# ElasticSDK: A Monitoring Software Development Kit for enabling Data-driven Management and Control in 5G

Xenofon Vasilakos[1,2], Berkay Köksal[1], Dwi Hartati Izaldi[1,3,4], Navid Nikaein[1], Robert Schmidt[1], Nasim Ferdosian[1], Riri Fitri Sari[3], and Ray-Guang Cheng[4]

[1]EURECOM, France, Email: firstname.lastname@eurecom.fr
[2]Department of Electrical and Electronic Engineering, University of Bristol, UK
[3]University of Indonesia, Indonesia Email: riri@ui.ac.id
[4]National Taiwan University of Science and Technology, Taiwan,
Email: crg@mail.ntust.edu.tw

***Abstract*—5G networks generate massive (quasi-) real-time data streams that different network apps can exploit to implement sophisticated single- or cross-domain control and management logic. This paper presents ElasticSDK, a Software Development Kit specially designed to abstract the development and chaining of such agile 5G monitoring apps for the control, management, and coordination of the underlying 5G network heterogeneous modules. Custom apps can collect, incrementally process and further expose flows in a flexible Pub/Sub fashion via appropriate SDK API calls, thus sharing both raw and complex data flows among themselves. Furthermore, the design of ElasticSDK allows respecting typical 5G data ownership and privacy models, as desired by the different 5G stakeholders ranging from physical infrastructure providers up to service providers over slicing. Finally, we provide two important contributions to the 5G open-source research community: (i) a RAN monitoring prototype implementation over the ElasticSearch and FlexRAN platforms that allows to demonstrate ElasticSDK app development and capturing hierarchical control features of typical SDN-enabled 5G architectures, and (ii) a first-ever publicly available dataset of realistic 5G RAN monitoring traces.**

***Index Terms*—5G mobile communication, network monitoring, SDK**

## I. Introduction

The anticipated complexity of 5th Generation (5G) mobile networks pushes towards minimizing human engagement and relying on vast amounts of (quasi-) real-time monitoring data in a new era of Data-Driven Control & Management (DDCM). A crucial aspect of DDCM regards delivering network *monitoring as a service* on a per-use-case basis. Currently, there are different monitoring tools targeting specific use cases and network domains, namely, the 5G Core Network (CN), the transport network, and the Radio Access Network (RAN), hence a requirement for a multiservice execution environment to *unify* all of the earlier into customized virtual views tailored to use case requirements. Dynamic bit rate video optimization poses such a representative example, requiring to utilize both User Equipments (UEs) and RAN information, each monitored by different tools. Last, complexity gets further amplified by the concept of network "slicing" into multiple (possibly recursive) logical networks (a.k.a. *"slices"*) composed of virtual and/or physical resources over a common infrastructure, with implications on monitoring spanning from practical up to policy contradictions between the various stakeholders.

The topic is highly important and contemporary, with various recent works relying on the combination of existing Monitoring Frameworks (MFs), e.g. the Machine Learning (ML)-based works of [1]–[4] or the knowledge-defined networks of [5], [6]. Nevertheless, none of these works is meant to address, nor can address, these challenges in 5G. What we need instead, is a properly designed monitoring *Software Development Kit (SDK)*.

To cover a gap in the 5G literature and open-source community, this paper presents ElasticSDK, an SDK adopting the design principals discussed [7] for *abstracting* the development as well as the chaining of *agile* and/or use case-specific monitoring *apps*. Based on this main contribution, we further discuss and contribute to the 5G open-source research community a publicly available RAN monitoring *prototype* SDK implementation, namely ElasticSDKv1.0, and (to the best of our knowledge) a first-ever publicly available dataset of *realistic* 5G RAN monitoring traces. In a nutshell, our main points of contribution can be summarized as:

- **SDK design:** ElasticSDK abstracts both the development and the chaining of agile monitoring apps in order to enable the formation of a technology-agnostic *application plane* for (i) implementing sophisticated control and management logics; (ii) providing a unified monitoring view to higher software layers; and (iii) satisfying the conflicting interests of all involved stakeholders with respect to (w.r.t.) data privacy/ownership, monitoring cost, accountability, etc. All above are enabled across single or multiple domains, over

This work has received funding from the European Union Horizon 2020 Framework Programme under grant agreement No. 761913 (SliceNet) and 762057 (5G-PICTURE).

flat or hierarchical architectures. Major consideration is put on the *elasticity* properties of our design. This refers to the ability to cope with the anticipated massive scale of data flows from heterogeneous sources in dynamic 5G environments, and the low *time-scales* and *high frequency* of raw and/or processed monitoring data.

