

Efficient High-precision Monitoring of Network Slices for 5G and Beyond Networks

by

Niloy Saha

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Niloy Saha 2026

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Israat Haque
Associate Professor
Faculty of Computer Science
Dalhousie University

Supervisor: Raouf Boutaba
Professor
David R. Cheriton School of Computer Science
University of Waterloo

Internal Member: Mina Tahmasbi Arashloo
Assistant Professor
David R. Cheriton School of Computer Science
University of Waterloo

Internal Member: Ali José Mashtizadeh
Associate Professor
David R. Cheriton School of Computer Science
University of Waterloo

Internal-External Member: Tahsin Reza
Assistant Professor
Department of Electrical and Computer Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This dissertation includes first-authored and peer-reviewed work that has appeared in conference proceedings published by the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM). The following is a list that serves as a declaration of the works included in this dissertation. I expand and revise material from the original publications.

Portions of Chapter 3

N. Saha, N. Shahriar, M. Sulaiman, N. Limam, R. Boutaba, and A. Saleh, “MonArch: Monitoring Architecture for 5G and Beyond Network Slices,” *IEEE Transactions on Network and Service Management*, vol. 22, no. 1, pp. 777–790, 2025. doi: 10.1109/TNSM.2024.3479246.

N. Saha, N. Shahriar, R. Boutaba, and A. Saleh, “MonArch: Network Slice Monitoring Architecture for Cloud Native 5G Deployments,” in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2023, pp. 1–7. doi: 10.1109/NOMS56928.2023.10154344.

N. Saha, A. James, N. Shahriar, R. Boutaba, and A. Saleh, “Demonstrating Network Slice KPI Monitoring in a 5G Testbed,” in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022, pp. 1–3. doi: 10.1109/NOMS54207.2022.9789904.

Portions of Chapter 4

N. Saha, M. T. Arashloo, N. Shahriar, and R. Boutaba, “Dynamic SLA-aware Network Slice Monitoring,” in *Proc. ACM SIGMETRICS*, 2026.

N. Saha, M. T. Arashloo, N. Shahriar, R. Boutaba, “Dynamic SLA-aware Network Slice Monitoring”, in *Proc. of the ACM on Measurement and Analysis of Computing Systems (POMACS)*. ACM Press. Vol. 10(1), No. 08, March 2026.

Portions of Chapter 5

N. Saha, N. Limam, Y. Xiao, and R. Boutaba, “Rethinking Telemetry Design for Fine-Grained Anomaly Detection in 5G User Plane,” in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2026.

Abstract

Next-generation mobile networks are undergoing a fundamental transformation to support a variety of services such as Augmented Reality (AR), vehicular automation, and smart cities, each with very distinct performance requirements. The primary enabler for this transformation is *network slicing*, which enables partitioning of the shared physical infrastructure into isolated logical networks, each tailored to the performance requirements of its target service through slice-specific Service Level Agreement (SLA) guarantees.

Realizing such network slices in practice depends critically on *automated* closed-loop management and orchestration mechanisms, where *monitoring data* drives analytical and control decisions. However, monitoring is a key bottleneck in achieving this goal: control logic can only optimize what it can observe, and observation is constrained by strict overhead, latency, and deployability limits. Existing monitoring solutions are domain-centric, statically configured, and not sufficiently expressive to capture the transient, end-to-end (e2e) performance signals that slice SLA assurance demands.

The overarching goal of this dissertation is to build the foundation for efficient, high-precision monitoring for network slices in 5G and beyond networks. Towards achieving this goal, we address three distinct but coupled challenges across the monitoring stack. First, we introduce MONARCH, a cloud-native architecture that abstracts slices as first-class monitored entities and enables direct computation of e2e slice Key Performance Indicators (KPIs) across heterogeneous domains. Next, we present SLICESCOPE, a principled framework for dynamically allocating limited monitoring budget across slices with heterogeneous SLAs. Finally, we develop KESTREL, a detectability-driven, sketch-based telemetry mechanism for detecting and attributing transient Quality of Service (QoS) anomalies in shared user-plane resources without per-packet overhead. Together, these contributions demonstrate that efficiency and precision in slice monitoring are jointly achievable through cross-layer design spanning architectural abstractions, control plane intelligence, and data plane telemetry.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Raouf Boutaba, for helping me navigate many challenges throughout this journey and helping me grow as a researcher. I feel very fortunate to have such a welcoming, supportive, and knowledgeable supervisor, who challenged me to think bigger, and whose confidence in my work encouraged me to reach for more than I thought initially possible.

I thank the members of my examining committee for their time, careful reading, and constructive feedback on this dissertation. I am grateful to Professor Israat Haque for serving as my external examiner and for her thoughtful comments. I also thank Professors Mina Tahmasbi Arashloo, Ali José Mashtizadeh, and Tahsin Reza for their valuable feedback and guidance during the examination process.

I am especially grateful to Professors Mina Tahmasbi Arashloo, Noura Limam, and Nashid Shahriar for their collaboration on research projects throughout this journey. I appreciate the many insightful discussions, which helped me refine my research ideas and taught me how to think more deeply about systems and networking. I am especially grateful to Professor Mina, whose dedication went above and beyond, and whose mentorship left a lasting mark on how I approach research.

I am grateful to the many people from the Computer Science Computing Facility (CSCF) who helped set up and maintain the research infrastructure. In particular, I would like to thank Costin, Harsh, and Tommy, whose prompt and reliable support was indispensable in keeping the research on track. I am also grateful for discussions with industry partners, including Rogers Communications and Expeto Networks, which helped ground this work in practical deployment constraints for cloud-native 5G systems, network slicing, and operational telemetry.

I would also like to thank my labmates and friends for the many stimulating discussions and enjoyable moments. I appreciate the many engaging conversations, both technical and otherwise, with Sulaiman, Rouili, Philippe, Amir, Setare, and Amr, which made the lab an enjoyable place to work. Beyond the lab, the adventures and travels across Canada with Philippe, Amr, Rouili, and Zangooui are among the fondest memories I will carry from my time during the PhD. I am also grateful to my friends beyond the lab, including Ujwal, Udani, and Indrani, for the many enjoyable hangouts that provided a welcome break from research. A special thanks to Glenn, who introduced me to the Canadian outdoors, which led to some of the most memorable experiences outside the lab.

Last but not least, I would like to thank my parents for their unwavering love, encouragement, and support throughout this journey.

Dedication

To my parents, whose love and support sustained me throughout this journey.

Table of Contents

Examining Committee Membership	ii
Author’s Declaration	iii
Statement of Contributions	iv
Abstract	v
Acknowledgements	vi
Dedication	vii
List of Figures	xiii
List of Tables	xix
List of Acronyms	xxi
1 Introduction	1
1.1 Challenges in Monitoring Network Slices	5
1.1.1 End-to-End Slice Visibility Across Domains	5
1.1.2 SLA-Aware Control of Monitoring Fidelity	6
1.1.3 Fine-Grained Detection and Attribution in the User Plane	7
1.2 Thesis Contributions	8
1.2.1 MONARCH: Architecture for Slice-Level Monitoring	9
1.2.2 SLICESCOPE: SLA-Aware Network Slice Monitoring	10
1.2.3 KESTREL: Lightweight Telemetry for Network Slice User-Planes	11
1.3 Summary of Results	12
1.4 Thesis Organization	13

2	Background & Related Work	15
2.1	Foundations: Slice Realization and Monitoring	15
2.1.1	Operational Identifiers and Slice Realization	16
2.1.2	Monitoring Fidelity: The Precision-Overhead Trade-off	18
2.1.3	Programmable Data Planes	20
2.1.4	Data Plane Telemetry Primitives	21
2.2	Architectures and Mechanisms for Slice Monitoring	23
2.2.1	Domain-centric vs Slice-aware Pipelines	23
2.2.2	Adaptive and Efficient Telemetry	24
2.2.3	KPI Computation and Analytics	25
2.3	SLA-Aware Network Slice Monitoring	26
2.3.1	Fixed-Rate and Uniform Slice Monitoring	27
2.3.2	High-precision Telemetry for Network Slices	27
2.4	User-Plane Telemetry and Anomaly Detection	29
2.4.1	User-Plane Telemetry Mechanisms	29
2.4.2	Anomaly Detection and Analytics in 5G Slices	30
2.5	Synthesis and Open Gaps	31
3	MONARCH: Architecture for Slice-Level Monitoring	34
3.1	Introduction	34
3.2	Characterizing the Slice Observability Gap	37
3.2.1	The State-Telemetry Mismatch	37
3.2.2	Operational Complexity of State Reconstruction	39
3.2.3	Design Requirements	40
3.3	MONARCH Design and Operation	40
3.3.1	Monarch Overview	41
3.3.2	Monitoring API	45
3.3.3	Flow of a monitoring request	47
3.3.4	Adaptive Monitoring	48
3.4	Prototype Realization	49
3.4.1	5G Testbed Environment	50
3.4.2	Architectural Realization	50
3.4.3	Use-Case Validation	52
3.4.4	Adaptive Monitoring Realization	54
3.4.5	Design and Implementation Challenges	56

3.5	Evaluation	57
3.5.1	Experimental Setup	58
3.5.2	Scalability with Slice Count	58
3.5.3	Telemetry Ingestion and Time-Series Behavior	61
3.5.4	Monitoring Intervals and Resource Efficiency	63
3.5.5	Query Processing and Slice-Level Aggregation	64
3.5.6	Adaptive Monitoring Trade-offs	66
3.5.7	Evaluation Summary	68
3.6	Discussions and Limitations	69
3.7	Chapter Summary	71
4	SLICESCOPE: SLA-Aware Network Slice Monitoring	73
4.1	Introduction	73
4.2	The Case for SLA-Aware Slice-Level Control	76
4.2.1	SLA Diversity in Modern Networks	77
4.2.2	Why Slice-level Monitoring?	77
4.3	Formalizing Closed-loop SLA-aware Slice Monitoring	78
4.3.1	Joint Multi-Slice Resource Allocation	79
4.3.2	Adapting to Network Dynamics	81
4.3.3	Requirements from the data plane telemetry primitive	82
4.4	SLICESCOPE: Realizing the Closed Loop with Change-Triggered INT	83
4.4.1	Why change-triggered In-Band-Network Telemetry (INT)?	84
4.4.2	Mathematical Framework for Modeling E and Γ	85
4.4.3	Telemetry Data Structures and Operations	88
4.4.4	Design and Implementation Challenges	92
4.5	Evaluation	94
4.5.1	Experimental Setup, Workloads, and Implementation	94
4.5.2	The Benefit of Per-Slice Closed-Loop Control	96
4.5.3	Data Plane Primitives for Slice SLA Monitoring	98
4.5.4	Microbenchmarks	100
4.5.5	Testbed Evaluation	103
4.6	Discussions and Limitations	104
4.7	Chapter Summary	106

5	KESTREL: Lightweight Telemetry for Network Slice User-Plane Monitoring	108
5.1	Introduction	108
5.2	Characterizing the Fine-Grained Telemetry Problem in 5G Data Planes . .	111
5.2.1	The Visibility Gap in UPF Telemetry	111
5.2.2	Empirical Characterization of Anomaly Signatures	112
5.3	Design of KESTREL	114
5.3.1	Design Objectives	114
5.3.2	Design Challenges and Solutions	115
5.3.3	Data Structure and Operations	116
5.4	Parameterizing KESTREL	118
5.4.1	Detectability Guarantees	118
5.4.2	Sketch Sizing under Detection Targets	120
5.4.3	Maintaining Diagnostic Sparsity Under Drift	121
5.4.4	Maintaining Collision Bounds under Traffic Heterogeneity	123
5.5	Prototype Realization	125
5.5.1	Prototype and Experimental Setup	125
5.5.2	Design and Implementation Challenges	127
5.6	Evaluation	128
5.6.1	Anomaly Detection Accuracy	129
5.6.2	Cost-Effectiveness	130
5.6.3	Responsiveness	132
5.6.4	Microbenchmarks and Design Insights	133
5.6.5	Maintaining Detectability Under Distribution Drift	134
5.6.6	Budget-Aware Width Allocation Across QIDs	138
5.7	Discussions and Limitations	140
5.8	Chapter Summary	142
6	Conclusion & Future Work	144
6.1	Summary of Contributions	144
6.2	Cross-Layer Synthesis	145
6.2.1	Slice SLA Violation and Progressive Attribution	146
6.3	Limitations and Design Tradeoffs	148
6.4	Threats to Validity	149
6.5	Future Research Directions	152

6.5.1	Causal Attribution Across the Slice–Infrastructure Boundary	152
6.5.2	Principled Scope Refinement Policies	153
6.5.3	Data and Control Plane Boundary in Anomaly Detection	154
References		155
APPENDICES		168
A	Experimental Testbed	169
A.1	Hardware and Deployment	170
A.2	5G Network and Slice Implementation	171
A.3	Traffic Generation	172
B	Proof of the Detectability Condition in KESTREL	174

List of Figures

1.1	Conceptual mapping of logical network slices to shared physical infrastructure. Each slice consists of a chain of virtual network functions (VNFs) and the virtual links interconnecting them, instantiated across heterogeneous domains, such as the Radio Unit, Edge Cloud, Transport Network, and Central Cloud.	2
1.2	The role of monitoring within the Monitor–Analyze–Plan–Execute (MAPE) loop. High-fidelity monitoring provides the critical input for network intelligence to maintain slice-specific SLAs.	3
1.3	Thesis overview and scope. The monitoring stack as the foundational enabler for closed-loop network automation, defining the scope of this thesis. Chapters 3–5 develop complementary contributions to slice monitoring: MONARCH provides slice-level monitoring abstractions, SLICESCOPE regulates telemetry fidelity under heterogeneous SLAs, and KESTREL enables fine-grained user-plane visibility for detecting transient performance anomalies. Together, these components provide the monitoring foundation required to drive automated data-driven management of network slices.	9
1.4	Summary of key empirical results across the thesis. (a) MONARCH establishes scalable e2e slice-level KPI monitoring, showing predictable overhead growth with the number of slices and bounded ingestion latency. (b) Building on this slice-centric visibility, SLICESCOPE demonstrates that allocating telemetry budgets according to SLA requirements yields improved cost–accuracy trade-offs compared to static monitoring. (c) Finally, KESTREL grounds telemetry efficiency in detectability, and shows that principled sketch parameterization enables accurate anomaly detection under fixed memory budgets.	13

2.1	Layering of operational identifiers in 5G slice realization. S-NSSAI anchors slice identity at the administrative level, QoS flows define per-traffic treatment, and TEIDs index session statistics in the user plane. Because slice semantics are distributed across these layers, slice-level monitoring requires cross-layer correlation.	16
2.2	Monitoring fidelity as a configuration space. Spatial, temporal, and statistical fidelity jointly determine the precision and scope of slice monitoring under resource constraints. The three thesis contributions emphasize complementary axes of this space: MONARCH (Chapter 3) provides monitoring abstractions at the slice level, SLICESCOPE (Chapter 4) allocates measurement rates under global budgets, and KESTREL (Chapter 5) preserves distributional signals under tight data plane constraints.	19
2.3	Basic operation of In-band Network Telemetry (INT). Packets are instrumented at the INT source and accumulate per-hop telemetry metadata as they traverse the data plane. At the INT sink, the metadata is stripped and exported to a collector, and the original packet headers are restored before forwarding to the destination.	22
2.4	Limitations of domain-centric monitoring pipelines. Computing slice-level KPIs (e.g., throughput or latency as defined in 3GPP [1, 2]) often requires correlating measurements exported independently by different network functions (e.g., SMF and UPF). In the absence of a slice-centric abstraction, this correlation must be implemented externally, leading to a lack of slice-centric view.	24
2.5	Heterogeneous monitoring requirements across network slices. While eMBB slices can operate with 10 – 50 ms latency, URLLC slices need continuous, high-precision tracking to ensure real-time compliance with strict 1 – 5 ms latency SLA targets.	27
2.6	Attribution challenges in multi-tenant user planes. Multiple GPRS Tunneling Protocol – User Plane (GTP-U) tunnels (Tunnel Endpoint Identifiers (TEIDs)) from different network slices converge into shared hardware queues at the User Plane Function (UPF). Congestion within these shared resources often manifests as transient distributional shifts (e.g., tail latency inflation) that are difficult to attribute to a specific slice using aggregate counters, necessitating high-fidelity telemetry capable of per-slice distributional resolution.	30

3.1	High-level architecture of MONARCH. MONARCH centralizes slice semantics within the monitoring layer by a) translating slice-level monitoring requests into segment-specific directives, b) aggregating telemetry across slice segments, and c) computing slice-level KPIs before exposing them through a unified Application Programming Interface (API). Cross-domain state correlation is performed once within the KPIs computation module rather than repeatedly by downstream applications.	41
3.2	Structure of a monitoring request intent object. Monitoring intent is defined by scope, KPI semantics, and operational policy.	47
3.3	MONARCH prototype integration with the Fifth Generation (5G) testbed. The control plane (dashed arrows) propagates monitoring intent to configure segment collectors, while the data plane (solid arrows) exports Network Function (NF) telemetry to the aggregation layer for slice-level KPI computation and retrieval.	51
3.4	MONARCH scales near-linearly with slice count without introducing centralized bottlenecks. CPU usage and monitoring overhead increase proportionally with the number of slices, while memory consumption remains nearly constant across monitoring intervals. Network Slice Segment Data Collector (NSSDC) dominates CPU utilization, and Monitoring Data Exporters (MDEs) account for the majority of monitoring overhead, reflecting MONARCH's slice-local telemetry architecture.	59
3.5	Slice scaling does not destabilize ingestion. Although the number of time series grows with slice count, ingestion latency remains stable due to distributed telemetry.	61
3.6	Distributed telemetry mitigates ingestion bottlenecks. Concentrating high time-series cardinality behind a single endpoint leads to sharp latency increases, whereas distributing telemetry across multiple endpoints preserves ingestion performance.	62
3.7	Query latency grows predictably with workload. Latency increases with datapoint volume and query arrival rate, particularly for complex slice-level KPIs. Horizontal partitioning of the aggregation layer reduces latency under fixed total load.	65
3.8	Adaptive monitoring preserves fidelity while reducing telemetry volume. Network slice throughput KPI under fixed 5s, fixed 10s, and adaptive intervals (top), with corresponding absolute error relative to 1s baseline (bottom). Adaptive monitoring increases sampling frequency during traffic bursts while reducing it during stable periods, maintaining low error with reduced monitoring overhead.	67

4.1	Closed-loop formulation of SLA-aware slice monitoring. The control plane continuously learns from telemetry observations that reflect slice-level performance, estimates monitoring error $E(k_{s,m})$ and overhead $\Gamma(k_{s,m})$, and tunes per-slice knobs $k_{s,m}$ of the data plane telemetry primitive to satisfy SLA objectives under resource constraints.	78
4.2	Analytical framework for change-driven telemetry insertion. (a) Accuracy-overhead trade-off as a function of the telemetry threshold $\Delta_{s,m}$: increasing $\Delta_{s,m}$ reduces telemetry overhead (blue) but increases monitoring error (red). (b) Example trace of the simulated cumulative metric difference $D(t, \Delta_{s,m})$, which mimics how $\delta_{s,m}$ accumulates in the data plane, and triggers telemetry insertion whenever it exceeds $\Delta_{s,m}$. (c) Probability density of $D(t, \Delta_{s,m})$, where the shaded region corresponds to the insertion probability $\beta(\Delta_{s,m})$	86
4.3	Per-slice telemetry state stored in each bucket A_{ij} of the bucket arrays. Each bucket maintains the slice key x , e2e estimates $(E_{prev}, E_{rep}, E_{last})$, auxiliary state V_{aux} (e.g., counters for packet loss), and a miss flag for recovery.	89
4.4	Telemetry header format. The fixed part (always present) includes a shim and per-hop metadata. The conditional part (inserted only when $\delta_{s,m} > \Delta_{s,m}$) carries updated e2e metric values E_{curr} and auxiliary state V_{aux} for computing metrics such as jitter or packet loss.	89
4.5	Pareto frontiers comparing per-packet overhead and tolerance violations across different workload mixes (SP, BAL, and LP). SLICESCOPE “moves” monitoring resources from less critical slices to more critical ones as needed. For <i>Static slice-agnostic</i> , points represent varying Δ values (1-20), for <i>Static slice-aware</i> , different combinations of Δ values for (URLLC, eMBB, mMTC) slices, and for SLICESCOPE, different values of tuning parameter λ	97
4.6	Comparison of different data plane monitoring primitives for per-packet e2e slice monitoring	99
4.7	Control plane scalability. (a) Runtime scaling with slice count for the Integer Linear Program (ILP) optimizer and heuristic fallback. (b) Monitoring quality across epoch lengths τ ; the heuristic achieves comparable overhead-violation trade-offs while running $\sim 12\times$ faster.	101
4.8	Bucket array sizing. Table-miss rate, memory usage, and overhead for different (d, w) configurations. A configuration of $d=2$, $w=4096$ achieves $< 3\%$ miss rate with minimal extra bandwidth overhead.	102

