Scalable flow analysis

Abhishek Kumar Sapan Bhatia

Abstract

While current toolkits for analysis of flow-records such as SiLK are powerful and versatile, real-time analysis of flow records at very large flow collection installations continues to be a challenge. In this paper we present a new approach for summarization and analysis of flow records. Through the use of approximate data structures, a large bulk of flow records is reduced to a compact representation that is 100 times smaller in volume than the original flow records. The techniques are suitable for implenting a small number of predefined queries that are evaluated repeatedly in a periodic manner. The operations involved in summarization and query processing are fast enough to keep up with 2.5 million flows per second in a software implementation running on general purpose hardware.

I. Introduction

Collecting flow records is the dominant method for retaining state about the traffic observed at various links in a network. Large flow collection installations primarily face bottlenecks of four types. First, the generation of flow records from the packet stream is a challenging task, especially at the higher end of link speeds. Second, the transmission of the flow records to a central operations center is problematic due to the large (and unpredictable) volume of flow records. The third issue is that of the vast amounts of storage and its management required in large flow collection installations. Finally, analyzing a large number of flow records has a high computational overhead that often grows super-linearly with the number of flow records being analyzed together.

In this paper, we present a new approach to flow analysis. Instead of providing deterministic answers that are guaranteed to be precise and accurate, this approach uses probabilistic techniques and provides approximate answers to most queries. But relaxing the requirement for deterministic accuracy in the favor of probabilistic guarantees allows for a solution that is significantly faster and more efficient in its communication and storage requirements.

The approach presented in this paper is not meant to be a replacement for deterministic flow analysis tools such as SiLK [1], [2], [3]. Instead, we believe the two to be complementary . Deterministic analysis provides precise and unambiguous solutions, and provides a high degree of flexibility in the definition of the queries, albeit at high computational, storage and transmission overheads. The approach presented here trades off the deterministic guarantees for efficiency gains that allow such analysis to be performed at scales that are two to three orders of magnitude larger. The improved scalability of this approach implies that it provides an excellent means to perform routine analysis for a high volume of flow records, identifying a small number of suspicious cases that merit further investigation. It is also a suitable candidate to build situational awareness systems that evaluate a fixed set of queries over the flow data in a periodic fashion.

In the next section we provide a taxonomy of queries posed during flow analysis. Section III looks at an example query and provides a solution using the proposed approach. Section IV generalizes this solution to present a comprehensive summarization and analysis system for automated, routine analysis of flow data. We discuss the advantages and limitations of this approach in section V and conclude with pointers to future work in section VI.

II. Taxonomy of Queries

We divide the universe of queries that can be posed against flow records into three classes. First, there are the *aggregate queries*, that refer to various totals among the dataset in question. Typical examples of aggregate queries are estimating the total number of flows, the total number of sources by IP, the total number of destinations identified by IP alone or by `<destination IP, destination port, protocol>` tuples. Such aggregate quantities are the broadest and most important indicators capturing the details of network activity.

The second family of queries covers *distributional queries*, pertaining to questions such as “what is number of sources that have contacted exactly one destination within my network?” or “what fraction flows have less than 10 packets?”. The broadest distributional queries ask for the entire observed distribution of a metric, such as the distribution of flow sizes, or the distribution of number of destinations contacted by various sources. Answers to distributional queries are more precise and more sensitive indicators of various kinds of network activity and attacks. For example, a small DoS attack on a web-server might not show up as a large increase in the total number of flows, but it will cause a more significant change in the number of single-packet flows.

Finally, *identity queries* try to identify specific entities that are outliers according to some metric evaluated over the dataset. For example, “which sources have contacted more than a hundred unique destinations” or “which destinations have received more than two thousand flows” are queries that identify individuals that exceed respective thresholds for the number of unique destinations and the total number of flows.

