Monarch: Monitoring Architecture for 5G and Beyond Network Slices

Niloy Saha*, Nashid Shahriar†, Muhammad Sulaiman*, Noura Limam*, Raouf Boutaba* and Aladdin Saleh‡
{n6saha, m4sulaim, n2limam, rboutaba}@uwaterloo.ca, nashid.shahriar@uregina.ca, aladdin.saleh@rci.rogers.com
*University of Waterloo, Canada, †University of Regina, Canada, ‡Rogers Communications, Canada Inc.

***Abstract*—Data-driven algorithms play a pivotal role in the automated orchestration and management of network slices in 5G and beyond networks, however, their efficacy hinges on the timely and accurate monitoring of the network and its components. To support 5G slicing, monitoring must be comprehensive and encompass network slices end-to-end (E2E). Yet, several challenges arise with E2E network slice monitoring. Firstly, existing solutions are piecemeal and cannot correlate network-wide data from multiple sources (e.g., different network segments). Secondly, different slices can have different requirements regarding Key Performance Indicators (KPIs) and monitoring granularity, which necessitates dynamic adjustments in both KPI monitoring and data collection rates in real-time to minimize network resource overhead. To address these challenges, in this paper, we present *Monarch*, a scalable monitoring architecture for 5G. Monarch is designed for cloud-native 5G deployments and focuses on network slice monitoring and per-slice KPI computation. We validate the proposed architecture by implementing Monarch on a 5G network slice testbed, with up to 50 network slices. We exemplify Monarch's role in 5G network monitoring by showcasing two scenarios: monitoring KPIs at both slice and network function levels. Our evaluations demonstrate Monarch's scalability, with the architecture adeptly handling varying numbers of slices while maintaining consistent ingestion times between 2.25 to 2.75 ms. Furthermore, we showcase the effectiveness of Monarch's adaptive monitoring mechanism, exemplified by a simple heuristic, on a real-world 5G dataset. The adaptive monitoring mechanism significantly reduces the overhead of network slice monitoring by up to 76% while ensuring acceptable accuracy.**

***Index Terms*—5G, Network Slicing, KPI, Monitoring, Open5GS**

I. Introduction

Network slicing enables the creation of multiple isolated virtual networks for different services on a common physical infrastructure, but also increases network complexity and necessitates the use of Artificial Intelligence/Machine Learning (AI/ML) techniques for automated orchestration and management [1]–[3]. State-of-the-art algorithms for network slicing are data-driven and rely on large amounts of data collected from the network. Effective network monitoring is a fundamental component of these systems; it is the process of constantly monitoring the network and network components by collecting data about their state and behavior. To support 5G slicing, monitoring must be timely and accurate, and encompass network slices E2E.

In this context, a variety of monitoring solutions exist; unfortunately, they have several limitations. First, 5G slices span multiple technological domains and necessitate unified monitoring encompassing different network segments on a per-slice basis, as illustrated in Figure 1. To check if 5G services satisfy the desired Quality of Service (QoS) requirements, standardization bodies such as 3GPP have defined several slice KPIs such as throughput, latency, and reliability [4]. These KPIs are often composite, requiring the collection and correlation of multiple infrastructure and network function (NF) related metrics from different network segments. However, existing monitoring approaches predominantly concentrate on individual NFs or segments. For instance, cloud monitoring tools like OpenStack Ceilometer [5] and Nagios [6] facilitate tracking performance metrics such as CPU and memory utilization of NFs and the infrastructure on which they are hosted. For the Radio Access Network (RAN) domain, proprietary monitoring solutions exist from vendors such as Ericsson [7], and open-source solutions such as FlexRAN [8]. While efforts in literature, such as [9], [10], aim to enhance efficiency, they primarily center on individual NFs.

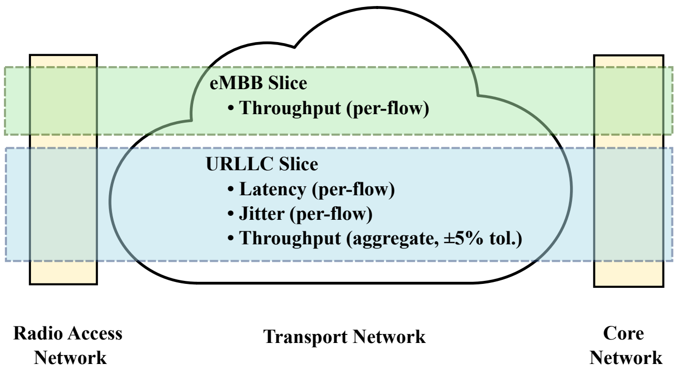


Fig. 1: Figure illustrating two distinct network slices: enhanced mobile broadband (eMBB) and ultra-reliable low-latency slice (uRLLC), each spanning diverse technological domains with varying KPIs.

Second, network operators are expected to provision several network slices on their 5G infrastructure, catering to the needs of different 5G services. The ability to support a large number of network slices is essential for 5G and beyond networks. Emerging applications like cloud gaming, AR/VR, holographic communication, and vehicle-to-everything (V2X) communication have diverse requirements that demand customized network slices to meet their specific needs. In scenarios like smart cities [11], and massive IoT deployments, the ability to scale up to hundreds or thousands of slices ensures that the network can effectively manage and isolate resources across a wide range of use cases. Fixed high-granularity monitoring across all slices, however, can lead to excessive use of network resources (i.e., overhead) and produce redundant (i.e., unnecessary) data. Therefore, network slice monitoring should also scale to the number and variety of slices and be cost-effective in minimizing monitoring overhead while guaranteeing monitoring accuracy. As the number of slices grows, the monitoring system must efficiently manage the increased data volume generated by each slice. The monitoring architecture must scale with the number of slices to check adherence to slice-specific SLAs and to enable precise per-slice resource allocation and admission control. Additionally, network slices may have different requirements in terms of resources and QoS. This in turn requires slice-specific KPI monitoring at different levels of granularity. The recent literature [12]–[15] on monitoring for 5G and beyond networks has focused on addressing the scalability aspect, but falls short in providing support for network slicing. Existing solutions also lack concrete implementation for computing slice KPIs and APIs for specifying monitoring requests at the network slice level. Some works including [12], [13] have no support for network slicing, while others such as [14], [15] mention slicing from an abstract architectural standpoint without providing details on how actual slice monitoring and computation of KPIs may be achieved.

Third, the next generation of 5G and beyond networks is progressively embracing cloud-native technologies (such as containers) to support multiple distributed NFs per network slice. This requires network management to leverage container orchestration tools such as OpenShift and Kubernetes. A related challenge for monitoring is that traditional monitoring tools cannot easily observe ephemeral cloud-native elements such as containers.

To address the aforementioned challenges of 5G network slice monitoring, a new comprehensive solution is needed, which meets the following requirements: (1) Scalable monitoring that adapts effectively to the growing number of network slices and seamlessly integrates into cloud-native 5G environments. (2) Ability to define monitoring requests at a high level of abstraction, ensuring ease of use in network automation. (3) Mechanisms to effectively balance the trade-off between monitoring accuracy and overhead, optimizing resource utilization and network performance.

To fulfill these requirements, this paper presents Monarch, a comprehensive and scalable monitoring architecture for 5G and beyond networks. Monarch focuses on monitoring network slices, computing slice KPIs, providing an API for specifying monitoring requests, and effectively managing the trade-off between monitoring accuracy and overhead. In this context, we make the following contributions:

- We present Monarch, a scalable monitoring architecture for cloud-native 5G network deployments, for providing a holistic view of E2E network slices (§III-A).
- We design a northbound API that allows for specifying monitoring requests at various levels e.g., slice level and network function level (§III-B), as well as an adaptive monitoring heuristic designed to strike a balance between monitoring accuracy and overhead (§III-D).
- We implement a prototype of Monarch and integrate it with a 5G network slice testbed comprising of popular open-source projects UERANSIM and Open5GS (§IV). We exemplify Monarch's role in 5G network monitoring by showcasing two scenarios: monitoring KPIs at both the slice and network function levels (§IV). To foster collaboration and further development, we make the implementation publicly available [16].
- We evaluate the Monarch prototype (§V) and demonstrate that Monarch can scale well with the number of slices — *i.e.*, monitor up to 50 network slices, without a significant increase in overhead and resource usage. Based on real 5G traffic traces, we show that Monarch can reduce overhead by up to 76% while maintaining acceptable accuracy using a simple adaptive monitoring heuristic.

This paper builds upon our prior work on network slice monitoring [17], significantly expanding its scope. First, we enhance the related works section by incorporating relevant studies on monitoring data extraction, KPI computation, AI/ML solutions, and the 3GPP network data analytics function (NWDAF)-based solutions within the context of 5G network slice monitoring. Second, we refine the monitoring API to align it more closely with 3GPP specifications for defining 5G KPIs. Additionally, we provide detailed examples illustrating the API's usage for specifying requests at both the network slice and network function levels. Third, we introduce an adaptive monitoring mechanism into Monarch based on a simple heuristic and evaluate its performance using real-world 5G traffic traces, presenting insights into the accuracy-overhead trade-off of network slice monitoring. Fourth, we significantly extend our experimental results by deploying and evaluating Monarch with up to 50 network slices. We provide a detailed breakdown of Monarch's resource usage, highlighting the relative contributions of key components. Further, we conduct additional experiments to analyze the factors influencing monitoring data ingestion time, which helps guide best practices for optimizing data ingestion and minimizing latency. Lastly, we release the implementation code demonstrating Monarch's integration with Open5GS [18], a widely-used 3GPP R16 compliant 5G core implementation.

