Classification-assisted Query Processing for Network Telemetry

Gioacchino Tangari
Macquarie University
gioacchino.tangari@mq.edu.au

Marinos Charalambides
George Pavlou
University College London
firstname.lastname@ucl.ac.uk

Clara Grazian
University of New South Wales
c.grazian@unsw.edu.au

Daphne Tuncer
Imperial College London
d.tuncer@imperial.ac.uk

***Abstract*—Network telemetry systems are responsible for providing operators with the necessary information to secure and manage a network. As such, they should be able to respond, in real-time, to a number of queries on events ranging from performance impairments to attacks. While existing measurement techniques can swiftly collect all the relevant data from high-speed traffic streams, significant processing costs are faced when aggregating and evaluating such data to generate elaborate query responses. This paper explores the use of machine-learning approaches for reducing the run-time processing cost of monitoring queries. A novel processing workflow is introduced, which entails classifiers trained over recent traffic, automatically tuned to match accuracy requirements of query responses, and applied to sampled subsets of raw measurement data. Experiments with representative monitoring queries on recent CAIDA traffic traces demonstrate promising improvements on the query processing capability, by up to 3x, while maintaining accuracy levels above 98%.**

I. Introduction

Network telemetry is vital for network management and security. By extracting and processing a wide range of network metrics it enables relevant knowledge to be generated, which can be used for various tasks including anomaly detection [28], root cause analysis [4], and traffic engineering [22]. Modern network telemetry approaches [32], [13], [18], [16] follow the principles of *software-defined measurements* [31], [30], [21]. They provide a high-level interface that hides measurement details to register monitoring requirements, and automatically configure measurement operations to satisfy hardware constraints. In general, they operate as query processors that receive declarative monitoring commands or *queries*, extract raw measurement data from the traffic streams, and process the data to identify a variety of network events and report them in query responses.

Recent research has empowered network telemetry with the right tools to extract raw measurement data from traffic streams at line rate. Hashing techniques and sketch-based approaches [9], [10], [30], [14] have emerged as standard ways to collect measurement data with a limited computation and memory footprint. While these advances have contributed to the development of efficient extraction mechanisms, the task of *processing* the collected data remains a costly operation. As networks move to larger speed and scale, it becomes crucial to reduce this cost in order to cope with massive traffic volumes while providing real-time responses to heterogeneous sets of queries. The efficient processing of measurement data can enable dynamic resource management applications to operate on shorter reconfiguration cycles, and it allows for faster detection of network events – a key requirement to timely react to security threats.

The main methods proposed in the literature for improving the data processing efficiency have mostly been focusing on ad-hoc design optimizations, tailored to specific telemetry systems [22], [13], [24]. This paper presents a novel approach, applicable to a wide range of queries, for reducing the cost of monitoring query processing. In contrast to previous work, our approach leverages intelligent filtering of the raw measurement data collected through the monitoring pipeline, as opposed to system-specific optimizations. We use a methodology based on machine-learning classification algorithms to infer query results from small measurement subsets, and take decisions on the portions of data that can be proactively filtered prior to the execution of standard data processing operations. In particular, the proposed approach uses lightweight classifiers that *learn* traffic properties based on recent measurements. To protect query responses from classification errors, the classifiers are automatically configured to discard decisions with low confidence levels, according to operators' requirements on the query response accuracy.

To demonstrate its capabilities, we integrate our solution to a state-of-the-art traffic analysis tool based on software packet-processing [22], [1], and we experiment with representative query examples. The results, obtained with CAIDA traffic traces from 2018 [2] and using short 10-seconds *training* sets, show that large fractions of the extracted measurement data – more than 50% and even up to 90% in some cases – can be filtered out while satisfying accuracy requirements above 98%. This leads to substantial processing cost reductions and up to 3x improvements of the query processing capability.

II. Query-based network telemetry

Network telemetry systems must satisfy two fundamental requirements: (i) provide a generic, *declarative* interface, based on the definition of monitoring *queries*, and (ii) cope with the complexity of query processing, from compilation to the final push of monitoring reports, while satisfying stringent monitoring accuracy goals.

Query name	Description	Processing workflow
Heavy hitters [22]	A traffic aggregate (srcIP) exceeding volume K_{hh}	**extract** 5-tuple bytes, srcIP **aggregate** on srcIP (*sum* bytes) **evaluate** sum $> K_{hh}$
DDoS attack [30]	A host (dstIP) reached by more than K_{ddos} unique sources	**extract** 5-tuple srcIP, dstIP **aggregate** on dstIP (*count distinct* srcIPs) **evaluate** count $> K_{ddos}$
Slowloris attack [13]	A host (srcIP) opening more than K_{sl} connections, with average rate below B_{sl}	**extract** 5-tuple bytes, srcIP **aggregate** on srcIP (*mean* bytes, *count*) **evaluate** mean $< B_{sl}$ & count $> K_{sl}$
Bursty source [22]	A host (srcIP) generating more than $X\%$ *bursty* connections, *i.e.,* connections with more than $Y\%$ packets coming in bursts	**extract** 5-tuple, #pkts, #pkts-in-burst, srcIP **aggregate** on srcIP (*isBursty()*) **evaluate** $\%$(isBursty() $== 1) > X\%$

TABLE I: Representative monitoring queries

A. Monitoring queries

Monitoring queries are declarative commands issued to identify a variety of events related to the network performance and security. From a logical perspective, they entail combinations of three building blocks. The first is the *match & extract* component, which selects the portion of the network traffic being in the scope of the query, and extracts raw information based on packet size, header fields, or timestamp. The second one, *evaluate*, corresponds to the set of logical and arithmetic operations that check extracted information against a set of conditions (query predicates). The last component, *aggregate*, is the state aggregation function (e.g., sum, mean, count, stddev) applied to monitoring data for evaluation or reporting purposes. These components can be used to build complex query-processing workflows, and *aggregate-evaluate* functions can be iterated to check different predicates at different levels of data aggregation.