In theory, there is an immense number of possible queries that can be posed against a given collection of flow records. But in practice, there are two kinds of queries that are actually evaluated. The first set is that of *routine queries* that are evaluated regularly, in a periodic manner, over the flow records collected in regular monitoring intervals. Typically this set includes a small number of queries of all three types described above. The results to these queries can be used to visualize the

Number of Sources	32	64	128	256	512	1024
Execution Time (sec)	9	50	581	922	1573	2464

TABLE I
TIME TAKEN TO EXECUTE AN EXAMPLE SILK QUERY OVER 512K (2^{19}) FLOW RECORDS.

current status of the network traffic, or in other words, provide situational awareness about the network traffic. Anomaly detection systems can track the results of such queries to detect and flag significant deviations from the norm, perhaps raising an alarm for an analyst to evaluate additional, non-routine queries. This second set represents what we call *forensic queries*. Such queries are typically part of an investigation prompted by specific network events or perhaps even an alarm raised on the basis of the information provided by the routine queries. The range and objective of forensic queries is significantly larger than that of routine queries. In this paper, we propose a more efficient approach for evaluating routine queries, keeping in mind that such queries are evaluated repeatedly and over all data that is collected. Forensic queries on the other hand, are evaluated less frequently , and are better implemented using traditional flow analysis tools such as the SiLKtools that can support a range of complex queries, albeit with significantly higher costs in computational, storage and transmission complexity. We now provide an example query and its probabilistic implementation. We will generalize from that implementation to present a broader system for routine analysis of flow data.

III. AN EXAMPLE QUERY

Consider a query for identifying sources that have contacted an unusually large number of destinations within the monitored network. For a large network, evaluating such a query periodically over the flow records corresponding to the inbound traffic can expose a variety of activity such as port scans, flooding attacks and automated worm or botnet spreading attempts. Tracking the set of heavy hitters in this query, i.e., the sources sending the largest number of flows into the network, and changes in this set, can yield a list of suspects warranting further investigation.

Using SiLK, an implementation of this query might look like:

```
%rwuniq --distinct-destinations data.raw
```

As demonstrated in Table I, the total running time for this query increases with the total number of sources. More significant is the fact that all the half million flow records in this example need to be available at the analysis station for this query to be evaluated. Also, the amount of memory required to implement this query and the total number of memory accesses both increase no slower than the total number of flows, the total number of sources, and the number of distinct destinations per source. In the event of an actual attack or scan, the situation is further exacerbated due to the increase in the number of distinct sources and/or destinations, often by several orders of magnitude, caused by such events.

We now describe a probabilistic solution implementing this query. Consider a simple array of counters, indexed by a hash function, such that the range of the hash function equals the number of counters in the array. Now for every flow record, a hash of the source IP can be used as an index into the counter array, to locate the corresponding counter. This counter can be incremented for every flow, thus accumulating the count of the total number of flows from the corresponding source IP. Sources sending a large number of flows can be identified by selecting the flows that cause any counter to reach a predetermined threshold value. This solution is approximate in the sense that collisions in hashing can result in counter values being incremented due to more than one source IP hashing to the same location. However, with a reasonably large array, say with 2^{18} counters, the probability of collisions among two large sources is quite small. Such data structures, also known as *sketches*, have been studied recently for their suitability in estimating various statistics about network traffic. We refer the reader to [4], [5] for a detailed treatment of the accuracy of estimation using such sketches.

This solution can track the total number of flows corresponding to each source, but requires another component to track the set of unique destinations contacted by each source. We implement this functionality by adding a Bloom Filter to estimate whether a `<source IP, destination IP>` tuple is unique. Bloom Filters [6] are probabilistic data structures capable of answering set membership queries in very efficient manner. To count the number of unique destinations contacted by each source in an approximate manner, a second counter array is maintained, but updated only for previously unseen `<source IP, destination IP>` tuples.

