Industry Paper: Continuous Distributed Monitoring in the Evolved Packet Core

Romaric Duvignau
Marina Papatriantafilou
{duvignau,ptrianta}@chalmers.se
Chalmers University of Technology
Gothenburg, Sweden

Konstantinos Peratinos
konstantinos.peratinos@gmail.com
Chalmers student and Ericsson intern
Gothenburg, Sweden

Eric Nordström, Patrik Nyman
eric.r.nordstrom@ericsson.com
patrik.nyman@ericsson.com
Ericsson
Gothenburg, Sweden

ABSTRACT

For performance analysis, optimization and anomaly detection, there is strong need to monitor industrial systems, which, along modern datacenter infrastructures, feature a high level of decentralization. Continuous Distributed Monitoring (CDM) corresponds to the task of continuously keeping track of statistics from different distributed nodes. We review the feasibility of implementing state-of-the-art CDM algorithms in a large-scale, distributed, performance-critical production system. The study is on the Evolved Packet Core (EPC), an inherently distributed component of the 4G architecture, for processing mobile broadband data. In this work, we propose adjustments to classical models that are needed to account for communication and computation delays and the hierarchical architecture present in production systems. We further demonstrate efficient CDM implementations in the EPC, and analyze trade-offs of accuracy versus savings in communication, as well as availability of the monitoring system.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; • **Networks** → **Mobile networks**; *Network monitoring*; • **Information systems** → *Data streams*; *Distributed retrieval.*

KEYWORDS

continuous distributed monitoring, evolved packet core, real-time

ACM Reference Format:
Romaric Duvignau, Marina Papatriantafilou, Konstantinos Peratinos, and Eric Nordström, Patrik Nyman. 2019. Industry Paper: Continuous Distributed Monitoring in the Evolved Packet Core. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3328905.3329761

1 INTRODUCTION

For any computing system, monitoring is an important task that quickly becomes necessary to assess performance, allocate the appropriate amount of resources to fulfill service level agreements and achieve high utilization through the information it can provide. Additionally, monitoring data can be used for a variety of applications, ranging from computing usage statistics and detection of abnormal behaviors, to providing feedback on the design of the system. Efficient monitoring continuously provides an accurate view of the whole system and its respective subsystems, while minimizing the cost associated with the monitoring task.

Recent years have seen a clear growth of the number of Internet of Things devices, that preponderantly connect to the Internet using mobile broadband. The Evolved Packet Core [11] (EPC) is the core network of Long Term Evolution (LTE), the current mobile broadband standard (also referred as 4G), supporting the vision for bridging the gap between cellular networks and the Internet. The EPC architecture is inherently distributed: packet processing is performed on distributed nodes that are also duplicated to provide fault tolerance; each such node employs dedicated workers that must be able to handle processing of up to millions of packets per second. Monitoring is of utmost importance in such large distributed systems, but it must induce minimal load on the processing units and on the communication network (e.g. [9]), shared with the dense traffic processed by the EPC, so that the costs associated with it do not overweight its benefits.

In this industrial experience report, we provide algorithmic implementations of state-of-the-art Continuous Distributed Monitoring (CDM) methods, suitably tuned and adapted to fit a high-performance production system. We discuss here design and implementation challenges faced when porting event-based CDM to large scale distributed systems, survey the feasibility of dynamic real-time monitoring in the EPC and explore obtainable communication gains through a thorough analysis of gathered statistics. In general, there is a lack of insights from system-level perspectives and this work highlights needed adjustments to known models and practical aspects to consider. It further attests how adaptable and efficient (in time, communication bandwidth, energy and other system resources) the actual solutions are at production level systems, where the load and the size place challenges in scalability, as attesting algorithms efficiency solely based on theoretical bounds and simulated data cannot capture the entire truth. We have found that adjustments of classical models are required to account for (unreliable) communication and computation delays that are commonly abstracted in the literature. Indeed, systems with high loads cannot always keep the pace of all monitored statistics gathered from all distributed nodes and message propagation comes with non-negligible latency. However, the algorithms implemented in this work reduce the burden on the internal network and we show that more responsive monitoring is achievable when less data is sent through the

DEBS '19, June 24–28, 2019, Darmstadt, Germany

ACM ISBN 978-1-4503-6794-3/19/06…$15.00
https://doi.org/10.1145/3328905.3329761

network. Moreover, the abstract architecture used in the analytical literature is not representative of hierarchical distributed systems that are in place in modern data centers. Finally, this work provides further guidelines to implementing necessary monitoring systems in the upcoming 5G replacement for the EPC [1].