Table I reports representative monitoring queries that are used in this paper. Raw measurement data is extracted and stored for each 5-tuple flow as in [22]. This data is then periodically processed by *aggregate-evaluate* functions to produce query responses, based on a short (milliseconds or tens of milliseconds) query reporting period. These monitoring queries are further detailed below.

Heavy hitters (**HH**). This query returns all hosts (*i.e.,* source IP addresses) whose generated traffic exceeds a bytes threshold K_{hh} during the last query reporting interval. The results on heavy hitters are widely used by network management, e.g., for the detection of anomalies, to decide on traffic engineering configurations, to unveil server load *inbalance* in data centers.

DDoS attack (**DDoS**). This query returns DDoS victims [30], *i.e.,* those hosts (destination IP addresses) that have been contacted by more than K_{ddos} other hosts (source IP addresses).

Slowloris attack (**Slowloris**). Slowloris belongs to the category of "slow" denials of service. This attack consists in sending data over a large number of connections all with a very slow rate, without hitting the idle connection timeout value on the server. The query returns Slowloris-attack sources, *i.e.,* those source IP addresses generating more than K_{sl} connections, all with byte rate below B_{sl}.

Bursty flow source (**Bursty**) A TCP / UDP flow is *bursty* if more than $Y\%$ of its packets come in a burst [22], *i.e.,* with short interarrival times. This query returns bursty flow sources, *i.e.,* hosts (source IP addresses) for which more than $X\%$ of generated flows are bursty. The information about bursty flow sources can serve several network management tasks, for example the analysis of Incast [7] congestion (by correlating such information with packet loss).

B. The challenge of efficient query processing

With networks evolving to larger scale and higher traffic speeds, the challenge for telemetry systems is to handle heterogeneous sets of queries on short timescales (e.g., 10ms epochs [22]) while facing large numbers of concurrent flows (1000+ on 10ms intervals [25]) and increasing packet rates (10Gbps+ on a single processor core in software-packet processing platforms [22], [14]). As a result, a vast amount of information needs to be processed to build query responses, especially for stateful monitoring [24], [22] (e.g., groupby-like state aggregation), with significant time and computation consumed for the *aggregate* and *evaluate* execution.

Recent research has investigated how to efficiently match-on-traffic and extract raw measurement data from a packet stream at line rate and with reduced memory footprints. To this end, sketch-based techniques [30], [20], [14], [15], heap-based solutions (e.g., top-k [19] counting), or simple hash tables [22], [3] can be adopted. However, it is still unclear how to curb the cost of query processing when looking at the aggregation and evaluation of such amounts of measurement data. Ad-hoc design optimizations have been proposed to improve efficiency, e.g., [13], [24], [22], but their applicability is limited to specific systems/implementations or is bound to the use of programmable hardware-switch architectures.

Generally speaking, different approaches can be explored to generate query responses at a reduced cost [6]. One possibility is to reduce the pace at which raw data is processed for reporting the query results. However, performing the *aggregate-evaluate* operations less frequently leads to increasing accumulation of measurement data over longer periods of time, which does not result in substantial benefit. In Fig. 1, we test

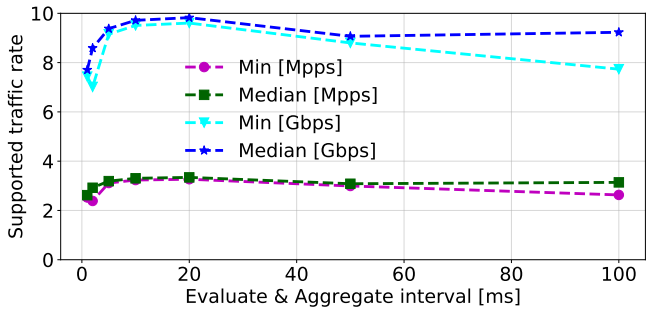


Fig. 1: Effects of tweaking the reporting frequency

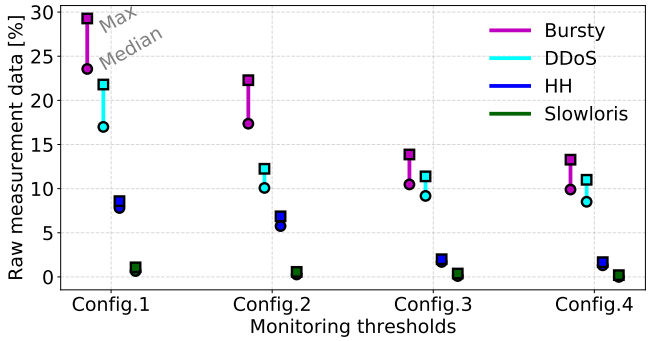


Fig. 2: % of raw data retained in query responses

the impact of relaxed query-results reporting on a software packet-processing pipeline based on Trumpet [1], using a single CPU core[1]. As input, we use 10min of CAIDA traffic [2] and apply the queries of Table I. To quantify the query processing cost, we analyze how it affects the monitoring performance in terms of supported traffic rate. As shown in Fig. 1, increasing the reporting period from $1ms$ to $10ms$ (Trumpet default) and $20ms$ improves the maximum traffic rate by approx. 20% and 25%, respectively. Going beyond $20ms$, the performance declines as more raw data accumulates prior to processing – more CPU cycles are consumed by aggregate-evaluate operations.