4.9	Real-time tracking of slice metrics on Tofino. SLICESCOPE adapts telemetry reporting rates as slice activity and latency variation change over time.	103
5.1	Anomaly signatures require appropriate monitoring granularity. (a) Congestion produces sustained tail latency on specific QoS Flow Identifiers (QFIs). (b) Policy abuse appears benign in QFI aggregates but is visible at the TEID-level. (c) Microbursts vanish in second-level averages yet emerge with sub-second resolution. (d) Contention induces distributed oscillations in Inter-Arrival Time (IAT) across flows.	113
5.2	Data structure and operations of KESTREL. Buckets extend Count-Min Sketch (CMS) to record packet/byte totals, latency histograms, IAT histograms, and policer colors. Example shows an arriving packet (TEID=87, QFI=3, Queue Identifier (QID)=2, latency=25 μ s, IAT=18 μ s, color=yellow) being hashed to a bucket: totals and color counters are incremented, latency/IAT bins are updated using QID-specific edges, and the last is timestamp refreshed (updates in red).	115
5.3	Detectability-driven parameterization of KESTREL. (a) Anomalies are detectable when the diagnostic-region signal exceeds sketch collision noise εN_T ; increasing sketch width w lowers this noise floor. (b) Traffic drift increases baseline diagnostic occupancy N_T , violating the target $N_T \leq \rho N$ and necessitating re-binning. (c) Under a fixed on-chip memory budget W_{total} , per-queue widths w_q are allocated to equalize worst-case detectability (minimax) across heterogeneous queues.	119
5.4	Detection accuracy across telemetry approaches. (a) In single-anomaly scenarios, KESTREL achieves higher AUPRC than coarse counters and selective sampling. (b) Under mixed anomaly workloads, KESTREL maintains strong detection performance, demonstrating robustness to concurrent anomaly types.	130
5.5	Detection accuracy vs telemetry cost (Pareto tradeoff). KESTREL with $w=512, d=3$ achieves AUPRC 0.81 at approximately 6 Mbps export rate, providing roughly $2\times$ higher accuracy than <i>3GPP-PM</i> and an order-of-magnitude lower telemetry volume than Δ - <i>SMP</i>	131
5.6	Responsiveness measured by median time-to-first-detection (TTFD). KESTREL achieves sub-second detection latency while maintaining the highest F1 score among evaluated schemes. The table reports per-anomaly breakdowns, showing that KESTREL detects short-lived events (e.g., microbursts) rapidly and consistently outperforms coarse counters.	132

5.7	Per-window pipeline latency. Median (p50) and interquartile range (IQR) of e2e processing time per 1 s window, with time breakdown across ingestion (I), feature extraction (F), and ML detection (ML).	133
5.8	Accuracy across binning strategies. Adaptive target-occupancy binning yields consistently higher Area Under the Precision–Recall Curve (AUPRC) across anomaly types compared to other binning strategies.	134
5.9	Transient response under sustained distribution drift (QID 6). Adaptive re-binning events (vertical dotted lines) counteract drift-induced accumulation in the diagnostic region, restoring occupancy toward the target ρ . Static binning leads to sustained saturation under the same drift episode. .	135
5.10	Aggregate control accuracy across queues. Mean absolute deviation from the target occupancy ρ under static and adaptive binning. Adaptive re-binning reduces deviation by an average of 87% across 8 queues.	136
5.11	Threshold sensitivity analysis. Online diagnostic thresholds selected by KESTREL compared to oracle thresholds computed from raw traces (log–log scale). The strong correlation ($r = 0.75$) indicates that adaptive re-binning preserves sparsity without collapsing the diagnostic region into the tail. . .	137
5.12	Benefit of budget-aware width allocation. (a) QIDs exhibit highly heterogeneous width requirements due to differences in traffic volume and diagnostic occupancy. (b) Under constrained budgets, optimized allocation significantly lowers the collision-induced noise floor (up to 38%), while the gap diminishes once sufficient memory is available.	139
A.1	e2e 5G experimental testbed. The platform spans a Radio Access Network (RAN), a Tofino-based programmable transport network, and a cloud-native 5G core. Telemetry from the RAN, transport, and core is aggregated in the monitoring cluster, while the compute server runs the telemetry optimizer (SLICESCOPE, Chapter 4) and anomaly detection pipeline (KESTREL, Chapter 5). The multi-hop transport topology is emulated on a UfiSpace S9180-32X Tofino switch [120] using external loopbacks.	170

List of Tables

2.1	Synthesis of related work for the network slice monitoring. While prior work provides individual mechanisms for scalability and precision, they lack the vertical integration and slice-centric approach needed for slice SLA monitoring.	32
3.1	Gap analysis of existing 5G monitoring architectures. Comparison of prior systems with MONARCH across key design requirements for e2e slice monitoring. Legend: supported (✓), not supported (✗), partial (◐).	38
3.2	Prototype realization of MONARCH’s components. Each logical module is realized using commodity cloud-native technologies; the mapping reflects an implementation choice rather than an architectural constraint.	52
3.3	Impact of generation vs. ingestion interval on CPU usage. Increasing the ingestion interval provides an effective reduction in CPU consumption.	63
3.4	Slice-level labeling reduces query latency. Session-level labels require slice-scoped queries to aggregate over many network-function/session series, whereas MONARCH exposes pre-resolved slice-level labels.	65
3.5	MONARCH’s adaptive monitoring achieves better accuracy–overhead trade-off compared to commonly used fixed-frequency. Lower error (MAE, MAPE) and competitive data savings ratio (DSR) result in the highest expected utility (EU).	66
4.1	SLA-aware multi-slice joint resource allocation	80
4.2	Comparison of representative telemetry approaches against data plane requirements (§4.3.3) and closed-loop control challenges (§4.3.1 and §4.3.2). Only SLICESCOPE satisfies all criteria. ✓ supported, ✗ not supported, ◐ partial support.	85

4.3	Example execution of change-triggered telemetry on a two-hop path ($s_1 \rightarrow s_2$) with threshold $\Delta = 2$. For each packet, the switch computes the local metric L_{curr} , updates the e2e estimate E_{curr} , and inserts telemetry only when the deviation $ E_{curr} - E_{rep} $ exceeds Δ . The example illustrates selective insertion, state updates, and bounded per-hop error under skipped reports.	90
4.4	Summary of slice types and workloads used in the evaluation. Each slice type reflects a class of applications with similar SLA characteristics; many instances of each slice type are generated, each with its own SLA within the specified range. Bandwidth values refer to typical per-user usage; total slice bandwidth is computed as: (App. BW) $\times$ (#users per slice), which is 3 – 10 (Ultra-Reliable Low-Latency Communications (URLLC)), 10 – 20 (Enhanced Mobile Broadband (eMBB)), and $\sim 10,000$ (Massive Machine-Type Communications (mMTC)). Representative applications include industrial automation (URLLC), UHD video streaming (eMBB), and sensor telemetry (mMTC). Details on workload generation are discussed in §4.5.1.	95
4.5	Tofino hardware resource utilization under different bucket-array configurations. For the selected configuration ($d = 2, w = 4096$), SLICESCOPE consumes 12.5% of hash units, 27.1% of SALUs, and 9.4% of SRAM. . . .	104
5.1	Anomaly scenarios used in our evaluation. Single anomalies (1–4) and mixed anomalies (5).	126
5.2	Telemetry approaches evaluated in our testbed.	128
A.1	Hardware components of the experimental testbed. The monitoring cluster comprises two monitor nodes and one collector node. The transport switch implements a multi-hop topology via external loopbacks. All server nodes run Ubuntu 22.04. Chapters 3–5 are evaluated on this shared infrastructure.	171

List of Acronyms

- 3GPP** 3rd Generation Partnership Project 5, 8, 26, 35, 45, 52, 70, 96, 109, 112, 125, 140, 141, 153, 171
- 5G** Fifth Generation v, xv, 1–3, 5, 6, 10–12, 16, 22, 23, 25–31, 33–38, 43, 49–52, 56–58, 70–72, 76, 77, 94–96, 103, 109, 111, 114, 125, 141, 143, 144, 148, 150, 153, 154, 172, 173
- 5QI** 5G QoS Identifier 53
- AI/ML** Artificial Intelligence/Machine Learning 25, 30, 31, 49, 117, 125
- ALU** Arithmetic Logic Unit 92, 104, 115, 127, 134, 140
- AMF** Access and Mobility Management Function 16, 50, 171
- API** Application Programming Interface xv, 23, 24, 28, 36, 40–42, 45–49, 51–53, 57, 58, 69, 71, 126, 127, 144
- AR** Augmented Reality v, 39
- ASIC** Application-Specific Integrated Circuit 20, 21, 92, 105, 115, 116, 125, 127, 140, 151
- AUPRC** Area Under the Precision–Recall Curve xviii, 12, 129–131, 134
- CDF** Cumulative Distribution Function 98, 122
- CMS** Count-Min Sketch xvii, 22, 99, 100, 110, 115–120, 123, 126, 140, 174
- CNI** Container Network Interface 171
- CU** Centralized Unit 17, 109, 111, 141

DCAE Data Collection, Analytics, and Events 42

DDoS Distributed Denial of Service 22

DKW Dvoretzky–Kiefer–Wolfowitz 122

DU Distributed Unit 17

e2e end-to-end v, xiii, xvi, xviii–xx, 1–6, 8–10, 12, 13, 16, 18, 23, 25, 26, 28, 32, 34–36, 38, 58, 66, 70, 71, 73–76, 79, 83–85, 88–91, 93–95, 98–100, 103, 105, 106, 108, 133, 140, 144–147, 153, 169, 170, 172

eBPF Extended Berkeley Packet Filter 25, 141

eMBB Enhanced Mobile Broadband xx, 1, 7, 27, 73, 77, 94, 95

gNB Next Generation Node B 50, 103, 128, 170–173

GTP GPRS Tunneling Protocol 96, 103

GTP-U GPRS Tunneling Protocol – User Plane xiv, 17, 30, 52, 109, 111, 117, 125, 128, 173

IAT Inter-Arrival Time xvii, 110, 113–118, 120, 124–126, 133, 134

ILP Integer Linear Program xvi, 79, 81, 82, 100, 101, 106, 125, 138

INT In-Band Network Telemetry 7, 20–22, 28, 29, 36, 38, 43, 74–76, 81, 83–86, 88, 92–96, 98, 99, 105, 106, 126, 169, 172

IoT Internet of Things 112, 123, 126

IQR Interquartile Range 133

JSD Jensen–Shannon Divergence 135

KPI Key Performance Indicator v, xiii, xv, 4–6, 9, 10, 12, 13, 17–19, 23–26, 28, 29, 32, 34–49, 51–58, 66, 67, 69–73, 144–147, 152, 153, 172

MAD Mean Absolute Deviation 136

MAPE Monitor–Analyze–Plan–Execute 2, 5, 34, 45

MDE Monitoring Data Exporter xv, 43, 44, 48, 50–56, 58–60, 62–64, 68–71

ML Machine Learning 142

mMTC Massive Machine-Type Communications xx, 1, 77, 79, 94, 95, 98

MPLS Multiprotocol Label Switching 96

MTU Maximum Transmission Unit 88, 92

NBI Northbound Interface 42

NF Network Function xv, 9, 23–25, 30, 31, 35–38, 42–44, 46, 47, 50–54, 58, 60–62, 64, 68, 70, 71, 153

NFVO Network Function Virtualization Orchestrator 43

NSS Network Slice Segment 42, 43

NSSDC Network Slice Segment Data Collector xv, 43–45, 47, 48, 50, 51, 59, 60, 63, 64, 66, 68, 69

NST Network Slice Template 45

NWDAF Network Data Analytics Function 26, 31, 70, 141

ONAP Open Network Automation Platform 42, 56

P4 Programming Protocol-Independent Packet Processors 11, 20, 21, 172

PISA Protocol-Independent Switch Architecture 20

PRB Physical Resource Block 128

QER QoS Enforcement Rule 111, 113

QFI QoS Flow Identifier xvii, 17, 19, 37–39, 46, 53, 61, 62, 64, 109, 111–113, 115, 117, 125, 126, 128, 129, 145–147, 152, 154, 173

QID Queue Identifier xvii, 110, 115–118, 122–125, 127, 131, 134, 136, 138–141

QoS Quality of Service v, 7, 8, 11, 13, 17, 18, 27, 29, 31, 39, 46, 53, 77, 109, 111, 112, 115–117, 126, 140, 142, 146, 150–152, 173

RAN Radio Access Network xviii, 2, 5, 11, 23, 25, 26, 31, 35, 37, 43, 51, 70, 71, 141, 170, 172, 173

RCA Root Cause Analysis 152, 153

RL Reinforcement Learning 25, 56

S-NSSAI Single Network Slice Selection Assistance Information 16, 17, 37, 52, 53, 146, 152, 153

SALU stateful Arithmetic Logic Unit 20, 21

SBA Service-Based Architecture 17, 35

SDR Software-Defined Radio 50, 103, 128, 171, 173

SLA Service Level Agreement v, x, xiii, xvi, xix, xx, 1, 3–13, 15, 17–21, 23, 26–29, 32, 34–36, 39, 40, 69, 72–79, 81–85, 87, 89, 91, 93–98, 100, 102, 104–108, 111, 140, 144–149, 151, 152, 173

SMF Session Management Function 17, 24, 38, 39, 47, 50, 53, 58, 62, 141, 171

SMO Service Management and Orchestration 43

SRAM Static Random-Access Memory 21, 28, 31, 112, 123, 124, 134, 138, 140

SRv6 Segment Routing over IPv6 96

TEID Tunnel Endpoint Identifier xiv, xvii, 17–19, 30, 37–40, 53, 64, 96, 109–115, 117, 118, 125, 126, 128, 140, 142, 145, 147, 152–154, 173

TPC Telemetry Primitive Contract 76, 78, 82–84, 88, 105–107, 145

trTCM Two-Rate Three-Color Marker 113, 117, 125, 172

TTFD Time to First Detection 132

UE User Equipment 50, 58, 103, 111, 126, 171–173

UPF User Plane Function xiv, 3, 7, 17, 24, 30, 37–39, 42, 44, 47, 50, 52, 53, 58, 61, 62, 109–112, 115–117, 123–125, 127, 128, 131, 134, 140, 141, 146, 147, 153, 169, 171, 172

URLLC Ultra-Reliable Low-Latency Communications xx, 1, 7, 18, 19, 21, 27, 28, 73, 77, 79, 82, 83, 94, 95, 98

V2X Vehicle-to-Everything 39

VNF Virtualized Network Function 1, 2, 56, 71

VoIP Voice over Internet Protocol 126

VXLAN Virtual eXtensible Local Area Network 172

WRR Weighted Round Robin 113, 125

XDP eXpress Data Path 141

XGBoost eXtreme Gradient Boosting 126

Chapter 1

Introduction

Fifth Generation (5G) and beyond mobile networks are expected to support a diverse range of services, including Enhanced Mobile Broadband (eMBB) (e.g., high-bandwidth video), Ultra-Reliable Low-Latency Communications (URLLC) (e.g., vehicular automation), and Massive Machine-Type Communications (mMTC) (e.g., smart-city deployments). These services impose fundamentally different requirements on the network in terms of throughput, latency, reliability, and connection density [3]. For example, URLLC slices are expected to provide very low loss rates (e.g., $<0.001\%$) and end-to-end (e2e) latency (e.g., 1–5 ms) while having relatively flexible bandwidth requirements (e.g., 1–10 Mbps) [7, 9, 10]. In contrast, eMBB slices have higher bandwidth requirements (e.g., up to 1000 Mbps) but tolerate higher packet loss (e.g., $<1\%$) and e2e latency (e.g., 10–50 ms) [7, 9, 10]. Supporting such heterogeneous service objectives on a shared physical infrastructure requires a fundamental departure from the best-effort design principles used for earlier generations of mobile networks.

Network slicing. Network slicing has emerged as the primary abstraction for addressing this challenge. Slicing partitions a shared physical network into multiple logical networks (slices), allowing operators to tailor each slice to its target service through resource allocation and performance guarantees [3]. A slice is realized by chaining together Virtualized Network Functions (VNFs) and the virtual links interconnecting them, orchestrated to satisfy a specific service objective. Modern deployments may support dozens to hundreds of concurrently active slices, each governed by distinct Service Level Agreements (SLAs) and associated with different lifetimes and traffic characteristics [98, 99]. Network slicing transforms the network from a monolithic system into a collection of coexisting, independently managed virtual networks.

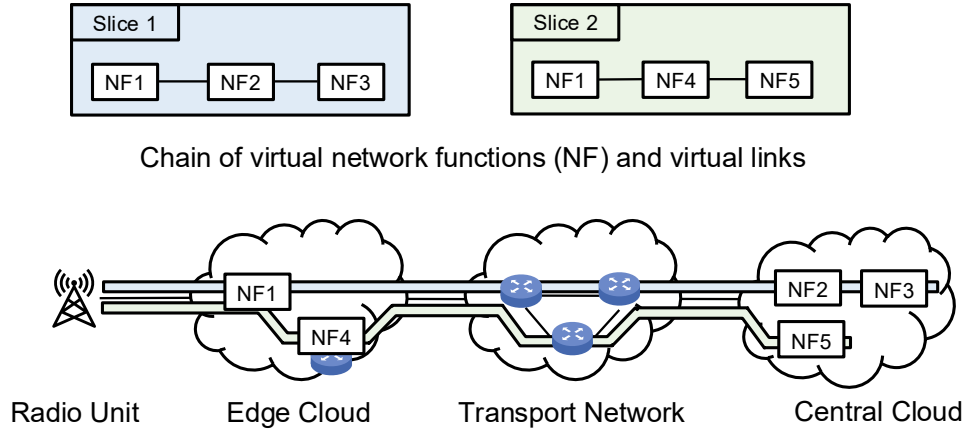


Figure 1.1: **Conceptual mapping of logical network slices to shared physical infrastructure.** Each slice consists of a chain of virtual network functions (VNFs) and the virtual links interconnecting them, instantiated across heterogeneous domains, such as the Radio Unit, Edge Cloud, Transport Network, and Central Cloud.

Figure 1.1 grounds this abstraction by illustrating how logical slices are realized as an e2e systems spanning the entire network from the Radio Access Network (RAN) to the 5G Core. Each slice is instantiated by placing and configuring VNFs and their associated virtual links across heterogeneous domains, including radio, transport, edge, and central cloud resources, based on the slice’s service requirements. While slices are logically isolated, they may share underlying physical resources such as transport links, compute nodes, and hardware queues. This resource sharing creates fundamental challenges for performance assurance and diagnosis, since slice-level behavior emerges from interactions across domains, timescales, and planes.

Closed-loop automation depends on monitoring. To address the growing complexity of network management, both industry and standardization efforts increasingly rely on closed-loop automation. In such systems, monitoring data feeds analytics and decision-making logic, which in turn drives control actions such as resource scaling, reconfiguration, or fault mitigation. A commonly used conceptual model for such network automation is the Monitor–Analyze–Plan–Execute (MAPE) loop (Figure 1.2) which organizes the control logic as a sequence of measurement, reasoning, decision, and actuation stages. In sliced networks, the monitoring stage is especially critical: it determines which aspects of each

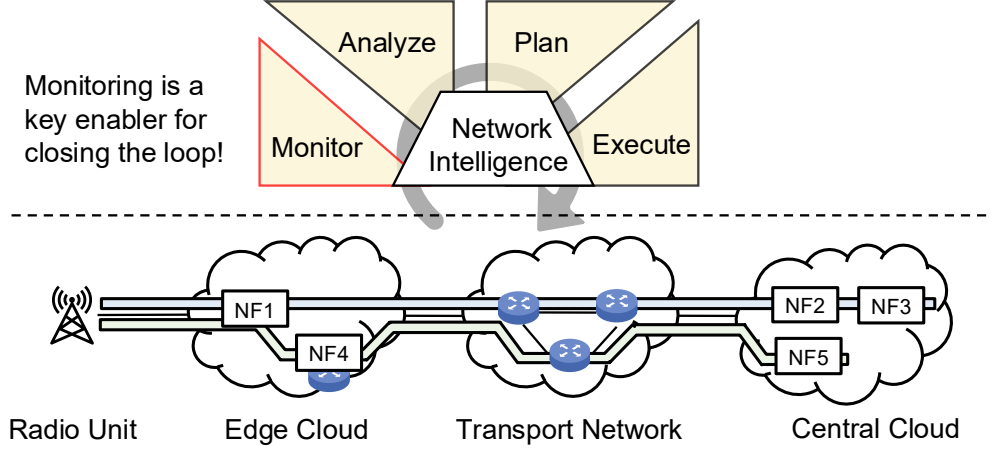


Figure 1.2: **The role of monitoring within the Monitor–Analyze–Plan–Execute (MAPE) loop.** High-fidelity monitoring provides the critical input for network intelligence to maintain slice-specific SLAs.

slice are visible to the intelligence layer and at what fidelity and timescale. Thus, monitoring shapes the effectiveness of all downstream control decisions.

Core thesis lens: telemetry as the limiting factor. In closed-loop network slice automation, *telemetry is the bottleneck*: control can only optimize what it can observe, and observation is constrained by strict overhead, latency, and deployability limits. This creates a fundamental tension in sliced networks: operators need *SLA-relevant, e2e, fine-grained visibility* to detect transient violations, yet must deliver it *efficiently and at scale* across many heterogeneous slices in a shared infrastructure. To this end, this dissertation asks: *How can slice monitoring be designed and controlled to maximize SLA-relevant visibility under strict resource budgets?*

Why monitoring is fundamentally harder for slices. Monitoring requirements for network slices differ fundamentally from those in traditional networks. SLAs for slices are defined over *e2e performance metrics* such as throughput, latency, jitter, and loss [96]. At the same time, slice workloads can rapidly change due to traffic bursts, mobility, and orchestration actions. Consequently, monitoring systems must provide *high-precision visibility* into *e2e performance* to capture transient anomalies, yet remain *efficient* enough to scale with dynamic workloads. Coarse-grained approaches (*e.g.*, periodic polling) can hide short-lived performance degradations that impact real-time slice behavior. For instance, consider the 5G user plane, where a User Plane Function (UPF) multiplexes traffic from

many logical flows onto shared forwarding and queuing resources. In such settings, a coarse 1 second monitoring interval can hide microbursts in which the 95th percentile latency briefly exceeds tens of milliseconds, even while slice-level average metrics appear to satisfy SLA targets [97].