The paper is organized as follows. Section 2 presents a high-level view of the EPC and an overview of CDM. Section 3 presents the system model and how it relates to classical CDM models, selected suitably adapted algorithms from the literature, and design and implementation choices to adapt CDM to the EPC, including two statistics that we are interested in monitoring. Section 4 provides an extensive study of the implemented algorithms from several runs gathered from live experiments in the EPC. Finally, Section 5 gives concluding remarks and perspectives for further research.

2 PRELIMINARIES AND BACKGROUND

2.1 The Evolved Packet Core

This is a main component in LTE/4G mobile broadband infrastructure. In the LTE, traffic from mobile users is forwarded from base stations to the EPC before the latter forwards it to its destination (e.g. an Internet server). The EPC is a distributed system made of sub-components responsible for different functions (e.g. mobility management, billing, QoS). Among them and of interest in our study is the *Packet Gateway*, a part of the EPC's data-plane functions, that is in charge of processing data traffic through dedicated nodes (e.g. aggregators, cf. Figure 1 and outline in § 3.1) .

2.2 Continuous Distributed Monitoring

Distributed monitoring is a generalization of classic monitoring (measuring a single node internal statistic) that places more focus on communication efficiency rather than computing performance. In most forms of CDM, a specific node C acts as the *coordinator*, i.e., a sink node where monitoring messages are sent, whereas all other nodes (also named *sites*) act as *observers* of a continuous stream of values. The goal is to compute a function f (the *monitoring function* or *task*) at C, from the distributed input, in a communication-efficient manner. Observers can be thought of as routers in a computer network, base stations in a cellular network, or even services within a single system. In what follows, the number of observers is fixed to k, and we consider that they do not communicate with each other. In CDM, an *item* refers to an application-specific *type* of observation detected at a remote site; such detection is referred as an *event*. A *monitoring decision* is defined as a choice made by an observer, after an event occurred, whether to inform C or stay silent. A further CDM refinement requires f to be computed within a *sliding window* encompassing the last n events or n units of time.

Simple Approaches: The most straightforward approach to CDM is for each observer to forward all its observations towards C. Such strategy does not scale when the number of observers and observations is large. A second approach, *polling*, consists in retrieving statistics at a constant rate from observers. A major drawback of this is that the monitoring process is dependent on the polling frequency, and narrow gaps in the polling intervals may easily overload the network whereas larger gaps may lead to reduction in the accuracy [5]. *Sampling* aims to reduce the number of measurements on the system by fetching data from a few nodes and/or statistics. This approach may decrease a lot the burden associated with monitoring but cannot be applied to find under- and over-used components in the system as whenever those elements are sparse, they will not be captured by the sampling mechanism. Moreover, all the aforementioned approaches may send superfluous updates when it is not needed (e.g. when the system parameters remain stable). To overcome those shortcomings, the last decade has seen intensive research on the topic.

Monitoring Tasks: The most common monitoring problem is *counting* the number of events within some sliding window W, or since the start ($|W| = \infty$). The approximate version allows to report values within ε of the current count, i.e., for every W, the reported value at the coordinator $\hat{c}$ is so that $|\hat{c} - c| \leq c\varepsilon$, where c is the exact number of monitoring events within W. Other very popular problems for distributed monitoring include computing frequency of items [6] and most popular items [18].

Monitoring Models: The *CDM* model [5] aims for the coordinator to compute an aggregate function of the union of the k observers' stream of data, directly connected to C through a bidirectional channel. The model is asynchronous and each observation is considered an event, after which a monitoring decision is computed and a message transmitted to C, so that f is updated at C. Most algorithms developed in this model make use of the interchangeability of events to compute efficiently classical data mining functions. An example is *Geometric Monitoring* [14] that allows efficient threshold monitoring of functions computed over network-wide aggregates. The *Distributed Data Streams* (DDS) model [4] is a variant of the CDM model with two differences: nodes have a unidirectional communication channel towards C and must use sublinear space (so they are only able to account for approximation of their input stream at all time). Due to the unidirectional channel, observers can only rely on their own observations to take monitoring decisions. The *Distributed Online Tracking* model [17] is unidirectional and also synchronous, working in rounds. It also features the possibility to have intermediary nodes between the observers (leaf nodes) and the coordinator, playing the role of *relays* and being in charge of aggregating locally statistics before forwarding some data to C.