An alternative approach is to filter the original (raw) information sets considered in the *evaluate* and *aggregate* steps. This is promising when only fractions of the raw measurement data are retained to craft query responses. In fact, this applies to all queries concerning events such as unusual or suspicious behaviors or specific traffic patterns [22], [13], [32]. Fig. 2 shows the percentage of raw measurement data being used for query responses on the same setup of Fig. 1. To make query results more or less selective with respect to the total extracted measurement data, we vary the thresholds of Table I[2]. As observed, despite differences between queries, large fractions of raw measurement data (65%+) could always be scrapped without influencing the query responses, with peaks above 90% for the most selective configurations.

As the example shows, there is potential in reducing the cost of query processing by filtering the raw measurement data before the *aggregate* and *evaluate* steps are executed. However, to exploit this a few key challenges need to be addressed. First, a solution is required, applicable to a wide range of monitoring queries, to accurately predict portions of the query results, thus enabling intelligent filtering of the raw data. Second, this should guarantee significant processing cost reductions with respect to standard *aggregate* and *evaluate* operations. Lastly, the accuracy of query results should not be compromised. We address these challenges with a novel, *classification-assisted*, query processing approach, which is described in the next Section.

III. Classification-assisted query processing

To achieve intelligent filtering on the raw measurement data, we enhance the standard query processing workflow with a *classification* functionality, whose goal is to reduce the volume of data in input to standard *aggregate* and *evaluate* steps. The proposed methodology relies on the ability to derive classifiers for each query type in order to predict query results from subsets of the raw measurement data. In particular, classifiers are constructed by *learning* traffic properties from recent measurement results.

The main motivation for exploring a machine-learning approach is the flexibility towards heterogeneous monitoring queries and measurement datasets. Existing techniques for predicting traffic monitoring results generally suffer from being query-specific. For example, ProgME [31] provides a method for early Heavy Hitter identification based on testing of probability ratios, while the work in [17] defines a prediction algorithm, namely Threshold Random Walk, specific to Port Scan detection. In contrast, off-the-shelf machine learning classifiers do not require any particular knowledge on the learning datasets, and can automatically build models for measurement data following unknown distributions. This makes them suitable to a wide range of monitoring queries.

Overview The proposed methodology is presented in Fig. 3. The core of our solution is a set of machine-learning classifiers, one for each query, which take samples of the raw measurement data as input and generate predictions on query-related events. Each classifier uses a small subset of the raw counters obtained from *match & extract* operations as features for the prediction algorithm. For example, for a Heavy Hitters query, the classifier receives a sample containing the byte count of few 5-tuple flows with the same srcIP x, and returns a probabilistic answer to the predicate *is x a heavy hitter?*. Classifiers are trained on recent monitoring query results, *i.e.,* using labelled samples (*i.e.,* for which the ground-truth label is known) extracted from recent measurement data. The output of classifiers is the key to exclude portions of the raw measurement data from standard *aggregate* and *evaluate* processing. For instance, answering predicate *is x a heavy hitter?* allows raw counters matching srcIP x to be filtered

[1]We use a 3GHz CPU, with 8MB shared L3 cache

[2]We select 4 threshold configurations between the ranges $K_{hh} \in [1KB, 100KB]$, $K_{ddos} \in [10, 100]$, $X_{bursty} \in [10, 100]$, $K_{sl} \in [5, 50]$

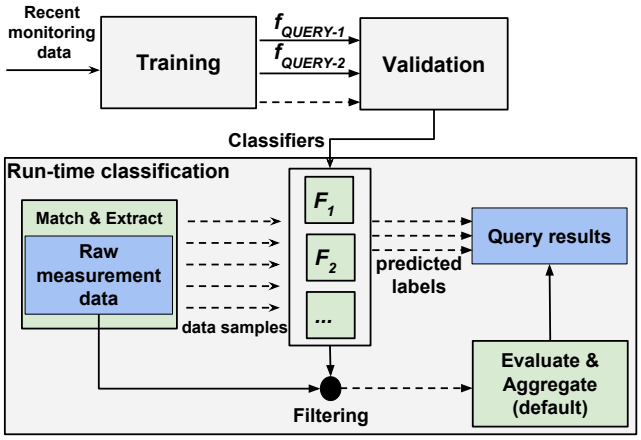
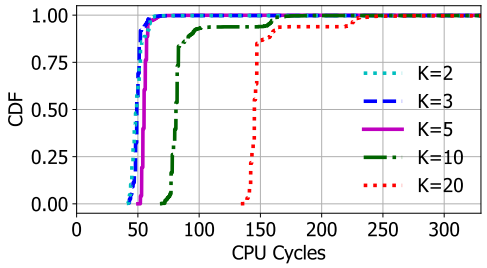
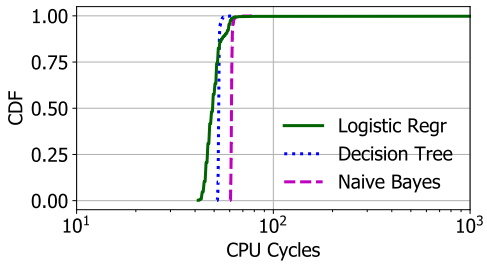


Fig. 3: Classification-assisted query processing



(a) Logistic-regression classification, different values of K



(b) Different classification algorithms

Fig. 4: CPU overhead (cycles) of individual classifications.

out, and in case of positive output to proactively add x to the query response.