Existing solutions force a difficult trade-off. Fine-grained monitoring mechanisms such as per-packet telemetry [15, 59, 119] or event-based reporting [54, 75, 133] can detect microbursts, but applying them uniformly is computationally and bandwidth-prohibitive at scale. On the other hand, compact summaries (including sketch-based approaches [127, 128]) reduce overhead but can degrade accuracy when configured statically and slice-agnostically, or when workload dynamics and traffic drift violate implicit assumptions. Finally, relying on traditional coarse-grained monitoring approaches [74, 82] can avoid these costs but obscures transient behaviors necessary for ensuring slice SLA compliance. This trilemma is exacerbated when considering network slices, because monitoring must not only be precise and efficient, but also adaptable to slice-specific SLAs that vary across services and evolve over time [96].

A cross-layer diagnosis. The challenges above reflect a cross-layer monitoring problem that cannot be resolved at any single layer of the monitoring stack:

- *Architectural level.* Monitoring for slices is fragmented across heterogeneous domains and control plane services, obscuring e2e visibility. Existing abstractions primarily expose metrics at the level of individual network functions or infrastructure components, rather than directly providing slice-level Key Performance Indicators (KPIs) defined over e2e behavior.
- *Monitoring control level.* Static or infrastructure-centric policies, as used in existing works, allocate monitoring resources uniformly, and thus fail to account for heterogeneous slice SLAs and evolving workload dynamics.
- *User-plane level.* Transient anomalies that can affect slice SLAs are either masked when using coarse aggregation or exposed at prohibitive cost when using high-fidelity telemetry. This severely limits practical observability and attribution of SLA-relevant events.

The challenges mentioned above compound across layers. Architectural fragmentation limits what monitoring systems can observe; rigid control policies force a choice between visibility gaps and prohibitive overhead; and insufficient user-plane visibility undermines e2e anomaly attribution. Efficient and high-precision slice monitoring, therefore, requires a holistic approach that jointly addresses architectural visibility, monitoring control, and data plane observability.

THESIS STATEMENT

This dissertation addresses the challenge of achieving efficient high-precision monitoring of network slices by decomposing it along three complementary dimensions: architectural abstraction, SLA-aware fidelity control, and fine-grained user-plane attribution.

From diagnosis to research questions. The cross-layer diagnosis above points to three coupled design problems: (i) exposing slices as first-class monitoring entities and computing e2e slice KPIs across heterogeneous domains, (ii) dynamically regulating monitoring fidelity under heterogeneous SLAs and evolving network conditions, and (iii) detecting and attributing transient user-plane anomalies under tight data plane constraints. These problems give rise to the research questions that guide the remainder of this thesis.

1.1 Challenges in Monitoring Network Slices

In this section, we articulate these challenges in more detail, structuring them along the layers of the monitoring stack and formulating the corresponding research questions.

1.1.1 End-to-End Slice Visibility Across Domains

To support monitoring-driven decision making in sliced networks, monitoring systems must expose slice-level performance in a form that higher-level applications can directly consume (*cf.* the MAPE loop in Figure 1.2). While slicing enables operators to define service-level objectives at slice granularity, existing monitoring infrastructures largely expose low-level, domain-specific metrics. Bridging this gap requires rethinking how slices are represented and surfaced within the monitoring architecture.

Architectural abstraction gap. As shown in Figure 1.1, network slices are e2e and span multiple technological domains, including the RAN, transport network, edge cloud, and 5G core [39]. While each domain exposes its own monitoring interfaces, data formats, and timescales, SLA for slices are defined over *e2e slice KPIs*, such as throughput, latency, and reliability, as specified by 3rd Generation Partnership Project (3GPP) [1]. These KPIs are composite: they require the collection and correlation of measurements across multiple network functions and infrastructure components constituting the slice.

However, many monitoring solutions remain domain- or infrastructure-centric, focusing on individual functions or resources rather than treating the slice itself as a *first-class monitored entity* [40, 84]. As a result, operators and higher-level management applications lack a unified and explicit view of slice performance, and e2e KPIs must be inferred indirectly from fragmented measurements.

Decoupling monitoring from control logic. This fragmentation complicates monitoring-driven control. Applications for slice scaling [108], admission control [109], or SLA enforcement must either re-implement KPI aggregation themselves or rely on ad-hoc correlations across heterogeneous sources. This tight coupling hinders modularity, reuse, and evolvability of closed-loop control functions, and obscures monitoring as a shared service.

Impact of cloud-native deployments. These challenges are compounded in cloud-native 5G deployments, where control plane and user-plane functions are decomposed into microservices that can scale and migrate independently [14, 69]. While beneficial for elasticity, this decomposition fractures the monitoring context: migration and scaling obscure the linkage between resource-level events and slice-level performance. Consequently, architectures that remain at the level of individual functions or resources lack the context required to disentangle slice-specific behavior under dynamic conditions.

Research Question RQ1 (*Chapter 3*)

How can a monitoring architecture abstract network slices as first-class entities and expose slice-level KPIs through a unified interface, enabling control applications to reason about e2e slice performance across heterogeneous domains?

1.1.2 SLA-Aware Control of Monitoring Fidelity

Once slices are exposed as first-class entities (§RQ1), monitoring-driven applications can specify slice-level monitoring intents, such as continuous visibility into e2e latency or loss. These intents must be translated into concrete actions: which metrics to collect, where to collect them, and at what fidelity. While slice-centric abstraction addresses *what* to monitor, it leaves open a fundamental question of *how precisely* each slice should be monitored.

Heterogeneous SLAs and monitoring demands. Operational networks may host dozens to hundreds of concurrently active slices, each governed by distinct SLAs over

metrics such as throughput, latency, jitter, and loss [12, 13, 23]. These SLAs impose fundamentally different monitoring requirements. URLLC slices require fine-timescale, high-fidelity monitoring to detect transient latency or loss violations, whereas eMBB or best-effort slices can tolerate coarser visibility [7, 8, 10]. Applying fixed, uniform configurations across such heterogeneous slices leads to systematic inefficiencies: either excessive overhead from over-monitoring benign slices, or insufficient visibility for slices with stringent Quality of Service (QoS) requirements.

This challenge is compounded by dynamics: the set of active slices, their traffic characteristics, and their priorities can change rapidly due to slice churn, workload variation, and orchestration actions. A configuration that is well-tuned at one moment can become too costly or too imprecise shortly thereafter. Fine-grained primitives (e.g., In-Band Network Telemetry (INT)) and compact distributional summaries (e.g., sketches) provide powerful visibility mechanisms [15, 119, 127, 128], but their effectiveness depends critically on run-time parameterization: over-provisioning incurs prohibitive overhead, while conservative configurations risk missing short-lived anomalies or degrading accuracy under drift and collisions [96].

The fidelity–overhead control challenge. Prior adaptive telemetry mechanisms are often infrastructure-centric and do not explicitly reason about slice-level semantics, SLA objectives, or global monitoring budgets. This limits their ability to allocate monitoring resources in proportion to heterogeneous requirements and evolving priorities. Addressing these limitations requires mechanisms that dynamically regulate monitoring fidelity across slices in an SLA-aware, budgeted, and robust manner.

Research Question RQ2 (*Chapter 4*)

How can monitoring fidelity be dynamically regulated across slices with heterogeneous and evolving SLAs, ensuring high-precision visibility for critical services while strictly adhering to global resource budgets?

1.1.3 Fine-Grained Detection and Attribution in the User Plane

Once slice-level visibility is established (§RQ1) and monitoring fidelity is regulated (§RQ2), a remaining challenge lies in the user plane, where many SLA violations originate. Data plane components such as the UPF aggregate traffic from multiple slices and flows onto shared resources, making them critical points for detecting QoS degradations that threaten

slice SLAs [3, 7]. However, user-plane operation imposes fundamental constraints on what can be observed and attributed at scale.

Transient anomalies and distributional visibility. Many SLA violations are caused by short-lived phenomena such as microbursts, queue oscillations, and transient congestion. Detecting these events requires *distributional* visibility (latency tails and sub-second fluctuations), rather than coarse averages or long-term aggregates. Standardized 3GPP user-plane counters are typically polled at coarse granularity and expose only aggregate statistics, which can mask brief but severe latency excursions that violate slice SLAs [2]. Such transient behaviors are invisible to low-frequency polling, yet are precisely the events that impact latency-sensitive services.

Attribution under shared data plane resources. Effective remediation further depends on attribution: identifying which slice, flow, or tunnel is responsible for an observed degradation. In practice, data plane resources such as hardware queues and on-chip memory are shared across many tunnels and slices, entangling performance effects. Per-packet telemetry (e.g., postcard-style export) can expose fine-grained behavior but is impractical at production line rates due to bandwidth and processing cost [15, 78]. Sketch-based telemetry offers a promising alternative by summarizing behavior under strict constraints [63, 127], but introduces trade-offs in accuracy, parameter sensitivity, and attribution fidelity when state is shared across flows and slices. Resolving these trade-offs is essential for practical high-precision user-plane monitoring at scale [97].

Research Question RQ3 (*Chapter 5*)

How can we achieve high-fidelity visibility into transient QoS anomalies (e.g., microbursts), and attribute these anomalies to specific slices, without incurring the prohibitive overhead of per-packet monitoring?

1.2 Thesis Contributions

This dissertation develops a cross-layer framework for efficient, high-precision monitoring of network slices in 5G and beyond networks. Figure 1.3 provides a structural overview of the dissertation and illustrates how the contributions align across the monitoring stack. Guided by the research questions above, the contributions span the monitoring stack—from architectural abstractions for e2e slice visibility, to SLA-aware control of monitoring fidelity,

to lightweight user-plane telemetry mechanisms for detecting and attributing transient performance anomalies.

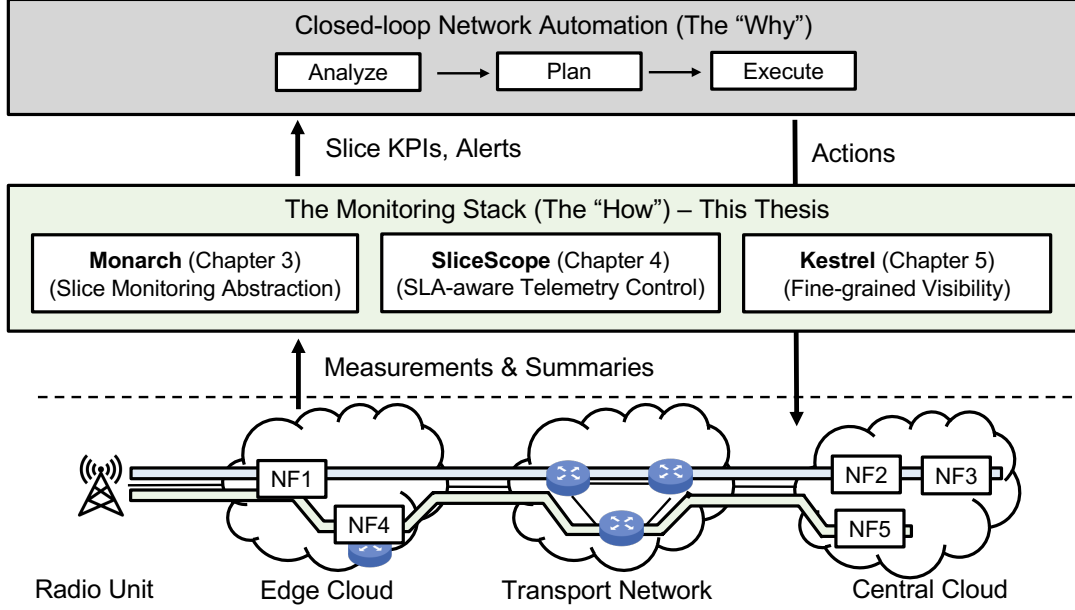


Figure 1.3: **Thesis overview and scope.** The monitoring stack as the foundational enabler for closed-loop network automation, defining the scope of this thesis. Chapters 3–5 develop complementary contributions to slice monitoring: MONARCH provides slice-level monitoring abstractions, SLICESCOPE regulates telemetry fidelity under heterogeneous SLAs, and KESTREL enables fine-grained user-plane visibility for detecting transient performance anomalies. Together, these components provide the monitoring foundation required to drive automated data-driven management of network slices.

1.2.1 MONARCH: Architecture for Slice-Level Monitoring

This contribution addresses Research Question RQ1 and is developed in Chapter 3. Prior monitoring systems are infrastructure- and Network Function (NF)-centric: the slice, though the unit at which SLAs are defined and resources provisioned, is not itself a monitored entity, so e2e slice KPIs must be reconstructed downstream by every consuming application. We introduce MONARCH, a monitoring architecture that elevates network slices to first-class monitored entities, enabling explicit reasoning about e2e slice performance across heterogeneous domains. The key contributions are as follows:

- We present a scalable, cloud-native monitoring architecture that abstracts slices as first-class entities and enables direct computation of e2e slice KPIs from distributed measurements.
- We design a unified, slice-centric monitoring interface that decouples monitoring-driven applications from domain-specific telemetry sources, improving modularity, reuse, and evolvability within closed-loop control systems.
- We demonstrate the practicality and scalability of MONARCH on the shared 5G testbed described in Appendix A, showing that slice-level monitoring scales to increasing numbers of slices with predictable overhead and bounded ingestion latency.

1.2.2 SLICESCOPE: SLA-Aware Network Slice Monitoring

This contribution addresses Research Question RQ2 and is developed in Chapter 4. Prior adaptive telemetry mechanisms are infrastructure-centric and SLA-agnostic: they tune monitoring parameters without reasoning about slice-level objectives or a shared monitoring budget, and so cannot allocate fidelity in proportion to heterogeneous, evolving SLAs. We introduce SLICESCOPE, a principled framework for dynamically regulating monitoring fidelity across slices with heterogeneous SLAs. By explicitly reasoning about the accuracy–overhead trade-off under global budgets, SLICESCOPE provides high-precision visibility for critical slices while operating within strict resource constraints. The key contributions are as follows:

- We formalize slice monitoring as a closed-loop resource allocation problem, enabling systematic reasoning about accuracy–overhead trade-offs under global budgets.
- We identify the minimal data plane properties required to support SLA-aware monitoring control, enabling monitoring fidelity to be regulated in accordance with heterogeneous SLAs and evolving traffic dynamics.
- We validate SLICESCOPE on the programmable-switch component of the shared testbed, demonstrating substantial reduction in violations of per-slice monitoring-error tolerances for critical slices.

1.2.3 KESTREL: Lightweight Telemetry for Network Slice User-Planes

This contribution addresses Research Question RQ3 and is developed in Chapter 5. Existing user-plane telemetry is flow-aggregate or volume-oriented, exposing per-class counters or heavy-hitter summaries with no attribution of anomalies to the slice traffic responsible. A QoS degradation is visible in aggregate but its source cannot be isolated. We introduce KESTREL, a lightweight user-plane telemetry mechanism for detecting and attributing transient QoS anomalies in shared data plane resources. By capturing distributional performance signals under strict memory and bandwidth constraints, KESTREL enables fine-grained anomaly detection without the overhead of per-packet monitoring. The key contributions are as follows:

- We characterize the telemetry requirements of slice-level anomaly detection in 5G user planes, showing that representative anomalies manifest as distributional signals such as latency tails, sub-second microbursts, and inter-arrival irregularities.
- We design a detectability-driven sketch-based telemetry mechanism that captures per-flow or per-tunnel distributional behavior without maintaining explicit per-flow state, enabling deployment within the constraints of hardware-accelerated data planes.
- We provide formal guarantees and empirical validation on the programmable user-plane component of the shared testbed, showing that accurate detection and attribution of transient anomalies is achievable with orders-of-magnitude lower telemetry overhead than packet-level monitoring approaches.

Together, these contributions demonstrate that efficiency and high precision in network slice monitoring are not mutually exclusive. Rather, they can be jointly achieved through cross-layer design that aligns architectural abstractions, control plane intelligence, and user-plane telemetry mechanisms to support scalable, SLA-aware operation of 5G and beyond networks, as summarized in Figure 1.3.

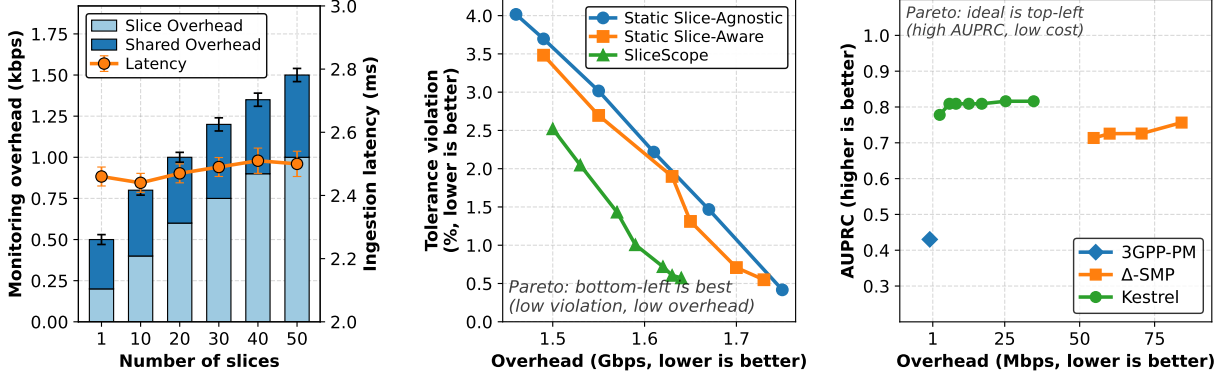
Experimental foundation. The systems developed in this dissertation are implemented and evaluated on a common experimental substrate for slice-aware 5G monitoring, described in Appendix A. The testbed integrates a cloud-native 5G core, RAN simulation and emulation, programmable transport on Intel Tofino, monitoring infrastructure, and trace-driven traffic generation into a single slice-aware deployment. Realizing this substrate required per-slice core-network instantiation, line-rate Programming Protocol-Independent Packet Processors (P4) data planes, and hardware QoS enforcement. It is therefore a

supporting contribution of the thesis: it provides the practical environment needed to validate slice-level KPI reconstruction, SLA-aware telemetry control, and fine-grained user-plane anomaly detection under consistent deployment assumptions. The software developed for this work is released as six public repositories spanning the full stack, from testbed provisioning and identifier-controlled traffic generation to the three system prototypes (MONARCH, SLICESCOPE, and KESTREL); Appendix A provides the complete list. Each chapter activates a different subset of the platform, but the shared testbed connects the three contributions experimentally and grounds the design choices in deployable 5G system constraints.

1.3 Summary of Results

Figure 1.4 provides a consolidated view of the dissertation’s empirical findings, illustrating how each contribution improves scalability, cost–accuracy trade-offs, or anomaly detectability within its respective layer of the monitoring stack.

- **Slice Monitoring Abstraction (MONARCH).** *Key result:* MONARCH shows that e2e slice-level KPI monitoring is feasible at realistic scales. Monitoring overhead grows predictably with the number of slices, exhibiting linear scaling in slice-specific components while shared monitoring components remain largely constant; the overhead growth factor further reduces by roughly $3\times$ when moving from fine-grained (1 s) to coarser (10 s) monitoring intervals. Ingestion latency remains bounded within $< 3\text{ ms}$ as the number of slices increases, well within the requirements of real-time control loops. We revisit these results in detail in Chapter 3.
- **SLA-aware fidelity control (SLICESCOPE).** *Key result:* SLICESCOPE shows that allocating a fixed telemetry budget using SLA context yields consistently better cost–accuracy trade-offs than static monitoring. Across realistic workloads, SLICESCOPE reduces violations of per-slice monitoring-error tolerances by up to $\sim 2.3\times$ at comparable overhead relative to both slice-agnostic and static slice-aware baselines, by dynamically shifting monitoring fidelity toward more critical slices. These results are examined in detail in Chapter 4.
- **User-plane anomaly detection (KESTREL).** *Key result:* KESTREL demonstrates that accurate user-plane anomaly detection can be achieved with substantially lower telemetry cost by grounding sketch parameterization in detectability. In our evaluation, KESTREL achieves Area Under the Precision–Recall Curve (AUPRC) 0.7–0.82 with only



(a) **MONARCH** (Ch. 3): Slice monitoring shows predictable monitoring overhead and bounded ingestion latency. (b) **SLICESCOPE** (Ch. 4): SLA-aware telemetry allocation improves cost-accuracy trade-offs. (c) **KESTREL** (Ch. 5): Detectability-driven telemetry enables accurate anomaly detection at lower cost.

Figure 1.4: **Summary of key empirical results across the thesis.** (a) MONARCH establishes scalable e2e slice-level KPI monitoring, showing predictable overhead growth with the number of slices and bounded ingestion latency. (b) Building on this slice-centric visibility, SLICESCOPE demonstrates that allocating telemetry budgets according to SLA requirements yields improved cost-accuracy trade-offs compared to static monitoring. (c) Finally, KESTREL grounds telemetry efficiency in detectability, and shows that principled sketch parameterization enables accurate anomaly detection under fixed memory budgets.

5–10 Mbps export overhead, delivering up to $2\times$ higher accuracy than standards-based counters and up to $10\times$ lower overhead than sampling-based approaches, while remaining robust to traffic drift. We analyze these trade-offs in depth in Chapter 5.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 surveys related work on monitoring architectures, telemetry, and network slicing. Chapter 3 presents MONARCH and its e2e slice monitoring architecture. Chapter 4 presents SLICESCOPE and its SLA-aware telemetry control mechanisms. Chapter 5 presents KESTREL and its sketch-based approach to QoS anomaly detection in the user plane. Chapter 6 concludes this thesis by synthesizing the contributions and discussing limitations and future directions. Appendix A describes the experimental testbed used across the dissertation, including the cloud-native 5G core,

RAN components, programmable transport, monitoring cluster, and traffic-generation infrastructure.

Chapter 2

Background & Related Work

This chapter establishes the technical foundations and surveys related work necessary to contextualize the research questions posed in Chapter 1. While Chapter 1 motivated the need for efficient and high-precision slice monitoring, here we formalize the operational concepts underlying slice realization and monitoring, and examine prior approaches through the lens of the monitoring stack introduced earlier.