Related Work and Similar Approaches: Needed adjustments to theoretical models have been attested in previous attempts to make CDM algorithms feasible in practice as e.g. for sensor networks [16]. To the best of our knowledge, CDM algorithms and techniques, encompassing a large volume of research invested to find and prove (matching) lower and upper bounds on the communication complexity, have not yet been studied in industrial performance-critical systems. In parallel to analytical works, heuristic solutions to distributed queries have been explored (e.g. by using the well known adaptive filters [12] or for tracking popular items [2]). Note there also already exist *distributed monitoring systems* such as Magpie [3] and Dapper [15] for collecting information about request behavior, and Ganglia [10] for monitoring a cluster system. However, they manage to monitor large distributed systems by not being performed online (as in the earlier versions of Magpie), using some form of sampling (Dapper), or featuring large polling intervals (e.g. in the order of 30-60s for Ganglia) thus not very strictly complying with the objective of continuously monitoring the full set of nodes and statistics as discussed here.

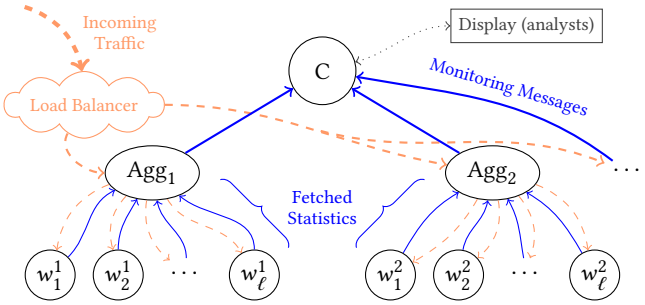

Figure 1: System architecture overview.

3 MONITORING THE EPC

3.1 Monitoring Architecture

We assign roles to specific EPC nodes following the CDM terminology: the **coordinator** node C is a specific process running in EPC's control plane and is responsible for maintaining up-to-date monitoring statistics about the whole system, and its sub-systems. The **observer** nodes are the main worker nodes and produce the performance statistics that we want to monitor. The **aggregator** (or *relaying*) nodes, are intermediary nodes on the communication path from observers to the coordinator lying in EPC's user plane; each aggregator is associated with a fixed set of observers so that by monitoring the observers we are also able to display statistics at an aggregator level. Figure 1 presents an overview of this simplified architecture: incoming traffic (dashed lines) is first received at the aggregator level before being assigned to particular observers, monitoring information (plain lines) are fetched from observer nodes before being transmitted to C through the aggregators, and C continuously displays a statistics summary to the analysts.

A main difference with models described in § 2.2 is that observers cannot be considered *anonymous*, as in the CDM model, but have identities and the function f to monitor is not over the union of streams (meaning each event occurring at another site will induce the same consequence on f) but rather has k as arity (each stream is differentiated) and requires all k streams to be monitored at all times (within some acceptable error bounds). The internal network used for communication between sites and C is bidirectional, and events are asynchronous. Contrary to most previous works, typical performance statistics (e.g. CPU utilization) are not proper *events* detected at a remote site but are rather obtained through system calls at one's convenience. The observation of a packet as mentioned in [5] is a good candidate for a monitoring event but in performance-critical systems as EPC, associating any computation with the processing of a single packet is ruining the intention of not degrading performance with the monitoring task. Since those system statistics do not translate into proper *events* of classical CDM algorithms, we need to introduce some form of polling mechanism to be able to recreate monitoring events at regular intervals. This is achieved by defining a *monitoring period* as a time interval where information will be gathered leading to the production of a single (multidimensional) "value" at the end of the period. Once this value has been generated, internal data structures (e.g. sliding windows) are updated and a monitoring decision occurs whether to transmit the updated metric to the coordinator (referred thereafter as a *monitoring update*) or not, depending on the particular monitoring algorithm which is actually running. A sliding window of fixed size (in terms of monitoring periods) is associated with each statistic of every observer node. Efficiency of CDM algorithms will be measured in terms of the number of monitoring updates sent, that is proportional to the number of bytes sent.

