Universidad Nacional del Litoral

Proyecto Final de Carrera

Diseño de un sistema de detección de anomalías en redes de computadoras.

Informe de avance 1

Pineda Leandro
Santa Fe
14 de octubre de 2016

1. Resumen

El presente documento muestra los resultados obtenidos de la etapa de investigación del proyecto final de carrera de Ingeniería en Informática. En primer lugar dará una breve descripción de los distintos métodos existentes para la detección de anomalías. Luego se presentarán las técnicas de modelado a utilizar para detectar anomalías en el tráfico de red. Finalmente se describirán las tecnologías a utilizar, la arquitectura del sistema y sus componentes.

2. Detección de anomalías

La detección de anomalías puede ser considerada como un problema de clasificación, donde comportamientos anómalos deben ser diferenciados de eventos normales[1]. A continuación se describen los distintos enfoques analizados:

2.1. Métodos de detección

2.1.1. Enfoque estadístico

Este enfoque es usado cuando los comportamientos normales pueden ser ubicados en regiones de alta probabilidad de algún modelo estocástico, mientras que las anomalías pueden encontrarse en regiones de baja probabilidad de ese modelo[2]. El modelo estocástico es determinado *a priori* o es inferido a partir de un conjunto de datos. Bajo este enfoque, una anomalía es una observación que puede ser total o parcialmente irrelevante porque no está generada por un modelo estocástico dado.

Estos métodos de detección de anomalías ajustan un modelo estadístico (usualmente de comportamiento normal) a un conjunto de datos y luego, usando inferencia estadística, determinan si una nueva instancia pertenece a este modelo. Las instancias que tienen una baja probabilidad de ser generadas por el modelo ajustado son consideradas como anómalas.

2.1.2. Enfoque espectral

En algunas ocasiones muchas características de los patrones analizados son altamente independientes. Utilizar solo las dimensiones que son dependientes para describir los datos incrementa la precisión del modelo y reduce el costo computacional de los algoritmos. Matemáticamente, esta formulación es referida como reducción de dimensionalidades[3]. Podemos pensar esta reducción como una representación de los datos en un espacio de menos dimensiones, de forma que instancias normales y anómalas se vean drásticamente diferentes. Una técnica popular de reducción de dimensionalidad es PCA (*Principal Component Analysis*).

2.1.3. Enfoque basado en Machine Learning

En este enfoque se *entrena* un algoritmo con datos de entrenamiento de forma que este "aprenda" cómo debería ser un comportamiento normal en el tráfico de red. Así, las anomalías son identificadas en base a la experiencia previa. Un algoritmo de *machine learning* "aprende" una función que mapea todos las instancias de datos con alguno de los dos estados (usualmente representados con 0 y 1).

De acuerdo con la caracterización realizada en [2], podemos diferenciar los siguientes tipos de algoritmos basados en *machine learning*:

- **Basados en clasificación:** El objetivo de estos algoritmos es asignar cada dato a una clase (en este caso normal/anómalo), basandose en la información provista por un conjunto de características.
- **Algoritmos de vecino más cercano**[1]**:** Estos algoritmos usan diferentes funciones (basadas en alguna métrica como distancia o densidad) para medir la diferencia entre una instancia de los datos y su *k-ésimo* vecino más cercano[2]. Esta diferencia o distancia es un puntaje que puede ser utilizado para decidir si la instancia es o no una anomalía.

[1] En la literatura se encuentra como *nearest-neighbor*

- **Clustering:** Estos algoritmos buscan en los datos de entrenamiento grupos de instancias muy similares o cercanas entre si. Las anomalías pueden formar grupos de muy pocos elementos o no pertenecer a ningún grupo. Mapas auto-organizativos[4] y algorítmos de k-medias[5] son algoritmos clásicos de clustering.

