ATMoN: Adapting the "Temporality" in Large-Scale Dynamic Networks

Demetris Trihinas*, Luis F. Chiroque†, George Pallis*, Antonio Fernández Anta†, Marios D. Dikaiakos*

* Department of Computer Science, University of Cyprus
Email: { trihinas, gpallis, mdd }@cs.ucy.ac.cy

† IMDEA Networks Institute, 28918 Leganés, Spain
Email: { lf.chiroque, antonio.fernandez }@imdea.org

***Abstract*—With the widespread adoption of temporal graphs to study fast evolving interactions in dynamic networks, attention is needed to provide graph metrics in time and at scale. In this paper, we introduce ATMoN, an open-source library developed to computationally offload graph processing engines and ease the communication overhead in dynamic networks over an unprecedented wealth of data. This is achieved, by efficiently adapting, in place and inexpensively, the temporal granularity at which graph metrics are computed based on runtime knowledge captured by a low-cost probabilistic learning model capable of approximating both the metric stream evolution and the volatility of the graph topology. After a thorough evaluation with real-world data from mobile, face-to-face and vehicular networks, results show that ATMoN is able to reduce the compute overhead by at least 76%, data volume by 60% and overall cloud costs by at least 54%, while always maintaining accuracy above 88%.**

***Index Terms*—Dynamic Networks, Edge Computing, Temporal Graphs, Adaptive Monitoring**

I. Introduction

For a diverse set of applications, graphs have been used extensively to model links among entities in dynamic networks [1]. The network structure describing how the graph is wired allows us to study, predict and optimize the behavior of dynamic systems [2]. However, with the widespread adoption of social networks to depict digital interactions among "friendship" links [3], neural networks to chain metabolic reactions [4] and the prevalence of the Internet of Things to monitor the physical world [5], the dynamics shaping network evolution yield the need to appraise "time" in graph modeled networks.

Dynamic networks are modeled as time-ordered sequences of graphs in which links are short-lived and span only through the duration of the interaction between the nodes (e.g., online chat, email, phone call) [6]. In such graphs, the concepts of node adjacency and reachability crucially depend on the exact temporal ordering with the adjacency matrix describing the current graph snapshot for a fixed duration of time [7]. In the literature, these graphs are frequently mentioned as time-evolving graphs, time-varying graphs or, simply, temporal graphs [8]. Hence, temporal graphs have become very popular, featuring the analytic benefits of static graphs, while also improving graph exploration and navigation by retaining all temporal information and interactions.

Monitoring temporal graphs is widely used in a variety of services, such as high-frequency algorithmic stock trading, social network analysis, targeted advertising, and in intelligent transportation services. However, computing complex graph metrics, such as community and distance metrics, is a challenging task [9] [10]. In turn, if the nodes of the graph represent data sources distributed across the edges of an actual network, then these sources must timely disseminate and receive processed monitoring data [11]. As an example, consider a vehicular network where the expectations are that 1GB of telemetry data will be generated by self-driving vehicles every second [12]. This data must be processed instantly to keep vehicles safe on the road. It may not seem a challenge for one vehicle, but multiply this number by millions of inter-connected vehicles in an urban environment and even powerful organizations equipped with large-scale graph processing engines reach their limits as the velocity of data keeps increasing [7]. Thus, by the time the data reaches the cloud, it will be too late. Instead, data needs to be processed as close to the data sources as possible, directly at the edge of the network [13].

The remedy to the above challenges is to suppress large-scale temporal graphs with approximation techniques [14]. Ideally, an approximation technique dynamically adjusts the rate at which data are processed based on the current data stream evolution, such that when stable phases are detected, the data processing rate is reduced. However, current approximation techniques developed for edge and streaming settings are not suitable for temporal graphs since they do not take into account the dynamicity of the graph topology [15] [16]. For instance, although the graph metric stream may introduce phases of low variability, the graph topology structure can still be extremely volatile. This is highlighted in Figure 1, where although the outgoing connections per node of a mobile network (average out

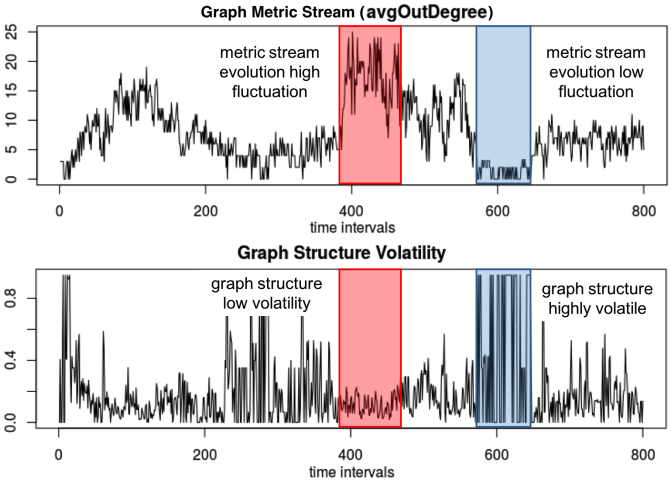


Fig. 1: Graph Metric and Topology Structure Volatility for a Mobile Network with Fixed Temporal Granularity

degree) are relatively stable in certain phases, it is actually a different set of nodes interacting in the network [17]. In light of this, ignoring that the particular connections actually span across different locations of the network, can severely affect capacity planning and service provisioning [14]. Thus, while reducing the metric processing rate preserves resources by computationally offloading graph processing engines, it hinders the challenge of missing structural changes in the graph topology which might capture and reveal significant insights.

The focal point of our work is to deliver an adaptive monitoring framework for dynamic networks which adjusts the network temporal granularity given an estimation model capturing both the metric stream runtime evolution and the volatility of the graph topology. Thus, our ultimate goal is to significantly ease processing on graph engines, and costs in general, while respecting at all times user-given accuracy guarantees.

The main contributions of our paper are:

- We introduce Adaptive and Topology-aware Monitoring for dynamic Networks. ATMoN is an open-source framework[1] developed to dynamically adjust the temporal granularity graph metrics are computed based on runtime knowledge captured by a low-cost probabilistic learning model approximating both the metric stream evolution and the runtime volatility of the graph topology. This computationally offloads graph processing engines and eases the communication overhead in edge computing networks where the wealth of data dissemination is plentiful.
- We present a thorough experimentation study to evaluate both the efficiency and accuracy of our solution. All testbeds utilize real-world and publically available datasets from mobile, face-to-face and vehicular networks. Results show that when graph processing engines embrace ATMoN, they can reduce the compute overhead by at least 76%, data volume by 60% and overall cloud costs by at least 54%, while always maintaining accuracy above 88% in comparison to other adaptive frameworks.
- We extend the open-source graph ecosystem of the R programming language [18] by enabling developers to model dynamic networks as adaptive temporal graphs. This extension creates new perspectives by facilitating developers to implement novel algorithms (i.e, ATMoN) that efficiently manage time-evolving graph-structured data streams.

The rest of the paper is structured as follows: Section 2 presents the related work, while Section 3 the problem statement. Sections 4-5 introduce ATMoN, while Section 6 presents the evaluation. Section 7 concludes the paper.