3.2 Selected CDM Algorithms

We have selected a set of algorithms representative of the models outlined in § 2.2, and customized them using ad-hoc heuristics tailored for the EPC environment. They all solve the counting monitoring task within some sliding window of n monitoring periods and assume that a single counter fits in $O(1)$ memory space. The naive solution (not considered here) would be to send to C all gathered statistics every monitoring period. We have instead implemented two *monitoring modes*: basic mode for perfect monitoring, and approximate one (tolerating a relative error of ε on the data gathered at C and featuring small space requirements per observer).

Basic Algorithm. Used as baseline in this work, it just sends updates to the coordinator for every statistic that has been modified since last message sent to C. This is a small optimization on the naive solution and can reduce drastically stable statistics. For each monitored statistic, a window containing the last n observed values is maintained. Once a monitoring period ends, the oldest item in each window is dropped and replaced by the newly acquired observation. It needs constant time at each observation and requires $O(n)$ words of memory to store the sliding window.

Simple Approximation. A natural algorithm one can think of is to maintain exactly a sliding window of the last n observed events and remember the last value v shared with the coordinator. For each new observation, the exact count c is computed from the sliding window, and transmitted to the coordinator if it lies ε-far from v, i.e., $|c - v| > c\varepsilon$. The sliding window is the same as the basic algorithm.

Approximate Algorithm. This algorithm presented in [4] provides a simple approach to monitor ε-approximate counts using logarithmic memory by using Exponential Histograms (EH) [7]. EH are approximate data-structures that counts the number of ones, with a relative error of λ, within some sliding window W of the last n binary events, and can be extended to support the sum of bounded positive numbers. In short, approximate counting is obtained by maintaining a sequence of *buckets*, where the range each bucket is responsible for grows exponentially. Each bucket aggregates a continuous section of W, and includes a timestamp indicating the time of the most recent event within the bucket. Working over a stream of R-bounded positive integers, a λ-approximate EH is capable of producing an approximate sum of the last n items within λ from the real one in constant time; moreover, it is able to process new arrivals in $O(\log R/\log n)$ amortized time, and uses only $O(\log n(\log R + \log n)/\lambda)$ bits of memory.

In the *approximate algorithm*, each of the k distributed streams is treated separately and maintain an $\frac{\varepsilon}{9}$-approximate EH for counting approximately the sum of all items over the sliding window. Using the EH, each site can provide at any given time t its approximate

count of items $\hat{c}(t)$. Denoting $\hat{c}(p)$ the last count forwarded to the coordinator at time t, so with $p < t$, two events are defined:

- **Up:** $\hat{c}(t) > (1 + \frac{4}{9}\varepsilon) \cdot \hat{c}(p)$
- **Down:** $\hat{c}(t) < (1 - \frac{4}{9}\varepsilon) \cdot \hat{c}(p)$

and whenever one of those occurs at time t, the remote site informs the coordinator of its current approximated count $\hat{c}(t)$. In [4], the algorithm is shown to use $O(\log(\varepsilon n)/\varepsilon)$ memory words of communication and can be further improved to yield the same complexity in terms of bits; it is asymptotically optimal in this regard.

3.3 Design and Implementation Choices

Implementing the Monitoring Logic. Monitoring updates are aggregated locally by aggregators before their transmission to C through a TCP session to avoid the overhead of adding a TCP/IP header for each update, hence, every monitoring period (up to) a single *monitoring message* per aggregator is transmitted to C. Moreover, for performance reasons, individual observers and aggregators are not globally time synchronized meaning monitoring messages will not be timestamped. However, local channels are time ordered, as this service is provided by the TCP connection between aggregators and the coordinator. To account for computation and communication delays, we will provide a monitoring service that will be mostly available (i.e., the displayed statistics correspond to stale values for only very short time spans as shown in § 4) but will not cover all elapsed time; this is due as a non-negligible part of the time is spent computing monitoring decisions, serializing and sending monitoring information. Since in our architecture, observers and aggregators may communicate via inter-process communication without using the internal network, monitoring logic has been implemented as part of the aggregators. Another advantage of such design is ensuring that the monitoring process do not interfere with the recorded system statistics, hence not harming the observers' performance (reported values correspond to the same actual statistics without monitoring running on top); this overall trades some CPU time at the aggregators for expected saving in communication.

