Anomaly Detection in Network Traffic Traces Using Latent Dirichlet Allocation

Benjamin D. Newton

***Abstract*—The detection of anomalies in network traffic can assist network operators in controlling and securing computer networks. Latent Dirichlet Allocation can be applied to computer network traffic, where word counts are replaced with packet counts, and documents, with user sessions. I describe the processing of network traces collected at UNC, and the results of running LDA on these traces. The resulting model can be used to detect anomalies in other network traces on the UNC network. Variational inference is run on a second trace, and three anomalies are detected and analyzed. A simulated Denial of Service Attack is also detected by the model.**

***Index Terms*—Latent Dirichlet Allocation, Computer Networks**

I. Introduction

DETECTING and correcting certain anomalies in network traffic is an important task for network operators who seek to ensure that computer networks run smoothly and efficiently. Malicious attacks must be detected and stopped, but the detection of anomalies which do not pose a direct security threat are also of interest. These anomalies may indicate areas of misuse in the network. Traditional systems for detecting malicious anomalies such as SNORT [1] and Bro [2] rely on rules written by security experts. These systems are, therefore, bad at detecting new threats. Latent Dirichlet Allocation (LDA) [3] may be able to help. LDA is a machine learning technique originally used for natural language processing, which can be applied to computer network data. Once a model of the traffic on a network is generated, it can be used to identify anomalous network traffic. The goal of this work is to use Latent Dirichlet Allocation to generate a model of the traffic between UNC clients and external external servers, and then to use this model to detect anomalies in other traces from the network.

A. Previous Work

Anomaly detection in computer networks is an active research area. Various systems have been proposed and implemented, including those which detect anomalies based on packet bytes [4], and behavioral distance [5].

Cramer and Carin [6] also use LDA for computer network traffic analysis. They used LDA to analyze traffic on the network of a small business. They obtained good results by analyzing the packet counts and connection initiation counts on various ports. This work differs from their work in three important ways. First, our data set is much larger, including over 17,000 client machines, compared to their 10-40. Second, they analyze the raw packets, whereas I analyze a higher-level representation inferred from the packets. Lastly, although part of the motivation of their paper is intrusion detection, they leave the use of their models for that purpose as a future work.

The organization of the remainder of this paper is as follows. In Section II I give an overview of the approach I used to obtain my results, and information about the data. Next Section III describes the LDA algorithm, and Section IV lists the results. I conclude and describe possible future work in V.

II. The Approach

It will be helpful to understand the networking data, and how it is processed, before giving an overview of the approach.

A. Network Data

The networking group at the University of North Carolina at Chapel Hill (UNC) has collected various TCP/IP header traces over the course of several years [7]. For these traces a monitor records the header of every TCP message flowing between the high-speed link connecting UNC to the router of its ISP. To overcome privacy concerns, the results are anonymized in a consistent manner, replacing every occurrence of each IP address with the a “random” IP address. The raw traces include the following for each TCP message: timestamp for the message, source and destination IP addresses and ports, flags set in the TCP message, sequence numbers (optional), ACK number (optional), window size, and occasionally some extra information for various TCP features. No application layer data is stored in the traces, and therefore, it is impossible to know explicitly what type of application layer data is being carried in each TCP message.

Each TCP message transmits data from a client to a server, or from a server to a client. TCP is connection oriented, which means that before a client and server can exchange messages they must go through a three message handshake procedure. This procedure establishes a connection between a specific client and server. Once a connection is established packets can be sent back and forth. Clients always initiate the TCP connections. A TCP connection can remain open for only a fraction of a second, or for several minutes. A connection can support the transfer a single TCP packet, or thousands. Further a connection can be categorized as either synchronous or concurrent. If messages are exchanged between the client and server in a ping-pong fashion, the connection is synchronous. If TCP messages are sent and received by either side at the same time, the connection is concurrent. A simple interaction with a web server is an example of a synchronous connection. The web browser sends a request to the server and then waits for the response, and only sends the next request after the response is received. The request and response can each be made up of many individual TCP packets, which are re-assembled on the receive side, into a single request or response. The request response interchange of a sequential connection has been defined as an epoch. A single epoch can take the place of may individual TCP messages. For web activity, an epoch can be thought of as the combination of a request for an “object” from the server, and the servers response including that object. I define a new term, sub-epoch, which can be thought of as just the request or response portion of an epoch.

