Differentially-Private Big Data Analytics for High-Speed Research Network Traffic Measurement

Oana-Georgiana Niculaescu
UMass Boston
onic@cs.umb.edu

Mihai Maruseac
UMass Boston
mmarusea@cs.umb.edu

Gabriel Ghinita
UMass Boston
gabriel.ghinita@umb.edu

ABSTRACT

High-speed research networks (e.g., Internet2, Géant) represent the backbone of large-scale research projects that bring together stakeholders from academia, industry and government. Such projects have increasing demands on throughput (e.g., 100Gbps line rates), and require a high amount of configurability. Collecting and sharing traffic data for such networks can help in detecting hotspots, troubleshooting, and designing novel routing protocols. However, sharing network data directly introduces serious privacy breaches, as an adversary may be able to derive private details about individual users (e.g., personal preferences or activity patterns). Our objective is to sanitize high-speed research network data according to the de-facto standard of *differential privacy (DP)*, thus supporting benefic applications of traffic measurement without compromising individuals' privacy. In this paper, we present an initial framework for computing DP-compliant big data analytics for high-speed research network data. Specifically, we focus on sharing data at flow-level granularity, and we describe our initial steps towards an environment that relies on Hadoop and HBase to support privacy-preserving NetFlow analytics.

1. INTRODUCTION

The last decade witnessed significant developments in the area of high-speed research networks. Such networks are characterized by high line rates (up to 100Gbps) and connect users from heterogeneous environments, such as academia, industry and government. The projects served by these networks have a diverse set of requirements with respect to quality of service, and often complex network configurations are needed. Existing network communication protocols need to be adapted or re-designed from scratch to cope with the increasing demands. To improve flexibility and configurability of operation, *software-defined networks (SDNs)* are becoming the technology of choice for the future. To better understand and monitor these networks, it is important to collect and share network measurement data, which can help both in improving network operations (e.g., detecting traffic hotspots, troubleshooting), as well as in network design (e.g., improving routing protocols, transport mechanisms, etc.). Due to the high bandwidth of research networks, such data are generated at high rates, and in high volumes, raising significant performance challenges to process them.

In addition, sharing such network data can introduce serious privacy breaches, as an adversary may be able to derive private details about individual users. For instance, traffic data may disclose the health status or political orientation of some users based on their visits to specific medical or news websites. One can also determine the activity patterns of individuals based on their access to e-mail, social media and other online services. Our objective is to sanitize high-speed research network data according to the de-facto standard of *differential privacy (DP)*. DP is a semantic model that prevents an adversary from learning whether data about a specific individual has been included or not in the release. This way, we can support the benefic applications of traffic measurement without compromising individual privacy. This objective has many challenges, due to the high rates and high volumes of traffic data.

In this paper, we present an initial framework for computing DP-compliant big data analytics for high-speed research networks. Specifically, we focus on sharing data at flow-level granularity, and we describe our initial steps towards a system that uses Hadoop and HBase to support privacy-preserving NetFlow analytics.

Section 2 introduces necessary background on the NetFlow protocol, big data tools and differential privacy. Section 3 presents the proposed framework for differentially-private processing of network flow data, as well as the main open research questions. We conclude with directions for future research in Section 4.

Attribute	Description
IN_BYTES	count of incoming bytes in the flow
OUT_BYTES	count of outgoing bytes in the flow
IN_PKTS	count of incoming packets in the flow
OUT_PKTS	count of outgoing packets in the flow
PROTOCOL	IP layer protocol (1=ICMP, 6=TCP, etc)
IPV4_SRC_ADDR	IPV4 source address
IPV4_DST_ADDR	IPV4 destination address
L4_SRC_PORT	TCP or UDP (layer 4) source port
L4_DST_PORT	TCP or UDP (layer 4) destination port

Table 1: NetFlow v9 attributes used by our framework

2. BACKGROUND

2.1 NetFlow Protocol. NetFlow is a network protocol for collecting IP traffic information developed by Cisco, and subsequently adopted by other manufacturers. A network flow is represented by a tuple with attributes such as: source and destination IP address, IP protocol, source and destination port numbers, type of service, timestamp, number of packets and bytes communicated, etc. A network flow models a logical end-to-end communication stream,

CODASPY'17 March 22-24, 2017, Scottsdale, AZ, USA

ACM ISBN 978-1-4503-4523-1/17/03.