II. Literature Survey

In recent literature, there have been several works that have focused on 5G network monitoring. We coarsely characterize them into the following categories:

Monitoring architectures. Several works [12]–[15] focus on scalable monitoring architectures for 5G networks. Perez *et al.* [12] present a monitoring architecture for a multi-site 5G platform. The proposed architecture consists of a two-level hierarchy of intra-site and inter-site publish-subscribe brokers, with infrastructure-specific agents for extracting and translating metrics from heterogeneous infrastructure components. While the authors address scalability, they do not discuss E2E monitoring or network slicing. Beltrami *et al.* [14] propose an architecture for monitoring E2E network slices across multiple heterogeneous domains. Their framework facilitates horizontal

TABLE I: Comparison of Monarch with state-of-the-art. Legends in the table represent supported (✓), not supported (✗), partial/limited support and/or implementation details missing (◆); "PM" refers to performance metrics.

Features	**Perez *et al.* [12]**	**Beltrami *et al.* [14]**	**Giannopoulos *et al.* [13]**	**Mekki *et al.* [15]**	**Monarch**
Scalability (§V-B)	✓	✓	✗	✓	✓
Monitoring request API (§III-B)	✗	✗	✗	◆	✓
Adaptive monitoring mechanisms (§III-D)	✗	✗	✗	✗	✓
5G PM and network slice KPIs (§IV)	✗	◆	✗	◆	✓
5G network integration (§IV)	✗	✗	✓	✓	✓
Code availability (§IV)	✓	✗	✗	✗	✓

and vertical elasticity within the monitoring infrastructure by dynamically instantiating and removing monitoring agents in the NFs related to a slice. The authors propose several components such as slice measurements aggregator and engine controller for tuning the configuration of the monitoring components (e.g., monitoring frequency), however, they do not provide any quantitative analysis or results related to the resource consumption of the proposed architecture.

In a similar vein, Giannopoulos *et al.* [13] present a monitoring framework for 5G systems, focusing on monitoring different levels of the 5G system. They demonstrate how open-source monitoring tools such as Netdata and Prometheus can collect system and network-level metrics by deploying their proposed framework on a testbed consisting of a physical gNB and the open-source Open5GS 5G standalone (SA) core. However, they do not discuss how to capture per-slice KPIs. Finally, Mekki *et al.* [15] present a scalable monitoring architecture for network slices, including collecting metrics from different network slice segments. This is achieved by deploying domain (e.g., RAN and edge/core) and slice-specific collection agents, instantiated per slice, which collect and aggregate monitoring data per slice. Slice identification is achieved using a custom protocol that encapsulates monitoring data with a header containing slice identifiers. While this architecture introduces several useful abstractions, the paper falls short in describing how slice-level aggregations are done, and how domain orchestrators compute KPIs from monitoring data of NFs in a slice.

AI/ML solutions. These works [9], [19], [20] leverage artificial intelligence (AI) and machine learning (ML) approaches to aid in reducing the communication overhead incurred in 5G network monitoring. In this context, Xie *et al.* [19] advocate for integrating joint monitoring and analytics in 5G networks to balance monitoring costs and service assurance. They propose partial monitoring to reduce monitoring overhead, achieved by a combination of feature selection, flow selection, and optimal probe placement. Their simulations show that analytics can inform monitoring frequency decisions, but use simplistic models and arbitrary probability values without concrete implementation. Sciancalepore *et al.* [9] and Plascinskas *et al.* [20] present similar ideas with more concrete implementation details. Sciancalepore *et al.* [9] improve a management and orchestration (MANO) system's decisions while minimizing the network monitoring load. The authors use a two-stage ML-based approach consisting of an unsupervised learning algorithm to cluster similar NFs into NF profiles, and a reinforcement learning algorithm to find a trade-off between monitoring load and good MANO decisions. Plascinskas *et al.* [20] extend this idea by utilizing the time-varying characteristics of different NFs to adjust the monitoring frequency of individual metrics in an NF. However, these works mainly focus on individual NFs and do not address a network slice as a whole.

Monitoring data extraction. These works [21], [22] focus on extracting monitoring data from different components of the 5G network. Both Janus [21] and 5GC-Observer [22] represent innovative approaches to monitoring complex aspects of 5G networks, leveraging eBPF technology to extract crucial monitoring data. Janus focuses on the RAN, enabling inline execution of custom codelets for real-time monitoring and control. By utilizing eBPF, Janus allows direct access to raw RAN data structures, facilitating the collection of diverse statistics and enabling real-time inference and control decisions. On the other hand, 5GC-Observer targets the 5G Core (5GC) system, providing insight into the exchanges between network functions, including telco-specific protocols. By harnessing eBPF, 5GC-Observer can monitor container-based 5GC network functions orchestrated by Kubernetes, collecting telemetry data in the form of measurements and logs without modifying network function code. These works are complementary to Monarch, and can be used to augment the monitoring data exporters (§III-A) used by Monarch.

KPI computation. Works such as [23] and [24] focus on KPI computation in 5G networks. Vasilakos *et al.* [23] proposed ElasticSDK, a software development kit (SDK) that provides a range of abstractions tailored specifically for monitoring tasks. The authors utilized a custom agent deployed on top of the FlexRAN controller to collect monitoring data and show that ElasticSDK can seamlessly establish a monitoring pipeline, where control plane applications access the database, compute processed values, and seamlessly write back the results. The paper discusses the need for a northbound API for monitoring requests but lacks details on implementing filtering and aggregation criteria, particularly concerning network slices spanning various network segments. Vietch *et al.* [24] advocate for the simultaneous consideration of E2E KPIs and individual performance metrics from the Network Function Virtualization (NFV) infrastructure to ensure the performance of 5G network slices. They substantiate their argument by examining two network slices - one critical and one non-critical - both residing on the same physical x86 infrastructure and sharing the same last-level cache (LLC). The authors illustrate how resource contention caused by the non-critical slice significantly impacts the performance of the critical slice, despite their separate virtual CPUs. Their findings emphasize that solely relying on either E2E KPIs or NFV infrastructure metrics is insufficient for gaining insights into such issues; rather, both aspects must be jointly considered to derive meaningful insights.

NWDAF based solutions. Several recent works have focused on NWDAF solutions within the 5G core network, addressing the need for advanced analytics capabilities. Garcia *et al.* [25] highlights NWDAF's crucial function in 5G network automation, focusing on its role in data collection and analytics exposure. While NWDAF utilizes standardized methods like the event exposure API for gathering data from 5G core NFs, it relies on operations, administration and management (OAM) systems for E2E KPIs. Mekrache *et al.* [26] present a microservices-based implementation of NWDAF integrated with the OpenAirInterface (OAI) 5G core and RAN, demonstrating its analytic capabilities through the detection of abnormal User Equipment (UE) traffic patterns using a Long Short-Term Memory (LSTM)-based autoencoder. Moreover, Manias *et al.* [27] propose a distributed NWDAF architecture with multiple instances placed at different network regions, highlighting the concept of central-edge aggregation for monitoring slice-level performance metrics and developing future forecasts. While existing works highlight various aspects of NWDAF analytic capabilities within the 5G core network, Monarch takes a distinct approach by focusing on the critical aspect of data collection and complementing the NWDAF's analytics functions.

Synthesis. The existing literature on 5G network monitoring demonstrates several limitations related to E2E network slice monitoring. While several studies have proposed monitoring architectures (as summarized in Table I) and AI/ML solutions, they often lack comprehensive support for network slicing and fail to provide detailed implementations using real-world 5G testbeds. Some works, such as [14], [15], [19], address slicing to some extent but lack specifics on slice KPI computation. Other approaches, such as [9], [20], may not directly apply to network slicing as they focus on individual NFs. The absence of a comprehensive solution for E2E network slice monitoring results in the following limitations:

- *Lack of 5G network slice monitoring data*: Most of the existing studies rely on either synthetic data or 4G data generated by emulating the 4G Evolved Packet Core (EPC). Alternatively, studies involving 5G networks lack publicly available implementations of a monitoring framework capable of easily collecting 5G network slice monitoring data. This scarcity of genuine 5G monitoring data poses a significant challenge in quantitatively evaluating any data-driven network automation solutions related to network slicing and 5G network slice KPIs.
- *Lack of quantitative analysis of monitoring cost*: Within the current literature, there is a lack of quantitative analysis of the trade-off between monitoring data volume and accuracy, particularly in the context of E2E network slice KPIs. Assessing the optimal balance between the required volume of monitoring data and the associated cost remains a critical aspect yet to be thoroughly explored.

This paper addresses this gap by introducing Monarch, a scalable monitoring architecture designed for 5G and future networks, with a particular emphasis on network slice-level monitoring and KPI computation. We have implemented and assessed this architecture using a cloud-native 5G testbed, providing an empirical examination of its effectiveness in computing 5G network slice KPIs. Moreover, we propose an adaptive monitoring heuristic aimed at minimizing monitoring overhead while ensuring accuracy, thereby enhancing cost-effectiveness. This algorithm is evaluated using a real-world 5G traffic dataset [28].

III. Monarch Design and Operation

A. Monarch Overview

Monarch is a monitoring architecture for 5G networks, focusing on E2E network slice monitoring. Figure 2 shows the architecture overview of Monarch, comprised of several high-level components, which are described below:

Request Translator. This component implements an abstraction layer that translates high-level monitoring requests specified by the monitoring API (§III-B) into low-level monitoring directives understood by the Monitoring Manager (*e.g.*, which NF instance to monitor). It communicates with an external slice orchestrator (e.g., ONAP [29]) to obtain the mapping between NF instances and slices.

Monitoring Manager. This component is responsible for converting the monitoring directives into monitoring configurations for each network segment, which can be understood by the network slice segment data collector (NSSDC). The monitoring manager also implements intelligent algorithms for monitoring, such as adaptive polling and KPI prediction.