2.1.4. El enfoque de *streaming*

En muchos casos, detectar anomalías en una red significa llevar registro de cambios significativos en los patrones de tráfico de red, como número de flujos activos o volumen de tráfico actual. Esto hace que sea necesario aplicar técnicas de detección que sean escalables, puesto que en redes donde circulan grandes volúmenes de información, registrar estos cambios en cada flujo de tráfico puede ser una tarea con un gran costo computacional. El enfoque de *streaming* analiza flujos continuos de datos y extrae información de cada uno, utilizando algoritmos discretos para detectar anomalías[6] y evitar así tomar muestras del tráfico cada cierto tiempo, práctica típicamente utilizada en los algoritmos de detección de anomalías para solucionar el problema de la escalabilidad. Sin embargo, atacar el problema de la escalabilidad con muestreo involucra una disminución en la precisión de los sistemas de detección de anomalías ya que los paquetes que no son tenidos en cuenta pueden contener información importante para determinar la existencia de tales eventos.

2.2. Comparación de los métodos

Los métodos supervisados (sin importar que enfoque utilicen) necesitan de grandes *dataset* de comportamiento normal o un conjunto de datos anómalos conocidos e identificados. Construir un modelo basado en una base de datos hace que calidad de las clasificaciones realizadas por el mismo dependa directamente de la calidad de los datos: un modelo no puede ser mejor que el *dataset* con el cual se lo construyó. Más aún, la implementación en diferentes ámbitos de producción hace necesario que el modelo sea recalculado para un conjunto de datos que son propios de la infraestructura de red los cuales, en general, no están disponibles.

Una desventaja de los métodos basados en modelos estadísticos es que pueden ser "entrenados" gradualmente de forma que el tráfico generado durante un ataque se identificado como normal. Además, la puesta en funcionamiento de este tipo de sistemas toma períodos largos de tiempo dado que la construcción de los modelos estocásticos involucra analizar grandes volumenes de datos. Por otro lado, los modelos no requieren conocimiento previo ya que tiene la habilidad de "aprender" el comportamiento esperado procesando los datos del tráfico de red[7].

Los enfoques de clustering y vecindad utilizan métodos no supervisados. Aunque esto es una gran ventaja con respecto a los *dataset*, su precisión es muy dependiente de las métricas utilizadas: el uso de una medida poco adecuada de proximidad afecta negativamente la capacidad de detección de los algoritmos de vecindad.

Las técnicas *on-line* generalmente muestrean los datos para reducir la carga computacional de los algoritmos, descartando información que puede ser importante para que el sistema clasifique adecuadamente un evento. En los últimos años, las arquitecturas de *streaming* distribuido (por ejemplo Flume[8], Apache Storm[9] y Spark Streaming[10]) permitieron desarrollar algoritmos que aprovechen la capacidad de cálculo paralelo para mejorar la eficiencia en la clasificación y solucionar el problema de la escalabilidad[11]. La ventaja más importante de estas técnicas es la posibilidad de realizar las tareas de detección en forma distribuida: esto permite combinar múltiples resultados de detección de forma de reducir la cantidad de falsos positivos, aspecto de gran importancia en los sistemas de detección de anomalías y que además permite escalabilidad. Este último aspecto es también de gran importancia: la reciente aparición de ataques distribuidos masivos[2] hace que sea necesario pensar el diseño de los nuevos sistemas de detección de anomalías para que funcionen en arquitecturas escalables y tolerante a fallas.

[1] término que refiere a los algoritmos que procesan la información del tráfico de red mientras se genera, opuesto a los algoritmos *batch*

[2] https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/

3. Método de detección propuesto

El tráfico de red puede ser caracterizado por la información disponible en la cabecera de los paquetes. El modelo TCP/IP establece que los host deben soportar como mínimo los protocolos IP, ICMP, TCP y UDP[12]. Estos tienen en común una dirección de origen y destino: en capa de red se utilizan direcciones IP[13] y en capa de transporte se utilizan puertos[14][15].