• **Prototype:** We contribute a prototype SDK implementation, namely `ElasticSDKv1.0`, which cooperates fully with ElasticSearch (ES) distributed search and analytics engine and Mosaic5G FlexRAN platform [8], [9]. Due to the SDK abstraction, `ElasticSDKv1.0` remains applicable to *other* underlying platforms and can be extended to cover the CN, as well. The prototype comes with an execution environment including a group of specific functions and methods that facilitates the life-cycle of ML apps for an intelligent, proactive and adaptive DDCM. Apps can interact with the underlying network modules via our SDK to access resource/state and control the corresponding behaviors, meeting with most typical *hierarchical* features of SDN-based 5G network architectures. This translates to the ability of `ElasticSDKv1.0` to gather monitoring data in both a local, e.g. a Base Station (BS), and a global/regional level (multiple BSs), matching the hierarchical structure of the underlying FlexRAN controller. Last, the prototype considers a hierarchical data-ownership model, allowing both infrastructure providers and slice owners to share *only* the data they wish to share with other stakeholders through add-on *filters*.

• **Demonstration & Publicly available 5G RAN traces:** Our demonstration shows that our design allows to produce, store and retrieve a rich set of monitoring data in a flexible manner by tuning the Pub/Sub monitoring *frequency* and adjusting data *granularity*, while identifying performance trade-offs and possible implications on (quasi-) real-time DDCM.

We further contribute [10] raw and processed 5G RAN monitoring data from the MAC, RRC and PDCP layers to the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD). To our knowledge, this is still a rare contribution of *real* 5G RAN monitoring data to the 5G community.

For the rest of this paper, Sec. II outlines and compares the features of popular community monitoring tools. Sec. III analyses the design challenges of our proposed design, followed by our demo prototype implementation in Sec. IV. Sec. V presents a data processing use case followed by a prototype demonstration in Sec. VI, before we conclude in Sec. VII.

## II. Background

Standard messaging queues such as RabbitMQ can be used to add a *Publish/Subscribe (Pub/Sub)* feature to a MF. However, such queues do *not* operate as storage units. They are designed for transferring queue content as a whole, but *not* for tuning the rate of data consumption by monitoring clients. Tools like Apache Kafka employ offset features allowing to skip queue messages; yet the clients remain responsible for processing the data received, as aggregation queries are not a common use case for messaging queue-based frameworks. Monasca is a fault-tolerant MF using a Rest Application Program Interface (API) for storing and querying metrics, as well as for history information. It uses *only* the *HTTP* protocol to simplify its design and to allow for a rich data description via the concept of "dimensions", i.e metrics in the form of (key, value) pairs. Last, Monasca employs real-time thresholding and alarms on metrics.

The work of [11] presents a generic comparison between agent-based and agent-less *cloud-based* monitoring. The authors discuss in depth the cloud configurations for both schemes alongside challenges and requirements. The proposed prototype uses Raspberry Pie to push data from the network to a data store monitored the Prometheus monitoring API. This design is claimed to scale well on small/medium-sized networks and with reasonably-sized data. Nevertheless, this assumption does not meet the massive size and frequency of the anticipated 5G network measurements. In addition, Prometheus does not offer sophisticated horizontal scaling or user management features that medium-large scale environments would require. ES is an Apache License-based distributed text search engine, hiding complexity behind a RESTful API [12] that can index data in very high frequencies. Moreover, it splits data into fields, so that client apps send complex read/write/aggregate queries in different granularity levels to different fields. While not integrating a Pub/Sub scheme, community-based solutions exist using Redis, Apache Kafka or Google Cloud Functions, which utilize the Logstash to facilitate Pub/Sub operations or database triggering functionality between ES clients. All solutions above are crafted to operate on a set of specific use cases up to a certain level of frequency and granularity. Taking a step forward, `ElasticSDK` addresses the challenges in 5G by covering the gaps related to monitoring, as identified in [13] and summarized in Table I. In further, `ElasticSDK` greatly covers the needs of heterogeneous 5G networks that generate massive and (quasi-) real-time data streams for DDCM purposes.