KPI Computation. These components are instantiated per monitoring request and contain the logic for network slice KPI computation. They are responsible for querying the data store, computing KPIs (e.g., by correlating data from different NFs) and subsequently writing the computed KPIs back to the data store. They contain the logic necessary for computing network slice KPIs (*e.g.*, those standardized by the 3GPP [4]) and are extendable via scripts to support custom KPIs. In this paper, we present a proof-of-concept implementation of network slice throughput KPI (§IV).

NSS Data Collector (NSSDC). This component is instantiated per network slice segment (NSS) and interacts with the NSS management function (NSSMF) (*e.g.*, service management and orchestration (SMO) in the RAN and network function virtualization orchestration (NFVO) in the core) to instantiate monitoring data exporters (MDEs) specific to that network slice segment. For an E2E slice comprising various network slice segments like RAN, edge, and core, each NSSDC instantiates MDEs tailored to the specific configuration needs of its segment. For instance, in the RAN, MDEs may connect to xApps, while in the core, they might collect data directly from 5G core NFs. The NSSDC configuration includes details, such as monitoring frequency and targeted instances of gNBs or NFs, specifying IP addresses or endpoints for communication.

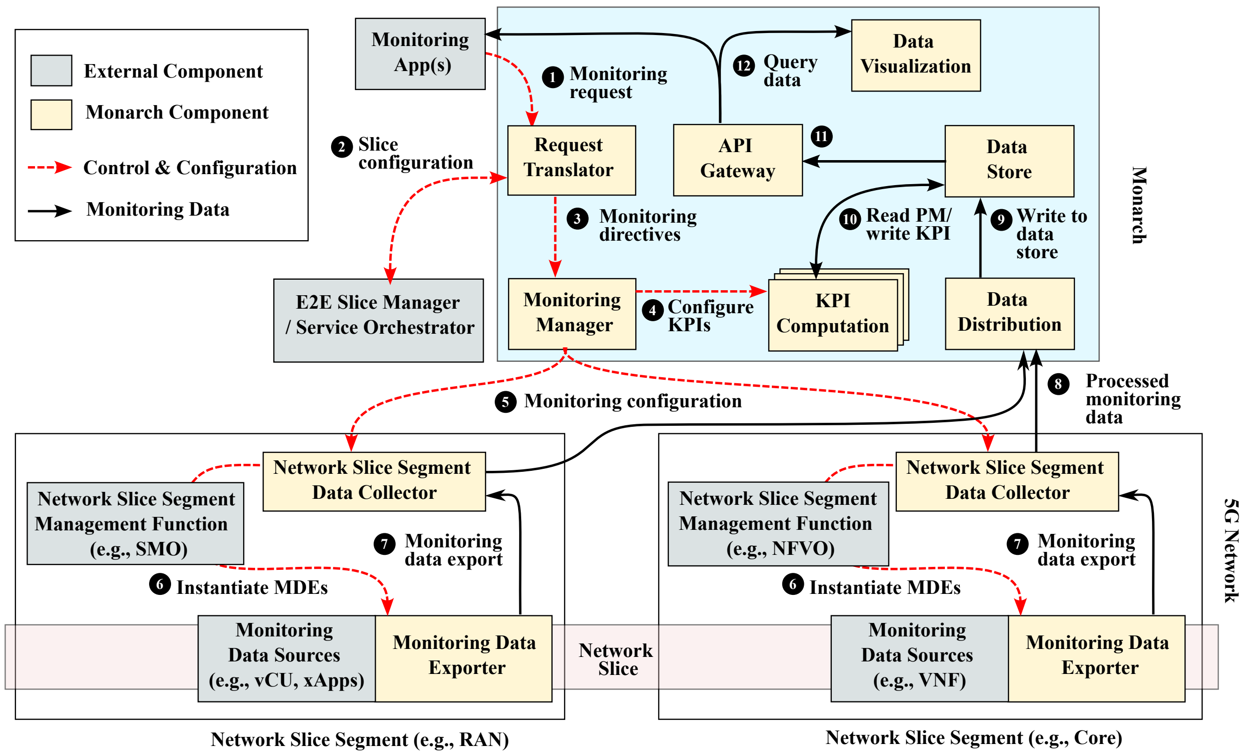


Fig. 2: Conceptual architecture for Monarch

The NSSDC also provides functions, such as short-term local data persistence, filtering, aggregation, and data transformation (*e.g.*, enriching the data by adding labels).

Monitoring Data Exporter (MDE). MDEs are lightweight containers responsible for extracting raw monitoring data from monitoring data sources (*e.g.*, NF, xApps, *etc.*), converting the data into a standardized format (*e.g.*, OpenTelemetry protocol[1]), and then exporting them to the NSSDC. MDEs can extract monitoring data either directly (*e.g.*, by interacting with the NF instance) or indirectly (*e.g.*, by querying monitoring xApps in the RAN).

Data Distribution. The data distribution component is responsible for collecting processed monitoring data from different NSSDCs. It implements horizontal scalability and load-balancing to handle large volumes of data.

Data Store. The data store component in Monarch is an abstraction of long-term persistent storage and is responsible for storing monitoring data as well as configuration data (e.g., templates for KPI computation modules).

API Gateway. The API Gateway acts as an intermediary between the monitoring request and the subsequent access to monitoring data by facilitating client access to monitoring data through request IDs. It is responsible for mapping monitoring request IDs to metrics, ensuring that users can be oblivious to how metrics are represented and collected in the underlying system. Authentication and access control mechanisms within the API Gateway enable multi-tenancy by only allowing users to retrieve data associated with their specific monitoring requests.

[1]https://opentelemetry.io/

Data Visualization. This component presents the monitoring data in a visual format using interactive dashboards. This allows users to glean insights by drilling down into specific data of interest, such as investigating the throughput of a particular session within a defined slice.

B. Monitoring API

A flexible northbound API for monitoring, which provides a high-level abstraction to monitoring applications, is essential for ensuring that the applications can focus on their business logic. Some existing works, such as [15], propose to leverage the 3GPP network slice template (NST) for this purpose. However, we argue that a separate northbound API for monitoring offers a few advantages:

- **Granular Control**. The dedicated monitoring API enables precise control over monitoring parameters such as KPIs and detailed sub-counters. It also allows for the modification and deletion of monitoring requests independent of the network slice configuration.
- **Separation of Concerns**. The monitoring API separates monitoring functions from network slice management and allows network administrators to concentrate on monitoring tasks without the complexities of slice creation and configuration.
- **Increased Flexiblity**. By not being bound to the NST, the dedicated monitoring API permits various monitoring applications to submit distinct monitoring requests for the same network slice, each with different parameters, thereby enhancing operational flexibility.

The proposed monitoring API provides a high-level abstraction to monitoring applications. The request structure is shown in Fig. 3. To submit a new monitoring request, clients (i.e., monitoring apps) initiate a POST request to the /api/monitoring-requests endpoint, providing a JSON payload describing the monitoring task. Details of a specific monitoring request can be fetched through a GET request to /api/monitoring-requests/request_id, where request_id uniquely identifies the monitoring request. Once the monitoring request has been submitted, the client can access the monitoring data by sending a GET request to GET/api/metrics?request_id=⟨request_id⟩. Fig. 3 shows how the monitoring API can be used to submit requests for both slice-level and NF-level KPIs.

C. Flow of a monitoring request

Here we discuss how the different components of Monarch work together to serve a network slice monitoring request. Figure 2 shows the flow of a monitoring request (Steps 1–12) through Monarch, which proceeds as follows:

- **Step 1**. Monitoring apps (*e.g.*, AI/ML-based slice management and orchestration applications) specify a high-level network slice monitoring request through the monitoring request API (*cf.*, §III-B) and receive a request_id.
- **Step 2**. The Request Translator communicates with an external slice orchestrator to gather information about slice configuration. This includes information such as network slice instances and their associated virtual NFs and virtual links.
- **Step 3**. The Request Translator parses the monitoring request and transforms it into monitoring directives understood by the monitoring manager (*e.g.*, which NF instances to monitor). For example, a monitoring request for network slice throughput KPI would be converted to monitoring directives for the associated session management function (SMF) and user plane function (UPF) instance(s) associated with that slice.
- **Step 4**. The Monitoring Manager instantiates the slice KPI computation component for the submitted monitoring request, using predefined logic for computing the specified KPI.
- **Step 5**. The monitoring manager translates the monitoring directives into monitoring configuration understood by the NSSDCs.
- **Step 6**: The corresponding NSSDC instantiates MDEs using the interfaces provided by the NSSMF, configuring them with the appropriate parameters for the monitoring data source, such as connection information and monitoring frequency.
- **Step 7**: The MDEs collect monitoring data at the specified frequency, transform them into a uniform format, and subsequently export the data to the NSSDC.
- **Step 8**: The NSSDCs perform their respective operations such as filtering, aggregation, and data transformation and subsequently relay the processed data to the Data Distribution component.
- **Step 9**: The data is written to the data store. This may involve adding monitoring data to a time-series database, as well as archiving it in long-term storage solutions, such as object storage.
- **Step 10**: The KPI computation component (instantiated in Step 4) reads the monitoring data from the data store, performs the KPI computation, and subsequently saves the computed KPIs in the data store.
- **Step 11 and Step 12**: The client queries the monitoring data through the API Gateway, using the request IDs generated in Step 1.