We proceed in two stages. First, we review the foundations of slice realization and monitoring, including identifier hierarchies (§2.1.1), the precision–overhead trade-off that defines monitoring fidelity (§2.1.2), and the programmable data planes and telemetry primitives on which high-precision monitoring is built (§2.1.3–2.1.4). Second, we survey related work along the three layers of the monitoring stack: architectures and mechanisms for slice monitoring (§2.2), SLA-aware monitoring control (§2.3), and fine-grained user-plane telemetry (§2.4). Throughout, we highlight recurring gaps that motivate the designs developed in Chapters 3–5.

2.1 Foundations: Slice Realization and Monitoring

Network slicing enables the deployment of multiple logical networks over a shared physical infrastructure. In 5G networks, this abstraction is realized through standardized control- and user-plane mechanisms that distribute slice semantics across multiple identifiers and components. While Chapter 1 motivated slicing from a service and Service Level Agreement (SLA) perspective, this section examines the operational structures underlying slice realization in practice. In particular, we examine how slices are identified and represented

across control and user planes, how architectural disaggregation distributes slice semantics across components, and how these structural properties constrain slice-scoped monitoring. These foundations provide the structural context for the monitoring mechanisms developed in Chapters 3–5.

2.1.1 Operational Identifiers and Slice Realization

Although a network slice is defined as an end-to-end (e2e) logical network spanning different domains (Figure 1.1), in Fifth Generation (5G) networks, its realization is distributed across multiple layers of operational identifiers, as illustrated in Figure 2.1. Each layer refines the slice abstraction for a distinct purpose (*e.g.*, subscription management, QoS differentiation, or packet forwarding), but no single identifier carries complete slice semantics throughout the control and user planes. This structural decomposition directly shapes how slice-level monitoring must be performed.

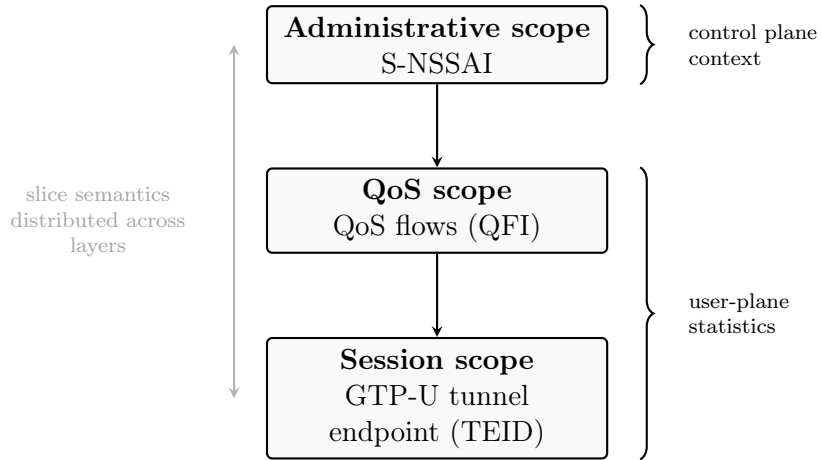


Figure 2.1: **Layering of operational identifiers in 5G slice realization.** S-NSSAI anchors slice identity at the administrative level, QoS flows define per-traffic treatment, and TEIDs index session statistics in the user plane. Because slice semantics are distributed across these layers, slice-level monitoring requires cross-layer correlation.

Administrative scope At the highest level, a slice is identified by the Single Network Slice Selection Assistance Information (S-NSSAI), which serves as the administrative identifier in the control plane, and is used for operations such as slice selection and policy management. Control plane functions such as the Access and Mobility

Management Function (AMF) and Session Management Function (SMF) associate user sessions with a specific S-NSSAI. The S-NSSAI therefore anchors slice identity within the control plane state. However, this identifier is absent in user-plane packet headers at forwarding elements such as the User Plane Function (UPF).

QoS scope Within each slice, user sessions are associated with multiple Quality of Services (QoSs) flows. Each such QoS flow is identified by the QoS Flow Identifier (QFI), and is characterized by performance attributes such as packet delay budget, priority, and packet error rate. These QoS flows allow differentiated treatment of applications sharing the same slice. However, SLAs are typically expressed at slice granularity [29], and thus, their satisfaction depends on the aggregate behavior of QoS flows across sessions in the slice.

Session scope In the data plane, traffic is carried over GPRS Tunneling Protocol – User Plane (GTP-U) tunnels identified by Tunnel Endpoint Identifiers (TEIDs). Each such tunnel may contain multiple QoS flows, depending on the user applications. Data plane operations such as packet processing, queueing, and per-session rate limiting are typically performed at this granularity. TEIDs therefore index per-session forwarding state in the data plane, but they do not explicitly encode slice semantics.

Architectural fragmentation under disaggregation. Beyond these identifiers at different layers, slice monitoring in modern mobile networks is complicated by the use of disaggregated architectures. In 5G and beyond networks, control plane functions, user-plane elements, and radio access components are decomposed into independently managed and scalable units. In the core network, this decomposition follows the Service-Based Architecture (SBA), where functions are implemented as virtualized microservices [3]. Similarly, in virtualized OpenRAN [77] deployments, radio functions are divided across Distributed Units (DUs) and Centralized Units (CUs).

While such disaggregation improves modularity, scalability, and interoperability, it also distributes slice-relevant state across multiple components. Slice membership and policy context may reside in one function (*e.g.*, the SMF), while forwarding counters and QoS statistics are measured elsewhere (*e.g.*, the UPF or the CU), exported through distinct telemetry pipelines and often at different timescales.

Implications for slice monitoring. The combination of identifier layering and architectural disaggregation means that no single component possesses a complete representation of slice behavior. Therefore, slice-level Key Performance Indicators (KPIs) must be reconstructed by correlating control plane context with distributed forwarding statistics. Ultimately, these architectural and operational choices, *i.e.*, where slice state is reconstructed,

how frequently measurements are collected, and how much detail is preserved, directly determine the *monitoring fidelity* (discussed next) available for slice-level management and orchestration.

2.1.2 Monitoring Fidelity: The Precision-Overhead Trade-off

We define *monitoring fidelity* as the trade-off between telemetry detail (precision and scope) and the resource overhead incurred to collect and export the relevant measurements. Slice monitoring needs both control plane state (*e.g.*, slice membership and session context) and data plane performance measurements (*e.g.*, latency, loss, and queue depth). However, these planes often evolve at very different timescales (*e.g.*, session management events occur at 1–10 s [3] while microburst events in the user plane occur at < 1 ms [54]) and expose different abstractions (*e.g.*, administrative scope and session scope; see §2.1.1). This means monitoring fidelity is fundamentally constrained by both cross-plane correlation complexity and telemetry overhead.

As a consequence of the architectural decomposition discussed in §2.1.1, telemetry often becomes the bottleneck in closed-loop network slice automation. Slice management and orchestration requires SLA-relevant, e2e visibility; thus, the measurements to drive such visibility must be collected across different layers and exported under strict resource and latency constraints. Therefore, the monitoring system must constantly balance how precisely slice behavior is observed against the resource cost of that observation.

Monitoring fidelity is crucial in determining whether slice-level KPIs accurately reflect SLA compliance, as well as whether short-lived QoS degradations can be detected and attributed. For example, a Ultra-Reliable Low-Latency Communications (URLLC) slice with a 5 ms latency budget may appear to satisfy its SLA if monitoring reports only 1 s averages; however, short-lived queue buildup can impact a subset of its sessions and violate tail latency targets, without being visible at the reported 1 s averages.

Detecting such events requires high-fidelity monitoring. This might involve, for example, latency measurements at sub-second intervals, maintaining per-TEID state, or preserving distributional summaries (*e.g.*, percentiles). These choices come at a cost: increased export bandwidth, collector processing, or data plane memory consumption. To systematically address these trade-offs, we characterize monitoring fidelity along three dimensions (Figure 2.2), each corresponding to a distinct design axis developed in this thesis.

Spatial fidelity (abstraction and scope) Spatial fidelity refers to the abstraction level at which monitoring entities are represented and measured before being exposed to

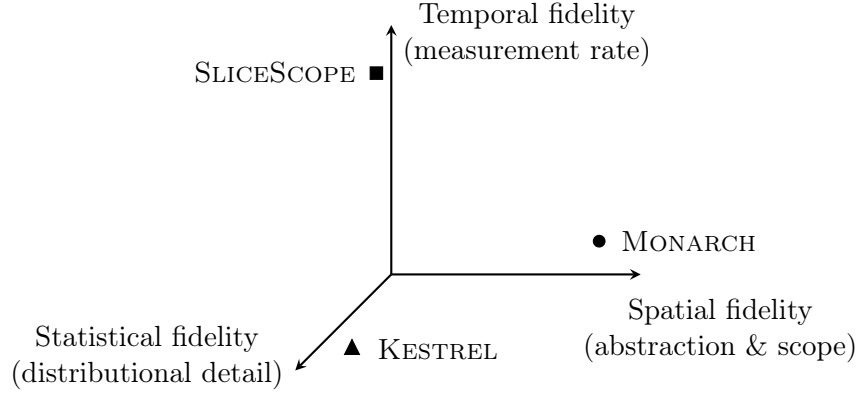


Figure 2.2: **Monitoring fidelity as a configuration space.** Spatial, temporal, and statistical fidelity jointly determine the precision and scope of slice monitoring under resource constraints. The three thesis contributions emphasize complementary axes of this space: MONARCH (Chapter 3) provides monitoring abstractions at the slice level, SLICESCOPE (Chapter 4) allocates measurement rates under global budgets, and KESTREL (Chapter 5) preserves distributional signals under tight data plane constraints.

control applications (e.g., per-slice, per-session, per-TEID/QFI, or per-queue). This abstraction determines how slice-level KPIs are computed and how degradations can be localized across domains.

For slice orchestration and management, control decisions are expressed at slice granularity. However, existing monitoring systems are typically infrastructure-centric: metrics are exported per network function, interface, or physical node. This limits effective spatial fidelity. Chapter 3 addresses this limitation by elevating network slices to first-class monitored entities and encapsulating cross-domain correlation within the monitoring layer, thereby aligning monitoring abstraction with management scope.

Temporal fidelity (measurement rate and allocation) Temporal fidelity denotes how frequently measurements are collected or exported. Low-frequency polling (e.g., 1 s intervals) can mask millisecond-scale microbursts that violate URLLC delay budgets, while high-frequency export increases telemetry overhead.

Because slices exhibit heterogeneous SLA requirements and evolving network conditions, uniform temporal configurations are inefficient. Chapter 4 treats temporal fidelity as a budgeted resource, dynamically allocating measurement rates across slices under explicit global export constraints.

Statistical fidelity (distributional visibility) Statistical fidelity captures the richness and accuracy of the statistics maintained or exported. Mean latency may appear nominal even when the 99th percentile violates SLA targets. Capturing distributional behavior *e.g.*, latency tails or inter-arrival irregularities, requires richer summaries such as percentiles or sketches, often under strict data plane memory limits.

Chapter 5 focuses on this dimension and presents a telemetry design that preserves distributional detectability under tight resource constraints, thereby enabling transient anomaly detection without per-packet monitoring overhead.

2.1.3 Programmable Data Planes

Modern networks are increasingly programmable across both the control plane, through software-defined networking and centralized controllers, and the data plane, through reconfigurable switching hardware. The mechanisms developed in this thesis depend specifically on *data-plane* programmability, which allows operators to define custom packet-processing logic at line rate rather than relying on fixed-function switching Application-Specific Integrated Circuits (ASICs). It is this capability that makes the temporal and statistical fidelity of §2.1.2 achievable in practice: per-packet telemetry insertion and in-network state must execute at line rate, which fixed-function hardware cannot provide. These same mechanisms, however, impose resource constraints that directly shape the data plane designs developed in Chapters 4 and 5.

The P4 pipeline model. Programmable switches are commonly described by the Protocol-Independent Switch Architecture (PISA) [118], in which packets traverse a programmable parser, a sequence of match-action stages, and a deparser, organized into separate ingress and egress pipelines. The behavior of each stage is specified in a high-level language such as Programming Protocol-Independent Packet Processors (P4) [118], which compiles to the target ASIC. Each stage matches packet fields against tables and applies corresponding actions, allowing custom headers (*e.g.*, In-Band Network Telemetry (INT) metadata) to be parsed, modified, and inserted as packets are forwarded. Because all processing occurs within a fixed pipeline traversed once per packet, the amount of computation and state accessible per packet is bounded by the number of available stages.

Stateful memory and its constraints. Telemetry primitives that maintain state across packets, such as the per-slice telemetry counters of Chapter 4 or the sketch buckets of Chapter 5, rely on *stateful memory* embedded in the pipeline. This state is held in register arrays accessed through stateful Arithmetic Logic Units (SALUs), which perform read-modify-write operations at line rate. Two constraints are central to data plane telemetry

design. First, a given register array can typically be accessed only once per packet within a single pipeline stage, so iterative or multi-pass computation over the same state is not directly supported. Second, SALUs are a scarce resource: each independently updated array consumes one, and a pipeline provides only a limited number. These constraints favor telemetry mechanisms with compact, fixed per-packet state and simple update logic over more expressive designs that would require recirculation, multi-pass processing, or large per-flow state.

Hardware realization. The systems in this thesis are implemented on the Intel Tofino [51], a widely used P4-programmable switching ASIC. On such targets, the relevant resources are the match-action stages, the on-chip Static Random-Access Memory (SRAM) that stores table and register state, the hash units that compute table and bucket indices, and the SALUs that update stateful memory. The data plane designs in Chapters 4 and 5 are structured around these resources, and their hardware cost is reported in terms of this utilization. We treat the efficient use of stateful memory as a first-order design constraint when developing high-precision, line-rate telemetry for network slices.

2.1.4 Data Plane Telemetry Primitives

Building on the programmable data planes described in §2.1.3, contemporary monitoring designs have moved beyond coarse-grained, polling-based mechanisms toward high-performance data plane primitives (*cf.* §2.1.2). In particular, INT algorithms and sketching algorithms have emerged as the primary mechanisms for achieving real-time, high-precision visibility.

INT. Traditionally, infrastructure monitoring has relied on a polling-based model where a centralized management system periodically retrieves counters from network elements. However, the coarse time granularity of polling is often inadequate for providing service assurance to mission-critical URLLC services. More recently, push-based streaming telemetry [19,119] has gained traction due to its ability to stream data directly from devices to analysis engines in near real-time.

Of particular note is INT [119], which leverages programmable data planes to embed detailed performance metadata such as per-hop latency, queue occupancy, and ingress/egress timestamps directly into live data plane traffic. As illustrated in Figure 2.3, telemetry data is added at every hop from the INT source to the INT sink, where it is stripped and forwarded to a collector for analysis. While INT provides high-fidelity per-packet visibility, it introduces a significant “telemetry tax” in the form of increased bandwidth and processing overhead [15]. The architectural challenges of mitigating this overhead while maintaining SLA awareness for specific slices are evaluated in §2.3.

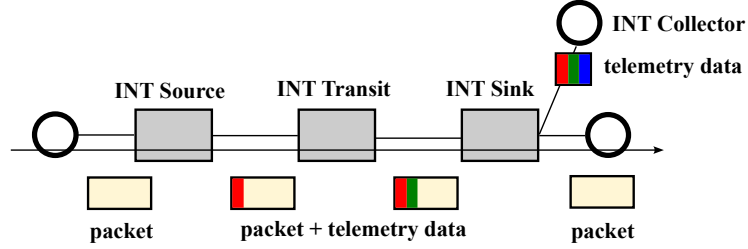


Figure 2.3: **Basic operation of In-band Network Telemetry (INT).** Packets are instrumented at the INT source and accumulate per-hop telemetry metadata as they traverse the data plane. At the INT sink, the metadata is stripped and exported to a collector, and the original packet headers are restored before forwarding to the destination.

Sketching Algorithms. Sketching algorithms use probabilistic data structures to compress large data streams in a way that allows querying approximate statistics about the entire data stream. A sketch usually leverages hashing to map flow keys into a table with multiple buckets/cells within constant time. The sketches, being concise summaries of data, maintain a very small amount of state, and are notably smaller than the input stream. Further, most sketches allow constant-time insertion and queries for fast processing of streaming data.

Prior work has shown how to use sketches to efficiently answer monitoring queries such as flow-size estimation, Distributed Denial of Service (DDoS) detection, and heavy-hitter detection [63, 127, 132]. Sketches include simple sketches, with inherent inaccuracies, or more sophisticated sketches that offer improved accuracy. A well-known simple sketch is the Count-Min Sketch (CMS) [30], which suffers from poor accuracy due to mismatches in the presence of skewed network traffic. More sophisticated sketches include ElasticSketch [127] and NitroSketch [63], which enhance accuracy at the cost of complexity, which often makes them impractical in real-world deployment. Recent work has attempted to combine the benefits of sketches with INT for fine-grained network monitoring.

In the context of 5G user planes, these primitives provide the foundation for statistical fidelity. However, as analyzed in §2.4, existing sketches primarily focus on per-switch aggregate counters and often lack the distributional detail or cross-domain correlation required to detect transient, slice-relevant performance anomalies.

Event-Driven Telemetry. Event-driven systems optimize telemetry overhead by exporting data only when specific, predefined conditions or “predicates” are satisfied. Unlike INT, which attaches data to packets regardless of network state, event-driven frameworks such as Marple [75], BurstRadar [54], and NetSeer [133] trigger reporting based on micro-

events like queue threshold crossings or packet drops. While these mechanisms significantly reduce measurement costs by ignoring *steady-state* traffic, they restrict visibility to *known failure modes*. As discussed in §2.4, this can be a limitation in 5G slices where transient, unknown performance shifts, rather than outright packet loss or drops, often precede SLA violations.

2.2 Architectures and Mechanisms for Slice Monitoring

2.2.1 Domain-centric vs Slice-aware Pipelines

Recent literature on monitoring for 5G and beyond networks has focused on addressing the scalability aspect, but often falls short in providing robust support for network slicing. Existing solutions frequently lack concrete implementations for computing slice-level KPIs and Application Programming Interfaces (APIs) for specifying monitoring requests at the network slice level. While some works [40, 84] offer no direct support for network slicing, others [14, 69] approach slicing from an abstract architectural standpoint without detailing the underlying mechanisms for slice monitoring and KPI computation.

Scalable Monitoring Architectures. Several works [14, 40, 69, 84] focus on scalable monitoring architectures for 5G networks. Perez *et al.* [84] present a monitoring architecture for a multi-site 5G platform consisting of intra-site and inter-site publish-subscribe brokers. This design utilizes infrastructure-specific agents for extracting and translating metrics from heterogeneous components. Similarly, Giannopoulos *et al.* [40] present a monitoring framework focusing on different levels of the 5G network, leveraging open-source tools like Netdata [76] and Prometheus [115] to collect system and network-level metrics. However, while these works address metrics collection and scalability, they do not discuss network slicing or slice KPI computation.

Slice-Aware Frameworks. To bridge the gap in slice monitoring, Beltrami *et al.* [14] present an architecture for monitoring e2e network slices across multiple heterogeneous domains. Their framework facilitates horizontal and vertical elasticity by dynamically instantiating or removing monitoring agents associated with the Network Functions (NFs) related to a slice. Although they propose components such as a slice measurements aggregator and an engine controller for tuning configurations (*e.g.*, monitoring frequency), they do not provide any quantitative analysis regarding the resource consumption of the proposed architecture. Finally, Mekki *et al.* [69] present a scalable architecture for network slices that collects metrics from various segments (*e.g.*, Radio Access Network (RAN),

edge, and core) via slice-specific collection agents. 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, it falls short in describing how slice-level aggregations are performed, or how domain orchestrators compute KPIs from the monitoring data extracted from NFs within a slice. This challenge is illustrated conceptually in Figure 2.4.

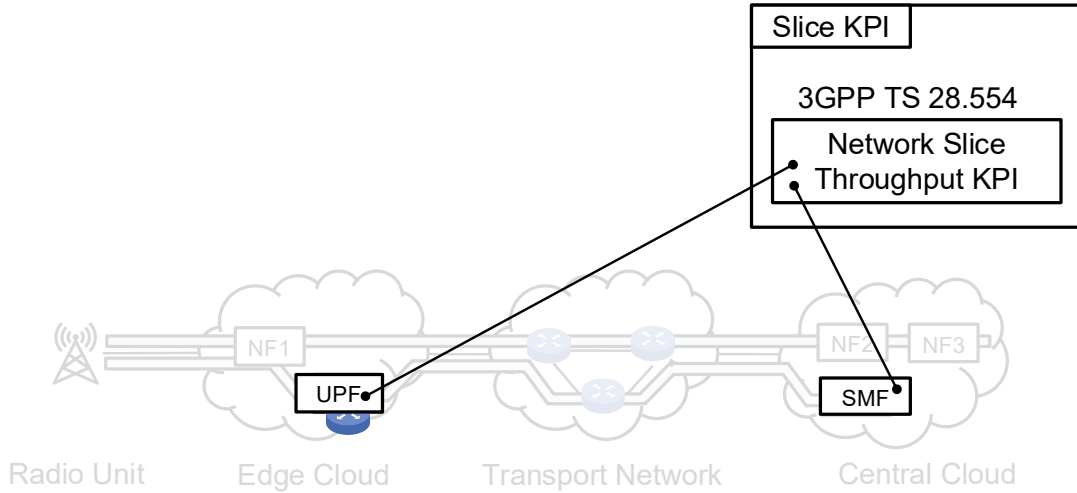


Figure 2.4: **Limitations of domain-centric monitoring pipelines.** Computing slice-level KPIs (e.g., throughput or latency as defined in 3GPP [1,2]) often requires correlating measurements exported independently by different network functions (e.g., SMF and UPF). In the absence of a slice-centric abstraction, this correlation must be implemented externally, leading to a lack of slice-centric view.