DOI: http://dx.doi.org/10.1145/3029806.3029841

which may comprise numerous individual packets, and thus is a compact but informative way of representing network traffic. We focus on the NetFlow version 9 specification, but we use only a subset of nine attributes out of the over 100 specified by NetFlow v9. Table 1 summarizes these attributes and their meaning.

2.2 Hadoop and HBase. High-speed research networks transfer very large amounts of data at high line rates, hence the amount of collected NetFlow data requires an environment that can provide large amounts of storage and efficient distributed processing. In our work, we make use of two prominent open-source big data tools, namely Apache Hadoop and HBase. Both Hadoop and HBase rely on the *Hadoop Distributed File System (HDFS)* environment [1]. HDFS provides a distributed file system that runs on two types of nodes: *NameNodes*, which store file system metadata, and *DataNodes*, which store the actual application data. The servers communicate with each other using the TCP protocol. To achieve reliability and high availability, HDFS replicates the data across several DataNodes, which also allows the processing of the data to be performed in distributed fashion.

On top of HDFS, Hadoop provides a framework for analyzing data using the MapReduce computation model [2]. A MapReduce program is composed of a *map* phase that performs filtering and sorting on the raw data, and a *reduce* phase that gathers the information from the map phase and performs a summarization operation (e.g., sum, max, etc.). In the map phase, data are partitioned into chunks processed by Map tasks in parallel on different nodes. Usually, each node works with local data (e.g., data stored on the HDFS system local to the node). This is an advantage as it reduces costly data transfers between different nodes. However, Hadoop does not provide any data indexing features, as it always performs a linear scan of the input. In our setting, we often access only a partition of the data (e.g., HTTP traffic, or net flows within a certain time range). HBase is a column-oriented big data store that supports efficiently random access. In HBase, one unit of data is represented by a row, and every row has a *row key* that uniquely identifies it. Each column of an HBase table represents one attribute of the row. For example, in our case a column could be the source IP, another column the destination IP, etc. HBase allows different attributes to be grouped together in *column families*, so that queries that retrieve attributes in the same family run faster. In the distributed HDFS, column families are stored as contiguous blocks of data.

2.3 Differential privacy (DP) [3] guarantees that for any two *sibling* datasets $\mathcal{D}_1, \mathcal{D}_2$ that differ in a single net flow π, the probability of an adversary learning which of the two datasets was used to obtain a certain output $\mathcal{A}$ is bounded by $\left|\ln \frac{Pr[\mathcal{A}(\mathcal{D}_1)]}{Pr[\mathcal{A}(\mathcal{D}_2)]}\right| \leq \epsilon$, where parameter $\epsilon > 0$ represents the *privacy budget*. To achieve privacy for numerical queries, the Laplace mechanism adds to each query result noise randomly distributed according to a Laplace distribution with parameter $\lambda = S/\epsilon$ where S is the *sensitivity* of the query, i.e., the maximum change in the result of the query for any two sibling databases. An important property of differential privacy is *sequential composability* which guarantees that executing algorithm $\mathcal{A}_1$ with privacy budget ϵ_1 followed by algorithm $\mathcal{A}_2$ with budget ϵ_2 produces a differentially private algorithm with parameter $\epsilon_1 + \epsilon_2$. This allows composing algorithms which use the results of simpler queries to produce a more accurate result for a highly sensitive query/algorithm.

3. PROPOSED APPROACH

3.1 System Architecture. Our contribution consists of a framework to support differentially-private analytics for a NetFlow v9 big data repository. The system architecture is presented in Figure 1. The input flows are collected in NetFlow v9 format from devices such as switches or routers. The flow data is pre-processed by retaining only the nine attributes from Table 1, and they are inserted in an HBase table of raw flows. The advantage of using HBase consists in the ability to perform random data access, as most user queries are interested in only a portion of the data (e.g., a certain time window, or data corresponding to a single protocol/service).