II. Related Work

Graph processing frameworks such as Giraph [10] and Powergraph [19], enable graph modeling and computation at scale. However, they assume the underlying graph structure is static. On the other hand, Iyer et al. [9] present GraphTau, a temporal graph processing framework built on top of Apache Spark, which efficiently unifies data and temporal graph processing. In turn, Han et al. [20] introduce Chronos, a graph engine designed and optimized specifically for running in-memory iterative graph computation on temporal graphs. However, both assume that the network temporal granularity is fixed and pre-defined. Graph-structured data is on the rise; from social and face-to-face networks to vehicular networks, applications that generate temporal graph structured data are ubiquitous. As IoT continues to spread across almost all industries it triggers a massive influx of big data. Undoubtedly, other than temporal graph processing frameworks, we also need algorithms to efficiently provision, manage and monitor IoT services [21]. In light of this, several efforts have been made to process and output complex network metrics. In particular, graph measurement techniques now embrace time as another dimension to graphs and develop "temporal" metrics to describe the network reachability [22], connectivity [23] and shortest paths [24]. However, some graph metrics are, in fact, intractable in practice, and cannot scale at will or be efficiently computed [8] [25].

Another approach successfully demonstrated in edge computing and streaming settings, is to provide approximate metrics values by adapting the monitoring intensity to scale and timely answer queries within certain accuracy guarantees when exact answers are not needed. This is achieved via a runtime estimation model capable of following the metric stream evolution, such that when phases of low variability exist, the metric computation rate is reduced to ease processing while also reducing the data volume. Fan et al. introduce FAST [15], an adaptive framework for differential privacy which estimates the metric computation rate based on an adjustment given by a PID controller fed with the current estimation error, the

[1] https://github.com/dtrihinas/ATMoN

time intervals between previously collected metric values and a given inaccuracy budget. In turn, AdaM [16] is an adaptive monitoring framework for IoT devices, which utilizes a probabilistic moving average as its estimation model and dynamically adjusts the temporal granularity of a metric stream based on the confidence of the algorithmic model to correctly estimate what will happen next in the metric stream. AdaM has been shown to achieve a balance between efficiency and accuracy even for metric streams featuring highly abrupt and transient changes. Nonetheless, while interesting solutions, FAST and AdaM are not suitable for dynamic networks as they limit their scope to adjusting the temporal granularity by only taking into account the metric stream runtime evolution and completely ignore the topology structure.

III. Problem Statement

Let $G := (V, E)$ be a graph representing a network, with $|V| = n$ a set of nodes, and $|E| = m$ a set of binary relationships depicting the active interactions among nodes. As the network evolves in time, it is modeled as a graph stream $\mathcal{G}$ indexed by $I \subseteq \mathbb{Z}^+$, with $\mathcal{G}$ denoting a succession of graph instances and formally defined as follows:

$$\mathcal{G} := \{G_i : i \in I\} = \{(V_i, E_i) : i \in I\} = (\mathcal{V}, \mathcal{E}) \quad (1)$$

In turn, let $\mu(G_i)$ denote a *graph metric* computed over the i^{th} instance G_i with each obtained measurement, described at the minimum, as a tuple (G_i, t_i, v_i) also comprised by a timestamp t_i and a value v_i. A measurement may include a set of other attributes, although for brevity, when describing a graph metric we will omit these attributes without loss of generality. Hence, a series of measurements over the graph stream is denoted as a *metric stream* and can be formally defined as follows:

$$\mu(\mathcal{G}) := \{\mu_i : i \in I\} = \{\mu(G_i) : i \in I\} \quad (2)$$

Fundamentally, measurements for a temporal graph are obtained periodically over a fixed period of time, denoted as Δ where $\Delta = t_i - t_{i-1}$. With this, Δ is pre-defined and known as the *temporal granularity* of the network so that the i^{th} graph instance is always modeled and processed at known time intervals with the i^{th} measurement obtained at time $t_i = i \cdot \Delta$. Due to its simplicity, this approach is widely adopted by graph processing engines [9] [18]. In this work, we argue that using a fixed and pre-defined Δ, over large-scale and highly temporal graph streams, features a number of constraints, especially when consecutive measurements do not vary. For example, consider the metric stream depicted in Figure 1. If a small Δ is used to process the metric stream, the graph engine is computationally stressed in order to output and disseminate timely measurements to nodes of the monitored network. As the graph grows, more resources (e.g., compute, memory, storage, bandwidth) are required to timely process the graph stream even if there are phases of low variability in the metric stream evolution. Instead, if a large Δ is used, then sudden events or significant insights might remain undetected. In general, because *monitoring depends on the evolution of the data in time, we argue that a fixed periodicity is not effective, as metrics and insights are only useful if collected in meaningful time intervals.*

To accommodate the above challenges, at any given time interval t_i we can dynamically adjust the temporal granularity Δ_i, based on some estimation model, denoted as $\rho(\mu(\mathcal{G}))$, capturing runtime knowledge of the metric stream evolution. Assume μ_i to be the latest computed measurement over $\mathcal{G}$ and that Δ_i accepts discrete integer values in the range $[\Delta_{min}, \Delta_{max}] \subseteq \mathbb{Z}^+$ without loss of generality. Now, suppose the temporal granularity of $\mathcal{G}$ is dynamically adjusted at runtime, so that when the metric stream evolution has not "changed" since last reported, Δ_i should be increased and when the evolution fluctuates, it should be decreased or restored to a minimum value. However, *how much "change" is "tolerable" depends on some evaluation metric (err), upper bounded by the maximum tolerable inaccuracy, denoted as η, given by the user.*

While this argument seems sound, and has been recently explored to adapt the periodicity of monitored nodes at the edge of a network [16] [15], in practice this approach is intractable for networks modeled as temporal graphs. This is due to the fact that current adaptive techniques completely ignore the structure of the network and adjust the metric computation rate solely based on the temporal evolution of the metric stream. Hence, although the metric stream evolution may undergo phases of low fluctuation, the network structure can still be extremely volatile. Therefore, an adaptive technique must extend the estimation model $\rho(\cdot)$ to also capture and acknowledge the graph structure volatility. Thus, in a similar fashion, let $\tau(\mathcal{G} \mapsto \mathbb{R})$ denote how the graph stream *structure volatility* evolves based on some metric capable of capturing the dissimilarity of consecutive graph instances [26].

$$\tau(\mathcal{G}) := \{\tau_i : i \in I\} = \{\tau(G_i) : i \in I\} \quad (3)$$

Our goal is to develop an estimation model $\rho(\mu, \tau)$ capturing at runtime the metric stream evolution and the graph structure volatility, and provide an adaptive function $f(\cdot)$, that outputs the maximum Δ to delay modeling G_{i+1} that will be used to obtain the next set of measurements $\vec{\mu}_{i+1}$. This must be achieved while respecting user-given accuracy guarantees based on some evaluation metric, denoted as $err(\eta)$. Thus, the following equation summarizes the problem:

$$\Delta^* = \arg\max_{\Delta} \{f(\vec{\mu}, \Delta, \rho(\mu, \tau), err(\eta)) \mid \Delta \in [\Delta_{min}, \Delta_{max}]\} \quad (4)$$