| Tool / Criteria | Storage | Filtering | Scalability | Delay |
|---|---|---|---|---|
| Apache Kafka | ++ | N/A | +++ | +++ |
| RabbitMQ | N/A | N/A | + | ++ |
| Monasca | +++ | ++ | ++ | + |
| Prometheus | +++ | +++ | ++ | + |
| InfluxDB | +++ | +++ | ++ | + |
| ElasticSearch | +++ | +++ | +++ | + |

TABLE I: Empirical performance of well-known platforms with useful features for monitoring systems as presented in Table 5.4 of [13]. Plus signs denote performance merit: fair (+); good (++); best (+++). "N/A" denotes non-applicability or lack of merit.

## III. SDK & Prototype Design Principals

We elaborate on the challenges for designing a proper SDK along with a detailed description of `ElasticSDK`[1],[2] and our prototype implementation `ElasticSDKv1.0`[3].

[1]Code access: gitlab.eurecom.fr/mosaic5g/store/blob/mon-sdk/sdk/lib/elasticmon_sdk.py.

[2]Tutorial wiki for the prototype: hackmd.io/zaLGcuLyROWP1O-Vu0bM_g#Install-and-configure-ElasticSearch-and-Kibana

[3]https://gitlab.eurecom.fr/mosaic5g/elasticmon/tree/develop

### A. Challenges

*1) Seamless Data Flow:* This is, perhaps, the greatest requirement for 5G monitoring due to a multitude of important parameters involved: (i) the massive data scale combined with (ii) very high recording frequencies and (iii) different levels of monitoring data granularity; the (iv) short time-scales such as during a 1 ms Transmission Time Interval (TTI); last, the necessity for the different control and management apps to (v) seamlessly exchange raw or structured data. The SDK must take all above into consideration and natively offer a set of fundamental processed data values, i.e. minimum, maximum and range values. From the implementation side, an appropriate data store must be selected to maintain a seamless data flow between the network controller and the custom apps that require online or history data, and network statistics.

*2) Stakeholders tussles and costs:* The *cost* of monitoring is high in 5G for the reasons mentioned above. At the same time, it is not always clear which stakeholder (i.e., the network slice owners and the infrastructure providers) has the responsibility, the capability and the will for bearing the cost of gathering and maintaining the monitoring data. Despite their conflicting interests (avoiding costs, and privacy and security policies), shareholders can still want to share monitoring data in trade for a mutually beneficial DDCM. A main SDK design challenge here regards enabling stakeholders with appropriate APIs that allow to flexibly share costs and responsibility. This can be done through customized views, levels of granularity and aggregation of shared data after stakeholder-defined rules.

*3) Monitoring APIs:* Custom apps must be provided with a proper set of APIs and abstraction layers that hide the technical aspects of the framework, while allowing essential filtering and aggregation operations on data, and to support heterogeneous underlying data store systems. A proper API must not only encourage the development of innovative operations, but also enable to set up rules and access limitation policies against damaging data or the network itself. Even more importantly, using appropriate Monitoring APIs in different network/slice instances can enable the collaboration of various MFs, as the latter become capable of processing same requests from monitoring apps via well-known API calls.

*4) Scalability & other 5G challenges:* The SDK APIs, the framework as a whole and, particularly, the components related to the underlying data store, must *scale* w.r.t. data volumes and the high frequency of interactions with the data processing apps. A 5G MF must have the right features for enabling alarms on time or even act proactively by reporting predicted future measurements and network states. The underlying design challenge here is to allow add-on ML/Artificial Intelligence (AI) app components such as Long Short-Term Memory Networks (LSTM) models to report important events for DDCM ahead of time, which is very significant for (quasi-) real-time network control decisions.