D. Monitoring algorithms

Fixed high-granularity monitoring across all slices can lead to excessive use of network resources (i.e., overhead) and produce redundant (i.e., unnecessary) data. Therefore, network slice monitoring should be cost-effective in minimizing monitoring overhead while maximizing monitoring accuracy. To address this challenge, Monarch supports adaptive monitoring algorithms, that can dynamically adjust monitoring parameters to optimize resource usage and enhance the efficiency of network slice monitoring processes. Monarch facilitates adaptive monitoring by adopting the *sidecar pattern* [30] for MDEs, allowing dynamic configuration of monitoring intervals per NF. The Monitoring manager can choose from a catalog of different adaptive monitoring algorithms. For example, for a critical network slice, an algorithm focused on optimizing accuracy may be chosen, while for non-critical slices, an algorithm prioritizing network overhead could be beneficial.

The placement of adaptive monitoring algorithms within the Monarch architecture depends on several factors, including the nature of the algorithms, the data sources they require, and the specific monitoring requirements of the network slices. Placing adaptive monitoring algorithms within the Monitoring manager allows for centralized control and coordination of adaptive monitoring strategies across the entire network. Alternatively, adaptive monitoring algorithms can be deployed within NSSDCs or MDEs, closer to the data sources. This approach leverages the proximity to raw monitoring data, enabling the algorithms to perform real-time analysis and generate adaptive monitoring decisions based on local observations. To demonstrate the adaptive monitoring capabilities of Monarch, we propose a heuristic-based adaptive monitoring algorithm (§IV, Algorithm 1). It is worth noting that while Algorithm 1 represents a simple heuristic, Monarch is not limited to this particular approach. Monarch is designed to accommodate a variety of monitoring algorithms, including more advanced ones based on AI/ML techniques.

IV. Proof-of-concept Implementation and Use-cases

Hardware. Our testbed setup, depicted in Figure 4, comprises two distinct 2-node Kubernetes clusters: one designated for the 5G network implementation and the other for Monarch deployment. The first cluster, dedicated to hosting the 5G network components, consists of two Intel NUC PCs equipped with Intel i7-6770HQ processors (4 cores) clocked at 2.600 GHz, each paired with 16GB of RAM. These nodes run Linux v5.15 and accommodate the NSSDC component of Monarch. Conversely, all remaining Monarch components are deployed on the second Kubernetes cluster, featuring Intel Xeon servers (E3-1230 v3) with 8 cores running at 3.7GHz, with 16GB

```
{
    "api_version": "1.0",
    "request_description": "Monitoring
    ↪  request for slice throughput",
    "scope": {
        "scope_type": "slice",
        "scope_id": "NSI01"
    },
    "kpi": {
        "kpi_name": "slice_throughput",
        "kpi_description": "Throughput of
        ↪  the network slice",
        "sub_counter": {
            "sub_counter_type": "SNSSAI",
            "sub_counter_ids": ["1-000001",
            ↪  "2-000002"]
        },
        "units": "Mbps"
    },
    "duration": {
        "start_time":
        ↪  "2023-12-01T00:00:00Z",
        "end_time": "2023-12-01T00:05:00Z"
    },
    "monitoring_interval": {
        "adaptive": true,
        "interval_seconds": 1
    }
}
```

(a) Monitoring request for slice-level KPI

```
{
    "api_version": "1.0",
    "request_description": "Monitoring
    ↪  request for the number of QoS
    ↪  flows",
    "scope": {
        "scope_type": "nf",
        "scope_id": "SMF01"
    },
    "kpi": {
        "kpi_name": "smf.num_qos_flows",
        "kpi_description": "Number of QoS
        ↪  flows at the SMF",
        "sub_counter": {
            "sub_counter_type": "5QI",
            "sub_counter_ids": ["1", "6"]
        },
        "units": "count"
    },
    "duration": {
        "start_time":
        ↪  "2023-12-01T00:00:00Z",
        "end_time": "2023-12-01T00:30:00Z"
    },
    "monitoring_interval": {
        "adaptive": false,
        "interval_seconds": 5
    }
}
```

(b) Monitoring request for NF-level KPI

Fig. 3: Structure of a monitoring request

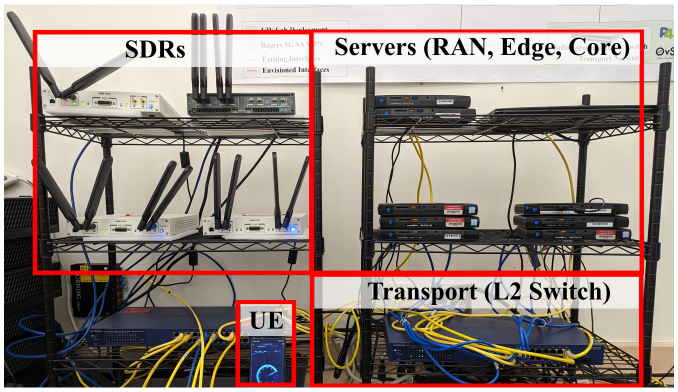


Fig. 4: Testbed infrastructure showing (from bottom left, clockwise): a) User Equipments (UEs) b) Software Define Radios (SDRs) connected to gNB, c) servers hosting 5G NFs, and d) transport network interconnecting different components.

of RAM. For the gNB implementation, we use srsRAN [31] deployed on an Intel Core i9-10980XE 3.0GHz PC running Ubuntu OS 22.04 with a low latency kernel. The gNB is directly connected to a Software Defined Radio (SDR) unit (USRP X310), and we utilize a Google Pixel 7 Pro as the User Equipment (UE) device.

5G network. Our 5G mobile core is based on Open5GS [18], an open-source implementation of 3GPP R16. The 5G NFs, *e.g.*, SMF and UPF are containerized and deployed on a Kubernetes cluster. We employ a software-defined VXLAN overlay utilizing OpenVSwitch (OvS) on the underlying physical substrate network to establish the transport network infrastructure. The integration of the 5G NFs with this transport overlay was accomplished via the OvS CNI plugin. The RAN segment is implemented using srsRAN [31], with the Google Pixel 7 Pro serving as our UE. To accommodate experiments necessitating multiple slices (more than two), we utilize UERANSIM [32], an open-source gNB and UE simulator, due to hardware limitations. Our experiments include two scenarios: i) using the UERANSIM simulator to demonstrate the efficiency and scalability of Monarch ii) utilizing real UEs and SDRs to replay real-world traffic traces, ensuring realistic results for adaptive monitoring evaluations. We have made the source code for deploying the network slicing testbed publicly accessible via our GitHub repository `open5gs-k8s` [33]. This repository includes scripts for setting up the Kubernetes cluster with requisite plugins, configuring multiple network slices, and custom versions of the Open5GS SMF and UPF NFs, offering per-slice metrics.

Monarch implementation. All Monarch components are instantiated as containers on a Kubernetes cluster, with the MDEs instantiated as sidecar containers inside Kubernetes pods. Figure 5 presents a logical diagram illustrating the interaction between the 5G testbed and Monarch, with a focus on monitoring data collection and control across various network segments. The diagram highlights two distinct network slices

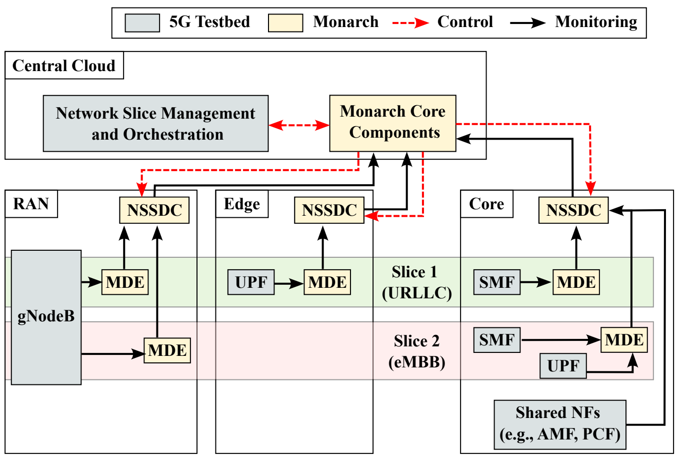


Fig. 5: Logical diagram showing the integration of Monarch with the 5G testbed, highlighting the flow of monitoring data and control messages between components

(Slice 1 and Slice 2) that traverse different segments, including the RAN, Edge, and Core. It demonstrates how Monarch components, such as Monitoring Data Exporters (MDEs), interface with key 5G network elements like gNodeB, UPF, and SMF within each slice. Table II presents an overview of how different components of Monarch are implemented. The container images, manifest files, and source code used for the implementation are publicly available on our GitHub repository `5g-monarch` [16]. The adaptive monitoring algorithm is incorporated into the MDE.

TABLE II: Implementation of Monarch components

Component	**Implementation**
Monitoring manager	Kubernetes API (v1.28.1)
KPI calculator	Python-based implementation
NSSDC	Prometheus (v2.47.0)
MDE	Exporters based on Prometheus and Open-Telemetry SDK
Data distribution	Thanos (v0.31.0)
Data store	Prometheus time-series database and Minio S3 object storage
API gateway	Python-based implementation leveraging Thanos Query API
Data visualization	Grafana

Use-cases. We illustrate Monarch's role in 5G network monitoring by showcasing two scenarios: monitoring KPIs at the slice and network function levels (for evaluation, see §V).