Measuring Metrics of Interest. We use two types of precision level for metrics: (i) **high granularity**, that allows computation of extra functionality (min, max, mean, quantiles, median, etc) within the monitoring period scope, and is obtained by updating frequency charts for the current period; (ii) **low granularity**, that displays only a single aggregate for each monitoring period. The former has higher cost but provides more live insights and is more versatile, allowing to change the monitoring aggregate function while running the system (e.g. how the statistics are displayed to the user) without having to update the observers (a potentially heavy task for large scale systems). For our evaluation, we have selected one archetypal statistic for each case, i.e. *CPU usage* and *packet processing rate*.

CPU Usage: A very important metric to detect over- or under-used nodes is processor utilization, characterized by the fraction of the full computing power that has been used in the last monitoring period. We will monitor it using high granularity tracking allowing by this way to observe individual distribution changes and sudden spikes between monitoring periods. In detail, during a monitoring period, P fetches are performed to retrieve CPU usage for the past 1ms (using system primitives that track the number of executed CPU cycles over time) and these values are used to obtain a frequency chart of F counts. This means that for each fetch, the fetched value is rounded to the closest frequency bin, and the corresponding count is incremented by one. Once the frequency chart is computed, the sliding window is updated for each of the F counts using either basic or approximate mode (cf § 3.2). For each count, if the value has changed (basic) or is outside of tolerated error (approximate), an update is sent along the next monitoring message for C to update that particular count of the respective observer. C keeps track of all frequency counts for every observer but it is oblivious of sliding windows (which are maintained locally for scalability reasons). C is able to provide in real-time the latest average CPU usage for an observer by computing its weighted frequency average: $\sum_{1 \le i \le F} i f_i / \sum_{1 \le i \le F} f_i$. Note the sum of the frequency counts may differ from P as some might not have been updated at every monitoring period (in the approximate algorithm).

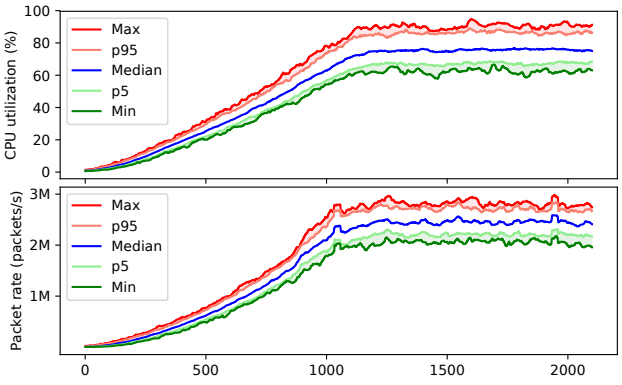

Figure 2: CPU utilization and packet proc. rate per round.

Packet Processing Rate: The packet processing rate, defined as the number of packets processed by an observer over some interval of time, is another good indicator of performance. During a monitoring period, for each processed packet at an observer, a dedicated counter is incremented by one. Once the monitoring period ends, the total number of packets processed during the period is retrieved and a single event is emitted. Contrary to the CPU usage, we will track a single counter for the packet processing rate and only the average value per monitoring period over the current sliding window is tracked at the coordinator.

4 EVALUATION

We evaluate the two continuous monitoring algorithms (namely basic and approximate modes) presented in § 3.2 in the Evolved Packet Gateway (EPG [8]), the core component of Ericsson's EPC infrastructure. The approximate algorithm has been executed with three error thresholds $\varepsilon = 5\%$, $\varepsilon = 10\%$ and $\varepsilon = 20\%$.