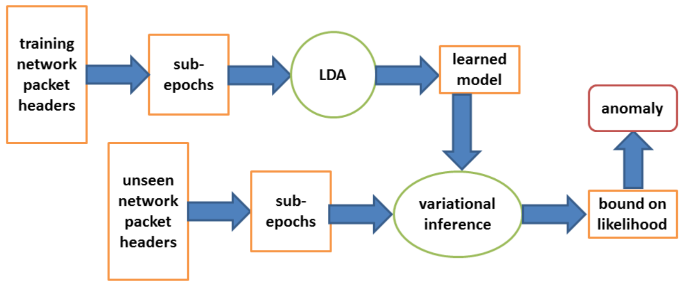
B. *a-b-t Model*

A TCP/IP header trace can be ”reverse compiled” into a higher-level representation. For every TCP/IP connection in the trace a connection vector is generated. The connection vector represents an entire single connection between ”a”, the connection initiator, and ”b”, the connection acceptor. The connection vector also stores the start time ”T” of the connection.

The network traffic is characterized as a set of request response interactions between ”a” and ”b”. Each request or response transfers one application-data unit (ADU), which is a generic term for the object or protocol element being transferred. As described above, each exchange (request then response) is called an epoch, and includes the sizes of the ADUs transferred, and the requester side ”think” or processing time, ”t”.

A network trace contains n connection vectors $C_1...n$, starting at times $T_1...n$. Each connection vector includes a set of k epochs, $C_i =< E_1, E_2, ..., E_k >$, where each epoch is defined by $E_i = (a_i, b_i, t_i)$. Each epoch includes a_i, the size of the i^{th} ADU sent from the connection initiator to the connection acceptor, b_i, the size of the i^{th} ADU sent from the connection acceptor to the connection initiator, and t_i, the ”think” or processing time between the receipt of the ith “response” ADU and the transmission of the $(i + 1)^{st}$ ”request”.

Fig. 1. An overview of the proposed use LDA to detect anomalies in network header traces.



C. *Overview*

The input to my method is a raw network header trace in pcap format. This is the training data set, which is now processed by a set of scripts and C programs resulting in the higher-level representation described above. I then process this data further to arrive at a set of sub-epochs associated with internal clients. These sub-epochs represent all of the traffic exchanged between internal clients and external servers, in both directions. Each sub-epoch is associated with an external client and port, and an internal client and port. All the sub-epochs associated with a certain internal client and port are grouped together. These sub-epochs represent the Internet session of a specific IP address internal to the UNC network. The “documents” in traditional LDA application are equivalent to the session of a specific IP address, and “words” are equivalent to the full external IP address and port number combinations. The “words” in each “document” are counted and then this data set is processed by LDA to yield a compact model of the data.

New network data can now be processed in a similar fashion obtaining “words” and “documents” as described above, but variational inference is now used to determine the bound on log likelihood for each document in the new dataset. This output is tested against a threshold, and any session with log likelihood less than the threshold is identified as anomalous. Figure 1 gives an pictoral overview of the proposed method.

III. Algorithm

The algorithm used is the original Latent Dirichlet Allocation proposed by Blei et al. [3]. As described above, LDA is a method for automatically discovering a given number of topics in a data set. It was originally used to determine the topics of text documents, but generalizes well to other domains. The graphical model representation of LDA is displayed in Figure 2.

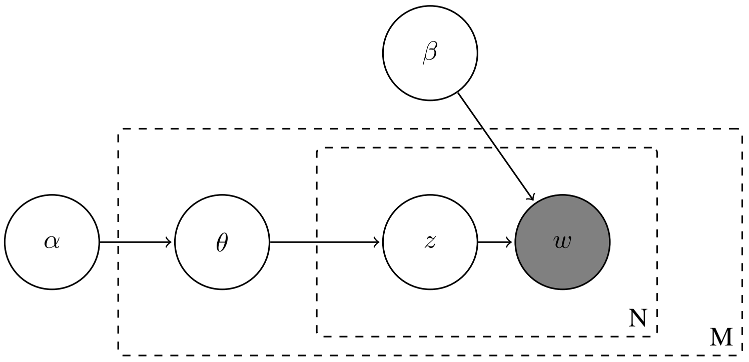


Fig. 2. The graphical model for LDA.