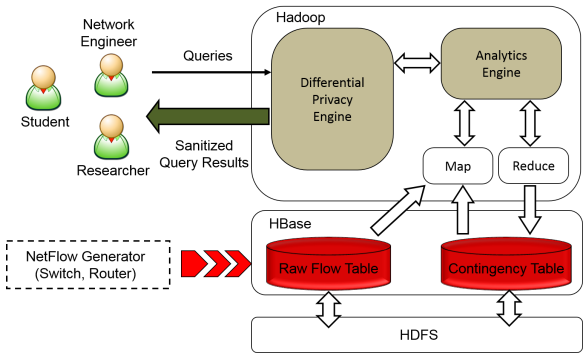

Figure 1: System Architecture

When users submit queries, the results are computed using a MapReduce job that accesses both the raw flow data, as well as the contingency tables that our system creates, and which will be described shortly. Both these tables contain sensitive data, so they are represented in red color. The analytics engine is responsible for performing the appropriate Map and Reduce phases, and may store additional results in the contingency table. Random noise is added to the results according to the DP model, and the noisy results are returned to the user.

3.2 Data Collection. We collect NetFlow data directly from network devices located at several Internet2 academic participants[1] (most manufacturers support NetFlow in their higher-end Layer 3 switches). Net flow metadata is generated in JSON format, which is delimited-text input. Each captured net flow contains numerous attributes, from which we select only a subset of interest (see Table 1 for the attributes we keep). Each flow record is generated after the session is closed, and is stored in the HBase repository. To enhance performance, before data insertion we convert numerical fields to integer types, which leads to a more compact representation (e.g., port 32768 can be represented as a two-byte integer, instead of a 5-byte string). We also convert IP addresses to binary format, which enhances operations such as flexible network mask computation. A sample of raw data format is illustrated in Table 2.

3.3 HBase Schema. The schema used to store flow data is very important, as it can have a significant impact on query performance. HBase stores data in sequence files, which contain key/value pairs. Our objective is to optimize the schema for rapid filtering using the row key and selection of attributes from specific column families. Most typical queries for the envisioned applications define a time range of interest, and require some statistic such as packet count or total bytes for flows between selected source and destination IPs. We devised the HBase schema in Table 3, where ts denotes the flow timestamp, BC and PC are byte and packet counts, respectively, and $md5$ is a hash over all flow attributes used to guarantee key uniqueness. The most important aspect in HBase schema design is the choice of row key. Our schema considers three types of row keys, corresponding to different types of queries. The general structure of our row keys comprises of several elements: (i) the time stamp at which the flow was generated, (ii) a type identifier (between 1 and 3) specifying how the row key must be interpreted, (iii) one attribute (sourceIP/destinationIP/port, based on the type value), and (iv) the $md5$ hash of the entire tuple.

[1]To preserve anonymity, we omit details about specific institutions.

Flow id	Source IP	Destination IP	Port	Protocol	Timestamp	Bytes Count	Packets Count
1	172.24.174.7	192.168.188.195	80	tcp	2016-02-24 19:18:28.0	1420	1
2	192.168.16.198	172.24.171.56	443	tcp	2016-02-24 19:18:28.0	104	1
3	172.24.171.250	172.24.171.255	138	udp	2016-02-24 19:18:28.0	229	1

Table 2: NetFlow data sample

The rationale for having three key types is to support efficiently distinct types of queries. Since most queries specify a time window, ts is a key prefix in all cases. Type 1 is designed for efficient processing of queries that ask for all flows originating at a given IP, whereas type 2 is relevant when the terminating end of a flow is specified in the query. Finally, type 3 is useful for queries that request statistics on specific service types (e.g., *http*, *ftp*, etc.). We also consider in the schema three different column families ($CF1$, $CF2$, $CF3$), which can drastically reduce the amount of data that needs to be scanned, based on the attributes requested by a query. The column families are a combination of the other attributes not included in the row key, and we also have a column family that combines the count information for bytes and packets.

3.4 Differentially-Private Queries. Querying big data repositories with differential privacy protection poses two challenges: achieving good query performance, and maintaining data accuracy. According to the Laplace mechanism [3], the noise added to a query result is proportional to the sensitivity of the query set. In turn, sensitivity is proportional to the maximum number of queries that overlap a specific region of the attribute space. To maintain data accuracy, it is important to limit the amount of overlap. This can be achieved by deciding upon a *strategy query set* [3], and answering all future queries based on the results to the strategy set.