### B. SDK architecture & prototype components

Figure 1a presents a *high-level* design overview of our SDK and its interfaces with the southbound Control Plane (CP) and the northbound custom monitoring apps. The producer API of the controller writes the measured data and statistics from the CP to the data store. These measurements are in an adjustable granularity. Consumer apps such as Control Applications (C-Apps) in the case of [8] pass their requests via a "FILTER" module internal to the SDK that performs filtering operations such as selecting desired data or aggregating results (max, min, range, count, per user, slice, cell, etc). The filtering module enables to also control the access and perform CRUD (Create, Read, Update, and Delete) operations to the underlying data store. Consumer apps retrieve data when a message is published on a subscribed channel or when a database watcher fires due to some metric threshold breach. It is important to note that `ElasticSDK` APIs allow a diverse set of monitoring systems and data store frameworks, to be used under the hood, as the APIs remain responsible for the conversion of client requests to the underlying monitoring modules and data stores.

As previously stated, facilitating different frameworks with a Pub/Sub module or database watchers can be handled by the `ElasticSDK` APIs to deliver the response over a different channel. Similarly, CP measurements should be sent over a separate channel forming a direct communication line with a higher write performance to the data store. `ElasticSDK` accepts requests from any platform able to send applicable requests via its *monitoring APIs*. Clients send requests and retrieve data through the Pub/Sub module by subscribing to channels or Database Triggers. After parsing incoming data, consumer apps can process input and generate more complex data. Note that consumers are responsible for both identifying the consumption method and for adjusting the required time between incoming data. Requests can also refer to aggregated results in-between time points to provide mean, median, min, max, range or count values to consumers, which may switch to producers to output data back to data store via `ElasticSDK` for other apps to consume.

## IV. Prototype Framework Implementation

The current prototype version, `ElasticSDKv1.0` works on top of the ES search engine, the hierarchical controller [8] of FlexRAN's programmable SD-RAN platform, and the OpenAirInterface (OAI) [14]. `ElasticSDKv1.0` can currently monitor RAN data as part of the Mosaic5G [15] community-led consortium for building agile 5G service platforms and opening fast wireless innovation. Figure 1b portrays the modules of `ElasticSDKv1.0`. FlexRAN's controller runs over an OAI user plane network infrastructure, while ES is used to store the control plane data from the southbound API (the FlexRAN *producer*). The FlexRAN Producer API is a dedicated lightweight app deployed over FlexRAN that stores raw monitoring data to ES. Once data get stored, apps can consume it through `ElasticSDK`'s API.

`ElasticSDKv1.0` can already support the most important envisioned features. First off, `ElasticSDKv1.0` employs a *configurable FlexRAN producer app*, in the southbound of `ElasticSDKv1.0` that can be configured using a REST

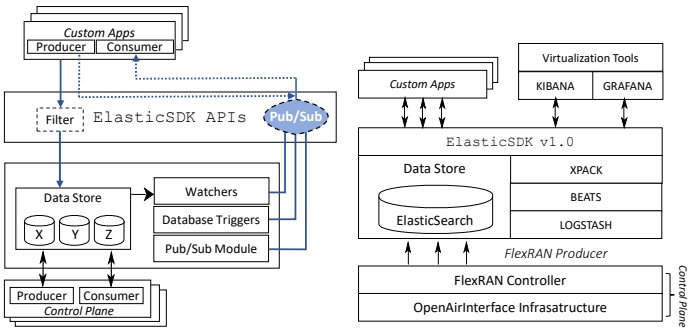



Fig. 1: (a) `ElasticSDK` design & interfaces. Notice the emphasized "Filter" and "Pub/Sub" modules in the SDK layer. (b) Prototype SDK implementation lying underneath Custom Apps and Visualization Tools.

north-bound API, holding the crucial role of continuously feeding `ElasticSDK` with monitoring data.

Second, the current version supports *multi-UE monitoring with per slice aggregation* regarding arbitrary numbers of UEs. UEs can be indexed by the FlexRAN producer app, hence monitored by the prototype. Data monitoring *aggregation* both per slice and UE is supported. In further, different instances of `ElasticSDK` can be used together over one or more data store instances, thus enhancing privacy guarantees via data storing isolation, alongside the filtering mechanisms that are native to ES or any other data store to be supported as well in future prototype versions.