- *Monitoring network slice KPI:* We use Monarch to monitor the throughput of a network slice instance (*cf.*, [4], Section 6.3.2, 6.3.3). The throughput of a network slice instance is determined by computing the packet size for each successfully transmitted IP packet through the network slice instance during each observation granularity period. This demonstrates the ability of Monarch to monitor KPIs at the slice level. Figure 3a shows the associated monitoring request, where we define the `scope` as slice, and establish per-slice aggregation by specifying the `sub_counter` of the `kpi` as `SNSSAI`. To compute the throughput of the network slice instance, Monarch aggregates the transmitted and received bytes for every PDU session associated with each UPF instance in the slice. This requires correlating information from both the SMF and UPF. To achieve this, Monarch instantiates two MDEs alongside the SMF and UPF, using the Kubernetes API. The MDE for SMF collects information about the mapping between slices and their active PDU sessions, while the MDE for UPF exports information about the bytes transmitted and received for each PDU session, properly labeled with information such as the originating UPF instance, direction (uplink/downlink), and network segment. The slice-specific KPI computation module then queries the data store to collect information about the slice and the number of bytes received and transmitted per active PDU session. It filters the list of PDUs for a given slice using S-NSSAI to PDU session mapping and aggregates the received and transmitted bytes for these filtered PDU sessions to calculate slice throughput. Monarch's ability to label metrics by direction and per PDU session provides flexibility for monitoring applications.
- *Monitoring network function KPI:* We use Monarch to monitor the number of QoS flows from the SMF (*cf.*, [34], Section 5.3.2). In 5G, a QoS flow represents the most granular level of QoS distinction within a session. This underscores Monarch's capability to monitor KPIs at the NF level. Figure 3 shows the associated monitoring request, where we define the `scope` as NF, and establish per-qos-flow aggregation by specifying the `sub_counter` of the `kpi` as `5QI`. This KPI is collected from the measurements exposed by the SMF. Note that we can specify a particular instance(s) of SMF by specifying one or more items in the `scope_id` field.

Adaptive monitoring. To exemplify the adaptive monitoring capabilities of Monarch, we propose a heuristic-based adaptive monitoring algorithm (Algorithm 1). We implement our adaptive monitoring heuristic by integrating it with our MDEs. The MDEs are implemented as Prometheus exporters responsible for pushing data to Prometheus remote write endpoints. This integration allows us to embed the monitoring algorithm directly within the MDEs, leveraging their proximity to the data source. We opt for this approach because the pull model of Prometheus does not support real-time on-the-fly adjustment of monitoring granularity. Integrating the heuristic within the MDEs enables the dynamic configuration of adaptive monitoring intervals, ensuring efficient resource usage and timely response to network dynamics.

Algorithm 1 operates on an input time series $x(t)$ (representing a monitored KPI (§IV)) and adapts its monitoring frequency to capture significant changes while minimizing resource usage. The algorithm maintains a sampling interval T, which determines how often data points are sampled from the time series. At each time step t, the algorithm evaluates whether it is appropriate to sample the data based on the current sampling interval and the observed changes in the time series. At each time step, the algorithm decides whether to sample the data based on two criteria — a) *time to sample*: it

Algorithm 1 Adaptive Monitoring Algorithm

```
Inputs: $x(t)$: original time series, $\delta$: threshold for significant change, $T$: sampling interval, $T_{min}$: minimum sampling interval, $T_{max}$: maximum sampling interval, $\alpha$: sampling interval increase factor, $\beta$: sampling interval decrease factor.
Output: : $x_s(t)$ : Sampled time series
 1: Initialize:
 2:     $T \leftarrow T_{min}$
 3: function SAMPLE($\tau_t, x(t)$)
 4:     if $t = 1$ then
 5:         $x_s(t) \leftarrow x(t)$                      ▷ Sample the time series
 6:     else
 7:         $T \leftarrow$ UPDATE_SAMPLING_INTERVAL($x(t)$)
 8:         if TIME_TO_SAMPLE($\tau_t, T$) then
 9:             $x_s(t) \leftarrow x(t)$                  ▷ Sample the time series
10: function TIME_TO_SAMPLE($\tau_t, T$)
11:     $\Delta t = \tau_t - \tau_{t-1}$
12:     if $\Delta t \geq T$ then
13:         return True
14:     return False
15: function SIGNIFICANT_CHANGE($x(t)$)
16:     $\Delta x = |x(t) - x_s(t-1)|$
17:     if $\Delta x > \delta$ then
18:         return True
19:     return False
20: function UPDATE_SAMPLING_INTERVAL($x(t)$)
21:     if SIGNIFICANT_CHANGE($x(t)$) then
22:         $T \leftarrow \max(T_{min}, T/\beta)$
23:     else
24:         $T \leftarrow \min(T_{max}, T * \alpha)$
25:     return $T$
26: for each timestep $\tau_t = 1, 2, \cdots$ do
27:     SAMPLE($\tau_t, x_t$)
```

checks if the time elapsed since the last sample exceeds the current sampling interval T (lines 10-14), and b) *significant change*: it examines whether the observed change in the time series exceeds a predefined threshold δ (lines 15-19). The algorithm adjusts the sampling interval depending on whether a significant change is detected. If a significant change is detected, the sampling interval is reduced by dividing it by a factor β to capture detailed variations. Conversely, if no significant change is observed, the sampling interval is increased by multiplying it by a factor α to reduce resource consumption.

V. Performance Evaluation

In this section, we present the evaluation results for Monarch, highlighting its performance across various aspects. Specifically, our findings illustrate Monarch's ability to efficiently scale its resource usage with the increasing number of slices (§V-B), its negligible impact on latency even with an increasing number of slices (§V-B), and the effectiveness of its adaptive monitoring mechanism in striking a balance between monitoring accuracy and overhead (§V-C).

A. Experimental Setup

Network slicing scenario. We create multiple network slices, ranging from 1 to 50, with each slice having a separate SMF and UPF, with all other 5G NFs common across all slices. We chose to configure network slices with individual SMF and UPF instances to customize resource allocation and monitoring for each slice while sharing other 5G NFs for resource efficiency and operational ease. Each slice accommodates one connected UE generating random traffic, except during adaptive monitoring experiments where traffic is replayed from the real-world 5G dataset. Leveraging Monarch, we gather the network slice monitoring data to compute network slice KPIs, as outlined in §IV. Each scenario is repeated 10 times to ensure robustness, and the average results are presented for analysis.

Dataset. To simulate real-world conditions, we utilize traffic traces sourced from a 5G dataset [28] obtained from a prominent mobile operator in South Korea. These traces encompass diverse 5G applications, such as cloud gaming and live streaming. By leveraging our 5G network testbed, which includes Pixel UEs and SDRs, we replay these traffic traces and employ Monarch to monitor the network slice KPIs. The efficacy of the adaptive monitoring algorithm is thoroughly evaluated against these realistic traffic patterns and KPIs.

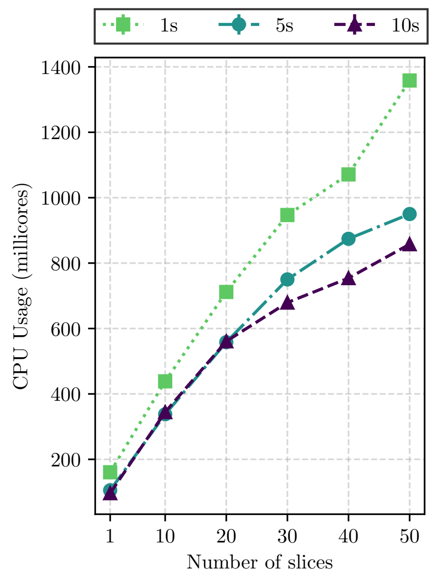
B. System Performance

Performance Metrics. We use the following performance metrics to evaluate the system performance of Monarch:

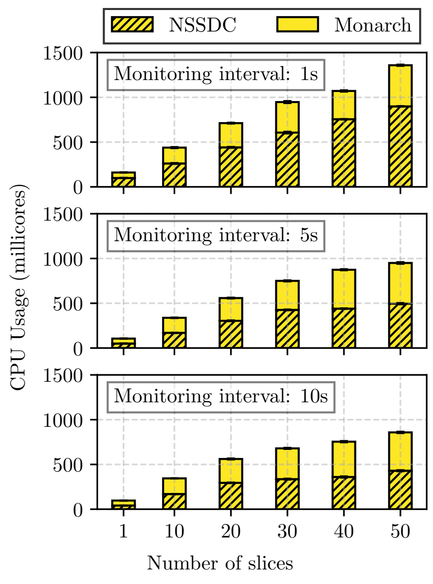
- **Resource usage**: The CPU and memory usage of different components of Monarch. We leverage CPU metrics exposed by cAdvisor integrated with Kubernetes to calculate CPU usage, given in millicores.
- **Monitoring overhead**: The amount of Monarch-generated network traffic (bytes/sec) to accomplish its monitoring tasks.
- **Ingestion time**: The time (in ms) to collect monitoring data from the NFs, given by Prometheus `scrape_duration` metric.

To assess the performance of Monarch, we conducted evaluations to measure the impact of the number of slices and the monitoring interval on the Monarch's resource usage. Specifically, we varied the number of slices (from 1 to 50) and monitoring interval (from $1s$ to $10s$) and used a separate instance of Prometheus serving as a meta-monitor to observe the resource usage patterns exhibited by various components of Monarch.

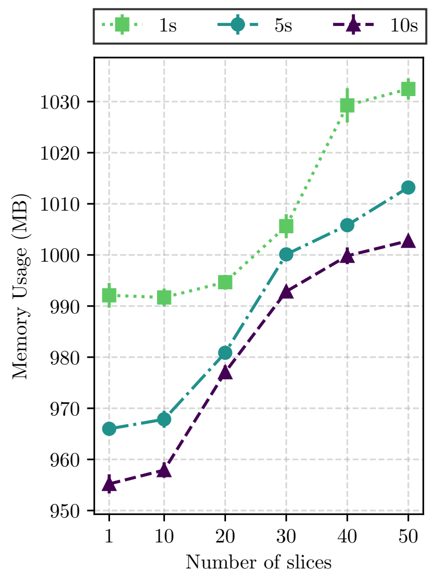
CPU usage. Figure 6a shows the CPU usage of Monarch with the number of slices in the network varying from 1 to 50, and monitoring intervals set at $1s$, $5s$, and $10s$. The CPU usage of Monarch is directly correlated with the volume of monitoring data collected, which increases linearly with the number of slices. Consequently, given factors, such as monitoring interval and metric cardinality, remain constant, the CPU usage of Monarch grows linearly as the number of slices increases. Specifically, when scaling the number of slices $50\times$ from 1 to 50, we observe a $8.5\times$, $9\times$, $8.9\times$ increase in CPU usage, for monitoring intervals of $1s, 5s$ and $10s$, respectively.