The design of the classifiers should satisfy two key requirements. First, classifiers should be computationally inexpensive, so that the benefits from raw data filtering are not undermined by the overhead of run-time classifications. To this end, we experiment with small-sized samples only and we adopt a logistic regression model whose run-time execution mainly consists of a single exp function. Second, query responses should effectively be protected from the erroneous predictions a classifier may generate. As such, classifiers are automatically configured to match user-specified requirements on the query result accuracy. Based on this configuration, obtained from an offline *validation* procedure, part of the classifier output is automatically rejected when the estimated error risk is not negligible. More specifically, each classifier output can fall under three cases: (i) the input sample belongs to a positive query result (e.g., classifier decision = x is a heavy hitter); (ii) the sample corresponds to a negative query result (e.g., decision = x is not a heavy hitter); (iii) the decision on the query result cannot be taken with a high level of *confidence* (possible error). Only in case (iii) the portions of measurement data to which the sample pertains are directed to standard *aggregate-evaluate*, while in case (i) the predicted result is directly included in the query response.

Workflow As depicted in Fig 3, our workflow includes three phases: *training* phase, where the classification functions are built, *validation* phase, where classifiers are configured to preserve query result accuracy, and *run-time classification* phase, where classifiers are run in the wild to take decisions on incoming traffic. These are detailed below.

A. Training

The training phase is divided in two steps: (i) *sampling*, which extracts training information (training sets) from recent measurement data, and (ii) generation of classification functions using the obtained training sets.

Sampling The training data is drawn from recent sets of raw measurements, generated over previous query reporting intervals, that have been processed through standard *aggregate-evaluate* operations. Specifically, the training set is created by sampling recent sets of raw measurements and by adding to each sample its ground-truth label represented as a binary indicator, e.g., the sample corresponds to a heavy hitter (label 1) or not (label 0).

To make classifiers more efficient, sampling should ideally be a query-dependent operation. This is because the size of the samples and the way they are extracted (e.g., transformation of the raw measurement set) depend on specific query characteristics, such as its execution workflow or how selective it is. In this work, we propose two sampling methods that cover a wide range of queries.

The first method is used when *aggregate-evaluate* operations contain a *cardinality*-based predicate, e.g., for *DDoS* in Table I. In this case, a random subset of the raw measurement sets is selected based on sampling factor k. The extracted values are then grouped on the *aggregate* key, e.g., the DstIP in DDoS detection, and each sample in the training set is a tuple containing the cardinality of a group and the associated ground-truth label. The second method covers cases where query predicates include aggregation functions such as mean, sum, stddev e.g., for *HH* in Table I. In this case, the initial set is first grouped on the *aggregate* key, e.g., the srcIP for *HH*. For each group, K values are then randomly selected.

Classification functions Once formed, the training set is used to build the classification functions. These are modelled following the *sigmoid* function of logistic regression $\frac{1}{1+e^{-z}}$, where z is a linear function with parameters $\theta_0, \theta_1, ..., \theta_K$ and K is the sample size. Such a model is selected due to its high interpretability, independence from complex tuning, and

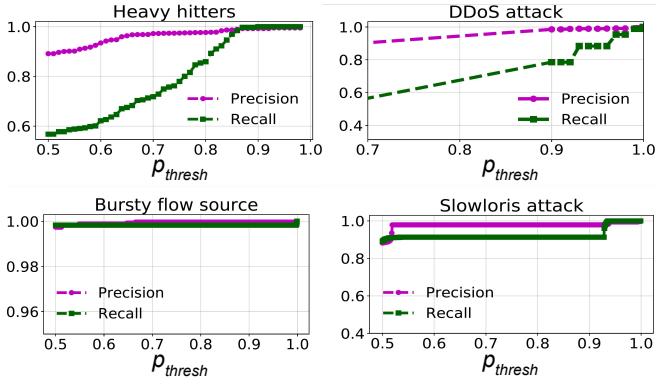


Fig. 5: Sensitivity analysis performed in *validation*

low computational cost. For each input sample, the trained model returns: (i) the predicted label $l_{predicted}$ (1 if the sample participates to a query-related event, 0 otherwise), and (ii) the probability estimates p_0, p_1 associated with each label (with $p_0+p_1 = 1$). Values of p_1 (p_0 respectively) indicate how likely a sample is to be labelled as 1 (0 respectively) in the ground-truth, and are used to quantify the confidence on individual classification decisions.

Cost of classification While the output of classifiers allows to reduce the data processing volume, the classifier execution itself requires some computation. Here, we show that only a limited execution cost is incurred by the model in use, and that the benefits acquired from raw data filtering are not voided by the classifier run-time. To quantify the cost of classifiers, we measure the CPU usage of individual classification executions. Results are shown in Fig. 4a in terms of CPU cycles for different values of the sample size K.

As depicted in the figure, the cost is $O(K)$ for $K \geq 10$. When $K \leq 5$, the cost is driven by the computation of the exp function (only significant operation). Note that 50 cycles ($K \leq 5$), corresponding to approximately $17ns$ on the server used[1], is no more than 25% of the time consumed for a single packet at 14.8Mpps (10 Gbps of small packets). However, classifications do not operate at the packet granularity, but at the one of query aggregation keys. Assuming query results at the 5-tuple flow granularity (worst case) and a reporting period of 10ms, the classifier overhead is $\frac{17ns \cdot N_{flows}}{10ms}$, close to 0.1% of CPU time for 1000 active flows in the 10ms intervals [25]. Furthermore, significant overhead reductions (up to 45%) can be achieved by pre-computing values of the sigmoid function $\frac{1}{1+e^{-z}}$, thus reducing the execution of the classifier to the update of z plus a small array lookup. Overall, this results in no more than 1 - 1.5% CPU overhead when 20 different measurement queries[3] are applied to the totality of traffic. While smaller sample sizes can further curb the classification overhead, the choice of smaller K can degrade the classification accuracy. We chose $K = 5$ for all experiments in Section IV, which guaranteed reasonable CPU overhead and only incurred limited accuracy reductions (less than 1%) compared to a larger sample size ($K = 20$).