Third, the prototype comes with all necessary *sample* monitoring apps for testing purposes such as for aggregation, filtering, selection, etc. It, also, comes with a *webserver for API calls* used to send/process API requests and a set of fundamental API calls including get/post/delete examples, UE count and other aggregation operations on monitoring fields like average, sum or mean. In addition, it contains all the necessary documentation about the usage and the configuration setup of FlexRAN Producer App.

Last, `ElasticSDK v1.0` employs useful modules and plugins, which we discuss in detail in Sections IV-A and IV-B.

### A. ElasticSearch

By leveraging ES API, `ElasticSDK v1.0` offers a *flexible* and *scalable* monitoring system based on a distributed data store that allows concurrent data-access to monitoring apps generating and exchanging (quasi-) real-time data. The data store can be effortlessly composed of many nodes with horizontal scaling, due to the ability of ES to manage multiple nodes. A clustered ES can store CP data from the southbound API. `ElasticSDK v1.0` provides bidirectional communication with clients, addressing the requirement of serial chain or parallel app development. ES is a text search engine that provides support for the filtering module required for data aggregation and pre-processing. Similarly, the security module corresponds to the `X-Pack` plugin, which provides user management features to provide authentication and privilege control. Next, the `Beats` plugin is used to monitor the health

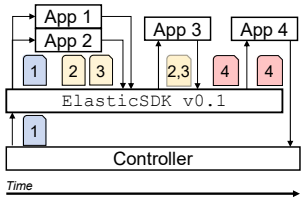



Fig. 2: Data exchange between the CP and custom apps.

of the nodes via observing resource usage, latency and so forth. Finally, the `Logstash` plugin is used for logging data store cluster events on node devices. Event listeners, alarms and triggers facilitate `Logstash` data in order to operate.

### B. FlexRAN Producer API

This is a dedicated app deployed over `FlexRAN` for storing data gathered from agents to the data store. It is lightweight to prevent any performance bottlenecks on other FlexRAN apps. Once data get stored, apps can consume it, thus `ElasticSDK` takes over the network load from the controller by becoming the end-point for apps. We note two critical performance points for this app: (i) high write granularity, and (ii) the delay between saving data to the data store and the data becoming available for consumption. We note an unavoidable delay between separate requests to ES, posing consideration about high frequency operations; yet ES includes the "Bulk API", which enables to perform multiple operations with single API calls.[4]

## V. Data Processing Use Case

Figure 2 exemplifies data exchange over time between apps running over the northbound API and their interaction with the controller. Apps can fetch data from the data store to generate further data by mining the original statistics and reports of the controller resulting in *new* data indexed back to `ElasticSDK`. Any app on the outer level of the network could fetch processed data for further processing and/or modify the state of the controller. In this context we identify a use case that combines (i) *serial-chain* and (ii) *parallel* data *flows* between apps. For common queries, practical apps can write processed data in the data store for other apps to use, thus avoiding unnecessary queries. This forms a serial app chain for write & read actions, using `ElasticSDK` as a *pipeline*.

Figure 3 shows a use case that combines both serial and parallel data flow pipelining. Notice the chain of control apps including *Positioning*, *Crowd distribution*, *Tracking* and *Handover control*, which are in a similar form to the one of Fig. 2. The *pipeline* starts with the FlexRAN controller writing raw data types and statistics of connected UEs. Such data include numerous metrics explained in [8]. Notice the use of a blue layer in the data flow pipeline at this stage. Positioning and Crowd apps consume mobility-related data *in parallel* from that blue layer such as Power Headroom Report (PHR), Reference Signal Received Power (RSRP),