Evaluation settings. The experiments correspond to about 4 hours of hardware statistics retrieved during 8 runs of a load testing procedure in an EPC testing configuration. The runs have been obtained by logging data at C during a stability load test of a small EPG testing environment. The network in the experiments is constituted of 2 aggregator nodes, each supervising 72 observers. We divide each run in *monitoring rounds* of 1s at the end of which the coordinator

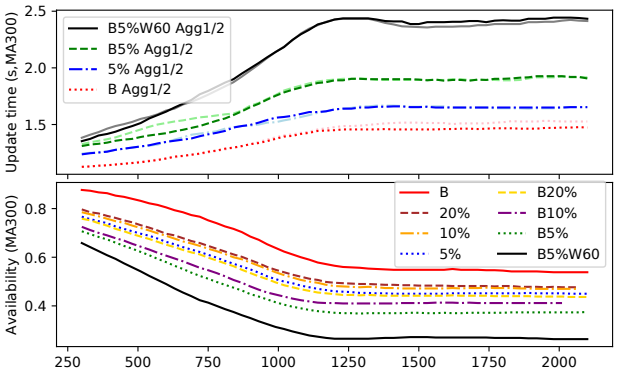

Figure 3: Monitoring availability per round.

node logs its current statistics (CPU average utilization, and number of processed packets) for each observer node. The *load test* is split into two parts: the first part lasting for 20min corresponds to an **increasing load** phase where packets are injected at a constantly increasing rate into the system (from 0 up to 430 M/s), then the second phase (that we measured for about 15min) corresponds to a **stable load** phase where packets are injected into the EPC at the maximal rate of the previous phase for the rest of the experiment. During both phases, the internal complex load balancing proper to EPG is at hand to route, assign and handle incoming packets, hence individual observers might receive different loads over time and different experiments thus produce non-deterministic results.

The 8 runs include 4 runs (B, 5%, 10%, 20%) where the monitoring algorithms have been executed in isolation: those runs are meant to benchmark the availability of the monitoring system; the 4 remaining runs (B5%, B10%, B20%, B5%W60) feature concurrent execution of both algorithms to recover the exact statistics at each round for measuring the approximation error. All runs except B5%W60 have been obtained computing statistics with a 10s sliding window, whereas B5%W60 uses a 60s window. For the concurrent runs, we removed from the gathered data 0.3%-0.6% of the rounds where only one algorithm had been processed by C but not the other one (hence making discrepant results on those rounds). In the evaluation, we set the monitoring period to 1s, $P = 1000$ and $F = 100$ for the polling frequency and granularity of the CPU usage.

Figure 2 (top) presents the maximum, minimum, median, 95% percentile and 5% percentile of recorded CPU utilization of all observers in each monitoring round (for run B; all runs present very similar profiles). The two phases are clearly distinguishable from the 1200 rounds mark (start of the second phase): in the increasing load portion CPU utilization are raising, and then in the second one, stay stable (within 65-85% utilization for 90% of the observers) for the rest of the experiment. The measured packet processing rate, Figure 2 (bottom), features more spikes or ups and downs (especially noticeable comparing the mean value of both statistics) due to the load-balancing of packet processing between observers and aggregators during the experimentation. Those spikes are inherent to each run and result in the average packet processing rate being slightly less predictable than the average CPU utilization.

Availability time. The *update time*, or equivalently the elapsed time between two consecutive updates, presented on Figure 3 (top) with a moving average of 5min, suggests that roughly we need 0.4-1.4s (in the stable phase) to serialize, send and deserialize monitoring messages for a monitoring period of 1s containing 1000 fetches for CPU and a single fetch for packet rate. The largest time between two updates for an aggregator is 4s for all runs except for the 5%W60 run, where it is 6s. Note that the measured availability[1] (Figure 3 bottom) is directly affected by the current load of the system and large availability is only noticed at small load, while an increasing load decreases the availability of the monitoring system.

From our results, we suggest in order to increase monitoring availability either to shorten monitoring period towards 250 fetches within 0.25s (decreasing accuracy but preserving responsiveness), or to make C only update its displayed results every 4 seconds (inversely conserving accuracy but decreasing responsiveness). Another approach can be to decrease granularity of the CPU statistics by, for instance, keeping only 20-25 frequency counters instead of 100 to decrease the number of data structures to update during monitoring decision and the time spent on serializing messages.