Podemos identificar entonces un flujo de datos entre dos *host* por una tupla de 5 valores (en adelante se hará referencia a estas tuplas como *keys*). Estas *keys* forman un espacio de 2^{104} características[3]:

$$\langle IP\ de\ origen,\ IP\ de\ destino,\ puerto\ de\ origen,\ puerto\ de\ destino,\ protocolo \rangle$$

Caracterizar tráfico normal en tiempo real utilizando un espacio de tantas dimensiones implica el uso de grandes cantidades de memoria. Más aún, las plataformas convencionales que utilizan sistemas de un solo procesador no proveen suficiente poder de cómputo para procesar cantidades masivas de flujos de datos en espacios de altas dimensionalidades.

Las técnicas basadas en conteo y las técnicas de basadas en *sketch*[16][17] pueden implementarse usando estructuras de datos eficientes para detectar anomalías en grandes espacios de características[11]. La primera técnica utiliza contadores que son actualizados a medida que se procesan nuevas tuplas. La segunda utiliza *sketchs*: son matrices que permiten representar un gran conjunto de datos como otro conjunto de datos de menor tamaño. Un buen *sketch* de un *dataset* es aquel que puede utilizarse en lugar del *dataset* en un cálculo, sin perder mucha precisión[18].

Las anomalías, llamadas *heavy keys*, pueden identificarse formulando el problema de detección en dos partes: detección de *heavy hitter* y detección de *heavy change*. El objetivo de la detección de *heavy hitter* es identificar el conjunto de flujos o *keys* que representan la mayor porción del tráfico de red o la capacidad del canal de comunicación[19]. Por otro lado, en el problema de detección de *heavy change*, el objetivo es detectar el conjunto de *keys* que tienen cambios abruptos en el volumen de tráfico que transportan, entre un período de tiempo y el próximo[16]. La detección de *heavy keys* puede realizarse mediante las técnicas de conteo y *sketch*, procesando *streams* de datos en tiempo real en un espacio sublineal en N. Además, el algoritmo es perfectamente paralelizable[18]. Más aún, las lecturas y actualizaciones a los valores del *sketch* son muy eficientes y pueden hacerse en tiempo constante[20].

4. Arquitectura del sistema

Los datos serán capturados utilizando la librería *libpcap*[4]. Luego, el *stream* de datos es procesado usando las técnicas de conteo y *sketch* mencionadas previamente para llenar las estructuras de datos necesarias. El módulo de detección de anomalías analiza las estructuras de datos calculadas y determina la presencia de *heavy keys*. Estas son almacenadas en un archivo de log. Finalmente la interfaz web presenta la información de las *keys* analizadas, las estadísticas del sistema y muestra las anomalías detectadas.

Tecnologías a utilizar

Data Streaming Frameworks

Actualmente la comunidad de software libre mantiene tres frameworks importantes para procesamiento de *streams* de datos. A continuación se da una descripción de cada uno:

- Spark Streaming: es una extensión del API de Apache Spark, un sistema de propósito general para procesamiento de datos en *cluster* de computadoras. Spark Streaming permite construir modelos de procesamiento de *streams* de datos de manera escalable, con garantía de que los datos serán procesados una y solo una vez, tolerante a fallos y de alto desempeño.

[3] Cada dirección IP es de 32 bits, cada puerto de 16 bits y el protocolo es un valor de 8 bits

[4] http://www.tcpdump.org/

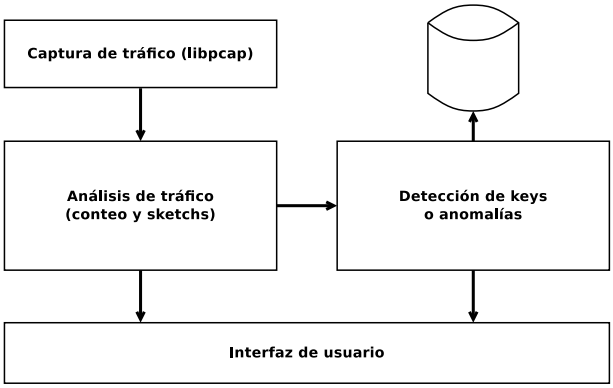

Figura 1: Arquitectura del sistema

Incorpora también funciones de alto nivel como *map*, *reduce*, *join* y *window*. Spark Streaming no procesa estrictamente los datos en tiempo real sino que procesa pequeños lotes o *mini-batches* a intervalos de tiempo dados.