[4]mosaic-5g.io/apidocs/flexran/

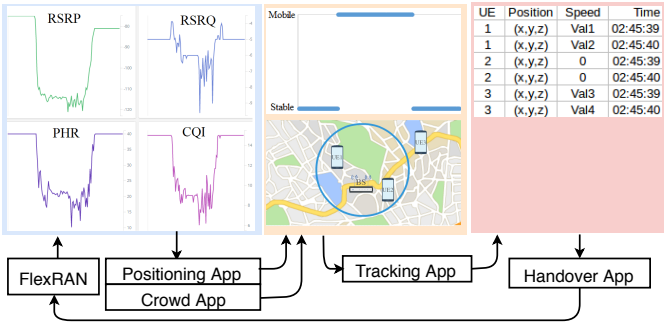
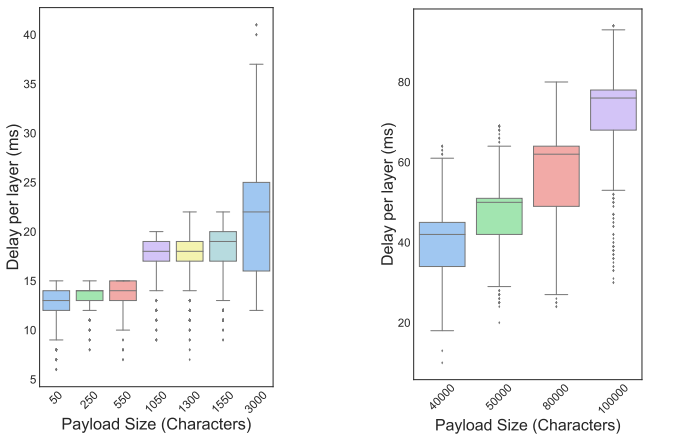



Fig. 3: Sample use case with serial and parallel data flow pipelining. Notice the three layers of the serial part of the chain from left to right: blue, feeding Positioning and Crowd Apps in parallel; yellow feeding the Tracking app; orange, feeding the Handover App.




(a) Common payload sizes.

(b) Large and massive payload sizes.

Fig. 4: Payload size (x-axis) in number of characters against round-trip delay (y-axis). Boxes include median values. Whiskers correspond to 95% confidence intervals with outliers marked with dots beyond whiskers.

Reference Signal Received Quality (RSRQ), Channel Quality Indicator (CQI). Based on this, the Positioning App computes the mobility status of each UE while the Crowd App computes an estimated positioning map of the UEs. Positioning and Crowd apps' output is stored back in the pipeline (yellow part). Then the Tracking App fetches the former joint output to compute things like the motion statistics of UEs over time, stored to the next (orange) layer of the data flow pipeline. This includes position changes, movement direction, velocity and other relevant motion statistics. Finally, a Handover App fetches the time series of the previous outputs to trigger a handover back to the CP for a subset of UEs based on the metric of interests such as Quality-of-Service (QoS), cell load, or user mobility.

Considering a common UE behaviour in the OAI LTE platforms with `FlexRAN` controller, we concluded that the connected devices have RSRP values between -75dBm to -128dBm and usually lose connection around -120dBm. A similar analysis is also made to the other metrics related to UE mobility. The total knowledge acquired from our own monitoring data enabled our network to develop a simple app chain that performs handover operations before QoS drops below a threshold.

## VI. Prototype Demonstration

We demonstrate in practice data flows exchange based on "read" and "write" app actions. We record *delay* and *disk* space measurements under *high frequency* usage by the `FlexRAN Producer API`. Due to space restrictions, we outline the essentials of our demo scenario below, and provide a meticulous description online at CRAWDAD [10] along with our publicly available RAN monitoring data traces.

### A. Data exchange delay

We adapt a serial chain data flow use case scenario, deploying 5 serial C-Apps that use `ElasticSDK` as a *pipeline* for exchanging data, and measure round-trip delay between "layers". A layer denotes one write action from a C-App and one read action from a sequence of C-App, yielding one measurement sample. In overall, we measure 700 data exchange delay samples using different payload sizes. The C-Apps are deployed on one machine that is equipped with a I7 7700HQ CPU with a clock speed of 2.8 GHz, a 16 GB of RAM, and a stable 50 Mbps Ethernet connection. To provide a more realistic evaluation, C-Apps communication is handled via an `ElasticSDK` instance running on a remote network, rather than a low latency local network MF instance.