Note also that no substantial gain can be obtained by choosing a different classification model. As shown in Fig. 4b, the run-time overhead of logistic-regression classifiers is slightly less than that of other popular (and simple) models such as Decision Tree and Naive Bayes. These results are obtained for $K = 5$ – additional costs similar to the ones in Fig. 4a can be also expected for these algorithms in case of larger sample size K, e.g., due to increasing Decision Tree depth, or more parameters required by Naive Bayes to estimate the probability of the sample belonging to a positive query result.

B. Validation

The goal of the *validation* phase is to refine the trained classifiers so that accuracy requirements of query responses can be met. Since the classifications are not error-free by definition, accepting all $l_{predicted}$ labels in output would result to incorrect query results. Intuitively, errors are particularly likely when the confidence on the classification is low, for instance when $(p_0, p_1) = (0.55, 0.45)$.

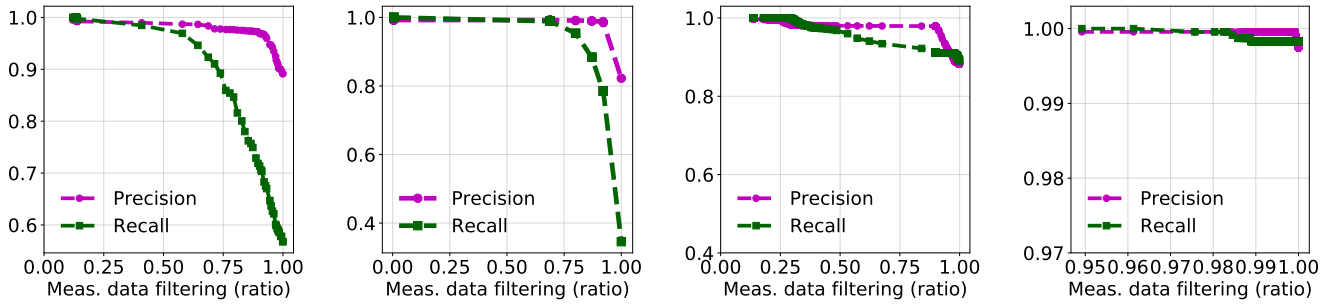
To correctly tune the classifiers, we need to determine the right ranges of probability estimates p_0, p_1 under which the predicted labels $l_{predicted}$ can be safely accepted. Specifically, a probability threshold p_{thresh} should be selected, such that classification results are accepted only if $max(p_0, p_1) \geq p_{thresh}$. A value $p_{thresh} = 0.5$ corresponds to accepting all $l_{predicted}$ labels. This setup maximizes the filtering, *i.e.,* the amount of raw data excluded from *aggregate-evaluate*, but at the same time it also maximizes the risk of injecting errors in the query responses.

The tuning is conducted by analyzing the relation between the accuracy of query responses and the different p_{thresh} setups. To this end, we use additional sets of labelled samples (*validation* sets) whose total volume is below 25% of the training set [4]. Accuracy is quantified using the *Precision*, *i.e.,* the ratio of the monitored query-related events that are *true*, and the *Recall*, *i.e.,* the fraction of detected *true* events. Fig 5 shows examples of this analysis for the queries in Table I. As observed, the value of p_{thresh} can have a significant impact on the query response accuracy, especially on the Recall (many events missed by the queries). In addition, different queries follow different trends, which indicates that query-specific p_{thresh} settings are required.

While several methods can be considered to select the value of p_{thresh}, the main approaches in the literature [23] [11] rely on *Receiver Operator Curve* (ROC) analysis. Specifically, different values of p_{thresh} determine different operating points on the ROC, and choosing the optimal p_{thresh} corresponds to finding a optimal ROC cut-off. A widely-used approach is to select the cut-off maximizing Youden Index [29], or equivalently, the highest Sensitivity+Specificity point of the curve. However, solving such an optimization problem for

[3]This cardinality is in line with the number of different measurement queries/applications used in [32] (*i.e.,* 17) or [13] (*i.e.,* 11)

[4]The choice reflects standard machine learning practices, where training/validation sets are 80/20 splits of total labelled-samples set



(a) Heavy hitter (b) DDoS attack (c) Slowloris attack (d) Bursty source

Fig. 6: Filtered raw measurement data

each monitoring query can be daunting. To reduce complexity, we resort to testing for each classifier a random set of p_{thresh} values in $[0.5, 1.0]$. After each test, the obtained Precision and Recall values (Fig 5) are compared against operator-specified thresholds. The selected p_{thresh} value is the lowest for which both Precision and Recall meet the requirements. For example, in the case of *HH* query in Fig 5 and for 98% Precision and Recall requirements, a value $p_{thresh} \approx 0.9$ is selected.

C. Run-time classification

Trained and configured classifiers are applied in the wild to pre-process the raw measurement data extracted for each active query. The run-time classification starts with the extraction of small samples of the raw-measurement set, performed through the same mechanism as the one adopted in the training phase. Each sample, corresponding to a specific *aggregate key* (e.g., a specific srcIP for HH detection), is then passed as input feature set to the query's classifier, which returns a pair $[l_{predicted}, (p_0, p_1)]$. If $max(p_0, p_1) \geq p_{thresh}$, the result is accepted, *i.e.,* the predicted query result $l_{predicted}$ is accepted and the portion of raw data matching the sample *aggregate key* is discarded. Otherwise, the measurement data is redirected to the *aggregate-evaluate* functions for standard processing.