IV. The ATMoN Library

To address the above challenges, we have designed and developed the ATMoN framework. ATMoN is a lightweight and open-source library developed to adapt the temporal granularity when computing graph metrics, thus

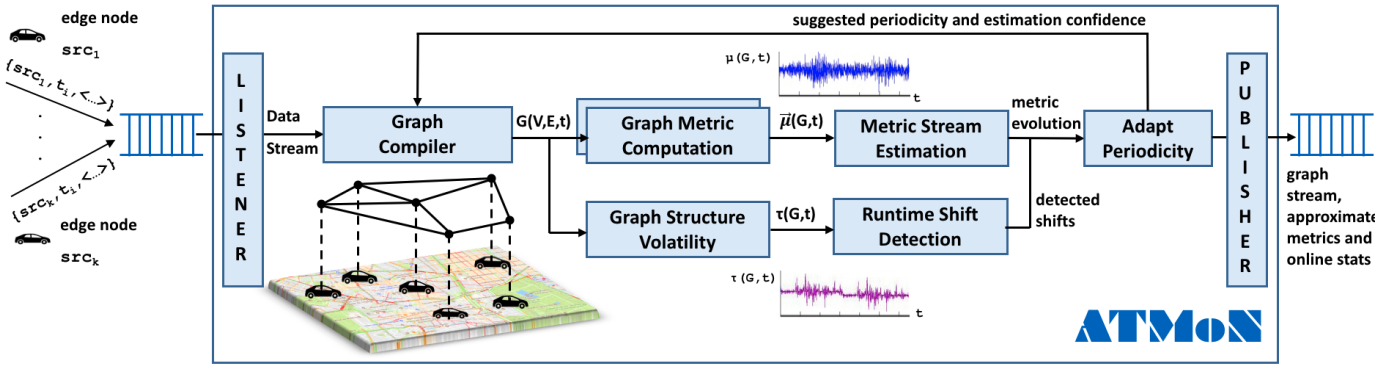


Fig. 2: High-Level Abstract Overview of the ATMoN Framework

computationally offloading graph processing engines and easing the communication overhead in dynamic networks. The current ATMoN prototype is developed in R and supports the underlying igraph engine, which is developed for C, R and Python. The igraph engine was selected for its open-source nature, allowing the code-base to be extended to provide the ability to adapt the graph temporal granularity, while igraph is also proven to be capable of handling and scaling large networks efficiently [18]. Nonetheless, while ATMoN supports the igraph engine, as it features no other external dependencies it can be ported to other popular graph processing frameworks.

Figure 2 depicts a high-level and abstract architectural overview of the ATMoN framework. The *Listener* module, continuously digests and queues incoming data updates from the nodes distributed across the monitored network. We consider an example drawn from the domain of vehicular networks. In such a setting, vehicles correspond to nodes and communication links to edges. Digested data is parsed by the *Graph Compiler* module, and depending on the current temporal granularity, the modeled graph stream is updated with the current network graph. Next, the current graph instance is passed to the *Graph Metric Computation* module so that the metrics of interest are updated. For our vehicular network example this could be the effective diameter, degree distribution and the number of graph components. A set of more 25 metrics are currently available by ATMoN, while users are able to define their own graph metrics by extending the ATMoN metric interface.

After processing the graph instance to obtain the current measurements, the *Metric Stream Estimation* module will update a local reference estimation module capable of capturing the current metric stream evolution, trend and variability. In parallel, the *Graph Structure Volatility* module will compute and update the graph topology volatility by detecting dissimilarities between the consecutive graph instances. The output of this module will be given to the *Runtime Shift Detection* module, which according to the error tolerance provided as input to ATMoN, will render if the network volatility is "tolerable" or not. Having updated the metric stream evolution and the network topology volatility, the *Adaptive Periodicity* module will return a new estimation for the graph stream temporal granularity and a confidence interval for the estimation. The Graph Compiler will then acknowledge the new temporal granularity to ease graph metric computation until this delay expires, thus, allowing the network to "breathe" by reducing resource consumption and the volume of generated data and, subsequently, cloud expenses. Finally, interested users and entities can access through the ATMoN *Publisher* API the approximated graph metric and topology volatility stream.

V. Temporal Granularity Adaptation

This section provides an in depth overview of ATMoN with focus on the algorithmic process (Algorithm 1) that updates the temporal granularity of dynamic networks modeled as temporal graphs.

Graph Compiler and Metric Computation. These modules directly map incoming monitoring updates from data sources to graph instances. To support this functionality we have extended the underling graph engine to support adaptive temporal graphs where users are required to provide: (i) the graph model; (ii) the algorithm that will be used to adapt the temporal granularity; (iii) the acceptable range for the temporal granularity; (iv) the maximum tolerable error; and (v) the list of metrics that will be computed by the graph engine. The graph model can be any model supported by the igraph engine, while in the case where igraph is not used, users must provide a direct mapping of how incoming data are modeled to graph instances and provide the respective interface.

```
gstream ← adaptive_graph<graphModel(V,E,...),
    algo=ATMoN, delta=(init,min,max), maxError,
    metric_list=list(...)>
```

Metric Stream Estimation. When a new measurement value v_i over the current graph instance G_i is available, this module updates the current metric stream evolution μ_i and variability σ_i, by using a moving average. This will give an initial estimation for the next measurement value, denoted as $\hat{v}_{i+1}$. Moving averages are easy to compute, though many types exist, and can be calculated on the fly

Algorithm 1 Adaptive Graph Temporal Granularity

Input: **init**: user-defined max tolerable error η,
per iteration: current graph G_i from the $\mathcal{G}$ stream
Output: Δ_{i+1} and estimation confidence c_i
Ensure: $\{\Delta_{i+1} \mid \Delta_{i+1} \in \mathbb{Z}^+ \text{ and } \Delta_{i+1} \in [\Delta_{min}, \Delta_{max}]\}$

$\vec{r} \leftarrow \{\}$ //result set
for each m **in** `metric_list` **do**
  $v_i \leftarrow$ `computeMetricUpdate`(G_i, m)
  $\mu_i, \sigma_i \leftarrow$ `updateMetricStreamEvolution`(v_i)
  $c_i \leftarrow$ `updateConfidence`$(\sigma_i, \hat{\sigma}_i)$
  $\vec{r} \leftarrow \vec{r} \cup (\mu_i, c_i)$
end for
$\delta_i \leftarrow$ `computeGraphDissimilarity`(G_i, G_{i-1})
$\tau_i \leftarrow$ `updateTopologyVolatility`(δ_i)
if `changeDetected`(τ_i) **then**
  `change` $\leftarrow$ `true`
else
  `change` $\leftarrow$ `false`
end if
$\Delta_i \leftarrow$ `updatePeriodicity`$(\vec{r}, \eta,$`change`$)$
return Δ_{i+1}, c_i

with only previous value knowledge. A cumulative moving average for streaming data is the Exponential Weighted Moving Average (EWMA), $\mu_i = \alpha\mu_{i-1} + (1-\alpha)v_i$, where a weighting parameter α, is introduced to decrease exponentially the effect of older values. However, the EWMA features a significant drawback; it is volatile to abrupt transient changes [16]. To overcome this drawback, we adopt a double Probabilistic EWMA (dPEWMA), which dynamically adjusts the weighting based on the probability density of the given observation evolution μ_i and trend x_i. The dPEWMA acknowledges sufficiently abrupt transient changes, adjusting quickly to long-term shifts in the metric evolution and when incorporated in our algorithmic estimation process, it requires no parameterization, scaling to numerous measurements. Equation 5 presents the dPEWMA where instead of a fixed weighting factor, we introduce a probabilistically adaptable weighting factor $\tilde{a}_i = \alpha(1-P_i)$. In this equation, the p-value, is the probability of the current v_i to follow the modeled distribution of the metric stream evolution.