Figure 4 depicts two boxplot diagrams of the measured round-trip delay times against the payload size of the exchanged data. Boxplot (a) regards common payload sizes (50 up to 3K characters of monitoring data), whereas boxplot (b) large and massive payload sizes (40K up to 100K characters) to investigate the delay performance under heavy monitoring load conditions. An average `FlexRAN` measurement of one UE is approximately 1.5K characters long in JSON format, having a round-trip time close to 15 ms. As the number of connected UEs increases, the data size grows as well, hence increasing the round-trip delay close to 50 ms for 30 UEs.

The results show an unavoidable delay cost even with very small payload sizes (Fig. 4(a)) and a *direct impact of payload size on round-trip time* from/to `ElasticSDK`. Both round-trip delay values and delay variability increase with payload size, particularly for ones $\geq 3K$. This is due to the large character sizes used to stress our system, particularly, in Fig. 4(b). The input generated by the first C-App arrives to the fifth C-App of the chain in 5x the round-trip delay for the given payload size. The cumulative delay of the chain creates a *time offset* to the output of the chain. This can lead to late or faulty decisions by consumers, thus it is strongly recommended for serial C-Apps to avoid exchanging unnecessary data and to preserve low payload sizes. Another way to decrease delay is by splitting the workload to fewer C-Apps. Based on the above, this would decrease the overall delay of the chain at the cost of a computational overhead per C-App.

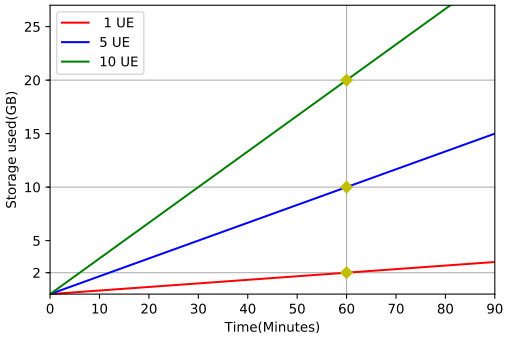



Fig. 5: Disk usage of `FlexRAN Producer`

### B. Disk space requirements

We record real-time data from the `FlexRAN` controller, using different numbers of connected UEs. Figure 5 portrays the storage amount used by the `FlexRAN Producer API` to store statistics with 1 measurement per 1 ms. This translates to 1000 measurements from each UE per second. Given this frequency and without any data replication, `ElasticSDK` requires ∼2GB of storage per hour to store the data of one UE. Note that the size of `FlexRAN` statistics in JSON format can change w.r.t. different variations of activated southbound network apps; yet we omit this change to provide a stable behaviour of the resource usage during our evaluation.

Providers can *lower* the recording *frequency* of statistics as well as of configuration settings that are rarely updated over long periods of time. This enables to fine-tune storage space and network traffic savings against a resulting time offset between latest actual metric changes and the time of consumed metric updates. To provide an example, we tuned `ElasticSDK` to record the configuration settings of the `FlexRAN Agent` once every 5 seconds. Within 3 hours of recording time merely 3 MB of storage was used for 2200 measurements. On a worst case scenario, this implies that a C-App can recognize a configuration change with a 5 second delay. On the contrary, increasing the recording frequency enables a more sensitive/precise monitoring analysis over metric changes at the cost of increased disk space consumption. Likewise, measurement sizes increase with the number of connected UEs, implying a higher storage demand (see Fig. 5).

The former verifies a dual trade-off: (i) between the data recording frequency and the update status of monitoring measurements, with the latter having implications on (quasi-) real-time DDCM; (ii) between the number of connected UEs and total storage consumption. We opt for dynamic recording frequency schemes that can be adjusted after metric changes. Nonetheless, such a system can be computationally costly for producer apps, and difficult to develop and adjust to network requirements. Finally, we note that UEs can arbitrarily (dis)connect to (from) the network, thus the needed resources should be available to support monitoring at all times.

### C. 5G RAN monitoring trace datasets

We collected 5G RAN monitoring data using our prototype implementation, which we contributed [10] to CRAWDAD. To our knowledge, this is still a *rare* contribution of *real* 5G RAN monitoring data to the 5G community, containing over a 100 metric categories from the MAC, RRC, and PDCP layers, as exposed by the FlexRAN controller[5] in JSON format. Data are organized in 5 different raw and 5 different corresponding processed datasets recorded for one enhanced NodeB (eNB) w.r.t. 5 different UE motion and distance patterns.