Packet Processing Rate. Figure 4 (bottom) illustrates the communication saving obtained by running the approximate algorithm. It presents the number of monitoring updates sent to C compared to the basic mode. The reduction is more important in the stable phase due to statistics remaining closer to the last sent value for longer period of time, and ranges from 5% for the least accurate algorithm to roughly 10% for the most accurate one. A larger window size further reduces the number of updates sent in the stable phase and has less influence in the unstable one (the 60s window run is close to its 10s equivalent in this portion of the experiment).

Figure 5 presents the results of the runs for the packet processing rate with boxplots of interquartile range with whiskers ranging up to 1.5 times above (or below) the first and third quartile. On the figure, the approximate algorithm is compared with the simple algorithm of threshold $\frac{5}{9}\varepsilon$ (this represents the maximum observed relative error, see two upper graphs on Figure 6). The implemented

[1] Availability is 1 if the coordinator receives update from both aggregators during a given round, 0.5 if it receives an update from one of them and 0 otherwise.

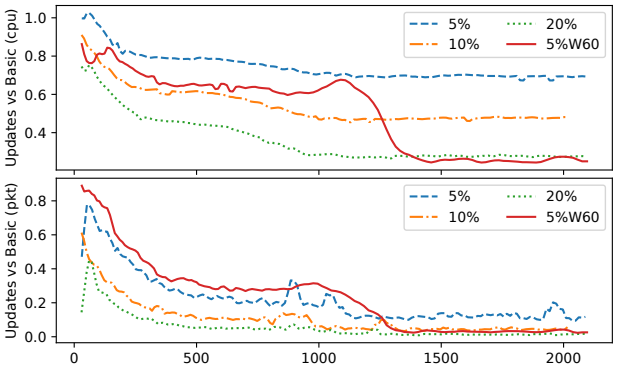

Figure 4: Number of monitoring updates per round.

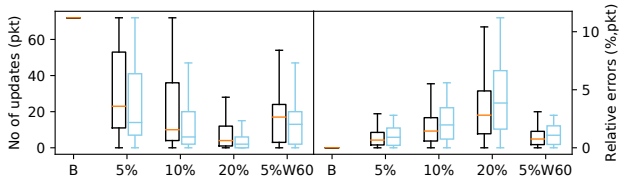

Figure 5: Statistics for packet proc. rate (Aggregator 1).

algorithm sends slightly more updates concerning the monitoring of the packet processing rate but features smaller relative errors (results have been filtered to a single aggregator node to compare with the basic algorithm that always send updates for all observers attached to a given aggregator, here 72).

CPU Usage. Following similar formatting as for packet processing rate, Figure 4 (top) illustrates the communication saving obtained by running the approximate algorithm and shows similar findings. Due to monitoring it using higher granularity (cf. § 3.3), the observed reduction is decreased compared with the packet rate monitoring and ranges from 30% to 70%, whereas the larger window size also slightly reduces number of updates during unstable load.

Absolute errors for 2 runs are depicted on Figure 6 (two lower graphs): for 10s window, errors are contained to within 0.5-1% of the truly measured value but for 60s they can reach up to 2.5% but remain very low on the increasing load phase (the larger window and a high rate of sending updates due to a higher rate of changes make the error small in absolute value).

5 CONCLUSIONS

We have presented in this experience report adjusted algorithmic implementations of state-of-the-art continuous distributed monitoring algorithms in the EPC. We provide keys to adaptations of those methods and the models needed to efficiently implement them in a production level, performance-critical system. In our evaluation, we have seen that only 6% of the monitoring data may need to be transmitted to the management node while ensuring an approximated monitored statistic within 1.6% apart from the one

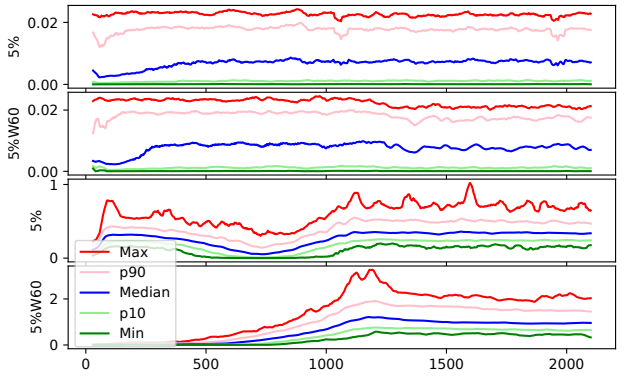

Figure 6: Approximation errors per round.