IV. Experimentation with multi-Gbps traffic analysis

We experiment with the proposed classification-assisted approach by applying it on a multi-Gbps traffic analysis tool. Our study is conducted in three steps. At first, we quantify the filtering of raw measurement data achieved by classifiers at run-time. Then, we investigate the resulting cost reductions in the processing of different monitoring queries. Lastly, we explore the extent to which the approach can meet stringent operator requirements on the accuracy of query responses. The testbed and datasets used for the experiments are described below.

Testbed We integrate classifiers with a multi-Gbps traffic inspector based on software packet-processing, which is running on a single CPU core[1]. The tool is based on the Trumpet framework [1], [22] and relies on a hash table, indexed on the flow 5-tuples, to buffer raw measurements extracted from the traffic stream. Query responses are periodically generated based on a fixed reporting interval $T = 20ms$. During each interval, measurement data samples (obtained from sampled flow-entries) are applied to the trained classifiers. If the classification decision is accepted, the flow-entries matching the *aggregate key* of the sample are excluded from further processing, and the predicted label is kept for the query response. For the remaining flow-entries, measurements are instead grouped by *aggregate key* and checked against the query predicates, which corresponds to the baseline *aggregate-evaluate* workflow.

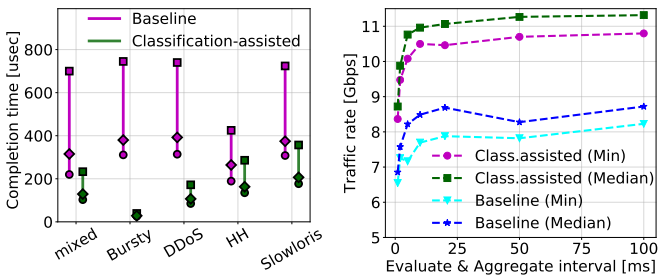
Datasets and training setup Experiments are performed using 1 hour of CAIDA traffic from the passive-2018 dataset [2]. Specifically, we rely on 60 different CAIDA pcap traffic traces, each of 1-minute duration, captured on a high-speed backbone link – more than one million packets per second – from New York to Sao Paulo.[5]

For each experiment, the classifiers are trained for 10 seconds, *i.e.,* using 10 consecutive seconds of CAIDA traffic, reserving the last 2s for *validation*. For the four queries in Table I, the average p_{thresh} values obtained from the validation are 0.86 (Heavy Hitters), 0.97 (DDoS attack), 0.92 (Slowloris attack) and 0.99 (Bursty Source). The classifiers are then used to process the following 30 minutes of traffic. Lastly, the default sampling setup used in the training phase is $k = 10\%$ and $K = 5$.

A. Measurement data volume

At first, we assess the potential of our classifiers to filter large data volumes (based on their prediction output) while generating accurate query responses. To this end, we run experiments with different p_{thresh} setups and, for each one, we measure the ratio of *filtered* measurement data, as well as the Precision and Recall of final query responses. The results are depicted in Fig. 6 and show that with a query response accuracy above 98% (both Precision and Recall), 50%, 55%, 70% and 98% of measurement data can be safely filtered for

[5]CAIDA traffic traces are widely used to evaluate traffic measurement approaches, and have been adopted in recent proposals on network telemetry such as [13] (10 minutes total CAIDA traffic used) and [32] (1 minute total CAIDA traffic used)



(a) $t_{proc.}$ (min, mean, max) (b) Supported traffic rate

Fig. 7: Benefits on the monitoring pipeline used

Slowloris, Heavy Hitters, DDoS, and Bursty Source queries, respectively. The levels of filtering are significant, despite using a small 10s training set, but different query behaviors can be observed. The different behaviors reflect how difficult it is for classifiers to discriminate between positive and negative samples, which in turn depends on query configurations and recent traffic characteristics. Interestingly, the query selectivity alone cannot explain the different filtering ratios. For example, Slowloris is the most *selective* query in Fig. 2 (*i.e.,* only a small fraction of the measurement data is retained to craft the query response), however no more than 50% of its measurement data can be filtered while meeting accuracy requirements.

B. Query processing cost

We now investigate the benefits of the classification-assisted approach on the query processing cost. Since our implementation is based on software packet-processing, such cost can be quantified by measuring the CPU time (t_{proc}) consumed to craft query responses from the raw measurement data extracted during each reporting interval. To experiment with different query workloads, we split the flow-address space and assign one query to each /12 prefix. In a *mixed* workload the query is randomly selected from Table I, while in other workloads all considered queries are of the same type. For all experiments performed, the classifiers are configured to achieve 98% Precision and Recall.

As shown in Fig. 7a, nearly 60% of CPU time is saved on average for a *mixed* workload, while the gain obtained for specific query types is in accordance with the filtering ratios in Fig. 6, e.g., the processing time for *Bursty* is reduced by more than 10x on average, as more than 90% of measurement data is filtered by classifiers. Reduced processing time can translate to more query responses handled over a reporting interval. In particular, assuming a constant time for raw data extraction[6], the guaranteed (minimum) number of simultaneous queries supported by our implementation is more than 3x higher than the baseline in *mixed* workload conditions.

Reduced query processing times can also improve the traffic speed supported by traffic analysis. To evaluate this, we split the experiments in 1 minute chunks, and for each chunk we measure the maximum traffic rate the monitoring implementation can cope with, *i.e.,* without dropping packets. As Fig. 7b shows, the classification-assisted approach can significantly speed up the monitoring pipeline, with gains in traffic rate up to 30%. Such gains (e.g., $+3$ Gbps) are also significantly higher than the ones operators could obtain (see Sec.II-B) by fine-tuning the reporting interval ($\leq +1.5$ Gbps).