LDA assumes that the data being processed was generated as follows:

- Pick N, the number of sub-epochs in the Internet session. (assumed to be drawn from a Poisson distribution)
- Pick θ, the distribution of topics in this Internet session, from a Dirichlet distribution parametrized by α.
- For each the N sub-epochs in the generated document:
  - Pick a topic from the multinomial distribution parametrized by θ
  - Pick a sub-epoch given the distribution of sub-epochs for the chosen topic

The remainder of the algorithm details can be found in [3].

IV. Results

A. 2008 trace

The data analyzed here is a trace of the traffic to and from UNC on Thursday January 10th, 2008. The trace is one hour in length and was taken between 2:00 to 3:00 p.m. This was a day at the very beginning of spring semester after classes at UNC had commenced. Table I contains a list of some of the statistics from this trace. This dataset was divided in half. The first half hour trace was used for training the topic model, and the last half hour of the trace was analyzed to detect anomalies.

External to internal combined header data size	22 Gigabytes
Internal to external combined header data size	19 Gigabytes
External to internal TCP and UDP packets	292 Million
Internal to external TCP and UDP packets	248 Million
Unique internal client IP addresses (sessions/documents)	17,463
Unique IP/Port combinations (words/unique sub-epochs)	51,266

TABLE I
STATISTICS OF 2008 UNC NETWORK TRACE

LDA was run on the first half hour trace, with $N = 5$. It is difficult to decipher the clear meanings of the topics found because the IP addresses are anonymized, so only the ports have meaning. Table II shows the ports of the port IP combinations which were most popular in each topic.

Topic 1	HTTP only
Topic 2	HTTP only
Topic 3	POP, SMTP, and HTTP
Topic 4	POP3 over SSL, HTTPS, and HTTP
Topic 5	HTTP and HTTPS

TABLE II
DISCOVERED TOPICS

The prevalence of HTTP data in the training data causes the HTTP port to be seen in each topic. It appears, however, that one topic includes a lot of secure mail activity, while another focuses on regular mail, and a third on secure HTTP.

To determine what anomalies exist in the last half hour of the trace I ran variational inference on the last half hour data given the model learned from the first half hour data. Figure 3 shows a graph of the results. I consider any spike on the graph which extends below $-1.5x10^6$ to be an anomaly. Figure 4 shows the same plot, but with the sessions sorted by log likelihood. It is easy to see that most sessions in the second half hour of the trace match well with the trained model, and there are only a handful which have very low log likelihoods.

I analyzed each of the anomalies detected in the last half hour of the trace. Table III lists the anomalies, and their details.

session id	details
17181	sending/receiving to/from 2,680 different external servers
12746	299,502 sub-epochs to a single SMTP server (every 6ms)
17170	sending/receiving to/from 2,400 different external servers

TABLE III
LIST OF ANOMALIES DETECTED

Fig. 3. The bound on log likelihood for the sessions in the last half hour of data.

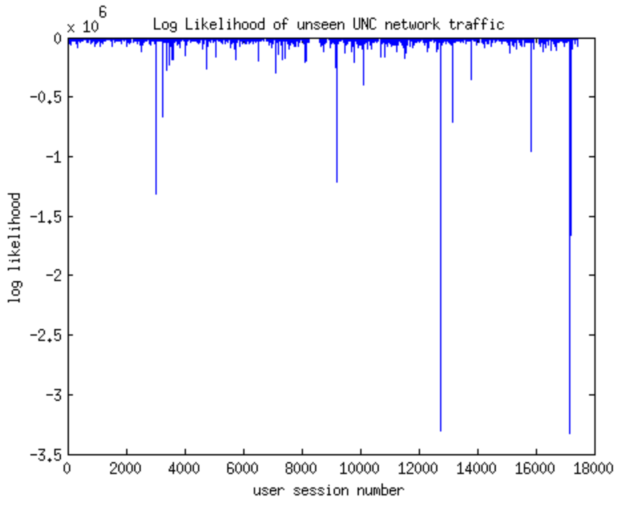
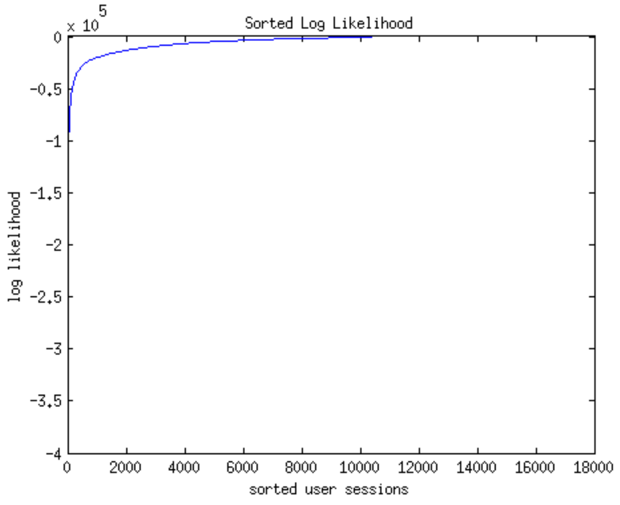