$$\begin{aligned} \mu_i &= \tilde{a}_i(\mu_{i-1} + x_{i-1}) + (1-\tilde{a}_i)v_i \\ x_i &= \xi\ (\mu_i - \mu_{i-1}) + (1-\xi)\ x_{i-1} \\ \mu_1 &= v_1,\ x_1 = 0,\ x_2 = v_2 - v_1 \end{aligned} \tag{5}$$

The logic behind probabilistic reasoning is that the current v_i depending on its p-value will contribute respectively to the estimation process. Therefore, we update the weighting by $1 - P_i$ so that sudden "unexpected" spikes contribute less in the estimation process. This allows the model to refrain from overestimating subsequent v_i's. In turn, if an "unexpected" value turns out to be a shift in the metric stream evolution, as the probability kernel shifts, subsequent "unexpected" values are awarded with greater p-values, allowing them to contribute more to the estimation process. Nonetheless, while adaptive weighting refrains the model from *overestimation at bursty time intervals*, it does not account for monotonic phases of upward and downward trends which often introduce *time lagging effects* in the estimation process. Hence, Equation 5 also features a trend component x_i, updated at each time interval, where ξ is a smoothing weight in the range $[0, 1]$ with values near 1 denoting a preference to favor recent trends. Thus, any lagging effects in the estimation process are reduced by boosting the moving average to the appropriate value base with the dPEWMA for $\hat{v}_{i+1}$ now also incorporating an additive trend component.

Assuming, a stochastic and i.i.d distribution as the bare minimum for a metric stream, we adopt a Gaussian kernel $N(\mu, \sigma^2)$, which satisfies the above requirements, with P_i the probability of v_i evaluated under a Gaussian distribution and computed by Eq. 6. Nonetheless, while a Gaussian distribution is assumed, if prior knowledge of the distribution is made available then only the p-value computation must change in the estimation process.

$$\begin{aligned} P_i &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{z_i^2}{2}) \\ Z_i &= \frac{\delta_i - \hat{\delta}_i}{\hat{\sigma}_i} \end{aligned} \tag{6}$$

At this point, the metric stream evolution and variability, encapsulated by $\hat{v}_{i+1}$ and $\hat{\sigma}_{i+1}$, can be efficiently updated with only previous value knowledge and without repeatedly scanning the entire stream ($N \to \infty$):

$$\begin{aligned} \tilde{a}_i &\leftarrow \alpha(1-P_i) \\ \theta_1 &\leftarrow \tilde{a}_i \cdot (\theta_1 + x_i) + (1-\tilde{a}_i) \cdot v_i \\ \theta_2 &\leftarrow \tilde{a}_i \cdot \theta_2 + (1-\tilde{a}_i) \cdot v_i^2 \\ \hat{v}_{i+1} &\leftarrow \theta_1 \\ \hat{\sigma}_{i+1} &\leftarrow \sqrt{\theta_2 - \theta_1^2} \end{aligned} \tag{7}$$

Having updated the metric stream variability, we proceed to compute the current metric confidence, denoted as c_i. The confidence is a ratio ($c_i \leq 1$) computed from the difference between the estimated and observed standard deviation, and is used as our error evaluation metric. The semantics behind the confidence is that: *the more "confident" the algorithm is, the larger the estimated temporal granularity can be for the reference metric stream.* This supports our framework to "reward" larger adjustments when estimations satisfy the accuracy requested by the user or rollback to a fixed approach when satisfactory estimations cannot be made. Hence, as $\hat{\sigma}_i \to \sigma_i$ the confidence $c_i \to 1$.

$$c_i = 1 - \frac{|\hat{\sigma}_i - \sigma_i|}{\sigma_i} \tag{8}$$

Graph Structure Volatility. This module anticipates and models the topology structure volatility for temporal graphs. Hence, after the graph stream is updated, we compute the dissimilarity of the current and previous graph instance, based on a metric capable of quantifying

Name	Description	Metrics	Vertices	Granularity	Duration
mit	A reality mining network obtained from an experimental study conducted at MIT to demonstrate how social structures are developed via online, mobile and direct conversations [27]	Max Clique	∼1000	5min	9 months
sg09	A face-to-face proximity network obtained from the Dublin Science Gallery in 2009 with data extracted from RFID tags worn by visitors to study human mobility and interactions [17]	Diameter Component Degree Distribution	∼14000	20s	3 months
vanet	A vehicular network from the city of Shanghai exploring wireless communication exchange among vehicles in short-lived proximity-based formed communities [25]	Effective Diameter Giant Comp. Ratio	∼75570	10s	1 day

TABLE I: Datasets Used to Compare the Frameworks Under Evaluation

topological differences. However, identifying and quantifying dissimilarities between graphs is a fundamental challenge with several metrics proposed, although most are limited to either extracting only partial information or are computationally demanding [9] [10].

Therefore, to measure the dissimilarity δ_i between two consecutive graph instances, we adopt the D-measure [26], denoted as $D(G_i, G_{i-1}) \mapsto \mathbb{R}$ and $D \in [0,1]$, which associates to each graph instance a set of probability distribution functions, representing node connectivity distances, and compares them, in terms of three graph metrics. The first captures global differences by comparing each graph average node distance distribution based on the Jensen-Shannon distance ($\mathcal{J}$), which is a method for measuring the similarity between probability distributions [28]. The second compares the connectivity of each node by looking at the local node dispersion ratio (NDr). The last term analyses the differences in the way this connectivity occurs, through the graph $\alpha-$centrality. We deem the D-measure an appropriate metric as it is capable of comparing graphs efficiently and with high precision, especially for networks with volatile connectivity, which is the case for real-world networks where nodes are not fixed and (dis-)appear in time (e.g., mobile, vehicular networks). We also note that the third term of the D-measure which is the computationally demanding term, can be ignored in scale-free settings, yielding a small imprecision penalty and reducing the complexity of the D-measure to $O(n \log n)$. Thus, the complexity is comparable to other topology metrics (e.g., assortativity, transitivity) but achieves more accurate results due to being able to detect dissimilarities even in the connectivity of graph instances [26].

$$D(G_i, G_{i-1}) \cong \sqrt{\frac{\mathcal{J}(dist(G_i), dist(G_{i-1}))}{\log 2}} + |\sqrt{NDr(G_i)} - \sqrt{NDr(G_{i-1})}| \tag{9}$$

At this point, although a threshold-based approach could be used to determine a runtime change in the graph structure (e.g., $\delta_i >$ `thres`), this is error-prone due to its sensitivity to short-lived spikes. Thus, having computed the dissimilarity between the consecutive graph instances, we then proceed to update the graph topology structure volatility evolution, denoted as τ_i, which is also modeled as a moving average adopting a dPEWMA and apply runtime shift detection to determine if the current topology structure volatility is "tolerable".