Figure 1 depicts the overall solution, consisting of two counter arrays and a Bloom Filter for determining new unique destinations for a given source. The first array provides the total number of flows per source and the second array provides the total number of unique destinations per source. The estimation techniques developed by Kumar et al. [4], enable the computation of the entire distribution of the number of flows sent by individual sources, and the number of unique destinations contacted by individual sources. This answers all distributional and aggregate queries about the statistics of total number of flows by source and total number of unique destinations by source. Finally, simple threshold sampling rules can be used to select a small number of records from sources that have sent more than T_{total} number of flows or contacted more than T_{unique} destinations. This allows for the identification of individual sources that have exceeded the set thresholds, thereby answering the identity queries about the outliers along these metrics.

In this section we have designed a solution that answers the aggregate, distributional, and identity queries for two metrics, the total number of flows by source and the number of unique destinations contacted by each source. The following section discusses how this approach can be further generalized to design a comprehensive system for routine analysis of flow data.

IV. A COMPREHENSIVE SYSTEM FOR ROUTINE ANALYSIS

The solution designed in the previous section used two arrays of counters and a Bloom Filter for resolving uniqueness

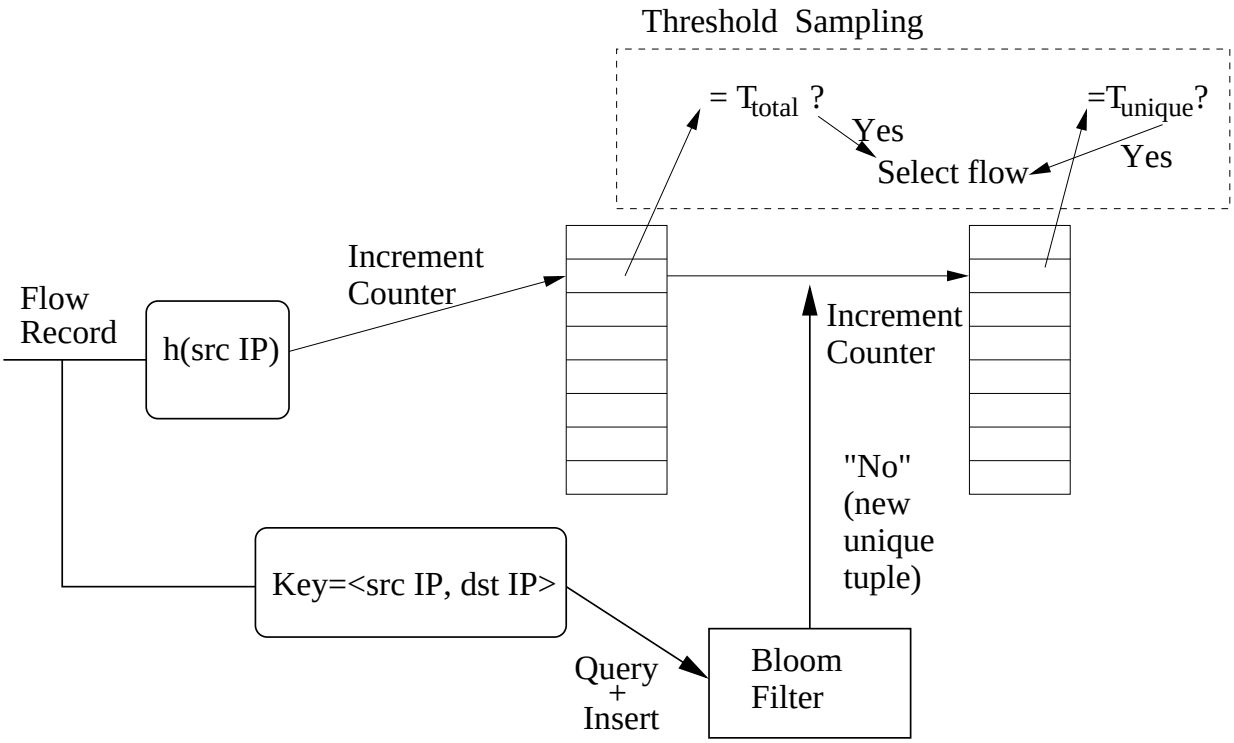

Fig. 1. Solution to track total flows and total unique destinations by source IP

Metric	Key field(s)	Aggregate Queries	Distributional Queries	Identity Queries
Bytes	Flow (5-tuple)	Total Flows	Flows with $x \leq$ bytes $\leq y$	Large flows
Packets	Flow (5-tuple)	Total Flows	Flows with x packets	Large flows
Total Flows	Source IP	Total sources	Sources sending x flows	Sources sending many flows ($> T_{total}$)
Unique Destinations	Source IP	Total sources	Sources contacting x destinations	Sources contacting many destinations (i.e., contacting $\geq T_{unique}$ destinations)
Total Flows	Destination IP	Total destinations	Destinations receiving x flows	Destinations receiving many flows
Unique Sources	Destination IP	Total destinations	Destinations contacted by x sources	Destinations contacted by many sources
Total Flows	DDP-tuple	Total DDP-tuples	DDP-tuples receiving x flows	DDP-tuples receiving many flows
Unique Sources	DDP-tuple	Total DDP tuples	DDP-tuples contacted by x sources	DDP-tuples contacted by many sources

TABLE II

METRICS TRACKED BY PROPOSED SYSTEM AND EXAMPLES OF SUPPORTED QUERIES.

of tuples to answer queries along two important metrics: the total number of flows and unique destinations by source IP. This solution can be generalized to comprehensively support routine analysis queries. Two observations facilitate this generalization: (i) The total number of metrics that are interesting enough to be tracked on a routine basis are very small, and (ii) A single Bloom Filter can be used universally to resolve all uniqueness questions.

