OF EXPERIMENT COMMANDS

tstat -l -i ens3f0 -s <data_output_path>	Start tstat live capture on interface with prescribed output directory
iperf3 -c <Server_IP >-t 10	Start iPerf3 TCP Bandwidth test for a duration of 10 seconds
iperf3 -s	Start iPerf3 Server
sudo iptables -A INPUT -m statistic mode random probability 0.005 -j DROP -p tcp src <	Drop packets with probability of 0.5%
tc qdisc change dev ens3f0 root netem delay 25ms 20ms distribution normal	Add 25ms of packet delay to interface with normally distributed noise
tc qdisc change dev eth0 root netem duplicate 1%	Add duplicate packets at 1%
tc qdisc change dev eth0 root netem corrupt 0.1%	Corrupt packets at 0.1%

TABLE III
SELECTED TSTAT DATA METRICS TABLE

Metric Name	Data Type	Metric Description
Client/Server IP addr	-	IP addresses of the client/server
Client/Server TCP port	-	TCP port addresses for the client/server
packets	-	total number of packets observed form the client/server
RST sent	0/1	0 = no RST segment has been sent by the client/server
ACK sent	-	number of segments with the ACK field set to 1
PURE ACK sent	-	number of segments with ACK field set to 1 and no data
unique bytes	bytes	number of bytes sent in the payload
data pkts	-	number of segments with payload
data bytes	bytes	number of bytes transmitted in the payload, including retransmissions
rexmit pkts	-	number of retransmitted segments
rexmit bytes	bytes	number of retransmitted bytes
out seq pkts	-	number of segments observed out of sequence
SYN count	-	number of SYN segments observed (including rtx)
FIN count	-	number of FIN segments observed (including rtx)
First time abs	ms	Flow first packet absolute time (epoch)
Last time abs	ms	Flow last segment absolute time (epoch)
Completion time	ms	Flow duration since first packet to last packet
C first payload	ms	Client first segment with payload since the first flow segment
S first payload	ms	Server first segment with payload since the first flow segment
C last payload	ms	Client last segment with payload since the first flow segment
S last payload	ms	Server last segment with payload since the first flow segment
C first ack	ms	Client first ACK segment (without SYN) since the first flow segment
S first ack	ms	Server first ACK segment (without SYN) since the first flow segment
C Internal	0/1	1 = client has internal IP, 0 = client has external IP
S Internal	0/1	1 = server has internal IP, 0 = server has external IP
C anonymized	0/1	1 = client IP is CryptoPAn anonymized
S anonymized	0/1	1 = server IP is CryptoPAn anonymized
Connection type	-	Bitmap stating the connection type as identified by TCPL7 inspection engine
P2P type	-	Type of P2P protocol, as identified by the IPP2P engine
HTTP type	-	For HTTP flows, the identified Web2.0 content
Average rtt	ms	Average RTT computed as time elapsed between data segment and corresponding ACK
rtt min	ms	Minimum RTT observed during connection lifetime
rtt max	ms	Maximum RTT observed during connection lifetime
Stdev rtt	ms	Standard deviation of the RTT
rtt count	-	Number of valid RTT observation
ttl_min	-	Minimum Time To Live
ttl_max	-	Maximum Time To Live
RFC1323 ws	0/1	Window scale option sent
RFC1323 ts	0/1	Timestamp option sent
window scale	-	Scaling values negotiated [scale factor]
SACK req	0/1	SACK option set
SACK sent	-	number of SACK messages sent
MSS	bytes	MSS declared
max seg size	bytes	Maximum segment size observed
min seg size	bytes	Minimum segment size observed
win max	bytes	Maximum receiver window announced (already scale by the window scale factor)
win min	bytes	Maximum receiver windows announced (already scale by the window scale factor)
win zero	-	Total number of segments declaring zero as receiver window
cwin max	bytes	Maximum in-flight-size
cwin min	bytes	Minimum in-flight-size
initial cwin	bytes	First in-flight size, or total number unack-ed bytes sent before receiving first ACK
rtx RTO	-	Number of retransmitted segments due to timeout expiration
rtx FR	-	Number of retransmitted segments due to Fast Retransmit (three dup-ack)
reordering	-	Number of packet reordering observed
net dup	-	Number of network duplicates observed
unknown	-	Number of segments not in sequence or duplicate which are not classified as specific events
flow control	-	Number of retransmitted segments to probe the receiver window
unnece rtx RTO	-	Number of unnecessary transmissions following a timeout expiration
unnece rtx FR	-	Number of unnecessary transmissions following a fast retransmit
!= SYN seqno	0/1	1 = retransmitted SYN segments have different initial seqno