Figure 2.4 illustrates a recurring limitation of domain-centric designs: even simple slice KPIs often require correlating measurements that originate in different functions and domains. For example, computing a slice throughput KPI requires joining session-to-slice context from the SMF with traffic statistics from the UPF, which are typically exported via separate monitoring pipelines. Without slice-centric abstractions and APIs, this correlation is manual, brittle, and difficult to perform in real time.

2.2.2 Adaptive and Efficient Telemetry

As monitoring fidelity (*cf.* §2.1.2) increases, the associated communication and processing overhead becomes a bottleneck. Therefore, to maintain efficiency, recent work has followed

two primary directions. First, using Artificial Intelligence/Machine Learning (AI/ML) techniques to dynamically regulate monitoring fidelity. Second, adopting lightweight instrumentation techniques to minimize the footprint of data collection.

Efficient monitoring mechanisms. Several works [85, 101, 125] leverage AI/ML to reduce the communication overhead in 5G monitoring. Xie *et al.* [125] advocate for integrating joint monitoring and analytics to balance monitoring costs with service assurance. They propose a "partial monitoring" approach achieved through feature selection and optimal probe placement. While their simulations suggest that analytics can inform monitoring frequency, the study relies on simplistic models without a concrete system implementation.

In contrast, Sciancalepore *et al.* [101] and Plascinkas *et al.* [85] provide more concrete systems including integration with management and orchestration systems for virtualized NFs. Sciancalepore *et al.* [101] use a two-stage approach consisting of unsupervised learning for clustering NF profiles and Reinforcement Learning (RL) to balance monitoring load against decision quality. Plascinkas *et al.* [85] extend this by utilizing time-varying characteristics to adjust the monitoring frequency of individual metrics. However, these works limit their focus to individual NF rather than considering the e2e network slice as a first-class entity.

Instrumentation Mechanisms. Some works, such as Janus [38] and 5GC-Observer [55] focus on low-level extraction of metrics from NFs using Extended Berkeley Packet Filter (eBPF) [32]. Janus targets the RAN, enabling inline execution of small snippets of eBPF code to facilitate real-time monitoring and control. Conversely, 5GC-Observer targets the 5G core, enabling the monitoring of container-based NFs without modifying their source code. These systems address the problem of *how* monitoring data can be extracted efficiently from 5G network components. In contrast, MONARCH (Chapter 3) addresses the orthogonal problem of *how* such data can be orchestrated, correlated, and processed to compute network slice KPIs.

2.2.3 KPI Computation and Analytics

Beyond architectural abstractions and efficiency mechanisms, a fundamental challenge lies in how slice-level KPIs are computed and standardized. This subsection reviews existing approaches to KPI computation and standardized analytics in 5G networks.

Mechanisms for KPI Computation. Vasilakos *et al.* [121] proposed ElasticSDK, a software development kit that provides abstractions for monitoring tasks via a custom agent deployed on the FlexRAN [37] controller. While it demonstrates how to setup an effective

monitoring pipeline for the RAN, it lacks details on implementing filtering and aggregation for slices. Veitch *et al.* [122] emphasize that solely relying on e2e KPIs is insufficient; they demonstrate how resource contention (*e.g.*, last-level cache sharing) between critical and non-critical slices can lead to degraded performance for critical slices, which cannot be diagnosed with just e2e KPIs. This underscores the need for monitoring architectures that allow monitoring both e2e KPIs, as well as the capability to drill down to lower levels.

The Role of NWDAF. To enable analytics-driven automation in 5G systems, 3rd Generation Partnership Project (3GPP) introduced the Network Data Analytics Function (NWDAF) as a standardized analytics function [6]. The NWDAF is envisioned to collect data from control plane and management interfaces, apply statistical or machine-learning models, and expose derived analytics to consumer functions such as policy control, network management, and orchestration systems. For the NWDAF, 3GPP defines standardized procedures for analytics generation and exposure within the 5G core architecture.

However, in the context of slice monitoring, the NWDAF focuses on analytics generation and distribution rather than on the mechanisms for cross-domain monitoring data collection or slice-level KPI computation. While the NWDAF can consume slice-scoped measurements, the 3GPP specification leaves the realization of e2e monitoring, metric correlation, and KPI computation pipelines to implementation-specific monitoring systems [4].

The Abstraction Gap

Existing monitoring architectures either emphasize scalable metric collection or standardized analytics exposure. However, they do not provide a unified, slice-centric abstraction that enables cross-domain correlation and computation of slice-level KPIs. As a result, slice monitoring remains fragmented across domains and dependent on ad hoc integration logic. This gap motivates MONARCH (Chapter 3).

2.3 SLA-Aware Network Slice Monitoring

Architectural support for e2e slice visibility is a necessary foundation, but it is insufficient in isolation. Even when metrics can be computed across domains, the fidelity of the resulting view has a significant impact on both monitoring overhead and the ability to detect SLA violations (§2.1.2).

2.3.1 Fixed-Rate and Uniform Slice Monitoring

Many existing monitoring systems (e.g., Prometheus [115]) rely on fixed polling intervals or static collection policies to balance overhead and accuracy. However, slice workloads exhibit pronounced temporal variability driven by changes in user demand and mobility [100]. A configuration adequate for steady-state operation may fail to capture rapid performance degradation, while high-frequency monitoring applied uniformly across all slices generates redundant data and excessive overhead [85, 101, 125].

Further, network slicing involves the coexistence of heterogeneous service objectives. As illustrated in Figure 2.5, an Enhanced Mobile Broadband (eMBB) slice may operate comfortably with a latency target of 10 – 50 ms, whereas a URLLC slice demands continuous tracking to ensure compliance with stringent 1 – 5 ms latency targets. Applying a uniform policy results in a sub-optimal trade-off: either over-monitoring of best-effort traffic or failing to detect transient QoS degradation in critical slices.

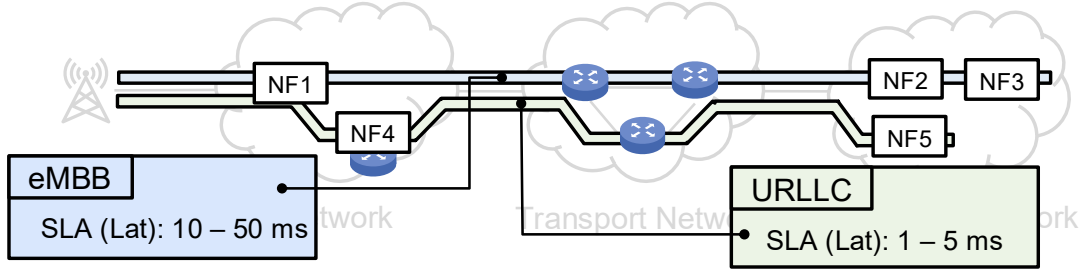


Figure 2.5: **Heterogeneous monitoring requirements across network slices.** While eMBB slices can operate with 10 – 50 ms latency, URLLC slices need continuous, high-precision tracking to ensure real-time compliance with strict 1 – 5 ms latency SLA targets.

2.3.2 High-precision Telemetry for Network Slices

To achieve the high-precision tracking required for critical slices, researchers have turned to data plane programmability, which enables the collection of telemetry at line rate. However, adapting these techniques, which were originally designed for high-performance data centers, to the context of 5G network slicing introduces significant challenges regarding overhead, flexibility, and SLA-awareness.

INT. While INT provides the high-precision primitives described in §2.1.4, its application to network slice monitoring is primarily constrained by a lack of SLA-awareness. First, *active measurement* approaches such as [16, 49] rely on injecting a minimum number of synthetic probes to cover all network links. While effective for monitoring link and port status, these are poorly suited for slice monitoring as they measure the performance of probes rather than actual slice traffic, potentially missing SLA violations occurring on user-plane flows.

Second, *selective in-situ measurement* approaches aim to monitor actual traffic with reduced per-packet overhead. PINT [15] uses probabilistic techniques to spread INT information across multiple packets. While accurate for long-lived flows, this may be unsuitable for the transient, short-lived events characteristic of URLLC traffic. Selective insertion methods like DeltaINT [102] and LINT [26] reduce overhead by embedding data only when states change. However, these solutions remain largely "slice-blind," lacking a mechanism to prioritize high-fidelity tracking for critical slices over best-effort ones.

Data Plane Sketches. In the context of 5G network slice monitoring, sketch-based approaches suffer from limited flexibility which manifests along two main dimensions. First, *task expressiveness and scope*: systems such as UnivMon [64] and Beaucoup [22] improve flexibility by supporting multiple monitoring tasks over shared sketch primitives. UnivMon provides a generic abstraction for a class of monotonic functions, while Beaucoup models queries in terms of key–attribute–threshold tuples. FlyMon [132] introduces runtime task reconfiguration by decomposing monitoring logic into key-selection and attribute-operation phases, enabling tasks to be composed dynamically via southbound APIs. However, these approaches focus on per-switch measurements (e.g., heavy hitters, super-spreaders) and do not natively support network-wide KPIs such as e2e latency, jitter, or slice throughput, which require cross-domain correlation beyond a single data plane element.

Second, *SLA-aware resource allocation*: while state-of-the-art sketches like ElasticSketch [127] and NitroSketch [63] include mechanisms to reduce collisions, they typically treat all traffic equally. In a multi-tenant 5G infrastructure, on-chip SRAM is a finite resource. Existing designs lack the control plane abstractions to dynamically allocate memory budgets or adjust sketch parameters based on slice-specific SLA priorities. Consequently, a best-effort slice can consume a disproportionate share of sketch resources, degrading the measurement precision for critical slices that require the strongest performance guarantees.

The Control Gap

Existing programmable data plane and adaptive monitoring techniques optimize

measurement overhead, but they treat monitoring resources as static infrastructure-level entities. They lack a slice-centric control loop capable of dynamically reallocating telemetry bandwidth and memory based on SLA priorities and observed slice conditions. This gap motivates SLICESCOPE (Chapter 4).

2.4 User-Plane Telemetry and Anomaly Detection

Accurate slice-level monitoring also depends on the fidelity of the underlying user-plane telemetry. While architectural abstractions enable slice KPI computation (§2.2) and adaptive control mechanisms regulate monitoring overhead (§2.3), the fidelity of the collected data determines whether transient QoS degradations can be observed and attributed. In 5G deployments, many performance anomalies manifest first as distributional shifts such as inflation of tail latencies or bursts of jitter (as we demonstrate empirically in Chapter 5). To address the need for such high-resolution visibility, this section surveys user-plane telemetry mechanisms and anomaly detection frameworks relevant to network slices.

2.4.1 User-Plane Telemetry Mechanisms

Building upon the primitives described in §2.1.4, we discuss how these approaches perform in terms of detecting transient, distributional anomalies across network slices.

INT. Beyond the control challenges discussed in §2.3, the fundamental limitation of INT variants like PINT [15] or LINT [26] lies in their impact on statistical fidelity. To reduce the overhead of per-packet telemetry, these approaches either sample packets or export data only upon state changes. While effective for reducing volume, such sparse sampling inherently filters out rare but critical distributional deviations, such as millisecond-scale jitter spikes that may occur between sampled packets or under the threshold of a change-trigger.

Event-Driven Telemetry. While event-driven systems like Marple [75] and Net-Seer [133] effectively minimize overhead by reporting only on specific predicates (*e.g.*, a queue overflow), they are fundamentally reactive. In 5G slices, SLA violations often manifest as subtle pre-failure shifts in latency distributions before packets are dropped. Because these systems ignore traffic that satisfies current predicates, they lack the capability to look back on earlier measurements and diagnose the root cause of an anomaly that was not predefined as a trigger.

Sketch-Based Telemetry. Although sketching algorithms provide space-efficient summarization (§2.1.4), existing designs like UnivMon [64] and FlyMon [132] are primarily optimized for volumetric counters (*e.g.*, heavy hitters, flow sizes). They do not natively target distributional metrics such as tail latencies or inter-arrival time variances.

In 5G user planes, this limitation is critical at convergence points such as the UPF (Figure 2.6). Since multiple TEIDs from different slices share physical buffering resources, congestion effects often manifest in latency tails rather than mean values [59]. Current sketch paradigms create a tension between visibility and scalability: we must either instantiate massive, per-flow sketches (prohibitive memory) or rely on aggregate counters (loss of slice-specific tail visibility).

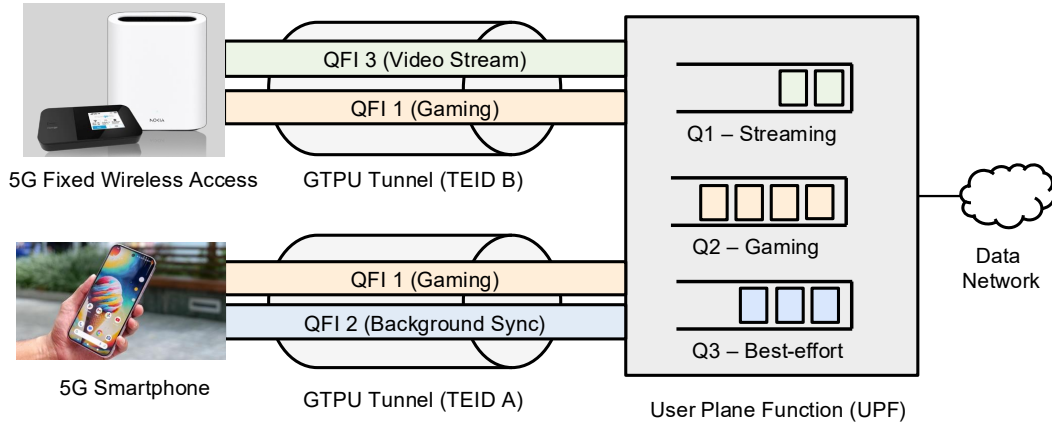


Figure 2.6: **Attribution challenges in multi-tenant user planes.** Multiple GTP-U tunnels (TEIDs) from different network slices converge into shared hardware queues at the UPF. Congestion within these shared resources often manifests as transient distributional shifts (*e.g.*, tail latency inflation) that are difficult to attribute to a specific slice using aggregate counters, necessitating high-fidelity telemetry capable of per-slice distributional resolution.

2.4.2 Anomaly Detection and Analytics in 5G Slices

Anomaly detection in 5G systems has primarily focused on control plane behavior and aggregate management-plane metrics. For the 5G core, approaches include sequence-based models for detecting abnormal NF interactions [112], AI/ML-based identification

of anomalous traffic patterns within NWDAF frameworks [70], and detection of service degradation in cloud-native NFs [71]. In the RAN, distributed detection models such as SpotLight [110] address anomalies across edge and cloud deployments, while other works investigate signaling storms and their mitigation [111].

Slice-level anomaly detection [18, 20] correlates multivariate slice KPIs to detect security and QoS threats. These approaches operate either on detailed packet traces, incurring substantial storage and computation cost, or on aggregate KPIs exported at relatively coarse granularity.

5G Core Analytics Frameworks. As established in §2.2.3, the NWDAF serves as the standardized engine for AI/ML-driven analysis within the 5G core. Recent works have utilized the NWDAF framework to automate the identification of anomalous traffic patterns and predict service degradation in cloud-native functions [70, 71]. However, it is important to distinguish between the analytics logic and the data source.

While these frameworks provide the *intelligence* for policy control, they remain analytics consumers. Their effectiveness is fundamentally capped by the fidelity of the telemetry provided by the underlying monitoring systems. As we demonstrate in Chapter 5, even the most sophisticated NWDAF models will fail to detect anomalies if the input data lacks the distributional resolution to capture transient jitter spikes or tail latency inflation.

The Visibility Gap

Existing telemetry paradigms face a fundamental tension between resource efficiency and statistical fidelity. While selective reporting and aggregate sketching reduce measurement overhead, they fail to preserve the distributional resolution, such as tail latency behavior, required to detect and attribute transient anomalies within shared user-plane resources. There is a need for a lightweight, distribution-aware data plane mechanism that enables slice-scoped attribution of QoS anomalies under strict SRAM constraints. This gap motivates Kestrel (Chapter 5).

2.5 Synthesis and Open Gaps

The prior work surveyed in this chapter establishes the foundational building blocks required for monitoring network slices, including scalable data collection pipelines, adaptive policies, and high-performance telemetry primitives. However, as synthesized in Table 2.1, these

capabilities are typically realized in isolation and through an infrastructure-centric lens that does not account for the unique requirements of network slicing. Specifically, three structural gaps remain unaddressed:

Layer/Approach Class	Capabilities Provided	Key Gap for Slice Monitoring
Architecture & Abstractions [4, 6, 14, 40, 69, 84]	Cross-domain metric collection and standardized analytics exposure (NWDAF).	Abstraction Gap: Slices are not first-class entities; KPI correlation and identifier resolution (S-NSSAI vs. TEID) remain external and brittle (§2.2).
Monitoring Control [15, 26, 85, 101, 102, 125]	Overhead reduction via dynamic frequency adjustment and selective telemetry export.	Control Gap: Decisions are infrastructure-centric and “slice-blind”; lacks the ability to prioritize telemetry resources based on slice-specific SLA priorities (§2.3).
User-Plane Telemetry [54, 59, 64, 75, 119, 127, 132]	High-precision per-packet visibility and space-efficient sketching of traffic counters.	Visibility Gap: Tension between overhead and precision; existing sketches lack the distributional resolution needed to detect and attribute tail-latency anomalies (§2.4).

Table 2.1: **Synthesis of related work for the network slice monitoring.** While prior work provides individual mechanisms for scalability and precision, they lack the vertical integration and slice-centric approach needed for slice SLA monitoring.

First, at the *architectural* layer, existing systems do not treat network slices as first-class abstractions, nor provide any means to express slice-level monitoring intent. Further, they do not encapsulate cross-domain correlation and identifier resolution within the monitoring layer, and thereby push the complexity of KPI computation into the management plane. This complicates the slice orchestration and management. Chapter 3 develops MONARCH to address this abstraction gap.

Second, in terms of *monitoring control*, current adaptive mechanisms focus on local efficiency rather than that of e2e slice monitoring. Without a control loop capable of integrating slice semantics and SLA priorities, telemetry resources are often allocated uniformly, leading to the “over-monitoring” of best-effort traffic at the expense of critical flows. Chapter 4 introduces SLICESCOPE to bridge this control gap via budget-aware fidelity allocation.

Third, at the *user-plane* layer, there exists a fundamental tension between visibility and limited data plane resources. Existing telemetry paradigms either mask fine-grained

behavior or incur high overheads and cannot scale to many concurrent slices. Further, the shared nature of 5G user-plane resources (e.g., shared queues) makes it difficult to attribute transient performance shifts to specific slice instances. Chapter 5 presents KESTREL to resolve this visibility gap using distribution-aware data plane primitives.

Taken together, these limitations identified in this chapter motivate a cross-layer approach to slice monitoring. Architectural abstractions, control plane adaptation, and user-plane telemetry primitives address distinct but interrelated design dimensions. This thesis develops complementary mechanisms at each layer to achieve efficient high-precision monitoring of network slices.

Chapter 3

MONARCH: Architecture for Slice-Level Monitoring

3.1 Introduction

In chapter 1, we identified a key architectural abstraction gap in monitoring of network slices: while slice Service Level Agreements (SLAs) are defined over end-to-end (e2e) Key Performance Indicators (KPIs), existing monitoring systems are largely domain- or network-function-centric. Slice-level semantics are therefore not explicitly represented in the monitoring layer; instead, e2e KPIs must be reconstructed from more fragmented measurements. This responsibility is pushed upward into the analyze-plan-execute stages of the Monitor-Analyze-Plan-Execute (MAPE) loop (*cf.* Figure 1.2), where control applications need to perform cross-domain correlation of network state before executing their business logic. As we argue in the remainder of this chapter, this misplaced abstraction boundary not only complicates decision making but also prevents slice monitoring from serving as a reusable, scalable foundation for closed-loop automation. To resolve this architectural gap, this chapter addresses Research Question RQ1:

How can a monitoring architecture abstract network slices as first-class entities and expose slice-level KPIs through a unified interface, enabling control applications to reason about e2e slice performance across heterogeneous domains?

We answer this question by designing and implementing MONARCH, a network slice monitoring architecture for cloud-native Fifth Generation (5G) deployments. Our design

is directly motivated by the mismatch in the capabilities provided by existing monitoring systems and the challenges of slice-level monitoring.

Abstraction challenges. Standardized slice KPIs defined by 3rd Generation Partnership Project (3GPP) [1] are inherently composite. Computing slice KPIs requires correlating network state distributed across network domains, and may involve potentially combining control plane state (e.g., session information) with user-plane measurements (e.g., traffic counters). For example, validating an e2e latency SLA requires aggregating latency information from both the distributed Radio Access Network (RAN) and the centralized 5G core. However, most existing monitoring systems provide mainly infrastructure or Network Function (NF) metrics (e.g., CPU and memory usage [74, 82]) or provide domain-specific telemetry without explicit slice-level abstractions [40, 84]. Even approaches that acknowledge slicing at a conceptual level lack concrete mechanisms for computing e2e slice KPIs and means of exposing such through unified monitoring interfaces [14, 69] that may be consumed by control applications.

Scalability challenges. Beyond abstraction, there exists a systems challenge in scaling monitoring with the number and diversity of slices. Operational deployments may host tens to hundreds of concurrently active slices [98, 99], where each slice might require slice-specific telemetry. Fixed high-granularity monitoring across all slices and KPIs, i.e., *always-on telemetry* can therefore lead to excessive resource usage and redundant data collection. As the number of slices grows, a practical monitoring architecture for network slices must be able to keep data ingestion latency bounded and maintain a predictable resource footprint. At the same time, different slices may require monitoring at different levels of granularity depending on their service objectives. This means a network slice monitoring system should provide flexible control over the monitoring scope and interval.

MONARCH overview. MONARCH addresses these challenges by elevating network slices to first-class monitored entities. It provides architectural support for slice-level KPI computation across heterogeneous domains. Rather than requiring monitoring-driven applications to manually correlate telemetry from individual network functions, MONARCH introduces a modular monitoring pipeline. This pipeline a) collects measurements from different slice segments, b) correlates control plane and user-plane information, and c) exposes slice-centric monitoring abstractions through a unified northbound interface.