Given the high dimensionality of flow data, it is somewhat counterintuitive that only a few metrics are of routine interest. Indeed, in a forensic investigation, an analyst might need to slice up the flow data along unusual dimensions. However, the key point here is that for *routine* analysis, only a small number of metrics are actually tracked. In practice, information about the eight metrics listed in Table II is adequate to provide situational awareness with enough sensitivity to flag most anomalies. This table also provides examples of queries supported by the sketches tracking the respective metrics. The abbreviation DDP-tuple stand for a `<destination IP, destination port, protocol>` tuple.

One may argue with the exact composition of the list of metrics in Table II, and we do believe that it can be improved with inputs from the community of analysts, but the interesting point here is that all the metrics in this list can be covered by eight corresponding arrays (sketches). The first two arrays, would track the number of packets and bytes on a per flow basis, in an approximate manner. To accommodate a huge number of flows in an array with fewer counters than flows, we can resort to multi-resolution techniques as presented in [4], using 1 million counters to track up to 50 million flows. The remaining six metrics can be tracked comfortably by arrays with 2^{18} or 256k counters each. With 32 bit counters, the overall size of these data structures would be 14 MB.

Threshold sampling would identify a small number of records that are interesting because they correspond to an outlier for one of the metrics. Note that for routine analysis, the goal is to identify a small number of interesting outliers. We assume that the detailed data will be available independently

for subsequent forensic analysis. For 50 million flows in a measurement, we conservatively estimate 2^{16} or 64K flow records, easily fitting within 2MB before compression, using a compact representation such as the SiLK raw format.

The second observation is that a single Bloom Filter can be used to determine uniqueness of tuples across the entire system. Bloom filters use an array of bits, or bit-vector, and a collection of k hash functions to insert and query for elements. For insertion, all k hash functions are computed and the corresponding bits are set to '1'. Upon being queried for a key, again all k hash functions are computed over the key, and the corresponding bits looked up. If all the bits are '1', the answer is "yes", i.e. the key was inserted previously, while if one or more bit is '0', the answer is "No", implying the key is a new unique item. Now, if the hash functions are chosen so they can take variable length inputs, the rest of the operations are transparent to the semantics of the keys, hence allowing us to use the same Bloom Filter for determining the uniqueness of various tuples with different fields. Continuing with the goal of processing 50 million flow records in one period, a Bloom Filter with a single hash function and 16 MB of storage would be sufficient to determine the uniqueness of various tuples with acceptable accuracy. If additional memory is available, these parameters can be tuned to provide optimal accuracy according to the method provided in [7]. Note that the Bloom filter is used as a local data structure used only while updating the sketches for each flow record. Once this process is complete, the Bloom Filter can be discarded; it does not need to be stored or transmitted.