Runtime Shift Detection. This is based on the CUSUM test, denoted as C_i, and is a hypothesis test for detecting shifts in the statistical properties of i.i.d timeseries [29]. Specifically, there are two hypothesis θ' and θ'' with probabilities $P(\theta')$ and $P(\theta'')$, where the first corresponds to the statistical distribution of the timeseries prior to a shift and the second to the distribution after a shift ($i > t_s$) with t_s denoting the time interval the shift occurs. The CUSUM is computed with sequential probability testing on the instantaneous log-likelihood ratio given for a metric stream at the i^{th} time interval, as follows:

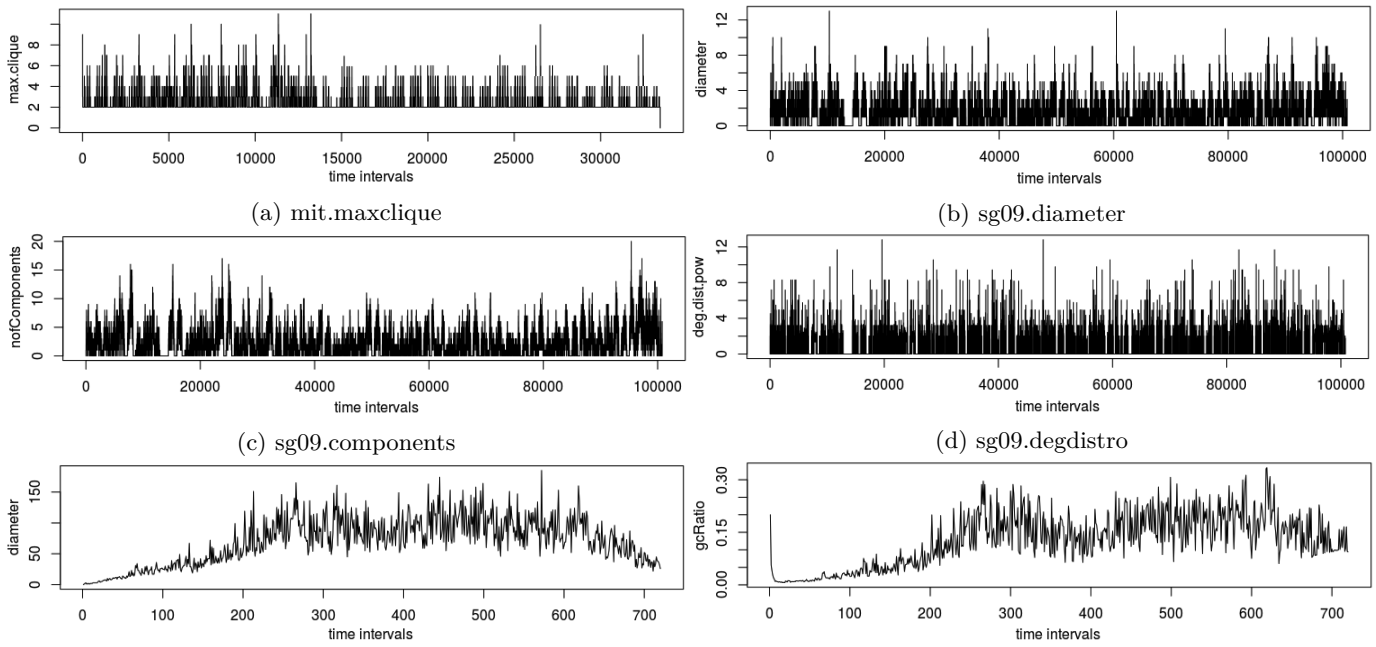
$$\begin{aligned} c_i &= \ln \frac{P(\theta'')}{P(\theta')} \\ C_{i,\{low,\ high\}} &= C_{i-1,\{low,\ high\}} + c_i \end{aligned} \tag{10}$$

where *low* and *high* denote the separation of the CUSUM to identify positive and negative shifts respectively.

The typical behavior of the log-likelihood ratio includes a negative drift before a shift and a positive drift after the shift. Thus, the relevant information for detecting a shift in how the volatility of the graph structure evolves, lays in the difference between the value of the log-likelihood ratio and the current minimum value. A decision function, denoted as L_i, is then used to determine a shift in the graph volatility evolution when its outcome surpasses a threshold ($L_i > h$) determined by the number of standard deviations respecting the user-given maximum tolerable inaccuracy.

$$\begin{aligned} L_{i,\{low,\ high\}} &= \{L_{i-1,\{low,\ high\}} + c_i\}^+ \\ t_s &= \underset{j \le s \le i}{\arg\min} \, (C_{s-1}) \end{aligned} \tag{11}$$

where $L^+ = sup(L, 0)$. Now, let us consider the particular case where the graph structure volatility τ undergoes possible shifts in its evolution modeled by a moving average. Thus, θ' and θ'' can be rewritten as τ' and τ'' respectively, with τ' representing the current evolution, while τ'' the estimated output of the dPEWMA with $\tau'' = \tau' + \epsilon$, and ϵ denoting the estimated magnitude of change in the graph structure volatility. As the structure volatility evolution is used to provide an estimation for $\hat{\delta}_i$, the magnitude of change is equal to $\epsilon = \hat{\delta}_i - \delta_i$. In turn, let $P(\tau')$ and $P(\tau'')$



(a) mit.maxclique (b) sg09.diameter

(c) sg09.components (d) sg09.degdistro

(e) vanet.effdiameter (f) vanet.gcRatio

Fig. 3: The Metric Streams from the Datasets Used to Compare the Frameworks Under Evaluation

be modeled and computed from Equation 6 when adopting a Gaussian kernel. With some calculations (omitted due to limited space), c_i is rewritten, to perform the decision-making with only previous value knowledge:

$$c_{i,\{low,\ high\}} = \pm\frac{|\epsilon|}{\sigma_\tau^2}\ (\delta_i - \tau' \mp \frac{|\epsilon|}{2}) \tag{12}$$

Update Temporal Granularity. Having updated the metric evolution and graph structure volatility we proceed to dynamically adjust the graph stream temporal granularity. To update the temporal granularity we adopt the approach proposed in [16] and extend it to acknowledge multiple metrics over the graph instance and the runtime volatility of the graph topology. Thus, in Equation 13, the estimated temporal granularity Δ_{i+1} is dependent to the current periodicity Δ_i, increasing if variability of the load decreases, and, in turn, decreasing if variability increases. The decision about how large of an adjustment is required, is dependent to the graph structural volatility and the "confidence" of the algorithmic process to (correctly) estimate and follow the runtime evolution of the monitored metric streams. Therefore, when the estimation model is "confident" of its estimations, and the graph volatility does not indicate a change in its evolution, the algorithm will award a larger periodicity.