MONARCH is designed for cloud-native 5G deployments where NFs are virtualized and deployed as containers; this allows them to be instantiated across domains (e.g., edge and core network) and dynamically scaled according to demand. This deployment model aligns with industry-wide shift towards cloud-native orchestration for telecommunication services, including the 5G Service-Based Architecture (SBA) [3, 28, 34]. MONARCH provides an

architectural foundation for slice-level observability for closed-loop network automation by decoupling control applications from domain-specific telemetry sources. The architecture is extensible to different telemetry mechanisms; more sophisticated monitoring techniques (*e.g.*, In-Band Network Telemetry (INT) and sketches) can be incorporated without modifying the slice-level abstraction exposed to applications.

Contributions. This chapter makes the following contributions:

- **Slice-native monitoring architecture.** We design and implement MONARCH, a monitoring architecture that abstracts slices as first-class entities and enables explicit computation of e2e slice KPIs from distributed NF-level telemetry.
- **Slice-centric monitoring API.** We introduce a northbound Application Programming Interface (API) that allows monitoring applications to specify slice-level and NF-level monitoring requests at a high level of abstraction, decoupling monitoring logic from heterogeneous telemetry sources and supporting configurable monitoring intervals.
- **Prototype implementation and open testbed validation.** We implement MONARCH as a cloud-native prototype on our 5G slicing testbed (*cf.* Appendix A). We demonstrate concrete slice-level and network-function-level monitoring use cases, and make the implementation publicly available [89].
- **Scalability and overhead evaluation.** Through experiments with up to 50 concurrent slices, we show predictable resource scaling and bounded ingestion latency (below 3 ms). Using real-world 5G traffic traces, we demonstrate that MONARCH’s adaptive monitoring mechanisms can reduce telemetry overhead by up to 76% while maintaining acceptable KPI monitoring accuracy.

Chapter roadmap. Section 3.2 characterizes the fundamental observability gap in network slice monitoring for cloud-native 5G deployments and derives the architectural requirements. Section 3.3 presents the design and operation of MONARCH, including the monitoring request flow and architectural components. Section 3.4 describes the prototype implementation and testbed integration. Section 3.5 evaluates scalability, ingestion latency, and monitoring overhead. Finally, Section 3.7 summarizes key insights and discusses how MONARCH provides the architectural foundation for the SLA-aware monitoring control mechanisms developed in Chapter 4.

3.2 Characterizing the Slice Observability Gap

As highlighted §3.1, standardizing slice KPIs is insufficient if the underlying telemetry cannot support them. Modern 5G networks comprise network state distributed across the RAN, Core, and Transport domains. We argue that this creates a fundamental *semantic gap*: slices are logical constructs defined in the control plane, but telemetry is generated by underlying resources spanning both physical infrastructure (e.g., hypervisors) and logical NFs (e.g., User Plane Function (UPF)) that are agnostic to these logical definitions.

This gap follows from the slice realization model introduced in §2.1.1: slice identity is anchored by Single Network Slice Selection Assistance Information (S-NSSAI) in the control plane, while user-plane measurements are commonly indexed by lower-level identifiers such as QoS Flow Identifiers (QFIs) and Tunnel Endpoint Identifiers (TEIDs). The related work surveyed in §2.2 shows that existing monitoring systems provide scalable metric collection and, in some cases, slice-aware collection pipelines, but they do not fully encapsulate this identifier resolution and KPI aggregation logic within the monitoring layer.

In this section, we characterize the structural disconnects that prevent existing monitoring systems from bridging this gap (§3.2.1), and analyze the operational penalties of attempting to resolve them through ad-hoc correlation (§3.2.2).

3.2.1 The State-Telemetry Mismatch

The observability gap stems from the orthogonal lifecycles of network slices and the underlying resources that support them. Unlike legacy networks where a “pipe” was a stable physical construct, a 5G slice involves a distributed collection of *ephemeral* state. This creates a structural mismatch: the *unit of management* (the slice) spans multiple domains and functions, but the *unit of monitoring* in most existing systems is fundamentally infrastructure-centric. These systems expose metrics at the level of individual network functions, containers, or compute nodes rather than at the level of network slices. Table 3.1 highlights this mismatch by contrasting existing 5G monitoring architectures in terms of the key capabilities needed for slice-level monitoring.

Existing 5G monitoring architectures. As Table 3.1 shows, several prior works propose architectures for scalable data collection [14, 69, 84]. Some systems also discuss integration with 5G deployments [40, 69]. However, they do not elevate slices to first-class monitoring entities: slice monitoring APIs are absent, adaptive slice-aware control is not supported, and slice KPI computation is at best partial. This limitation affects both simple and composite network slice KPIs. Although simpler KPIs (e.g., per-slice session counts) can

Table 3.1: **Gap analysis of existing 5G monitoring architectures.** Comparison of prior systems with MONARCH across key design requirements for e2e slice monitoring. Legend: supported (✓), not supported (✗), partial (◐).

Feature/Ref.	Perez [84]	Beltrami [14]	Giann. [40]	Mekki [69]	MONARCH (this chapter)
Scalability	✓	✓	✗	✓	✓
Slice monitoring API	✗	✗	✗	◐	✓
Adaptive / slice-aware control	✗	✗	✗	✗	✓
Slice KPI computation	✗	◐	✗	◐	✓
5G network integration	✗	✗	✓	✓	✓
Open-source availability	✓	✗	✗	✗	✓

be derived from control plane state alone, even then, applications that consume these KPIs, i.e., control applications, must reason in terms of individual network functions rather than querying a slice-level abstraction.

Commodity monitoring tools. Commodity observability systems such as Prometheus [115], OpenTelemetry [117], Thanos [27], Grafana [43], Nagios [74], and OpenStack Ceilometer [82] provide mature support for collecting, storing, querying, and visualizing metrics. Similarly, collectors based on exporters, xApps [77], eBPF [32] probes, INT [119], or sketches can expose measurements from specific network functions, devices, or domains. These mechanisms can serve as telemetry substrates for MONARCH; indeed, our lab deployment included Prometheus-based monitoring as part of the private 5G core deployed by our industry partner. However, they do not by themselves determine which NFs, sessions, TEIDs, or QFIs currently belong to a slice, nor do they define the aggregation semantics needed to compute slice KPIs. MONARCH adds this missing slice-semantics layer by translating slice-level monitoring intent into concrete telemetry targets, correlating control-plane context with user-plane measurements, and exposing computed slice KPIs through a unified interface.

The unresolved state-correlation problem. For more complex KPIs such as slice throughput or latency, the problem becomes more acute: they require attributing user-plane measurements to the correct slice, which involves correlating two independently maintained state domains. First, the *control plane state*, which defines the logical topology (e.g., which session belongs to which slice). This information is maintained by the Session Management Function (SMF) and is updated on events such as session establishment, session modification, etc. Second, the *user plane state*, which contains performance data (e.g., packet counters, latency). This information is typically maintained at the UPF but is indexed by ephemeral

(TEIDs) rather than slice identifiers.

In the absence of slice-centric monitoring abstractions, this complexity is externalized from the monitoring layer and pushed into the control logic. Next, we analyze the operational complexity that arises from this limitation.

3.2.2 Operational Complexity of State Reconstruction

We analyze two common monitoring scenarios where the lack of slice-native abstraction leads to observability failures.

Scenario A: The Opaque Tunnel Problem. Consider monitoring the aggregate throughput of a dedicated Augmented Reality (AR) slice [29]. The UPF exports counters for packets/bytes transmitted, which are indexed by ephemeral TEIDs. These identify sessions but do not encode the associated slice identity. Therefore, to compute slice throughput KPI, a monitoring system must join these UPF counters with SMF-maintained mappings that bind active TEIDs to their corresponding slice ID. For example, if the UPF reports 1 Gbps on TEID=123 (opaque), and the associated SMF maps TEID=123 to Slice A, then MONARCH correlates these to attribute 1 Gbps to Slice A’s throughput. Without this cross-correlation, traffic volume is observable but cannot be attributed to the correct slice. We refer to this as the *opaque tunnel problem*, i.e., the data plane sees the volume, but the control plane holds the label.

Scenario B: The QoS Blind Spot. Consider the case of a mission-critical Vehicle-to-Everything (V2X) slice requiring sub-10 ms delay [10]. The UPF exports latency measurements for packets marked with a specific QFI, e.g., $QFI = 9$. However, these QFI values are often reused across slices with similar delay profiles (e.g., cloud gaming). Therefore, a reported 50 ms latency spike for $QFI = 9$, by itself, cannot distinguish which slice is affected. Determining whether an SLA violation has indeed occurred requires correlating user-plane measurements with control plane policy context that binds the flow to a specific slice. We refer to this as the *Quality of Service (QoS) blind spot problem*, i.e., the violation is visible, but the victim cannot be determined.

The Cost of Decentralized State Reconstruction. In the absence of a unified slice monitoring architecture, this correlation logic is pushed downstream to control applications (e.g., resource scaling, admission control, or fault management systems). This leads to duplicated complexity: each consumer must independently subscribe to control plane events, maintain local mappings, and track ephemeral network state. This duplication places state reconciliation on the critical path of runtime decisions, which in turn limits automated network management.

3.2.3 Design Requirements

The characterization above reveals that effective slice monitoring cannot be achieved by simply aggregating infrastructure metrics. It requires an architecture that satisfies four specific requirements:

R1: Slice-Centric Abstraction. Network slices must be elevated to first-class monitoring entities. The system must decouple the definition of monitoring intent from the underlying infrastructure topology, shielding applications from physical resource volatility.

R2: Unified Northbound Interface. The architecture must provide a unified API that abstracts heterogeneous telemetry sources. Monitoring-driven applications should be able to query slice-level state without managing subscriptions to individual network functions or tracking instance migrations.

R3: Automated Cross-Domain Correlation. To verify SLA compliance for performance metrics (e.g., throughput, latency), the system must correlate control plane context (e.g., TEID-to-Slice mappings) with user-plane telemetry. This closes the semantic gap, ensuring that composite KPIs are attributed to the correct slice without requiring manual state reconstruction by consumers.

R4: Resource-Efficient Operation. To eliminate the overhead of decentralized state tracking, correlation logic must be executed once within the monitoring platform. The architecture must scale with the number of slices while preventing redundant computation and storage costs across downstream applications.

3.3 MONARCH Design and Operation

The design requirements above establish the architectural properties that a slice-centric monitoring system must satisfy. MONARCH realizes these properties through a modular architecture that encapsulates slice semantics within the monitoring layer. In this section, we first overview the design of MONARCH (§3.3.1), followed by the northbound monitoring API (§3.3.2). Subsequently, we discuss the flow of a monitoring request through the architecture (§3.3.3) and mechanisms included in MONARCH for adaptive monitoring (§3.3.4).

3.3.1 Monarch Overview

Architecturally, MONARCH is organized around three functional subsystems: (i) interpreting slice-level monitoring intent and coordinating monitoring scope, (ii) executing telemetry collection within network slice segments, and (iii) aggregating, correlating, and exposing slice-level KPIs. Figure 3.1 illustrates how these roles are realized through the major components of MONARCH. Next, we describe the components within each subsystem.

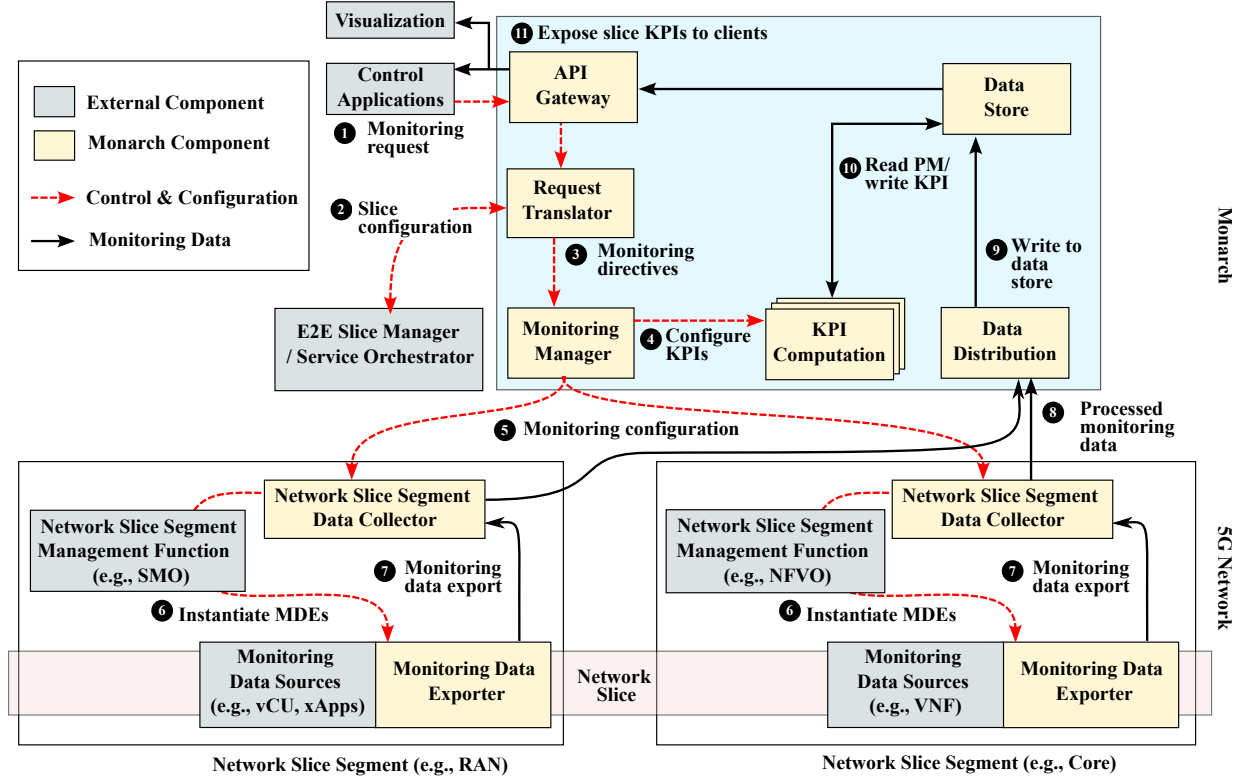


Figure 3.1: **High-level architecture of MONARCH.** MONARCH centralizes slice semantics within the monitoring layer by a) translating slice-level monitoring requests into segment-specific directives, b) aggregating telemetry across slice segments, and c) computing slice-level KPIs before exposing them through a unified API. Cross-domain state correlation is performed once within the KPIs computation module rather than repeatedly by downstream applications.

Slice-Intent Resolution and Coordination

This subsystem takes as input slice-level monitoring requests and translates them into monitoring scope and policies. It also coordinates the configuration of monitoring across different network segments. It comprises *API Gateway*, *Request Translator* and *Monitoring Manager*.

API Gateway The API Gateway implements the Northbound Interface (NBI) of MONARCH, through which monitoring applications submit requests and retrieve computed KPIs (*cf.* §3.3.2). To maintain slice-centric abstractions, MONARCH must expose slice KPIs through a stable interface independent of where and how telemetry was collected. This is enabled by the API Gateway; it maps monitoring request identifiers to the corresponding computed KPIs. It also enforces request-scoped access control. This allows for multi-tenancy and allows the internal telemetry schemas and infrastructure identifiers to evolve independently of client requests.

Request Translator This component implements the slice-centric abstractions by translating high-level monitoring requests received via the NBI (§3.3.2) into low-level monitoring directives (e.g., which NF instance to monitor). When an application submits a request using the monitoring API (§3.3.2), this component determines the required monitoring scope and resolves the relevant infrastructure instances by querying the external slice orchestrator (e.g., Open Network Automation Platform (ONAP) [116]).

By design, the Request Translator component of MONARCH is decoupled from slice lifecycle management and data collection mechanics. In contrast to existing frameworks such as ONAP, which focus on instantiating and managing data collection pipelines, the Request Translator focuses on translating slice-level monitoring requests into concrete monitoring scope and targets. It maintains the binding between high-level slice KPI and the set of active NF instances that constitute the slice. For example, in the case of slice scaling, e.g., by adding a new UPF instance, the Request Translator retrieves the updated information and dynamically updates the monitoring directives. This ensures that the monitoring scope remains accurate without having to embed slice-specific logic into the underlying data collection framework (e.g., ONAP’s Data Collection, Analytics, and Events (DCAE)). This keeps the architecture flexible and adaptable to infrastructure churn.

Monitoring Manager This component is responsible for converting slice-level monitoring directives into concrete monitoring configurations for each Network Slice Segments

(NSSs) (e.g., RAN, transport, or 5G core). While the Request Translator determines *what* slice-level state must be observed, the Monitoring Manager determines *how* that state is collected and maintained at runtime. The Monitoring Manager acts as the centralized coordinator of the monitoring layer: it manages monitoring scope, frequency, and resource allocation across different NSSs. This component is also responsible for coalescing overlapping requests. For example, if multiple applications request the same KPI at different intervals, the Monitoring Manager is responsible for deriving a unified telemetry plan at an appropriate base frequency to serve both requests without duplicating data.

Based on the requested KPI, it can also select the appropriate telemetry mechanism (e.g., polling-based metrics, INT-based delay measurement, or sketch-based summaries) and its associated parameters (e.g., target flows, reporting interval, sampling policy). It also implements monitoring intelligence, such as adaptive monitoring, to regulate monitoring overhead. We discuss adaptive monitoring in §3.3.4.

Segment-Level Telemetry Execution

This subsystem executes telemetry collection within different slice segments by configuring data collection mechanisms specific to each segment. It comprises of *Network Slice Segment Data Collector (NSSDC)* and *Monitoring Data Exporter (MDE)* components.

Network Slice Segment Data Collector (NSSDC) This component is instantiated per NSS and serves as the segment-level execution layer of MONARCH. It interacts with the corresponding management function (e.g., Service Management and Orchestration (SMO) or Network Function Virtualization Orchestrator (NFVO), as shown in Figure 3.1) to configure data collection leveraging telemetry mechanisms appropriate to its segment (e.g., xApps in the RAN, core NFs in the 5G core). It also provides capabilities such as short-term persistence, data filtering, and transformation (e.g., label enrichment) before forwarding the telemetry data upstream.

A single NSSDC instance may serve multiple monitoring requests simultaneously. Thus, they may share exporters and telemetry pipelines: this prevents redundant exporter instantiation and ensures that data collection is resource-efficient. The NSSDC is the actuation layer that acts on the decisions made by the Monitoring Manager. For example, for latency monitoring, it may activate an INT-based measurement pipeline by configuring relevant collectors and instantiating the corresponding MDEs to ingest INT reports (*cf.* INT-based slice latency tracking in SLICESCOPE, Chapter 4).

Similarly, for fine-grained data collection, it may deploy an MDE that retrieves UPF sketch summaries at the requested interval (*cf.* KESTREL, Chapter 5). This mechanism-agnostic design allows more sophisticated telemetry approaches to be integrated without altering the slice-level abstraction.

Monitoring Data Exporter (MDE) MDEs are lightweight containers responsible for extracting raw monitoring data from monitoring sources (e.g., NF, xApps), converting it into a standardized format (e.g., OpenTelemetry¹), and exporting it to the NSSDC. An MDE may operate per network function instance (e.g., retrieving counters or sketch summaries from a UPF) or per host when host-level instrumentation is used. The role of an MDEs is limited to telemetry extraction and normalization; slice-level aggregation is handled by the KPI computation module.

Slice-Level Aggregation and Exposure

This subsystem aggregates telemetry across segments, performs cross-domain correlation, and enables computation of slice-level KPIs. It comprises the *KPI Computation*, *Data Distribution*, and *Data Store* components.

KPI Computation The KPI Computation component is responsible for computing slice-level KPIs by centralizing aggregation and cross-domain correlation within the monitoring architecture. It adopts a modular execution model in which a shared execution environment provides common services such as data retrieval, time-window alignment, and correlation/aggregation. On top of this shared substrate, each monitoring request leads to the instantiation of independent computation modules that encapsulate the semantic definitions of individual KPIs (e.g., throughput, latency). MONARCH maintains a registry of KPI definitions that maps each KPI name to its semantic interpretation (*e.g.*, rate, counter, or distribution-based statistic) and the corresponding aggregation logic. The KPI Computation component consults this registry to instantiate the appropriate computation module for each monitoring request.

For each monitoring request, the component activates a computation module that computes the requested KPI by correlating control plane context with user-plane telemetry, including aggregation across multiple NFs and slice segments. When multiple requests share scope or KPI semantics, the runtime can reuse existing intermediate results to derive request-specific views (e.g., at different frequencies)

¹<https://opentelemetry.io/>

without duplicating data collection or computation. The KPI Computation component is extensible: new KPI definitions can be introduced as independent computation modules without modifying the shared execution environment. To validate this, we present a proof-of-concept implementation of slice throughput KPI in §5.5.

Data Distribution Slice-level KPI computation often requires consuming telemetry from many segments. To ingest data from multiple NSSDCs concurrently without causing bottlenecks, the Data Distribution component provides a scalable ingestion fabric. Its responsibilities include horizontal scalability and load balancing to sustain high-volume export.

Data Store MONARCH requires persistent storage to decouple telemetry collection from slice-level aggregation and analysis. The Data Store component maintains raw and processed monitoring data, intermediate aggregation results, and configuration artifacts (e.g., templates for KPI computation modules). This allows the reuse of processed data across multiple monitoring requests and supports time-windowed correlation across slice segments.