The focus of our work is to determine an appropriate set of queries that achieves low sensitivity, and based on which any query can be answered with fast performance and high accuracy. To achieve this goal, we build a number of *contingency tables*, which are the equivalent of the answers to the strategy queries. The contingency tables are multi-dimensional histograms, and each histogram bin contains the (noisy) answer to a statistical query. Below is a list of several typical queries that fit the strategy query profile:

Q1 : [Sum(BC), (timerange = α), ForEach srcIP]
Q2 : [Avg(PC), (timerange = β), ForEach destIP]
Q3 : [Sum(BC), (timerange = γ), ForEach port]

For instance, $Q1$ computes a histogram that contains for each IP the total number of bytes sent from that IP to all other destinations during the time period specified by $\alpha = [ts_{start}, ts_{end}]$. The result for $Q1$ can be determined using MapReduce on the raw flow table. Our framework creates a *Map* job for all those tuples that have the ts attribute contained within the range α, the key type 1 (because we want the row key to identify the source IP addresses) and the value of $srcIP$. The *Map* job will return all the matching tuples for that row key, and for each source IP there will be a count computed in the *Reduce* phase. Following the *Reduce* phase, all results will be stored in a contingency table inside HBase. According to the parallel composition property of differential privacy, the result for each distinct IP corresponds to a data partition that does not overlap with that of any other IP. However, the sequential composability property applies with respect to all other queries that specify the same IP in their predicates (e.g., $Q2$). One has to carefully quantify this overlap, which will determine the sensitivity value, and in turn the amount of noise that needs to be added to each bin.

Row Key	CF1	CF2	CF3
ts|1|srcIP|md5	destIP, port	destIP	BC,PC
ts|2|destIP|md5	srcIP, port	srcIP	BC,PC
ts|3|port|md5	srcIP	destIP	BC,PC

Table 3: HBase schema for raw flow tables

Each contingency table has its own specific structure, and we have to take in consideration what kind of direct queries would we need in order to return the users the result as fast as possible. The schema for the contingency table supporting queries $Q1 - Q3$ is illustrated in Table 4, where res denotes the resolution of the time slice supported (i.e., allow time ranges aligned at a multiple of minutes), qn denotes the query type (1-3), and q represents the query description, namely concatenated values of the attributes that are requested in the query. In the example of Table 4 all three queries have the res parameter as 15, which means that we will consider for our query all flows with a timestamp that is at most 15 minutes in the past from ts.

3.5 Research Challenges. There are several research challenges to be addressed within the proposed framework. One important aspect is choosing time granularities for strategy queries. One may allow reporting to be performed at hourly, daily and weekly intervals. According to sequential composition, doing so would triple sensitivity compared to having a single interval. In this case, one may choose to compute the results for weekly aggregation based on daily aggregation results, but in the general case, time intervals may be overlapping, rather than completely scoped, so sensitivity may need to be increased. Another important research challenge is privacy budget allocation. Instead of giving all granularities the same amount of budget, one can choose to use less budget for coarser granularities, where errors can be more easily absorbed.

Another important research question that may lead to gains in accuracy is considering hierarchies of attribute values. For instance, rather than considering all services independently, one can choose broader histogram bins, e.g, HTTP traffic may include ports 80, 443, 8080. This way, the histograms for ports become more coarse-grained, and the relative error will decrease. Of course, the trade-off is that one loses accuracy when individual ports are queried. A similar concept can be applied to the IP address space. One can choose bin granularity at the level of a class 'C' network for instance (netmask /24), or coarser granularity (e.g., /22), which may be still good enough to pinpoint cross-organization traffic.

4. CONCLUSION

In this paper, we proposed some initial steps towards efficient and accurate differentially-private analytics for high-speed research network measurement data. Our solution relies on Apache Hadoop and HBase big data tools. Query results are perturbed according to Laplace mechanism to achieve privacy. In future work, we plan to address the research challenges identified in Section 3.5.

5. REFERENCES

[1] R. Chansler, H. Kuang, S. Radia, K. Shvachko, and S. Srinivas. *The Architecture of Open Source Applications*, ISBN 978-1-257-63801-7. 2011.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design & Implementation*, pages 10–10, 2004.

[3] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

Row Key	CF4	CF5
ts|res|qn|q	noisyBC	noisyPC
ts|15|1|srcIP	noisyBC	noisyPC
ts|15|2|destIP	noisyBC	noisyPC
ts|15|3|port	noisyBC	noisyPC

Table 4: HBase schema for contingency tables