measured locally, on average . The preliminary results presented here, obtained after applying continuous distributed monitoring techniques in a production system, are of interest to guide further research in the field towards more dedicated monitoring algorithms. They advocate lightweight and adaptive algorithms and provide guidelines in setting distributed monitoring parameters, particularly useful for preparing the transition to the upcoming mobile broadband infrastructure. Such approaches can also be beneficial for continuation work, in the context of Service-Level-Agreements, as it is possible to identify trade-offs of various measures, including accuracy, efficiency, timeliness or freshness of information.

ACKNOWLEDGMENTS

We thank the reviewers for their helpful feedback and suggested improvements. We thank Sarkhan Ibayev, who, together with Konstantinos Peratinos, developed the prototype implementation in EPG during their master thesis [13]. The work was partially supported by the SSF project FiC, grant number GMT14-0032 and by the Chalmers Area of Advance framework project INDEED.

REFERENCES

[1] Mamta Agiwal, Abhishek Roy, and Navrati Saxena. 2016. Next generation 5G wireless networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 1617–1655.
[2] Brian Babcock and Chris Olston. 2003. Distributed top-k monitoring. In *Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of data*. ACM, 28–39.
[3] Paul Barham, Rebecca Isaacs, Richard Mortier, and Dushyanth Narayanan. 2003. Magpie: Online Modelling and Performance-aware Systems.. In *HotOS*. 85–90.
[4] Ho-Leung Chan, Tak-Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. 2012. Continuous monitoring of distributed data streams over a time-based sliding window. *Algorithmica* 62, 3-4 (2012), 1088–1111.
[5] Graham Cormode. 2013. The continuous distributed monitoring model. *ACM SIGMOD Record* 42, 1 (2013), 5–14.
[6] Graham Cormode, Minos Garofalakis, S Muthukrishnan, and Rajeev Rastogi. 2005. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proc. of the 2005 ACM SIGMOD*. ACM, 25–36.
[7] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM journal on computing* 31, 6 (2002), 1794–1813.
[8] Ericsson. 2019. Evolved Packet Gateway. https://www.ericsson.com/en/portfolio/digital-services/cloud-core/cloud-packet-core/evolved-packet-gateway
[9] Bastian Havers, Romaric Duvignau, Hannaneh Najdataei, Vincenzo Gulisano, Ashok Chaitanya Koppisetty, and Marina Papatriantafilou. 2019. DRIVEN: a framework for efficient Data Retrieval and clusterIng in VEhicular Networks. In *35th IEEE International Conference on Data Engineering (ICDE 2019)*. 1850–1861.
[10] Matthew L Massie, Brent N Chun, and David E Culler. 2004. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* 30, 7 (2004), 817–840.
[11] Magnus Olsson and Catherine Mulligan. 2012. *EPC and 4G packet networks: driving the mobile broadband revolution*. Academic Press.
[12] Chris Olston, Jing Jiang, and Jennifer Widom. 2003. Adaptive filters for continuous queries over distributed data streams. In *Proc. of the 2003 ACM SIGMOD*. 563–574.
[13] Konstantinos Peratinos and Sarkhan Ibayev. 2019. *EPGTOP: A tool for continuous monitoring of a distributed system*. Master's thesis. Chalmers University of Technology, Gothenburg, Sweden.
[14] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2007. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Transactions on Database Systems (TODS)* 32, 4 (2007), 23.
[15] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Technical Report. Google.
[16] Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, and Marina Papatriantafilou. 2018. Geometric Monitoring in Action: a Systems Perspective for the Internet of Things. In *2018 IEEE 43rd Conf. on Local Computer Networks (LCN)*. IEEE, 433–436.
[17] Mingwang Tang, Feifei Li, and Yufei Tao. 2015. Distributed online tracking. In *Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data*. 2047–2061.
[18] Ke Yi and Qin Zhang. 2013. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica* 65, 1 (2013), 206–223.