Fig. 4. The bound on log likelihood for sessions in the last half hour of data, but sorted by log likelihood



The first and third anomalies seem similar, each with connections to over 2,000 servers during the half hour trace. There is likely some sort of a web crawler running on these machines, and contacting tons of servers. It may be harmless, but probably still something a network administrator would want to be aware of. The second anomaly with nearly 300 thousand messages exchanged with an SMTP server, is a bit more troubling. It is possible that this was actually a malicious client participating in a Mailbomb attack. According to the DARPA Intrusion Detection Attacks Database [8] a Mailbomb attack "is one in which the attacker sends many messages to a server, overflowing that server's mail queue and possibly causing a system failure"

B. Denial-of-service Injection

To test whether the model would detect a threat I simulated a denial-of-service attach (DoS) emanating from a client at

UNC, by adding a new user session which was similar to other sessions except that it had 200,000 sub-epochs sent towards a single server. This simulates what I expect a SYN flooding denial-of-service attack would look like. Running variational inference on the model again yielded a log likelihood of $-2.4x10^6$, for that session. This is below the chosen threshold, and would be flagged as an anomaly.

V. Conclusion

Latent Dirichlet Allocation has proven itself to be useful at modeling network traffic. I have shown that a model learned using LDA on a network trace can detect anomalies in other network traces. As future work I would consider training on more data, and using traces from different days and times. More detailed analysis of the anomalies, would also be beneficial. As attacks become more prevalent and sophisticated, the application of machine learning to computer networking will be extremely important.

References

[1] M. Roesch, “Snort - lightweight intrusion detection for networks,” in *Proceedings of the 13th USENIX conference on System administration*, ser. LISA ’99. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238. [Online]. Available: http://dl.acm.org/citation.cfm?id=1039834.1039864

[2] V. Paxson, “Bro: A system for detecting network intruders in real-time,” in *Computer Networks*, 1999, pp. 2435–2463.

[3] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, “Latent dirichlet allocation,” *Journal of Machine Learning Research*, vol. 3, p. 2003, 2003.

[4] M. V. Mahoney, “Network traffic anomaly detection based on packet bytes,” in *Proceedings of the 2003 ACM symposium on Applied computing*, ser. SAC ’03. New York, NY, USA: ACM, 2003, pp. 346–350. [Online]. Available: http://doi.acm.org/10.1145/952532.952601

[5] H. Sengar, X. Wang, H. Wang, D. Wijesekera, and S. Jajodia, “Online detection of network traffic anomalies using behavioral distance.” in *IWQoS*. IEEE, 2009, pp. 1–9.

[6] C. Cramer and L. Carin, “Bayesian topic models for describing computer network behaviors,” in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, may 2011, pp. 1888 –1891.

[7] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott, “What tcp/ip protocol headers can tell us about the web,” *SIGMETRICS Perform. Eval. Rev.*, vol. 29, no. 1, pp. 245–256, Jun. 2001. [Online]. Available: http://doi.acm.org/10.1145/384268.378789

[8] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, “The 1999 darpa off-line intrusion detection evaluation,” *Comput. Netw.*, vol. 34, no. 4, pp. 579–595, Oct. 2000. [Online]. Available: http://dx.doi.org/10.1016/S1389-1286(00)00139-0