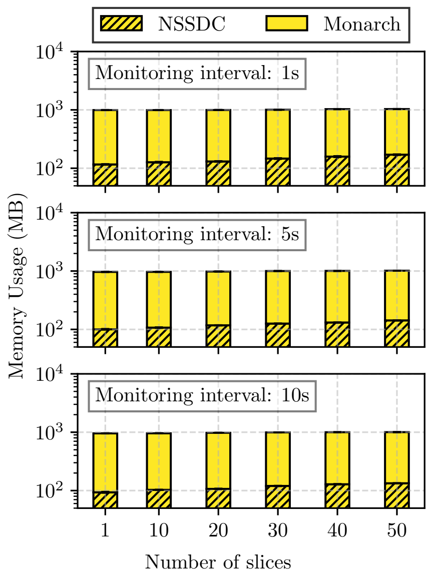
(a) CPU usage vs. slices



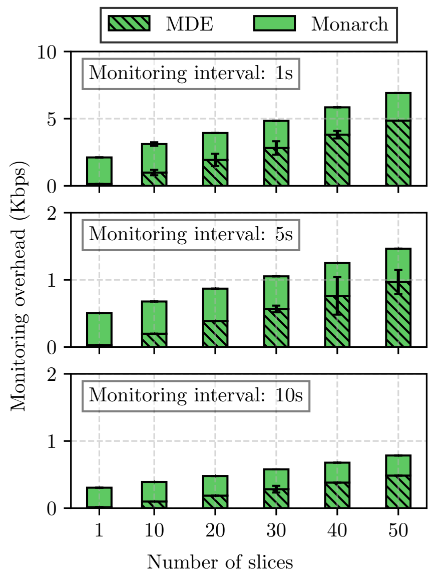
(b) CPU usage of NSSDC



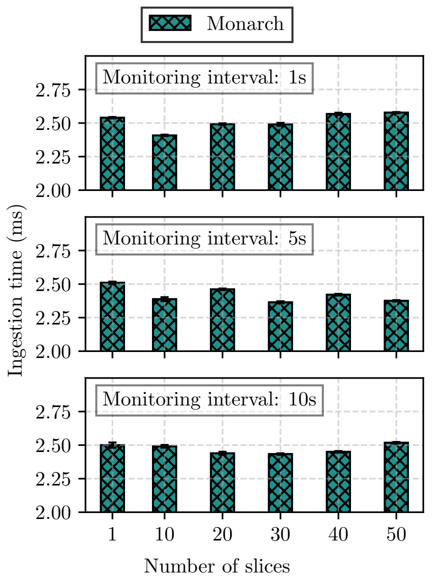
(c) Memory usage vs. slices



(d) Memory usage of NSSDC



(e) Monitoring overhead vs. slices



(f) Ingestion time vs. slices

Fig. 6: Monarch system performance

Figure 6b shows the CPU usage of the NSSDC component of Monarch. Given its role in data collection, the resource usage of this component is susceptible to change with both the number of slices and the monitoring interval. Notably, our analysis indicates that when there are 50 slices in the network, NSSDC contributes to 66%, 52%, and 50% of the total CPU usage of Monarch at monitoring intervals of $1s$, $5s$, and $10s$, respectively. Moreover, we observe a proportional increase in the share of total CPU usage attributed to NSSDC as the number of slices increases.

Memory usage. Figure 6c shows the memory usage of Monarch with the number of slices in the network varying from 1 to 50, and monitoring intervals set at $1s$, $5s$, and $10s$. In general, we observe as similar trend as CPU usage, where the memory usage of Monarch grows linearly as the number of slices increases; however, the increase is not very significant. Specifically, Specifically, when scaling the number of slices by $50\times$ from 1 to 50, we note a marginal $1.04\times$, $1.05\times$, and $1.05\times$ increase in memory usage for monitoring intervals of $1s$, $5s$, and $10s$, respectively.

Figure 6d shows the memory usage of the NSSDC component of Monarch, relative to the total memory usage. Unlike CPU usage, NSSDC's contribution to total memory usage is minor, with other components such as the data store occupying a larger share. Furthermore, our analysis reveals no significant augmentation in memory usage under lower monitoring intervals (higher frequency). This is because memory consumption is contingent upon the total number of data points collected before being flushed to disk, rather than the rate at which data is collected.

Monitoring overhead. Figure 6e shows the monitoring overhead of Monarch with the number of slices in the network varying from 1 to 50, and monitoring intervals set at $1s$, $5s$, and $10s$. We observe that monitoring overhead increases linearly with the number of slices. This is expected since we are collecting per-slice metrics. Specifically, when scaling the number of slices by $50\times$ from 1 to 50, we note $3.2\times$, $2.92\times$, and $2.6\times$ increase in memory usage for monitoring intervals of $1s$, $5s$, and $10s$, respectively.

Additionally, our analysis reveals the substantial impact of monitoring intervals on monitoring overhead, with monitoring intervals of $1s$ and $5s$ exhibiting $23\times$ and $4.86\times$ the overhead of the $10s$ interval, respectively, with 50 slices in the network. Further, we examine the contribution of MDEs relative to the total overhead. We define the MDE contribution to the total overhead as the volume of data (bytes/sec) used by the MDEs to collect raw monitoring data from the data sources. With 50 slices in the network, MDE overhead accounts for 69%, 66%, and 61% of the total overhead for monitoring intervals of $1s$, $5s$, and $10s$, respectively. These findings underscore the importance of selecting appropriate monitoring intervals to balance the trade-off between monitoring granularity and resource utilization.

Ingestion time. Next, we examine the ingestion time of Monarch, shown in Figure 6f. We observe minimal impact on ingestion time regardless of the monitoring interval or the number of slices, maintaining a consistent range between 2.25 to 2.75 ms. This consistency suggests Monarch's robust scalability with an increase in the number of slices, without introducing notable delays. However, we also find that while the monitoring interval does not significantly affect the ingestion time, it significantly impacts the accuracy of the monitoring results. In terms of fixed frequency monitoring, we find that a monitoring interval of $5s$ offers a good trade-off between monitoring overhead and accuracy. We examine this further in §V-C.

Factors affecting ingestion time. To explore the factors affecting ingestion time, we utilized Avalanche [35], a load-testing tool for Prometheus-compatible systems. Our investigation aimed to understand the impact of factors such as the number of time series and endpoints on ingestion time. Metrics, representing observable properties with defined dimensions (labels), generate multiple time series based on label variations. In the context of 5G network monitoring, metrics encompass various NF performance indicators, such as packet volume on the N3 interface of the UPF, with labels including annotations such as NF instance and network slice ID.

Figure 7a illustrates how the ingestion time changes with the number of monitored time series from a single endpoint. Here, an endpoint refers to a uniform resource locator (URL) that specifies the location of a resource (metric(s) to be monitored). We observe a gradual increase in ingestion time until it sharply escalates beyond a threshold (1000 time series), indicating degraded system performance. To assess the impact of endpoint numbers on ingestion time, Figure 7b explores the scenario with 10^4 time series being monitored. As the number of endpoints increases, ingestion time logarithmically decreases and stabilizes. This observation underscores the suboptimal practice of relying on a single endpoint for extensive data collection. In Monarch, this issue is mitigated by employing separate MDEs for each NF, thus averting the overload of a single endpoint by excessive metric collection tasks.

Further, we explore the number of time series monitored by Monarch, encompassing data collected for E2E KPI computation and self-monitoring for resource utilization. As depicted in Figure 7c, with 50 slices deployed in the network, Monarch monitors a total of 2934 time series. Notably, this number of time series is not aggregated from a single endpoint but distributed across multiple endpoints, highlighting Monarch's distributed data collection strategy for enhanced scalability and efficiency.

C. Fixed vs. Adaptive Monitoring

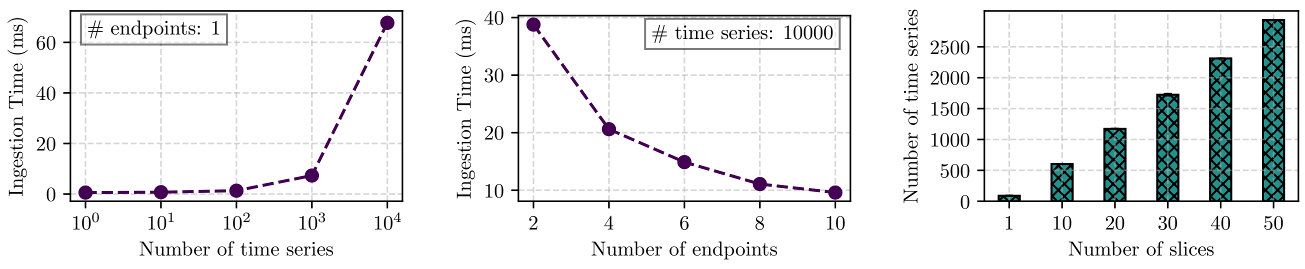
Finally, we assess the effectiveness of Monarch's adaptive monitoring mechanism, as detailed in §III-D. For our evaluation, we leveraged real-world 5G traffic traces sourced from a dataset [28], replayed via Pixel UEs through our testbed, and employed Monarch to compute E2E slice KPIs, as outlined in §IV. The experiment duration spanned three hours, with the results tabulated in Table III. However, for clarity, Figure 8 provides a snapshot of 300 seconds during the experiment period.