The sketches and selected records corresponding to outliers together make up the summary representation of the flow records. Here, they correspond to a total size of 16 MB before compression. For our example of 50 million flows in an observation period, this is roughly 100 times smaller than the actual flow records and can be transmitted easily to a central analysis server. The analysis server can then compute various estimates over these sketches, feeding a situational awareness or anomaly detection system. The next section discusses the benefits and limitations of this solution approach.

V. Benefits and limitations

As identified before, the solution approach presented here is complementary to the more flexible flow analysis paradigm and tools available to analysts today. The benefits of this approach lie in its speed and succinctness, while its limitations lie in the restricted set of queries that such a system can support.

- **Speed:** The first major benefit of the proposed approach is its high speed. Updating the sketches involves a small number of hash computations and memory lookups. A software implementation running on a 2.0 Ghz dual-Opteron system can process 2.5 million records per second, preparing the 16 MB of summaries to be shipped to the central analysis server. Running estimation algorithms at the analysis server is equally fast, taking under one processor-minute per sketch.
- **Succinctness:** Large flow monitoring installations typically have a distributed deployment, with *collectors* deployed at various locations in a network collecting flow records from adjacent router(s) and packing them into more succinct intermediate formats, before transmission to a central analysis installation. The approach presented in this paper enables creation of succinct summaries at distributed collectors that are about 100 times more compact than the packed formats of SiLK. This changes the issue of transmission bandwidth (and storage at the central server) from a major concern to a trivial overhead. Indeed in current installations, the large volume of flow records being transmitted, especially during attacks, are a major overhead in large ISPs, and cited as one of the main reasons against the deployment of flow-collection in the core.
- **Low Flexibility:** The benefits of this approach come with a significant limitation - the analysis only addresses a set of predefined queries. This implies that any forensic investigation into network events is likely to require evaluation of queries not supported by the sketches generated into this system. But this problem can be addressed by retaining the complete flow records at the distributed "collectors" till such time when a forensic investigation requires records from the corresponding period to be "pulled" to the central analysis station. Since such investigations are likely to be infrequent, at least relative to the repetitive evaluation of routine queries, such a 2-tier solution will provide all efficiency benefits of the sketch-based system while making available to the analyst all the power and flexibility of conventional flow analysis tools during specific investigations.

VI. Conclusions

Routine flow analysis tasks that periodically evaluate a fixed set of queries over the flow data collected in the corresponding period can be made significantly more efficient if approximate answers are acceptable in lieu of deterministic accuracy. The approach presented in this paper delivers a 100 fold reduction in the amount of data sent to a central analysis server for routine analysis. Future work on this subject includes the identification of important metrics to track and performance study of a complete, deployed system.

References

[1] D. Kompanek and M. Thomas, "Silk analysis suite," http://sourceforge.net/projects/silktools, 2003.

[2] C. Gates, M. Collins, M. Duggan, A. Kompanek, and M. Thomas, "More netflow tools: For performance and security," in *Proc. of LISA XVIII*, Atlanta, GA, Nov. 2004.

[3] J. McHugh, "Sets, bags and rock and roll," in *Proc. of the Ninth European Symposium on Research in Computer Security*, Sept. 2004.

[4] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *Proc. ACM SIGMETRICS*, June 2004.

[5] A. Kumar and J. Xu, "Sketch guided sampling – using on-line estimates of flow size for adaptive data collection," in *Proc. of IEEE Infocom*, Apr. 2006.

[6] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *CACM*, vol. 13, no. 7, pp. 422–426, 1970.

[7] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Fortieth Annual Allerton Conference on Communication, Control, and Computing*, 2002.