## VII. Conclusion

We present `ElasticSDK`, an SDK design for abstracting the development and the chaining of agile monitoring apps to enable the formation of a technology-agnostic application plane for implementing sophisticated control and management logics, as well as for providing a unified monitoring view and satisfying the conflicting interests of different stakeholders. Special design consideration is taken to facilitate massive scales of data coming from heterogeneous sources in dynamic 5G environments, in low time-scales and with a high frequency. Also, the SDK design employs a Pub/Sub data flow model for custom apps. Furthermore, we present a prototype RAN monitoring implementation to demonstrate our SDKs merits, and to produce and publicly contribute RAN monitoring dataset traces.

Future work includes exploring the potential of deploying multiple interacting `ElasticSDK` implementation instances to enable regional monitoring, hence enable different operators and slices to merge their network data into larger datasets. Additionally, we aim to investigate the implications of delay between processing apps in a serial chain, and try to lower it by, e.g., establishing direct communication channels between apps. Last, our prototype implementation will be extended to cover the 5G CN segment using the LL-MEC [16] platform.

## References

[1] S. Ayoubi et al., "Machine Learning for Cognitive Network Management," *IEEE Communications Magazine*, vol. 56, pp. 158–165, 2018.
[2] M. Zorzi, A. Zanella et al., "Cognition-Based Networks: A New Perspective on Network Optimization Using Learning and Distributed Intelligence," *IEEE Access*, vol. 3, pp. 1512–1530, 2015.
[3] S. Zander et al., "Sub-flow packet sampling for scalable ML classification of interactive traffic," in *IEEE Local Computer Networks, USA*, 2012.
[4] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 1-4, pp. 56–76, 2008.
[5] A. Mestres, A. Rodríguez-Natal et al., "Knowledge-Defined Networking," *Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.
[6] D. Clark et al., "A Knowledge Plane for the Internet," in *ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 3–10, 2003.
[7] C. Chang and N. Nikaein, "Closing in on 5G Control Apps: Enabling Multiservice Programmability in a Disaggregated Radio Access Network," *IEEE Vehicular Technology Magazine*, pp. 80–93, Dec 2018.
[8] X. Foukas, N. Nikaein et al., "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in *12th Intern. Conference on emerging Networking EXperiments and Technologies, CoNEXT, Irvine, California, USA, December 12-15*, pp. 427–441, 2016.

[5]For details, visit here: http://mosaic-5g.io/apidocs/flexran/#api-Stats

[9] R. Schmidt, C.-Y. Chang, and N. Nikaein, "FlexVRAN: A flexible controller for virtualized RAN over heterogeneous deployments," in *ICC 2019, 53rd IEEE International Conference on Communications, 20-24 May 2019, Shanghai, China*, 2019.

[10] B. Köksal, R. Schmidt, X. Vasilakos, and N. Nikaien, "CRAWDAD dataset eurecom/elasticmon5g2019 (v. 2019-08-29)." Downloaded from https://crawdad.org/eurecom/elasticmon5G2019/20190829, Aug. 2019.

[11] M. Brattstrom and P. Morreale, "Scalable Agentless Cloud Network Monitoring," in *4th IEEE Intern. Conference on Cyber Security and Cloud Computing, New York, NY, USA*, pp. 171–176, 2017.

[12] U. Thacker et al., "Performance of Elasticsearch in cloud environment with nGram and non-nGram indexing," in *IEEE Intern. Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016.

[13] J. Lessmann, M. Bredel et al., "D2.2 Architecture Design," 2015. H2020 SONATA project, Deliverable 2.2.

[14] N. Nikaein, M. Marina, S. Manickam, A. Dawson, R. Knopp and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM Sigcomm Computer Communication Review, Vol. 44, N5, October*, 2014.

[15] N. Nikaein and A. K. Chang, Chia-Yu, "Mosaic5G: Agile and flexible service platforms for 5G research," *ACM Sigcomm Computer Communication Review, Volume 48, N°3, July 2018*, 2018.

[16] N. Nikaein, X. Vasilakos, and A. Huang, "LL-MEC: Enabling Low Latency Edge Applications," in *IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, Japan, October 22-24*, 2018.