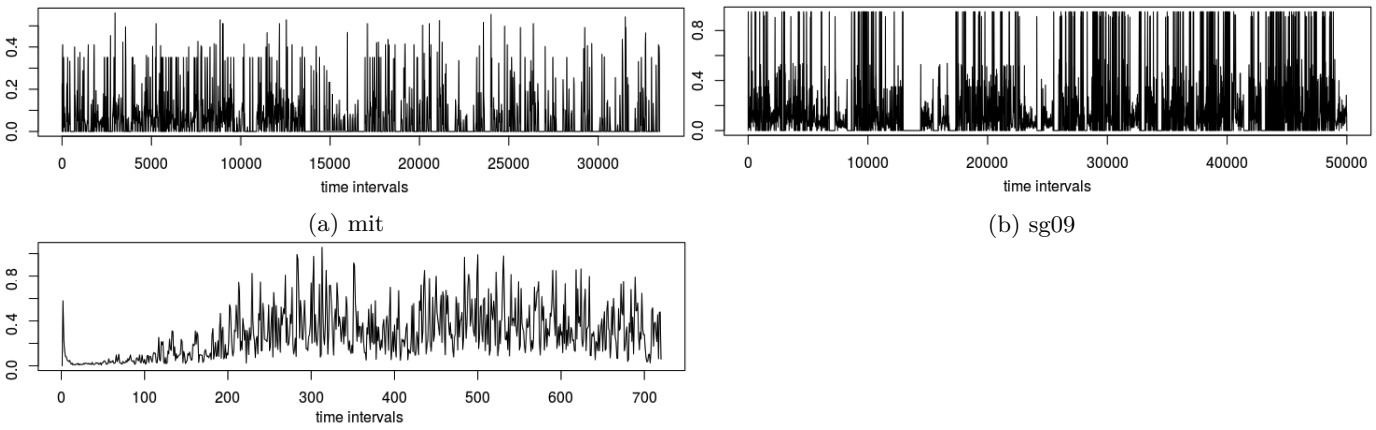
Intuitively, if $\eta \to 0$ then the algorithm converges to a fixed periodicity approach (unless an "exact" estimation is made). In turn, if $\eta \to 1$ an adjustment will take place on each interval even if a confident estimation cannot be made. In turn, if the algorithm process cannot provide a confident estimation within the user-given accuracy guarantees, even for a single metric stream, then the algorithm will rollback to the default periodicity Δ_{min}, in order to preserve accuracy at all times. The complexity of our approach is $O(n \log n)$, with time bounded by the computation of the dissimilarity metric (D-measure). All other calculations (e.g., dPEWMA, CUSUM) feature a constant complexity and are based on previous value knowledge. Moreover, the imprecision η, is the only parameter which is user-defined in the estimation process. Nonetheless, users are free to change: (i) λ which is an optional multiplicity factor (e.g. default $\lambda = 1$) to be used for a more aggressive approach; and (ii) the dPEWMA weights α and ξ, although due to the adaptive weighting process, these may take a wide range of values and can be left to default values for a small imprecision penalty.

$$\Delta_{i+1} = \begin{cases} \Delta_i + \tau_i \cdot \texttt{min}[\lambda(1 + \frac{c_i^m - \eta}{c_i^m})], & \forall m \mid c_i \geq 1 - \eta \text{ and } change = \texttt{true} \\ \Delta_i + \texttt{min}[\lambda(1 + \frac{c_i^m - \eta}{c_i^m})], & \forall m \mid c_i \geq 1 - \eta \text{ and } change = \texttt{false} \\ \Delta_{min}, & else \end{cases} \tag{13}$$

VI. Evaluation

In this section we present a thorough experimentation study based on real-world datasets to compare the overall accuracy and performance of ATMoN towards:

- **Baseline**, where the temporal granularity of the network is fixed and set to the dataset actual granularity while the graph engine is deployed without ATMoN;
- **AdaM**, a low-cost adaptive monitoring framework which embraces a PEWMA as its estimation model and dynamically adjusts the temporal granularity of



(a) mit

(b) sg09

(c) vanet

Fig. 4: Dataset Graph Topology Structure Volatility based on the D-measure: $D(G_i, D_{i-1}) \in [0, 1]$

the metric stream based on the confidence of the algorithmic model to correctly estimate what will happen next in the metric stream. Hence, AdaM adapts the graph temporal granularity solely based on the predicaments of the metric stream evolution.

- **AdaM-topo**, which is AdaM configured to dynamically adjust the network temporal granularity based on the graph topology structure volatility instead of the actual metric stream evolution. To make this possible we feed the structural volatility as a metric stream to the estimation module of AdaM.
- **AvgRandomWalk**, where a heuristic-based random walk is applied over the graph stream to select the runtime temporal granularity by taking the average of the 90th percentile over a series of 100 runs.

We include both AdaM and AdaM-topo in our evaluation to show that solely adjusting the temporal granularity based on either the metric stream evolution or the graph topology structure is restrictive and will yield high errors, especially for networks with highly volatile connectivity. We conduct each experiment with a tight inaccuracy budget configured to $\eta = 0.1$, unless otherwise stated, meaning that accuracy is expected to be at least 90%. For the frameworks using a moving average, we set the smoothing parameter to $\alpha = 0.45$ and the trend parameter to $\xi = 0.85$ which, after testing, is the best configuration for the AdaM framework.

Table I presents an overview of the datasets used to evaluate the approaches under-comparison. Instead of opting for trivial or simulated graphs datasets, we introduce graphs that model real-world and highly volatile dynamic networks where the nodes are actual data sources and are remotely distant from the graph engine. These datasets are available online so that our findings can be easily reproduced. Also, all datasets have been widely used in publications referring to dynamic networks with the metrics selected being metrics used in these publications and are of actual interest. Figure 3 depicts the graph metrics extracted from each dataset, while Figure 4 depicts for each dataset the volatility of the graph topology based on the D-measure.

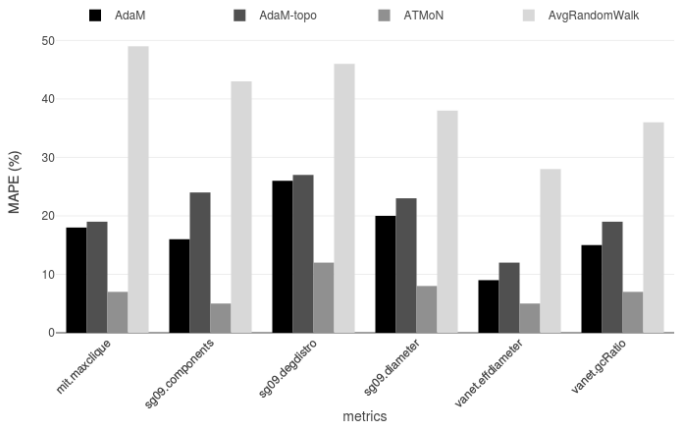
To emulate the behavior of the monitored sources comprising each network in an edge computing environment, a node emulator was developed. Upon instantiation, each emulator receives a unique ID directly mapping to a node of the graph and, from there on, it emulates the exact behavior of the respected node while also disseminating data updates to ATMoN. We deploy ATMoN and the graph engine in Docker containers on Google AppEngine with the testbed comprised of 8VCPUs and 8GB RAM. AppEngine was selected as a suitable cloud platform due to flexible pricing as compute resources are charged per actual cpu-time. Hence, services benefit in terms of cost when less processing load is applied. A similar scheme is applied for ingress and egress network traffic which is of significant importance in edge computing settings where data sources are distant from the cloud.