[6]In our implementation, this time is dominated by hashing and flow-entry retrieval executed for each packet, irrespectively of the query workload

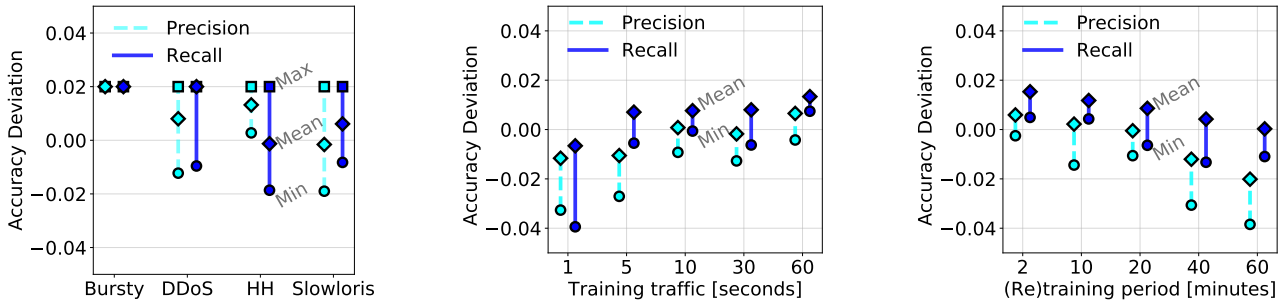
C. Query response accuracy

Finally, we evaluate the extent to which the classifiers can meet stringent requirements on the query response accuracy. We select a target accuracy of 98% (Precision and Recall) to be used in the validation phase, and for each 1-minute chunk we compute the deviation from this threshold. For example, a $+0.01$ Recall deviation corresponds to 99% Recall. As shown in Fig. 8a, our approach satisfies, on average, the desired accuracy levels, despite some differences between queries. Interestingly, negative deviations do not exceed -0.02 with the default training setup, *i.e.,* $10s$ training set for $30min$ traffic. However, different choices on (i) the amount of traffic (in seconds) used for training (Fig. 8b) and, (ii) the classifier re-training frequency (Fig. 8c), can have a noticeable impact on accuracy: up to 4% negative deviations recorded when training is infrequent, e.g., every hour, or when it relies on very small datasets, e.g., only 1s of traffic. Overall, the classifier training phase should be quick and lightweight in terms of dataset size in order to support the online re-training of classifiers on recent traffic. More extensive training could require more time and computation and as such undermine the resource savings from classifications, and based on Fig. 8b it would also bring limited accuracy improvements in our experimental settings.

V. Limitations of this study

Supported query types To be supported by the proposed workflow, a measurement query should meet two requirements. First, each query predicate (*i.e.,* each condition in the *evaluate* block) should produce a binary output (e.g., is srcIP x a heavy hitter (1) or not (0)?), since our solution relies on binary classifiers. Second, the conditions in the query predicates should be on raw counts (e.g., number of packets, flows, bytes) or on aggregate values such as mean or sum. While most of the monitoring queries considered in recent work [13] [22] [32] [24] satisfy these two conditions, several exceptions exist. Representative examples of unsupported queries are the *Flowlet size histogram* query in [24], which returns the histogram over length of flows, and the *Lifetime of connections* query in [32], which reports the duration of TCP connections. In addition to the above requirements, another key condition is the existence of labelled data for the query. In particular, if the training data contains no example of the query-related event, the query cannot be supported.

Impact of query selectivity By relying on data filtering to curb the query processing cost, our approach is highly dependent on the query selectivity levels. To obtain substantial savings, it requires that only small subsets of the measurement



(a) vs. query type (default training setup)

(b) vs. training traffic (*mixed* query workload)

(c) vs. re-training period (*mixed* query workload)

Fig. 8: Accuracy deviations from 98% Precision and Recall targets

data are retained to craft query responses, which in turn depends on the thresholds used in the query predicates. In particular, as shown in Fig. 2, less selective query configurations would result in smaller amounts of measurement data that can be *safely* discarded, *i.e.,* without incurring accuracy degradation. Hence, this would negatively impact the trade-off between resource savings and query response accuracy.

Impact of dynamic traffic patterns Experiments on CAIDA passive traces have shown that classifiers achieve high query response accuracy using only few seconds traffic as training data and then running with no further training for several minutes. However, in real network telemetry systems the classifiers could be exposed to more variable traffic patterns compared to the traces we used. This can negatively impact the query response accuracy if classifiers are not re-trained with an adaptive re-training period that follows the traffic dynamics.

VI. Related work

Sampling-based traffic monitoring Sampling techniques are widely used for measuring and analyzing network traffic with limited CPU and memory resources. A number of approaches relies on simple packet-based sampling (e.g., Netflow [8]), where on average 1 every N (e.g., 100, 1000) packets is captured and analyzed, to characterize traffic matrices and the flow size distribution [12] [27]. Other proposals applies more complex sampling schemes such as *flow-based* sampling (where only subsets of the flow space are tracked, e.g., based on hash ranges [26]) or *per-flow* packet sampling (where only a portion of every flow is captured, e.g., [5]).

Our approach also adopts sampling for the run-time traffic monitoring operations. However, while in previous work the sampling strategy acts on what packet or flow to capture/record, in our proposal all the raw data needed for the measurement queries are collected (*i.e.,* from all packets/flows), and sampling is used when processing such data to craft query responses. Compared to previous work, the advantage is that when the extracted data samples are not highly predictive for the query results (based on machine-learning classifiers), our workflow can still resort to the *standard* processing of the total set of measurement data, *i.e.,* with no sampling. This protects the telemetry system from the loss of monitoring result accuracy that is generally incurred by sampling-based measurement techniques [26] [27].