TABLE III: Evaluation of adaptive monitoring

Interval	MAE	MAPE	DSR	EU
5s	0.0301	0.2351	0.80	3.4027
10s	0.0760	0.5899	0.90	1.5257
Adaptive	0.0232	0.1742	0.76	**4.3444**

Table III shows several metrics including the mean absolute error (MAE), mean absolute percentage error (MAPE), data savings ratio (DSR), and expected utility (EU) for different monitoring strategies (fixed 5s/10s and adaptive). The DSR represents the reduction in monitoring overhead by monitoring data points at intervals relative to the base interval of $1s$. We choose $1s$ as the lowest granularity due to system performance constraints in Prometheus at intervals below this threshold. In Figure 8, the term "original" denotes the time series monitored at the base interval of $1s$. Additionally, we introduce the EU metric to assess the accuracy-overhead trade-off. EU is calculated as the DSR divided by the MAPE, where higher values indicate better performance. Notably, fixed-frequency monitoring at a $10s$ interval exhibits the best DSR, yet the proposed adaptive monitoring outperforms both $5s$ and $10s$ intervals in terms of EU.

Figure 8 illustrates a time series plot of network slice throughput KPI during our experiment, showcasing different monitoring intervals of $5s$, $10s$, and adaptive. Notably, the adaptive monitoring approach dynamically adjusts intervals according to traffic variations, effectively capturing sudden changes by reducing the interval and minimizing overhead during stable periods by increasing the interval. From the zoomed-in section in Figure 8 (top), we can see that the adaptive monitoring scheme captures more data points (compared to fixed-frequency monitoring) during the sharp dip in throughput

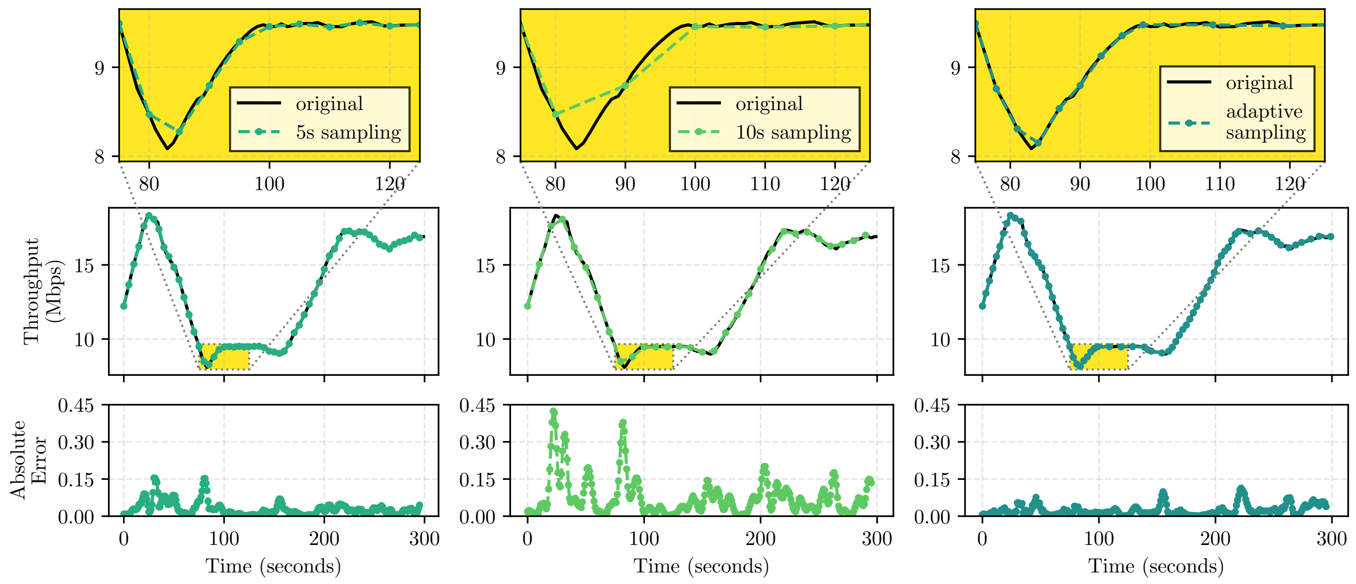


(a) Ingestion time vs. time series

(b) Ingestion time vs. endpoints

(c) Time series vs. slices

Fig. 7: Factors affecting ingestion time



(a) 5s sampling

(b) 10s sampling

(c) adaptive sampling

Fig. 8: Time series plot showing a) throughput at different monitoring intervals (top) and absolute error (bottom)

around the 80s mark and less data points during the relatively stable period from 100s to 200s. The bottom sub-figures display the corresponding absolute error values (compared to the baseline), highlighting the superior performance of adaptive monitoring in error reduction over fixed-frequency schemes.

VI. Discussions

Insights from Monarch. Monarch is a scalable monitoring architecture designed for cloud-native 5G deployments, focusing on E2E network slice KPIs. We discuss below a few key insights from Monarch:

- Monarch scales well with the number of slices and supports monitoring up to 50 network slices within a modest 2-node cluster deployment.
- While a notable portion of Monarch's CPU usage is attributed to the NSSDC component, we observe improved performance with higher monitoring intervals. This suggests potential benefits in employing adaptive monitoring strategies. Further, the inherently cloud-native design of Monarch facilitates horizontal scaling of NSSDC instances, offering a solution for managing increased computational demands.
- Ingestion time within Monarch is notably influenced by the number of endpoints, suggesting a best practice of horizontally scaling slices. For example, deploying separate instances of UPF for different slices can mitigate potential issues related to a large number of metrics generated from a single endpoint, thereby reducing monitoring delays.

NWDAF and Monarch synergy. NWDAF [3] is a fundamental function of the 5G core network, focused on network data collection and analytics, both essential for automated network management. On the other hand, Monarch is a scalable monitoring architecture designed specifically for cloud-native 5G network deployments. It gathers performance metrics from various network segments, providing a more comprehensive view of the E2E network slices. While NWDAF and Monarch serve distinct purposes, they can harmoniously collaborate to enhance 5G network slice monitoring. This collaboration can be understood in terms of the following aspects:

- *Enhanced Monitoring Scope.* While 3GPP provides standardized interfaces for NWDAF to collect data from 3GPP core NFs and application functions, it primarily focuses on the core network. This leaves out critical segments such as the transport and the RAN, which are also part of an E2E slice. Monarch addresses this gap by extending the monitoring scope to various network segments for E2E slice monitoring.

- *Enhanced scalability.* Recent works have proposed a hierarchical NWDAF architecture, where a central NWDAF aggregates data and insights from various edge NWDAFs. In a distributed NWDAF setup, efficient communication between the edge and central NWDAF is paramount, especially for data collection. Monarch aligns with this approach through its network slice segment data collector (NSSDC) component (§III-A), which is similar to the multiple instances of NWDAF deployed across various areas or regions as per 3GPP Release 17. Furthermore, Monarch includes algorithms for efficiently determining monitoring granularity, enhancing the scalability of monitoring (§III-D).

A combined deployment where Monarch and NWDAF work in synergy can significantly enhance 5G network slice monitoring. The architecture would involve Monarch collecting detailed performance metrics across all network segments, including those not covered by NWDAF, and NWDAF utilizing this data to provide aggregated insights and predictive analytics for network management.

Other use-cases for Monarch. Monarch is primarily designed to aggregate 5G performance metrics and compute E2E slice KPIs. As detailed in Section IV, Monarch can compute KPIs at both the NF and slice levels. However, Monarch's data collection relies on MDEs to gather monitoring data from sources that expose such data. Existing works like 5GC-Observer [22] and Janus [21] have showcased methods for extracting various 5G-specific metrics from the core and the RAN, respectively. Our future exploration includes integrating these approaches with Monarch's MDEs to broaden the spectrum of collected 5G metrics, thereby enabling diverse use cases. Additionally, for the RAN, we intend to utilize the ORAN RAN Intelligent Controller (RIC) to capture RAN-specific metrics using xApps.

Request translation. Presently, Monarch's implementation of the request translation module is limited, as it relies on manual processes. This limitation arises from the absence of readily available open-source E2E slice orchestrators. Looking ahead, we aim to explore automated translation methods for high-level monitoring requests and evaluate the potential of platforms like ONAP [29] for E2E slice orchestration capabilities.

VII. Conclusion

In this work, we present Monarch, a scalable monitoring architecture designed to provide E2E visibility of network slices and facilitate per-slice KPI computation. Monarch addresses several critical challenges associated with monitoring network slices in 5G and beyond networks such as scalability and E2E KPI computation. Through a proof-of-concept implementation of Monarch on a 5G network slice testbed, we demonstrated Monarch's ability to effectively monitor a wide range of network slices, accommodating up to 50 slices with consistent ingestion times, assuming each slice has a dedicated UPF. Further, our evaluations highlight the efficacy of Monarch's adaptive monitoring mechanism, which significantly reduces monitoring overhead while maintaining accuracy. Moving forward, we aim to improve the capabilities of Monarch towards addressing monitoring requirements in a dynamic network environment, particularly in automating request translation and integrating it with emerging slice orchestration platforms.

Acknowledgements

This work was supported in part by Rogers Communications Canada Inc. and in part by a Mitacs Accelerate Grant. We thank Amr Abouelkhair for his assistance in integrating Monarch with Open5GS.

References

[1] R. Boutaba, N. Shahriar, M. A. Salahuddin, S. R. Chowdhury, N. Saha, and A. James, "Ai-driven closed-loop automation in 5g and beyond mobile networks," in Proceedings of the ACM SIGCOMM FlexNets Workshop. Association for Computing Machinery, 2021, pp. 1–6.