- Apache Storm: es un sistema de cálculo distribuido en tiempo real. Los *streams* de datos son analizados como eventos en lugar de una serie de *mini-batches*. Esto significa que Storm tiene muy baja latencia y es ideal para datos que deben ser procesados como una sola entidad. Es escalable, tolerante a fallas, garantiza que los datos serán procesados al menos una vez.

  Apache Storm fue diseñado para ser utilizado con cualquier lenguaje de programación. Utiliza *Thrift*[5] para definir y ejecutar topologias por lo que las topologías pueden ser desarrollado en cualquier lenguaje.

- Apache Flink: es una plataforma de código abierto para procesamiento distribuido de datos mediante *batches* o *streaming*. A diferencia de los dos anteriores, Flink asegura que los datos serán procesados una única vez sin agregar latencia como como ocurre con Spark Streaming. Además, el mecanismo que implementa para asegurar tolerancia a fallos distribuida es liviano[21], lo que permite a la plataforma procesar grandes volúmenes de datos y mantener su consistencia.

  Flink únicamente soporta los lenguajes de programación Java y Scala. Esto debe ser así dado que implementa su propio espacio de memoria dentro de una máquina virtual Java, y debe ejecutar lenguajes compatibles.

Dadas los *framework* analizados, la selección adecuada para el desarrollo del proyecto es Apache Flink. En cualquier sistema de seguridad de información son necesarias tolerancia a fallos y alto desempeño.

Apache Storm ofrece alto desempeño, pero por su diseño asegura que las *keys* serán procesadas al menos una vez, sin dar garantías de repeticiones en las tuplas. Por otro lado, Spark Streaming no es adecuado para la aplicación que se quiere desarrollar pues tiene bajo desempeño debido al uso de *micro-batching*.

[5] http://thrift.apache.org/

Anexos

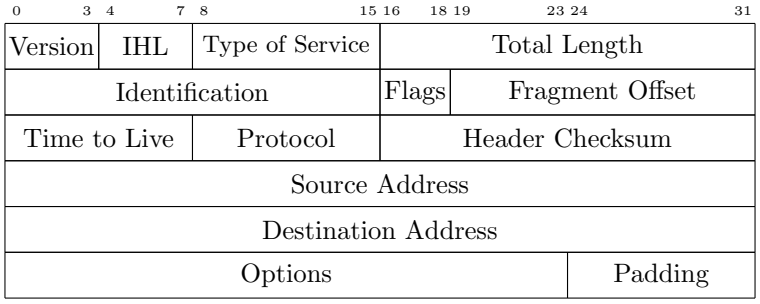

Figura 2: Cabecera IP

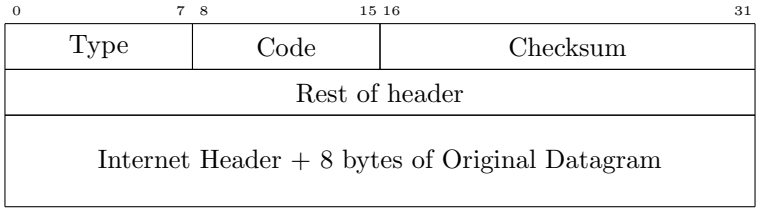

Figura 3: Cabecera ICMP

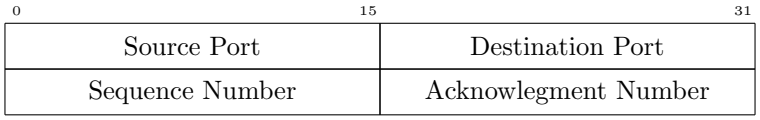

Figura 4: Cabecera UDP

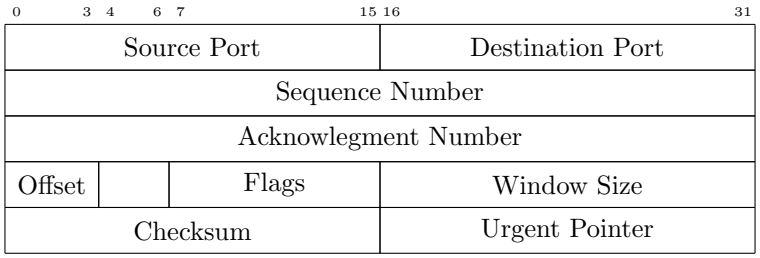

Figura 5: Cabecera TCP

Referencias

[1] S. A.-H. Baddar, A. Merlo y M. Migliardi, "Anomaly detection in computer networks: a state-of-the-art review",