A. Adaptive Technique Estimation Accuracy

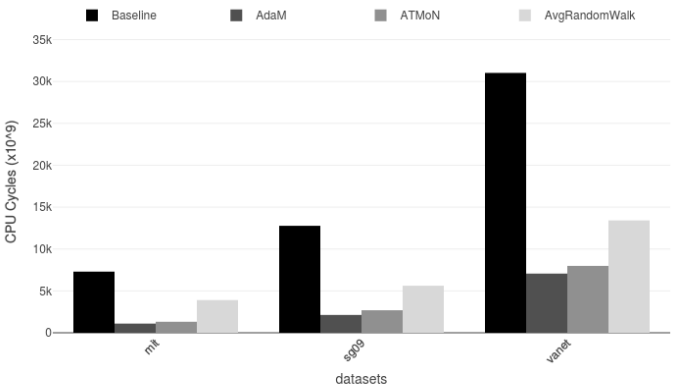
In the first set of experiments, we evaluate each of the adaptive techniques towards their estimation accuracy by measuring the mean absolute percentage error (MAPE) towards the dataset ground truth for each evaluated graph metric. Equation 14 depicts how the MAPE is calculated for each evaluated graph metric, where A_i is the actual metric value for the i^{th} datapoint and E_i is the estimated value. For each adaptive technique, when a datapoint is not present, E_i is considered the last reported value.

$$MAPE_n = \frac{1}{n} \sum_{i=1}^{n} |\frac{A_i - E_i}{A_i}| \cdot 100\% \tag{14}$$

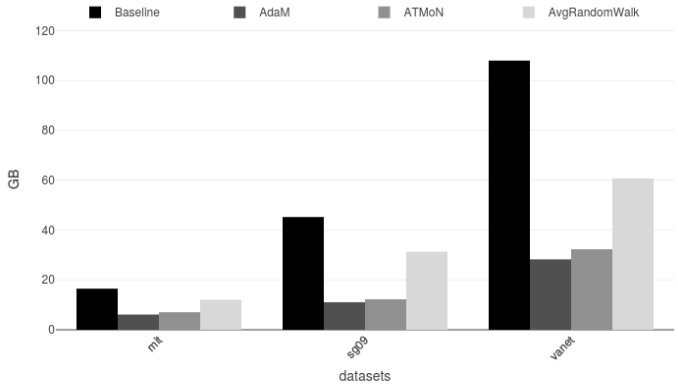
Figure 5a depicts the results of this experiment run. First, we observe that by applying a random walk, it is not feasible to approximate a graph metric stream, especially for metrics featuring highly abrupt and transient phases. Second, we observe that AdaM-topo yields higher errors than ATMoN and AdaM which shows that one cannot use only the graph structure volatility to adapt the temporal



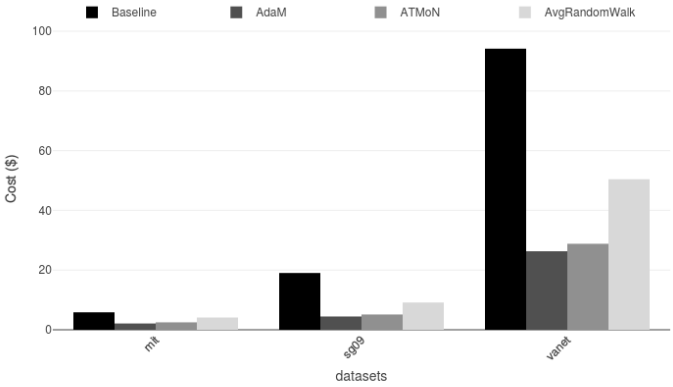
(a) MAPE per Evaluated Metric Compared to Baseline



(b) Hourly Compute Cycles



(c) Data Volume



(d) Daily Cloud Cost

Fig. 5: Accuracy, Overhead and Cost Comparison of the Techniques Under Evaluation

granularity of a metric stream. Third, we observe that ATMoN is the only adaptive technique able to satisfy the given accuracy guarantees ($> 90\%$), even for such volatile metric streams. The only exception is `sg09.degdistro`, which is the most complex and unpredictable metric, where the error is slightly above the desired, at 11%. Most importantly, the difference in error of ATMoN from both AdaM and AdaM-topo is significant. Specifically, the difference in error is, at all times, well above 10% for each experiment run. For the `sg09` and `vanet` networks the error difference even exceeds 15%. Thus, in contrast to adaptive techniques which only base the estimation process on the evolution of the metric stream, *ATMoN maintains user-given accuracy guarantees by acknowledging the volatility of the graph topology and adjusts the network temporal granularity based on knowledge of both the metric and graph evolution.*

B. Adaptive Technique Efficiency and Overhead

In the next set of experiments, we evaluate *efficiency* by measuring the overall overhead imposed to the underlying graph engine which must process the datasets. We measure: (i) *CPU Cycles* consumed to model the graph stream and process the load imposed by each dataset to output the depicted metric streams at runtime; (ii) *Data Volume* generated to store the modeled graph instances and the processed metric streams that must be disseminated at runtime to interested entities and the data sources comprising the network; and (iii) *Cloud Costs* incurred by running the testbed on the cloud based on the Google AppEngine pricing scheme. We note that, when calculating and depicting cloud costs, discounts and compute hours offered by Google are ignored. Also, for figure visualization clarity, in this experiment run we do not depict AdaM-topo which yields significantly higher errors than AdaM and also incurs higher overheads.

Figures 5b-5d depict the results of the experimentation run. First, we observe that ATMoN is able to significantly reduce both the compute overhead and the overall volume of generated data, when compared to the Baseline. Specifically, *ATMoN is able to reduce the computation overhead by at least 76% and data volume by at least 60%.* These numbers improve even more as the network grows and the graph engine is overwhelmed with data. In particular, for the `sg09` network, the compute overhead is reduced by 80%. For the `vanet` network, which is comprised by 75000

Trace \ η	0.01	0.05	0.1	0.15	0.2
mit	5.3	24.6	57.4	69.3	74.2
sg09	12.7	41.3	72.9	80.8	83.1
vanet	9.6	30.3	70.1	74.5	79.8

TABLE II: Data Volume Reduction (%) in Respect to Max Inaccuracy (η)

Trace \ η	0.01	0.05	0.1	0.15	0.2
mit	0.01	0.04	8.0	0.13	0.19
sg09	0.01	0.05	9.1	0.15	0.20
vanet	0.01	0.05	9.8	0.14	0.19

TABLE III: MAPE (%) in Respect to Max Inaccuracy (η)

nodes, the overall data reduction is 71%. This shows that by not embracing adaptivity in an edge computing environment, the overall system is significantly overwhelmed by the volume of data and the required processing which it incurs. In turn, due to constant communication between edge nodes and the cloud service, significant energy is consumed to preserve the wireless link. However, by reducing the compute and network requirements, not only is the network able to achieve greater scalability but the cloud costs are significantly reduced. Specifically, *with ATMoN utilized at the graph engine, an edge computing environment can reduce its daily costs by at least 54% with this number increasing as the network grows larger.*