[2] ETSI, "Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 3: Advanced topics," ETSI, Group Report (GR) ZSM 009-3, 08 2023, version 1.1.1.

[3] 3GPP, "5G System; Network Data Analytics Services; Stage 3," 3GPP, Technical Specification (TS) 29.520, 09 2023, version 17.12.0.

[4] 3GPP, "Management and Orchestration; 5G end to end Key Performance Indicators (KPI)," 3GPP, Technical Specification (TS) 28.554, 09 2020, version 17.0.0.

[5] OpenStack. (2021) Ceilometer project. [Online]. Available: https://docs.openstack.org/ceilometer/latest/

[6] (2021) Nagios. [Online]. Available: https://www.nagios.org/

[7] Ericsson. (2020) 5G RAN Slicing. [Online]. Available: https://www.ericsson.com/491c28/assets/global/eridoc/755404/09004cffc64e902f.pdf

[8] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in Proc. of the ACM CoNEXT, 2016, p. 427–441. [Online]. Available: https://doi.org/10.1145/2999572.2999599

[9] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "Z-TORCH: An Automated NFV Orchestration and Monitoring Solution," IEEE Transactions on Network and Service Management, vol. 15, no. 4, pp. 1292–1306, Dec. 2018.

[10] A. Plascinskas, X. Foukas, and M. K. Marina, "Towards Efficient and Adaptable Monitoring of Softwarized Mobile Networks," in Proc. of the IEEE/IFIP Network Operations and Management Symposium, Apr. 2020, pp. 1–6.

[11] W. Rafique, J. Barai, A. O. Fapojuwo, and D. Krishnamurthy, "A survey on beyond 5g network slicing for smart cities applications," IEEE Communications Surveys & Tutorials, pp. 1–1, 2024.

[12] R. Perez, J. Garcia-Reinoso, A. Zabala, P. Serrano, and A. Banchs, "A monitoring framework for multi-site 5G platforms," in Proc. of the European Conference on Networks and Communications (EuCNC), Jun. 2020, pp. 52–56.

[13] D. Giannopoulos, P. Papaioannou, L. Ntzogani, C. Tranoris, and S. Denazis, "A holistic approach for 5G Network Slice Monitoring," in 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Sep. 2021, pp. 240–245.

[14] A. Beltrami, P. D. Maciel, F. Tusa, C. Cesila, C. Rothenberg, R. Pasquini, and F. L. Verdi, "Design and implementation of an elastic monitoring architecture for cloud network slices," in Proc. of the IEEE/IFIP Network Operations and Management Symposium, Apr. 2020, pp. 1–7.

[15] M. Mekki, S. Arora, and A. Ksentini, "A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks," IEEE Transactions on Network and Service Management, pp. 1–1, 2021.

[16] 5G-Monarch repository on Github. [Online]. Available: https://github.com/niloysh/5g-monarch

[17] N. Saha, N. Shahriar, R. Boutaba, and A. Saleh, “Monarch: Network slice monitoring architecture for cloud native 5g deployments,” in IEEE/IFIP Network Operations and Management Symposium (NOMS), 2023, pp. 1–7.

[18] Open5GS. (2024) Open5GS github. [Online]. Available: https://github.com/open5gs/open5gs

[19] M. Xie, Q. Zhang, A. J. Gonzalez, P. Grønsund, P. Palacharla, and T. Ikeuchi, “Service Assurance in 5G Networks: A Study of Joint Monitoring and Analytics,” in Proc. of the IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), Sep. 2019, pp. 1–7.

[20] A. Plascinskas, X. Foukas, and M. K. Marina, “Towards Efficient and Adaptable Monitoring of Softwarized Mobile Networks,” in Proc. of the IEEE/IFIP Network Operations and Management Symposium, Apr. 2020, pp. 1–6.

[21] X. Foukas, B. Radunovic, M. Balkwill, and Z. Lai, “Taking 5G RAN Analytics and Control to a New Level,” in Proc. of the ACM MobiCom. ACM, 2023.

[22] A. Khichane, I. Fajjari, N. Aitsaadi, and M. Gueroui, “5gc-observer demonstrator: a non-intrusive observability prototype for cloud native 5g system,” in Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), 2023, pp. 1–3.

[23] X. Vasilakos, B. Köksal, D. H. Izaldi, N. Nikaein, R. Schmidt, N. Ferdosian, R. F. Sari, and R. Cheng, “ElasticSDK: A monitoring software development kit for enabling data-driven management and control in 5G,” in NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, Apr. 2020, pp. 1–7.

[24] P. Veitch, J. Browne, and J. Krogell, “An Integrated Instrumentation and Insights Framework for Holistic 5G Slice Assurance,” in Proc. of the IEEE Conference on Network Softwarization (NetSoft), Jun. 2020, pp. 247–251.

[25] M. A. Garcia-Martin, M. Gramaglia, and P. Serrano, “Network automation and data analytics in 3gpp 5g systems,” IEEE Network, pp. 1–1, 2023.

[26] A. Mekrache, K. Boutiba, and A. Ksentini, “Combining network data analytics function and machine learning for abnormal traffic detection in beyond 5g,” in Proceedings of the IEEE Global Communications Conference (GLOBECOM), 2023, pp. 1204–1209.

[27] D. M. Manias, A. Chouman, A. Al-Dulaimi, and A. Shami, “Slice-level performance metric forecasting in intelligent transportation systems and the internet of vehicles,” IEEE Internet of Things Magazine, vol. 6, no. 3, pp. 56–61, 2023.

[28] Y.-H. Choi, D. Kim, and M. Ko. (2023) 5g traffic datasets. [Online]. Available: https://dx.doi.org/10.21227/ewhk-n061

[29] The Linux Foundation. (2022) Open Network Automation Platform (ONAP). [Online]. Available: https://www.onap.org/

[30] B. Burns and D. Oppenheimer, “Design patterns for container-based distributed systems,” in Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16). Denver, CO: USENIX Association, Jun. 2016. [Online]. Available: https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/burns

[31] srsRAN project. [Online]. Available: https://github.com/srsran/srsRAN_Project

[32] A. Gungr. (2022) Ueransim. [Online]. Available: https://github.com/aligungr/UERANSIM

[33] “open5gs-k8s Github,” https://github.com/niloysh/open5gs-k8s.

[34] 3GPP, “Management and orchestration; 5G performance measurements,” 3GPP, Technical Specification (TS) 28.552, 09 2020, version 17.0.0.

[35] “Avalanche,” https://github.com/prometheus-community/avalanche.

Niloy Saha is a Ph.D. student at the University of Waterloo. He received his Master’s degree in Computer Science from the Indian Institute of Technology, Kharagpur, India. His research interests include next-generation mobile networks, network telemetry, and intelligent algorithms orchestration and management of 5G and beyond networks.

Nashid Shahriar is an Assistant Professor in the Department of Computer Science at the University of Regina. He received his PhD from the School of Computer Science, University of Waterloo in 2020. He is the recipient of 2023 Young Professional Award from IEEE Communications Society Technical Committee on Network Operation and Management and 2020 PhD Alumni Gold Medal and 2021 Mathematics Doctoral prize from the University of Waterloo. His research received several recognitions, including the IEEE/IFIP NOMS 2022 Best Student Paper Award, IFIP/IEEE IM 2021 Best PhD Dissertation Award, the IEEE/ACM/IFIP CNSM 2019 Best Paper Award, IEEE NetSoft 2019 Best Student Paper Award, and the IEEE/ACM/IFIP CNSM 2017 Best Paper Award. His research interests include resource optimization and monitoring in network function virtualization and software-defined networking and enhancing security and reliability of network slices in 5G and beyond mobile networks.

Muhammad Sulaiman (Student Member, IEEE) received BS in Electrical Engineering from the National University of Sciences and Technology, Pakistan, in 2019. He joined the University of Waterloo for a Master of Mathematics (MMath) in Computer Science in 2020 and transferred to the Ph.D. program soon after. His work was nominated for IEEE/IFIP NOMS best paper award in 2022 and 2023. His research interests include reinforcement learning, optimization of computer networks, and autonomous management and orchestration of mobile networks.

Noura Limam received the M.Sc. and Ph.D. degrees in computer science from the University Pierre and Marie Curie (currently Sorbonne University), France, in 2002 and 2007, respectively. She is a Research Assistant Professor of Computer Science with the University of Waterloo, Canada. She is an active researcher and a contributor in the area of network and service management. Her current interests revolve around network automation and cognitive network management. She is the Chair of the IEEE ComSoc Network Operations and Management Technical Committee, an Associate Editor of the IEEE Communications Magazine, and a Guest Editor of the IEEE Communications Magazine Network Softwarization and Management Series.

Raouf Boutaba received the M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Chair Professor and the Director of the David R. Cheriton School of Computer Science at the University of Waterloo (Canada). He is the founding Editor-in-Chief of the IEEE Transactions on Network and Service Management (2007-2010) and served as the Editor-in-Chief of the IEEE Journal on Selected Areas in Communications (2018-2021). He is a fellow of the IEEE, the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.

Aladdin Saleh is currently priming research and innovation activities at Rogers communications, among them the joint research partnership with the University of Waterloo on 5G and emerging technologies. He has over 20 years of industry experience in mobile telecom in Canada. He earned a Ph.D. degree in electrical and electronic engineering and an MBA in International Management, both from the university of London in the UK. He taught and conducted research on next-generation wireless networks at several universities as a full-time professor, Adjunct professor, and a visiting researcher. He is currently an Adjunct Professor with the Cheriton School of Computer Science at the University of Waterloo.