Satisfaction of Design Requirements. The design of MONARCH directly addresses the requirements of slice-centric monitoring identified in §3.2.3. First, by encapsulating slice semantics within the intent resolution and coordination subsystem, MONARCH promotes slices to first-class monitoring entities (R1). Second, the northbound API interface abstracts heterogeneous telemetry sources, which enables slice-level queries independent of instrumentation (R2). Third, KPI computation and correlation are performed within the monitoring layer, closing the semantic gap between slice KPIs and data sources (R3). Finally, by coalescing monitoring requests and sharing segment-level collectors, MONARCH avoids redundant data collection and duplicated computation, ensuring resource-efficient operation (R4).

3.3.2 Monitoring API

To realize the goal of slice-level monitoring, MONARCH exposes a dedicated northbound API. This API provides a high-level abstraction to applications that are interested in slice monitoring (*e.g.*, control applications in the MAPE loop; *cf.* Figure 1.2). This API allows MONARCH to separate monitoring intent from slice instantiation and orchestration, rather than binding monitoring to slice lifecycle primitives (*e.g.*, as done via 3GPP Network Slice Templates (NSTs) [69]). This separation ensures that higher-layer applications can

specify slice-level objectives without managing low-level telemetry configuration or tracking infrastructure/NF dynamics.

This design offers three architectural advantages. First, *granular control*: monitoring parameters, such as requested KPIs, sub-counters, reporting intervals, and duration, can be configured and modified independently of slice configuration. Second, *separation of concerns*: monitoring logic is decoupled from slice lifecycle management, reducing coupling between orchestration and observability. Finally, *operational flexibility*: multiple applications can submit distinct monitoring requests for the same slice, each with different metrics or temporal requirements, without altering the slice definition.

In MONARCH, a monitoring request is treated as a persistent monitoring object that remains active until explicitly removed. Because slice-to-instance bindings are maintained internally by the Request Translator (*cf.* §3.3.1), monitoring tasks remain valid even as underlying NFs scale or migrate. Furthermore, a single slice-level KPI request may implicitly span multiple network functions and slice segments; MONARCH resolves these dependencies internally, and the KPI Computation component performs the required cross-domain aggregation before exposing results through the API Gateway (*cf.* §3.3.1).

Structure of a Monitoring Request. MONARCH’s API allows the monitoring request to be structured hierarchically around three components: *scope*, *KPI definition*, and *monitoring policy*. The scope is used to identify the monitoring target using high-level abstractions. This can be, for example, at the slice level, or at the NF level for controlled drill-down during anomaly investigation (*cf.* §3.4.3). The KPI definition fields are used to specify the requested metric and optional associated sub-counters (*e.g.*, breakdown of the metric by QFI). Based on the specified metric, MONARCH can select the appropriate telemetry mechanisms (*e.g.*, instantiate SLICESCOPE, Chapter 4 for latency measurement). The specified sub-counters also inform the cross-domain aggregation decisions. For example, requesting `slice_throughput` broken down by `QFI:7` causes MONARCH to aggregate only the traffic associated with that QoS flow across all NFs in the slice. The policy fields define temporal parameters such as minimum reporting interval and duration, and an optional `adaptive` flag that tells MONARCH to dynamically regulate monitoring fidelity based on system conditions (*cf.* §3.3.4).

Example Request. Figure 3.2 illustrates a concrete example: a request targeting slice throughput for S-NSSAI 1-01 at a 1s reporting interval with adaptive monitoring enabled. This request implicitly spans all NFs associated with the specified slice. MONARCH resolves slice-to-instance mappings, aggregates relevant telemetry across segments, and returns the computed slice-level KPI through the northbound interface. Each submitted request is assigned a unique identifier, which can subsequently be used to retrieve configuration details

Scope (Target)	KPI Definition (Semantics)	Monitoring Policy
Type: slice	Name: slice_throughput	Interval: 1s
ID: S-NSSAI 1-01	Units: Mbps	Duration: 5 min
	Sub-counter: QFI {7, 9}	Adaptive: true

Figure 3.2: **Structure of a monitoring request intent object.** Monitoring intent is defined by scope, KPI semantics, and operational policy.

or query computed KPIs via the API Gateway.

3.3.3 Flow of a monitoring request

Having discussed the different components of MONARCH, in this section, we discuss how they work together to serve a network slice monitoring request. Figure 3.1 shows the flow of a monitoring request (Steps 1–12) through MONARCH, which proceeds as follows:

- **Step 1.** Monitoring apps, i.e., applications which want to consume slice KPIs (*e.g.*, slice management and orchestration), specify a high-level network slice monitoring request through the monitoring request API (*cf.* §3.3.2) 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 SMF and 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 configurations understood by the NSSDCs.

- **Step 6:** The corresponding NSSDC instantiates MDEs using the interfaces provided by the segment management function, 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 KPI in the data store.
- **Step 11:** Client applications (e.g., control loops or visualization tools) access the computed slice KPIs through the API Gateway using the corresponding request identifiers.

3.3.4 Adaptive Monitoring

The monitoring API (§3.3.2) allows clients to specify the minimum reporting interval, using the `interval` field. However, continuously operating at fixed high granularity across all slices can lead to excessive use of network resources (*i.e.*, *overhead*) and produce redundant (*i.e.*, *unnecessary*) data. Network slices have heterogeneous slice criticality and traffic dynamics [12, 13] and thus, fixed high-granularity monitoring across all slices is inefficient.

To address this, MONARCH supports adaptive monitoring algorithms that dynamically adjust telemetry parameters to balance monitoring accuracy and resource consumption (*cf.* R4 in §3.2.3). This is enabled by setting the `adaptive` flag in a monitoring request. Once this is enabled, the Monitoring Manager can adjust the effective reporting interval at runtime based on factors such as traffic conditions, slice priority, and overall system resource utilization.

Semantic correctness. A critical impact of enabling adaptive monitoring is that it changes the cadence at which telemetry reports are generated. This can bias KPI estimates if measurements collected at different intervals are aggregated naïvely. MONARCH prevents this by decoupling monitoring policy from KPI semantics. This is done in two ways.

First, the semantic definition of each KPI (*e.g.*, rate, counter, or distribution-based statistic) is maintained in a KPI registry. The KPI computation module then fetches the semantics based on the KPI name when a monitoring request is specified. Second, every reported KPI value is associated with the *time interval* over which it was computed. Together, these mechanisms ensure that the KPI meaning remains consistent and that telemetry reports can be aggregated correctly even when reporting intervals change due to adaptive monitoring.

For example, consider the following scenario. For rate-based KPIs such as slice throughput, a longer-horizon average can be computed correctly even with reports at varying intervals with the exposed interval information. This can be done as follows:

$$\bar{v} = \frac{\sum_i v_i \cdot \Delta t_i}{\sum_i \Delta t_i},$$

where v_i is the reported KPI value and Δt_i is the interval represented by that report. By weighting each value according to the time span it represents, the resulting estimate remains invariant to changes in telemetry sampling rate.

To demonstrate this capability, we implement a heuristic-based adaptive sampling algorithm (§5.5, Algorithm 2) that adjusts monitoring intervals according to observed traffic dynamics. While this heuristic serves as a proof of concept, MONARCH is agnostic to the specific adaptation strategy and can accommodate more sophisticated control mechanisms, including Artificial Intelligence/Machine Learning (AI/ML)-based approaches.

3.4 Prototype Realization

To validate the feasibility of MONARCH’s architectural abstractions, we implement a cloud-native prototype integrated with a 5G network slicing testbed. The goal of this prototype is to demonstrate that the key features, such as slice-centric monitoring, cross-domain correlation, and adaptive interval control, can be realized in a practical deployment.

This prototype implementation represents one concrete realization of the architecture. Several components (*e.g.*, the API Gateway and KPI computation modules) are implemented

as simple Python scripts for rapid prototyping. These services are logically decoupled and can be replaced with production-grade equivalents (*e.g.*, Golang-based microservices or service-mesh gateways) without altering the architectural principles described in §3.3.1. Below, we discuss specific aspects such as testbed environment (§3.4.1), realization of MONARCH components (§3.4.2), illustrative use-cases implemented using the prototype (§3.4.3), and adaptive monitoring realization (§3.4.4).

3.4.1 5G Testbed Environment

We deploy MONARCH on a Kubernetes-based network slicing testbed comprising containerized core network functions, based on Open5GS [80], and a radio access network consisting of both hardware deployment and software emulation. A detailed description of the hardware and networking setup is provided in Appendix A; in this chapter, we focus only on aspects relevant to slice-centric monitoring.

Slice Realization. Network slices are instantiated by deploying dedicated SMF and UPF instances per slice, while shared control plane functions (*e.g.*, Access and Mobility Management Function (AMF)) remain common across slices. This configuration allows each slice to maintain independent state while allowing realistic resource sharing across the infrastructure.

Monitoring Integration. MONARCH components are deployed alongside the 5G infrastructure as containerized services. MDEs are instantiated per NF instance, enabling telemetry to be collected independently for each slice segment. Segment-level collectors (NSSDC instances) ingest metrics from these exporters and forward them to the Data Distribution component.

Slice Scaling and Workload Modes. For scalability experiments (up to 50 slices), we use UERANSIM [44] to emulate Next Generation Node Bs (gNBs) and User Equipments (UEs). For adaptive monitoring evaluation, we replay real-world 5G traffic traces through a physical UE and Software-Defined Radio (SDR)-based gNB to capture realistic traffic dynamics.

3.4.2 Architectural Realization

MONARCH is implemented as a set of containerized services deployed on Kubernetes [114]. Each MONARCH component listed in Table 3.2 is packaged and deployed as a Kubernetes workload: stateless control-plane components run as microservices, stateful telemetry

components such as Prometheus and Thanos run as containerized monitoring services, and MDEs are deployed as sidecar containers next to the monitored NFs. Figure 3.3 illustrates the interaction between the 5G testbed and MONARCH, highlighting how slice-level monitoring spans the RAN, edge, and core segments while maintaining centralized KPI computation.

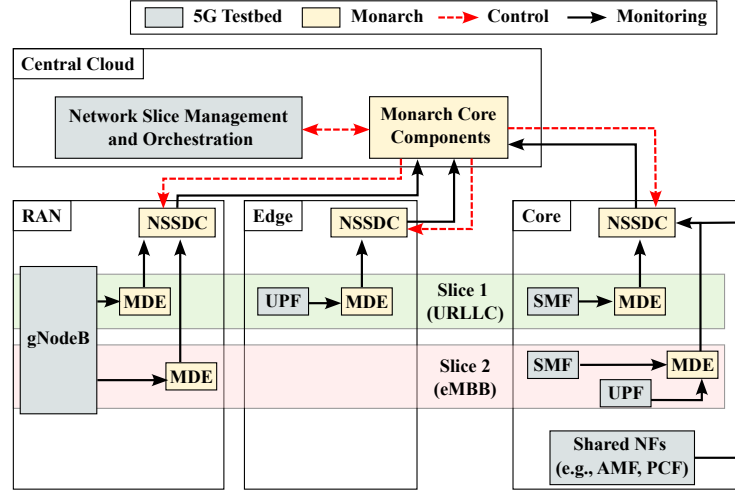


Figure 3.3: **MONARCH prototype integration with the 5G testbed.** The control plane (dashed arrows) propagates monitoring intent to configure segment collectors, while the data plane (solid arrows) exports NF telemetry to the aggregation layer for slice-level KPI computation and retrieval.

Implementation and Deployment. Table 3.2 summarizes how abstract architectural components are instantiated in the prototype. Segment-level telemetry execution is realized using MDEs implemented as Prometheus-compatible exporters [115] and OpenTelemetry-based collectors [117]. Prometheus instances serve as NSSDC components, while Thanos [27] provides federated query and distributed data aggregation across collectors. Slice-level KPI computation and API services are implemented as lightweight Python microservices. These components are logically decoupled from the telemetry pipeline and can be replaced with production-grade implementations (e.g., Go-based services or service-mesh gateways) without altering MONARCH’s architectural separation. The focus of this prototype is on validating slice-centric abstraction and correlation semantics.

MDEs are deployed as sidecar containers alongside network function instances, ensuring that telemetry extraction remains local to each slice segment. The NSSDC instances ingest metrics per segment and forward them to the slice-level aggregation layer. Although

Table 3.2: **Prototype realization of MONARCH’s components.** Each logical module is realized using commodity cloud-native technologies; the mapping reflects an implementation choice rather than an architectural constraint.

Component	Prototype Realization
API Gateway	Python REST service + Thanos Query [27]
Request Translator	Python microservice
Monitoring Manager	Python control service (Kubernetes [114] client)
NSSDC	Prometheus [115]
MDE	Prometheus exporters [115] + OpenTelemetry collectors [117]
Data Distribution	Thanos Query + Sidecar [27]
Data Store	Prometheus TSDB [115] + Object storage (MinIO) [73]

slice-level KPI computation is logically centralized, it operates over distributed collectors via Thanos, preventing ingestion bottlenecks while preserving unified slice abstractions. Adaptive interval control is realized by allowing MDEs to regulate their telemetry collection/export loop. The Monitoring Manager computes the effective reporting policy per monitoring intent, while MDEs act as segment-level executors that enforce updated intervals. This design preserves MONARCH’s architectural separation of policy and execution and avoids reliance on dynamic reconfiguration of Prometheus scrape jobs.

Extensibility. Although our prototype was implemented using Prometheus-compatible tooling, the MONARCH itself is technology-agnostic. Push-based streaming systems, in-network telemetry mechanisms, or sketch-based summaries (*e.g.*, Chapter 5) can be integrated without modifying slice-level abstractions or KPI semantics.

3.4.3 Use-Case Validation

We validate MONARCH through two use-cases implemented on the 5G testbed described in §3.4.1. These use-cases demonstrate how MONARCH’s monitoring API can be used to realize (i) slice-level KPI that requires correlating metrics from different NFs, and (ii) localized NF-level monitoring for diagnostics.

Slice KPI Monitoring. Recall from §2.1.1 that slice realization spans multiple operational identifiers across the control and user planes. We implement slice-level monitoring of average downlink throughput for a given S-NSSAI, as defined in 3GPP TS 28.552 [2]. This KPI requires correlation of control and user plane state: GPRS Tunneling Protocol – User Plane (GTP-U) byte counters at the UPF are indexed by forwarding rules, not

slice identifiers, while slice identity exists exclusively in control plane state at the SMF. Computing slice throughput, therefore, requires resolving the chain: S-NSSAI $\rightarrow$ QFI $\rightarrow$ TEID $\rightarrow$ forwarding rules $\rightarrow$ byte counters, followed by aggregation across all sessions and UPFs serving the slice. Upon receiving a slice-scoped monitoring request, MONARCH performs this multi-stage resolution internally and aggregates counters across all matching sessions and UPFs. From the application perspective, the monitoring intent reduces to a single declarative request: “monitor slice throughput for S-NSSAI X,” with all underlying complexity fully encapsulated within MONARCH. Algorithm 1 summarizes the internal logic executed by the KPI computation module when serving this slice-scoped throughput intent.

Algorithm 1 Slice throughput computation inside MONARCH. Control plane slice/session context is correlated with user plane counters to derive slice-level throughput.

Require: Slice identifier s (e.g., S-NSSAI), time window W

Ensure: Downlink slice throughput $\text{thr}(s, W)$

```

1:  $\mathcal{S} \leftarrow \text{QUERYSMF}(s)$   $\triangleright$  resolve S-NSSAI  $\rightarrow$  active PDU sessions and forwarding rules
2:  $B \leftarrow 0$ 
3: for each session  $u \in \mathcal{S}$  do
4:    $p \leftarrow \text{RESOLVEUPF}(u)$   $\triangleright$  UPF instance serving session  $u$ 
5:    $B \leftarrow B + \text{BYTESDELTA}(p, u, W)$   $\triangleright$  downlink byte delta over  $W$ 
6: end for
7: return  $\text{thr}(s, W) \leftarrow \frac{8B}{|W|}$   $\triangleright$  convert to Mbps in API layer
```

NF-Level Drill-Down. We also implement NF-scoped monitoring of the number of active QoS flows at an SMF, aggregated per 5G QoS Identifier (5QI). Unlike slice-level throughput, this KPI can be derived solely based on control plane session state. Each active session maintained by the SMF contains associated QoS flow descriptors, including their 5QI values. Computing the KPI therefore involves enumerating active sessions, grouping QoS flows by 5QI, and computing per-class counts within the specified time window.

In terms of operational steps, upon receiving an NF-scoped monitoring request, the Monitoring Manager resolves the target SMF instance and deploys an appropriate MDE alongside it. The MDE extracts per-5QI flow counters directly from SMF session context, and the KPI computation module performs aggregation at NF scope before returning results via MONARCH’s northbound API.

Key Takeaway. Both use cases above are realized through the same monitoring interface. Moving from the slice scope, which requires multi-stage metric correlation and

aggregation, to the NF scope, which requires only localized state, does not alter how higher-level applications interact with the monitoring system. MONARCH internalizes the slice semantics and correlation logic within the monitoring layer, eliminating the need for downstream applications to reconstruct distributed network state.

3.4.4 Adaptive Monitoring Realization

Next, we describe how adaptive monitoring is implemented in our prototype. This implementation reflects practical constraints of the Prometheus ecosystem used in our prototype and is not intrinsic to MONARCH.

Export-Side Adaptation. In our prototype, adaptive monitoring is implemented inside the MDE process; essentially following a *push* model. Each MDE continuously observes local KPIs at a fixed minimum interval $T_{\min}$ (*e.g.*, 1s), but adaptively controls the *export interval* to the monitoring backend.

We chose this design for two reasons. First, our experiments showed that Prometheus’s pull model does not support low-latency dynamic reconfiguration of scrape intervals. Reloading scrape configurations (by changing the config files) introduces non-trivial delay and operational complexity. Second, as shown in §3.5.4, resource consumption in MONARCH is dominated by telemetry *ingestion and processing* rather than *metric generation* at the source. Reducing export frequency, therefore, directly targets the primary CPU and overhead bottleneck. Based on these observations, we embed the adaptation logic in the exporter and use Prometheus’s Remote Write endpoint, following a push model.

Policy–Execution Separation. Although the *push* model requires that adaptation logic executes inside the MDE, the policy however, remains centrally defined. The Monitoring Manager determines adaptation parameters ($T_{\min}$, $T_{\max}$, δ , α , β) based on slice-level monitoring intent. These parameters are distributed to the MDEs via control plane updates. Based on these policies, the MDEs then enforce the effective reporting policy locally. This preserves MONARCH’s architectural separation: the Monitoring Manager defines slice-level monitoring policy, while the MDEs act as segment-level executors.

Example. To understand how the above works, consider a monitoring request for slice throughput on S-NSSAI 1-01 with **adaptive** mode enabled. The Monitoring Manager assigns adaptation parameters ($T_{\min} = 1\text{s}$, $T_{\max} = 5\text{s}$) and distributes them to the MDEs responsible for the slice segments. During stable traffic periods, exporters increase their export interval up to $T_{\max}$, reducing telemetry volume. When rapid KPI changes are detected, the MDEs immediately reduce the interval to $T_{\min}$ and export updates. Here, the exporters themselves do not determine these bounds; rather, they enforce the centrally

defined policy. Below, we describe the simple adaptive export algorithm used in our prototype.

Algorithm 2 Exporter-side adaptive monitoring. KPIs are observed at a fixed minimum interval ($T_{\min}$) while the export interval is adjusted based on detected changes.

Inputs: $x(\tau)$: KPI observed every $T_{\min}$; δ : change threshold; $T_{\min}, T_{\max}$: min/max export interval; $\alpha > 1$: increase factor; $\beta > 1$: decrease factor

Output: Exported samples $x_s(\tau)$

```

1:  $T \leftarrow T_{\min}$ ;  $\tau_{\text{last}} \leftarrow 0$ ;  $x_{\text{last}} \leftarrow \perp$ 
2: for each observation time  $\tau$  (every  $T_{\min}$ ) do
3:   observe  $x(\tau)$ 
4:    $\Delta x \leftarrow |x(\tau) - x_{\text{last}}|$   $\triangleright \Delta x = \infty$  if  $x_{\text{last}} = \perp$ 
5:   if  $\Delta x > \delta$  then  $\triangleright$  rapid change: export immediately, decrease  $T$ 
6:      $T \leftarrow \max(T_{\min}, T/\beta)$ 
7:   else if  $\tau - \tau_{\text{last}} \geq T$  then  $\triangleright$  scheduled export: increase  $T$ 
8:      $T \leftarrow \min(T_{\max}, T \cdot \alpha)$ 
9:   else
10:    continue
11:  end if
12:   $\text{EXPORT}(x(\tau)); x_s(\tau) \leftarrow x(\tau); \tau_{\text{last}} \leftarrow \tau; x_{\text{last}} \leftarrow x(\tau)$ 
13: end for
```

Adaptive Export Algorithm. Algorithm 2 describes the exporter-side adaptation logic. Each MDE observes the local KPI at a fixed minimum interval $T_{\min}$ (e.g., 1s), ensuring that short-lived dynamics are always detectable locally. However, values are exported to the monitoring backend only when required. The algorithm maintains a dynamic export interval T and tracks the last exported value x_{last} . At each observation step, there are two main cases: (i) if the KPI changes significantly ($\Delta x > \delta$), the exporter reacts immediately by reducing T and exporting the new value. This ensures rapid response during traffic bursts. (ii) otherwise, export occurs only when the elapsed time since the last export exceeds the current interval T . In stable periods, T is gradually increased (up to $T_{\max}$); this reduces export frequency and hence ingestion overhead.

This design ensures that adaptation operates on the export path rather than the observation path. As shown in §3.5.4, ingestion and downstream processing dominate CPU consumption. By reducing export frequency during stable periods while preserving fast local observation, the Algorithm 2 directly addresses the main resource bottleneck without sacrificing responsiveness.

Control Semantics. The Monitoring Manager decides the adaptation parameters in Algorithm 2 ($T_{\min}, T_{\max}, \delta, \alpha, \beta$) per monitoring request. These parameters are then sent to MDEs via control plane updates. The MDEs execute the adaptation logic locally and enforce the effective export interval. Thus, policy remains centralized, while enforcement occurs at segment-level telemetry executors. This separation also allows the policy computation in the Monitoring Manager to evolve independently (*e.g.*, rule-based heuristics as well as learning-based controllers such as Reinforcement Learning (RL)).