When comparing ATMoN to AdaM, the former presents a slightly lower compute and data volume footprint which also results in lower cloud costs. The difference in the compute and data volume overhead between ATMoN and AdaM is at most 5% and 7% respectively. This is primarily due to the constant complexity of AdaM, in contrast, to ATMoN where complexity is time-bounded by the dissimilarity metric computation to capture the graph topology volatility. However, for this slight overhead sacrifice, ATMoN yields significantly lower estimation errors. Specifically, ATMoN overall yields 1.5-2x less error compared to AdaM. Hence, by dynamically adjusting the temporal granularity of large-scale dynamic networks, *ATMoN is able to computationally offload graph metric computation, significantly reduce data volume and cloud costs, while maintaining, at all times, given accuracy guarantees.*

Finally, we experiment with the overall data volume reduction in respect to different settings of the maximum tolerable inaccuracy when the graph engine is integrated with ATMoN. Table 2 depicts the results of this analysis where, as expected, relaxing accuracy guarantees provides dynamic and latency-sensitive networks "breathing space" by requiring less compute effort to output graph insights. Nonetheless, the most interesting findings for this experiment are presented in Table 3. In particular, we observe that *at no point does ATMoN violate the accuracy requested by the user even for tight error bounds.* This is due to the fact that the ATMoN estimation process will not output an adjustment for the graph metric computation rate when a "confident" estimation cannot be made to ensure that the accuracy guarantees given by the user are obeyed at all times.

VII. Conclusions and Future Work

In this paper, we have presented ATMoN, a novel adaptive framework for monitoring applications that are modeled as temporal graphs. ATMoN provides a flexible architecture to inexpensively and dynamically adapt the temporal granularity graph metrics are computed. This significantly eases graph processing and also reduces data volume and cloud costs. To achieve this, ATMoN uses a low-cost probabilistic learning model that approximates both the metric stream evolution and the volatility of the graph topology structure, while respecting user-given accuracy guarantees. ATMoN is available as open-source and can be served as the vehicle for a number of ongoing research efforts, such as monitoring network dynamics (e.g., fake news epidemiology), community structure and density (e.g., capacity provisioning), abnormal behavior detection (e.g., malicious attacks) and identifying recurring events (e.g.,trends and seasonal patterns).

In the imminent future, ATMoN will be extended to support multivariate graph metric streams. This will allow ATMoN to acknowledge structural changes not only at a graph scale but also in graph communities to dynamically adjust the network temporal granularity at a finer granularity. Consequently, graph engines will be able to allocate processing time only to truly volatile segments of the graph.

Acknowledgement. This work is partially supported by the Regional Government of Madrid (CM) grant Cloud4BigData (S2013/ICE-2894) co-funded by FSE & FEDER, the EU Commission in terms of the H2020 projects Unicorn (IA 731846) and RECAP (RIA 732667), and the NSF of China grant 61520106005.

References

[1] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 177–187.

[2] P. Holme and J. Saramaki, "Temporal networks," *Physics Reports*, vol. 519, no. 3, pp. 97 – 125, 2012, temporal Networks.

[3] H. Efstathiades, D. Antoniades, G. Pallis, M. D. Dikaiakos, Z. Szlavik, and R.-J. Sips, "Online Social Network Evolution: Revisiting the Twitter Graph." in *2016 IEEE International Conference on Big Data*, 2016.

[4] N. Masuda, K. Klemm, and V. M. Eguíluz, "Temporal networks: Slowing down diffusion by long lasting interactions," *Phys. Rev. Lett.*, vol. 111, p. 188701, Oct 2013.

[5] J. Traub, S. Breß, T. Rabl, A. Katsifodimos, and V. Markl, "Optimized on-demand data streaming from sensor nodes," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: ACM, 2017, pp. 586–597.

[6] V. Kostakos, "Temporal graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[7] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 721–732, May 2014.

[8] A. Li, S. P. Cornelius, Y.-Y. Liu, L. Wang, and A.-L. Barabási, "The fundamental advantages of temporal networks," *Science*, vol. 358, no. 6366, pp. 1042–1046, 2017.

[9] A. P. Iyer, L. E. Li, T. Das, and I. Stoica, "Time-evolving graph processing at scale," in *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, ser. GRADES '16. New York, NY, USA: ACM, 2016.

[10] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.

[11] D. Trihinas, G. Pallis, and M. Dikaiakos, "Monitoring Elastically Adaptive Multi-Cloud Services," *IEEE Transactions on Cloud Computing*, vol. 4, 2016.

[12] L. Mearian, "Self-driving cars could create 1GB of data a second," https://www.computerworld.com/article/2484219/.

[13] W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[14] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "ADMin: Adaptive Monitoring Dissemination for the Internet of Things," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[15] L. Fan and L. Xiong, "An adaptive approach to real-time aggregate monitoring with differential privacy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2094–2106, Sept 2014.

[16] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "AdaM: an Adaptive Monitoring Framework for Sampling and Filtering on IoT Devices," in *IEEE International Conference on Big Data*, 2015, pp. 717–726.

[17] L. Isella, J. Stehle, A. Barrat, C. Cattuto, J.-F. Pinton, and W. V. den Broeck, "What's in a crowd? analysis of face-to-face behavioral networks," *Journal of Theoretical Biology*, vol. 271, no. 1, pp. 166 – 180, 2011.

[18] R igraph, "http://igraph.org/r/."

[19] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, Hollywood, CA, 2012, pp. 17–30.

[20] W. Han, Y. Miao, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, W. Chen, and E. Chen, "Chronos: A graph engine for temporal graph analysis," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: ACM, 2014, pp. 1:1–1:14.

[21] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, Nov 2016.

[22] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 145–156.

[23] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, "Community structure in time-dependent, multiscale, and multiplex networks," *science*, vol. 328, no. 5980, pp. 876–878, 2010.

[24] R. K. Pan and J. Saramäki, "Path lengths, correlations, and centrality in temporal networks," *Physical Review*, vol. 84, no. 1, p. 016105, 2011.

[25] M. D. Nicholas Loulloudes, George Pallis, "The Dynamics of Vehicular Networks in Large-Scale Urban Environments," in *1st IEEE International Conference on Collaboration and Internet Computing*, ser. IEEE CIC 2015, 2015, conference.

[26] T. A. Schieber, L. Carpi, A. Díaz-Guilera, P. M. Pardalos, C. Masoller, and M. G. Ravetti, "Quantification of network structural dissimilarities," *Nature communications*, vol. 8, p. 13928, 2017.

[27] N. Eagle and A. (Sandy) Pentland, "Reality mining: Sensing complex social systems," *Personal Ubiquitous Comput.*, vol. 10, no. 4, pp. 255–268, Mar. 2006.

[28] D. M. Endres and J. E. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1858–1860, July 2003.

[29] Y. Luo, Z. Li, and Z. Wang, "Adaptive cusum control chart with variable sampling intervals," *Computational Statistics & Data Analysis*, vol. 53, no. 7, pp. 2693 – 2701, 2009.