[2] V. Chandola, A. Banerjee y V. Kumar, "Anomaly detection: a survey", *ACM Comput. Surv.*, vol. 41, n.º 3, 15:1-15:58, jul. de 2009, ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. dirección: http://doi.acm.org/10.1145/1541880.1541882.

[3] J. Wang, *Geometric Structure of High-Dimensional Data and Dimensionality Reduction*. Springer Berlin Heidelberg, 2012, ISBN: 9783642274978. dirección: https://books.google.com.ar/books?id=0RmZRb2fLpgC.

[4] T. Kohonen, ed., *Self-organizing Maps*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997, ISBN: 3-540-62017-6.

[5] J. A. Hartigan y M. A. Wong, "A K-means clustering algorithm", *Applied Statistics*, vol. 28, págs. 100-108, 1979.

[6] G. Cormode y M. Thottan, *Algorithms for Next Generation Networks*, 1st. Springer Publishing Company, Incorporated, 2010, ISBN: 1848827644, 9781848827646.

[7] M. H. Bhuyan, D. K. Bhattacharyya y J. K. Kalita, "Network anomaly detection: methods, systems and tools.", *IEEE Communications Surveys and Tutorials*, vol. 16, n.º 1, págs. 303-336, 2014. dirección: http://dblp.uni-trier.de/db/journals/comsur/comsur16.html#BhuyanBK14.

[8] *Apache flume*. dirección: https://flume.apache.org/.

[9] *Apache storm*. dirección: http://storm.apache.org/.

[10] *Apache spark streaming*. dirección: http://spark.apache.org/streaming/.

[11] Q. Huang y P. P. Lee, "Ld-sketch: a distributed sketching design for accurate and scalable anomaly detection in network data streams", en *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, IEEE, 2014, págs. 1420-1428.

[12] R Braden, *RFC 1122 Requirements for Internet Hosts - Communication Layers*, 1989. dirección: http://tools.ietf.org/html/rfc1122.

[13] J. Postel, ed., *Rfc 791 internet protocol - darpa inernet programm, protocol specification*, Internet Engineering Task Force, 1981. dirección: http://tools.ietf.org/html/rfc791.

[14] J. Postel, *User Datagram Protocol*, RFC 768 (Standard), Internet Engineering Task Force, 1980. dirección: http://www.ietf.org/rfc/rfc768.txt.

[15] ——, *Transmission Control Protocol*, RFC 793 (Standard), Updated by RFCs 1122, 3168, Internet Engineering Task Force, 1981. dirección: http://www.ietf.org/rfc/rfc793.txt.

[16] B. Krishnamurthy, S. Sen, Y. Zhang e Y. Chen, "Sketch-based change detection: methods, evaluation, and applications", en *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, ACM, 2003, págs. 234-247.

[17] S. Muthukrishnan, *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[18] E. Liberty, "Simple and deterministic matrix sketching", en *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2013, págs. 581-588.

[19] C. Estan y G. Varghese, *New directions in traffic measurement and accounting*, 4. ACM, 2002, vol. 32.

[20] A. Goyal, J. Jagarlamudi, H. Daumé III y S. Venkatasubramanian, "Sketching techniques for large scale nlp", en *Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop*, Association for Computational Linguistics, 2010, págs. 17-25.

[21] K. M. Chandy y L. Lamport, "Distributed snapshots: determining global states of distributed systems", *ACM Transactions on Computer Systems (TOCS)*, vol. 3, n.º 1, págs. 63-75, 1985.