Network telemetry systems Several network-telemetry systems have been recently proposed [13] [24] [32] [22], which rely on declarative, query-based interfaces to express a range of different telemetry tasks. To support larger traffic volumes and query workloads, these proposals provide different, system-specific, solutions. Sonata [13] splits *aggregate-evaluate* load between programmable switches and telemetry stream collectors at servers. Marple [24] relies on a memory backend on commodity hardware to facilitate stateful *group-by* operations. Trumpet [22] adopts double-buffering of measurement data to interleave the *aggregate-evaluate* steps on previous data with *match & extract* on new traffic. In contrast with these system optimizations, our proposal consists in a new query processing workflow that achieves processing cost reductions through intelligent data filtering.

VII. Conclusion

We have investigated a novel approach, applicable to a wide range of monitoring queries, to reduce the cost of measurement data processing. This relies on the ability to train lightweight machine-learning classifiers used to achieve intelligent filtering of data. Experiments conducted on representative query examples and recent CAIDA traffic traces have shown that such an approach can effectively reduce processing workloads while preserving reporting accuracy, even when using no more than 10s training for 30min traffic. Despite the large volume of traffic data on which our approach was tested, the use of a single set of traces still limits our conclusions on the applicability of the approach under more heterogeneous traffic conditions. We plan to address this limitation in future work by experimenting with different sets of real-world traffic traces, as well as on extensive sets of synthetic traffic patterns. Future work will also investigate the applicability of the approach to broader query sets, and it will explore how to dynamically set the training duration and re-training frequency based on query and traffic properties (e.g., for different times of the day).

REFERENCES

[1] "Trumpet," 2016. [Online]. Available: https://github.com/USC-NSL/Trumpet

[2] "The caida ucsd anonymized internet traces dataset - march 2018," 2018. [Online]. Available: http://www.caida.org/data/passive/passive_dataset.xml

[3] O. Alipourfard *et al.*, "Re-evaluating measurement algorithms in software," in *Proceedings of HotNets-XIV*, 2015.

[4] B. Arzani *et al.*, "Taking the blame game out of data centers operations with netpoirot," in *Proceedings of SIGCOMM '16*, 2016.

[5] M. Canini, D. Fay, D. J. Miller, A. W. Moore, and R. Bolla, "Per flow packet sampling for high-speed network monitoring," in *COMSNETS'09*, 2009.

[6] S. Chaudhuri *et al.*, "Approximate query processing: No silver bullet," in *Proceedings of SIGMOD '17*, 2017.

[7] Y. Chen *et al.*, "Understanding tcp incast throughput collapse in datacenter," in *Proceedings of ACM WREN '09*, 2009.

[8] B. Claise, "Cisco Systems NetFlow Services Export Version 9. RFC 3954."

[9] G. Cormode *et al.*, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, 2005.

[10] G. Cormode. *et al.*, "Finding frequent items in data streams," *Proc. VLDB Endow.*, vol. 1, no. 2, Aug. 2008.

[11] C. Cortes and M. Mohri, "Auc optimization vs. error rate minimization," in *Proceedings of NIPS'03*, 2003.

[12] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 933–946, 2005.

[13] A. Gupta *et al.*, "Sonata: Query-driven streaming network telemetry," in *Proceedings of SIGCOMM '18*, 2018.

[14] Q. Huang. *et al.*, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of SIGCOMM '17*, 2017.

[15] Q. Huang *et al.*, "Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference," in *Proceedings of SIGCOMM '18*, 2018.

[16] IETF, "In-situ Operations, Administration, and Maintenance (IOAM)," https://datatracker.ietf.org/wg/ioam/about/, 2018.

[17] J. Jung *et al.*, "Fast portscan detection using sequential hypothesis testing," in *Proceedings of IEEE Symposium on Security and Privacy*, 2004.

[18] A. Khandelwal *et al.*, "Confluo: Distributed monitoring and diagnosis stack for high-speed networks," in *Proceedings of NSDI'19*, 2019.

[19] A. Metwally *et al.*, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of ICDT'05*, 2005.

[20] M. Moshef *et al.*, "Scream: Sketch resource allocation for software-defined measurement," in *Proceedings of CoNEXT '15*, 2015.

[21] M. Moshref *et al.*, "Dream: Dynamic resource allocation for software-defined measurement," in *Proceedings of SIGCOMM '14*, 2014.

[22] M. Moshref. *et al.*, "Trumpet: Timely and precise triggers in data centers," in *Proceedings of SIGCOMM '16*, 2016.

[23] M. Mozer *et al.*, "Prodding the roc curve: Constrained optimization of classifier performance," in *Proceedings of NIPS'01*, 2001.

[24] S. Narayana *et al.*, "Language-directed hardware design for network performance monitoring," in *Proceedings of SIGCOMM '17*, 2017.

[25] A. Roy *et al.*, "Inside the social network's (datacenter) network," in *Proceedings of SIGCOMM '15*, 2015.

[26] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "Csamp: A system for network-wide flow monitoring," in *NSDI*, 2008.

[27] V. Sekar, M. K. Reiter, and H. Zhang, "Revisiting the case for a minimalist approach for network flow monitoring," in *IMC'10*, 2010.

[28] K. Xie *et al.*, "On-line anomaly detection with high accuracy," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, 2018.

[29] W. Youden, "Index for rating diagnostic tests," 1950.

[30] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings of NSDI'13*, ser. nsdi'13, 2013.

[31] L. Yuan *et al.*, "Progme: Towards programmable network measurement," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, Feb. 2011.

[32] Y. Yuan *et al.*, "Quantitative network monitoring with netqre," in *Proceedings of SIGCOMM '17*, 2017.