The prototype shows how MONARCH’s main principles, such as slice-centric abstractions, cross-domain correlation logic, and adaptive monitoring, can be realized in a practical cloud-native 5G deployment. Next, we evaluate this system to quantify its scalability, resource efficiency, and monitoring accuracy.

3.4.5 Design and Implementation Challenges

Realizing MONARCH on a cloud-native 5G testbed surfaced several challenges that arise from the gap between the architectural model and the capabilities of available infrastructure and tooling. We describe the most significant of these and the design decisions taken in response, as they clarify which aspects of the prototype reflect MONARCH’s architecture and which reflect constraints of the underlying tooling.

Dynamic reconfiguration under a pull-based collector. MONARCH’s adaptive monitoring requires that the effective reporting interval of a monitored KPI be adjusted at runtime (§3.3.4). The Prometheus-based collection substrate used in the prototype follows a pull model in which scrape intervals are defined in static configuration, and our experiments showed that reloading this configuration introduces non-trivial latency and operational overhead. Low-latency interval adaptation is therefore difficult to achieve through scrape reconfiguration alone. To resolve this, adaptation logic is relocated to the MDE, which observes the local KPI at a fixed minimum interval and regulates the export interval through Prometheus Remote Write, following a push model (§3.4.4). This decision is an implementation response to the collector’s constraints rather than a property of the architecture, and it preserves the separation between centrally defined policy and segment-level enforcement.

Absence of mature slice orchestration. The Request Translator resolves slice-level monitoring intent into concrete infrastructure targets, which requires access to slice-to-instance bindings from an external slice orchestrator (§3.3.1). In practice, mature open-source orchestrators for dynamic, cloud-native 5G slicing are not yet available; existing platforms such as ONAP [116] were designed primarily for Virtualized Network Function

(VNF)-based deployments, and their slice management capabilities are not well established for this setting. In the prototype, these bindings are therefore supplied through explicit configuration. This substitution isolates the orchestration dependency at a single, well-defined interface, so that automated resolution through orchestration APIs can be incorporated without altering the remainder of the architecture. Such integration with emerging cloud-native orchestration platforms remains an orthogonal problem and is left to future work.

Aggregation consistency under variable intervals. Adaptive monitoring changes the cadence at which telemetry reports are produced, which can bias KPI estimates when measurements collected at differing intervals are aggregated without accounting for their durations (§3.3.4). Preserving correct KPI semantics under runtime interval changes therefore required decoupling monitoring policy from KPI definition. MONARCH maintains the semantic interpretation of each KPI in a registry and associates every reported value with the interval over which it was computed, which allows rate-based KPIs to be aggregated as duration-weighted averages that remain invariant to the sampling interval. This ensures that adaptive monitoring reduces telemetry volume without compromising the correctness of the computed KPIs.

Prototype scope. Finally, several components, including the API Gateway and the KPI computation modules, are implemented as lightweight Python services to support rapid prototyping (§3.4.2). These services are logically decoupled from the telemetry pipeline and may be replaced with production-grade equivalents, such as compiled microservices or service-mesh gateways, without altering the architectural principles evaluated in this chapter. The prototype is intended to validate slice-centric abstraction and correlation semantics rather than to provide a performance-optimized implementation.

3.5 Evaluation

In this section, we evaluate whether MONARCH can sustain slice monitoring at scale without incurring prohibitive resource or latency costs. In particular, we examine: (i) scalability with increasing slice count (§3.5.2), (ii) robustness of telemetry ingestion under high metric cardinality (§3.5.3), (iii) resource efficiency (§3.5.4), (iv) scalability of centralized slice-level query processing (§3.5.5), and (v) the accuracy-overhead trade-off enabled by adaptive monitoring (§3.5.6).

All experiments are conducted on our 5G slice testbed described in §3.4.1.

3.5.1 Experimental Setup

Network slicing scenario. We create multiple network slices, ranging from 1 to 50, with each slice having a separate SMF and UPF, and 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 §3.4. Each scenario is repeated 10 times to ensure robustness, and the average across the results is presented.

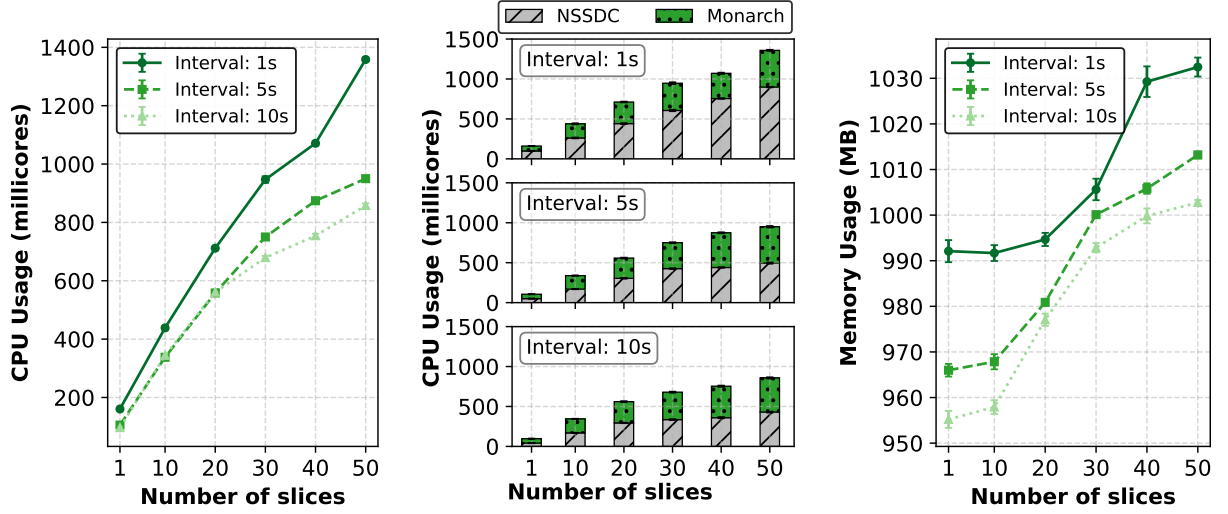
Dataset. To simulate real-world conditions, we utilize traffic traces from a public 5G dataset [24]. These traces include diverse 5G applications, such as cloud gaming, live streaming, and buffered video streaming. We replay these traffic traces over our 5G testbed (*cf.* Appendix A) and employ MONARCH to monitor the network slice KPIs. The efficacy of the adaptive monitoring algorithm is evaluated against these realistic traffic patterns and KPIs.

Evaluation Metrics. We evaluate MONARCH using the following metrics:

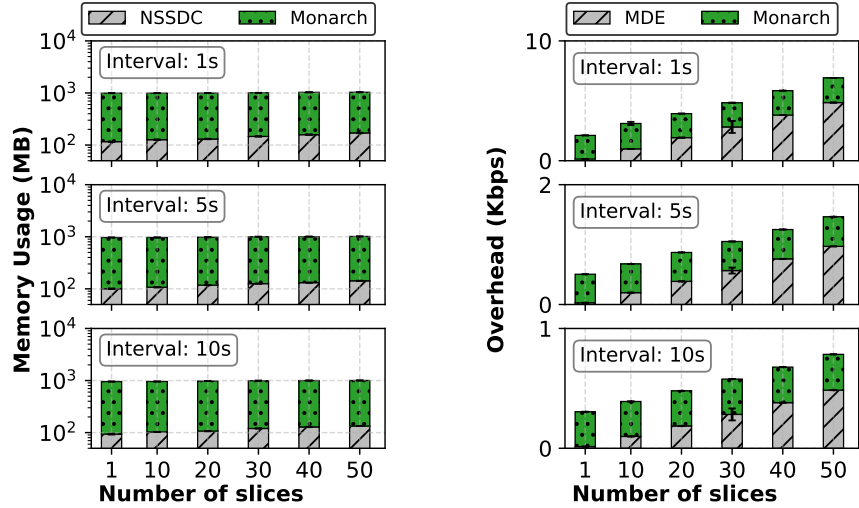
- **Resource usage:** CPU (in millicores) and memory consumption of MONARCH components, obtained via Kubernetes [114] API and cAdvisor [41] metrics.
- **Monitoring overhead:** Volume of monitoring traffic (bytes/sec) generated by MONARCH, including data exported by MDEs and inter-component communication.
- **Ingestion time:** Time required to collect monitoring data from network functions, measured using Prometheus [115] `scrape_duration`.
- **Query latency:** E2e response time for slice-level monitoring queries issued to MONARCH’s API Gateway, measured under varying slice counts and query complexity.

3.5.2 Scalability with Slice Count

We start our evaluation by asking: *How does MONARCH scale with the number of slices?* (R1, §3.2.3). To evaluate MONARCH’s scalability, we vary the number of active slices from 1 to 50 while considering different monitoring configurations at 1s, 5s, and 10s intervals. For each configuration, we measure the CPU usage, memory consumption, and monitoring



(a) CPU usage scales near-linearly with slice count (b) NSSDC dominates CPU utilization under slice scaling (c) 50× slices leads to $\approx 5\%$ more memory consumption



(d) NSSDC contributes minimally to memory usage (e) Monitoring overhead grows proportionally with slices

Figure 3.4: **MONARCH scales near-linearly with slice count without introducing centralized bottlenecks.** CPU usage and monitoring overhead increase proportionally with the number of slices, while memory consumption remains nearly constant across monitoring intervals. NSSDC dominates CPU utilization, and MDEs account for the majority of monitoring overhead, reflecting MONARCH’s slice-local telemetry architecture.

overhead. The goal of these experiments is to determine whether slice-centric monitoring introduces bottlenecks as the slice count increases.

CPU Usage. Figure 3.4(a) shows that CPU usage increases approximately linearly with the number of slices across all monitoring intervals. Scaling the number of slices by $50\times$ (from 1 to 50) results in an $8.5\times$ – $9\times$ increase in CPU usage, indicating sub-linear growth relative to slice count. We also examine the contribution of the NSSDC component (Figure 3.4(b)). The NSSDC accounts for the largest share of CPU consumption; with 50 slices, NSSDC accounts for 66%, 52%, and 50% of total CPU usage at monitoring intervals of 1s, 5s, and 10s, respectively. This is expected due to the fact that the NSSDC is the primary data collection layer for each network segment. From the figure, we observe that the relative contribution of NSSDC dominates the compute cost under slice scaling. In §3.5.5, we show that horizontal scaling of NSSDC instances mitigates this effect and preserves low query latency even at high slice counts and metric cardinality.

Memory Usage. Figure 3.4(c) shows that memory usage exhibits near-constant behavior as slice count increases. Scaling the number of slices from 1 to 50 ($50\times$) results in a very modest $1.04\times$ – $1.05\times$ increase in memory consumption across monitoring intervals. In contrast to CPU usage, memory growth remains modest because the main memory consumers in MONARCH (e.g., the time-series database and ingestion components) rely on shared data structures that do not scale with the number of slices. While slice count increases telemetry volume, it does not introduce a proportional per-slice in-memory state. We also examine the contribution of the NSSDC to memory usage. As shown in Figure 3.4(d), NSSDC only accounts for a minor fraction of total memory usage. These results indicate that scaling the number of slices does not lead to high memory pressure in MONARCH.

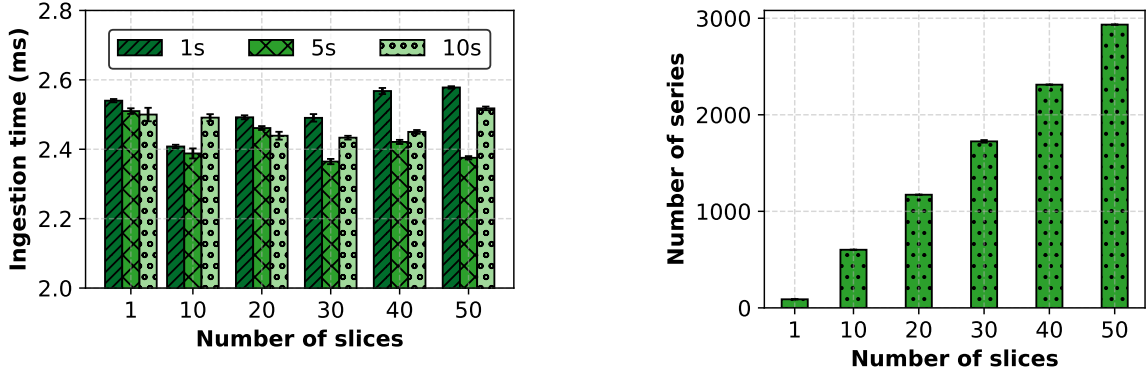
Monitoring Overhead. Figure 3.4(e) demonstrates that monitoring overhead increases approximately linearly with slice count. This is expected for per-slice metric collection. Increasing the number of slices by $50\times$ results in a $2.6\times$ – $3.2\times$ increase in monitoring traffic, depending on the monitoring interval. We further analyze the contribution of MDEs, which are responsible for extracting raw metrics from NFs. With 50 slices, traffic generated by MDEs accounts for 61%–69% of total monitoring overhead. This indicates that monitoring cost is dominated by telemetry extraction rather than internal aggregation or query processing. It also highlights the importance of MONARCH’s centralized slice-centric design, which avoids redundant data extraction when multiple monitoring requests share the same scope.

Take-away. The results show that MONARCH scales approximately linearly with slice count in both CPU usage and monitoring overhead, while maintaining stable memory consumption. MONARCH’s resource usage is dominated by the NSSDC and MDE compo-

nents; this aligns with MONARCH’s architecture, where slice-specific telemetry is collected independently per slice segment. The results do not show indications of super-linear growth in resource consumption, implying that MONARCH’s slice-centric design is able to sustain slice-centric monitoring.

3.5.3 Telemetry Ingestion and Time-Series Behavior

Next, we examine the robustness of MONARCH’s data ingestion pipeline under increasing slice count and time-series cardinality (R3, §3.2.3). In particular, we evaluate (i) data ingestion latency under slice scaling, and (ii) the impact of time-series cardinality and endpoint distribution on ingestion behavior.



(a) Ingestion time remains stable (2.25–2.75 ms) under slice scaling (b) Per-slice telemetry drives linear cardinality growth in the number of monitored time series

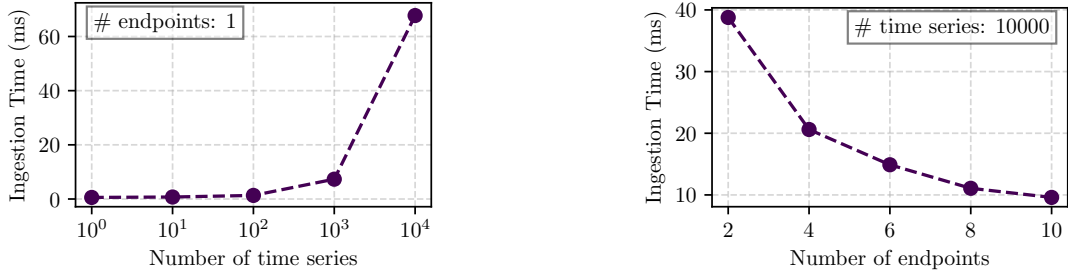
Figure 3.5: **Slice scaling does not destabilize ingestion.** Although the number of time series grows with slice count, ingestion latency remains stable due to distributed telemetry.

Ingestion Latency. Figure 3.5(a) shows that ingestion time remains stable as the number of slices increases from 1 to 50, ranging between 2.25–2.75 ms across monitoring intervals. This indicates that slice scaling does not introduce additional latency in the ingestion pipeline. Figure 3.5(b) shows that time-series cardinality grows approximately linearly with slice count. In slice monitoring, a single metric (*e.g.*, packets/bytes on the UPF interface) may generate multiple time series due to label dimensions such as slice ID, QFI, or NF instance. As the number of slices increases and additional network functions are instantiated, the number of time series also increases accordingly. Despite this increase

in telemetry volume, ingestion latency remains stable; this shows that higher cardinality alone does not degrade ingestion performance.

While monitoring interval affects telemetry volume and monitoring accuracy (discussed in §3.5.6), it has a negligible impact on ingestion latency. These results show that MONARCH’s ingestion pipeline remains robust under slice scaling due to its distributed telemetry collection model. To understand why this is the case, next, we examine how endpoint distribution influences ingestion behavior under high cardinality.

Impact of Time-Series Cardinality and Endpoints. To isolate the factors influencing ingestion latency, we use Avalanche [86] to emulate various stress workloads. In particular, we vary (i) the number of time series exposed by a single endpoint *e.g.*, a UPF exporter emitting per-slice and per-QFI throughput counters, and (ii) the number of endpoints exporting a fixed number of time series, corresponding to multiple NF instances (e.g., several UPFs or SMF) each serving different slices.



(a) Single-endpoint cardinality causes saturation (b) Distributing endpoints restores stability

Figure 3.6: Distributed telemetry mitigates ingestion bottlenecks. Concentrating high time-series cardinality behind a single endpoint leads to sharp latency increases, whereas distributing telemetry across multiple endpoints preserves ingestion performance.

Figure 3.6(a) shows that ingestion latency increases gradually as the number of time series exposed by a single endpoint grows, and rises sharply beyond approximately 10^3 series. This effectively indicates endpoint saturation. To examine whether this limitation is due to the *total cardinality* or due to its *concentration*, we fix the number of time series to 10^4 and vary the number of endpoints from 2 to 10. Figure 3.6(b) demonstrates that distributing 10^4 time series across multiple endpoints reduces ingestion latency and stabilizes performance.

This behavior directly reflects MONARCH’s telemetry execution model. In MONARCH, MDEs are instantiated per network function (and therefore per slice segment), preventing large per-slice cardinality from being concentrated behind a single MDE. As slice count

increases, telemetry is thus partitioned across multiple MDEs and endpoints, avoiding the saturation effects observed in the single-endpoint scenario.

Take-away. MONARCH maintains stable ingestion latency under slice scaling despite near-linear growth in time-series cardinality. Saturation occurs only when high-cardinality telemetry is concentrated behind a single endpoint. MONARCH instantiates MDEs per network function and thus distributes telemetry across endpoints to avoid such centralized bottlenecks.

3.5.4 Monitoring Intervals and Resource Efficiency

While the previous subsection established predictable scaling with slice count, we now examine how monitoring intervals influence resource consumption. In particular, we identify which elements of the telemetry pipeline dominate CPU consumption and monitoring overhead, and which control knobs provide meaningful resource savings (R4).

As shown in §3.5.2, CPU consumption under slice scaling is dominated by the NSSDC component, while monitoring overhead is primarily attributable to MDEs. We next analyze how generation and ingestion intervals influence these dominant costs.

Table 3.3: **Impact of generation vs. ingestion interval on CPU usage.** Increasing the ingestion interval provides an effective reduction in CPU consumption.

Generation (s)	Ingestion (s)	CPU (mCores)	Δ vs. 1s/1s
1	1	1382	—
1	10	854	−38%
10	10	822	−41%

Impact of Generation vs. Ingestion Frequency. Monitoring interval directly influences telemetry volume and processing cost. As shown in Table 3.3, increasing the ingestion interval from 1s to 10s reduces CPU usage from 1382 mCores to 854 mCores, accounting for a 38% reduction even when metric generation remains at 1s. In contrast, reducing generation frequency from 1s to 10s while keeping ingestion at 10s yields only an additional 3–4% reduction (from 854 to 822 mCores). This indicates that ingestion and downstream processing dominate CPU cost, whereas optimizing metric generation has a marginal impact. The dominant contribution arises from telemetry ingestion, parsing, aggregation, and storage operations performed within the monitoring layer.

We observe similar behavior in monitoring overhead: increasing the ingestion interval proportionally reduces exported telemetry traffic, while reducing the generation frequency provides negligible additional savings. This further confirms that resource consumption is primarily driven by ingestion frequency rather than metric production at the source.

Take-away. Resource consumption in MONARCH is dominated by telemetry *ingestion and processing* rather than *metric generation*. Controlling ingestion frequency, therefore, provides the most effective means for reducing CPU usage and monitoring overhead. This observation directly motivates the adaptive monitoring mechanism described in §3.3.4, which dynamically regulates the ingestion interval to balance monitoring fidelity and resource efficiency.

3.5.5 Query Processing and Slice-Level Aggregation

Since MONARCH performs cross-domain correlation and aggregation centrally (R3), this section answers two questions. First, does exposing pre-resolved slice-level labels reduce query cost relative to the low-level representation that applications would otherwise query over? Second, given that aggregation is centralized, does query latency scale predictably without becoming a bottleneck? We answer the first with a direct comparison of label representations, and the second along three workload dimensions: datapoint volume, query arrival rate, and horizontal scalability.

Effect of slice-level labeling. We first isolate the benefit of attaching slice-level labels at collection time rather than forcing applications to resolve slice membership at query time. A slice’s telemetry is fragmented across many time series indexed by low-level identifiers (*e.g.*, TEIDs, QFIs, or sessions), and without slice context a slice-scoped query must aggregate over all of them. In a representative deployment with 50 NFs and 100 sessions per NF, this low-level representation exposes ≈ 5000 time series, which a slice-scoped query must filter and aggregate using identifiers such as `pod` and `seid`. In MONARCH, the MDEs and NSSDCs already hold the slice-to-instance context resolved by the Request Translator, so they enrich telemetry with stable slice-level labels (`slice_type`, `slice_id`) as it is collected. The slice association is therefore resolved once during ingestion rather than recomputed on every query, and a slice-scoped query touches only the target slice’s series. As Table 3.4 shows, this reduces slice-query latency by roughly an order of magnitude.

Impact of Datapoint Volume. Having shown that slice-level labeling reduces the *number* of series a query touches, we turn to the second question: how latency scales with query *workload*. Figure 3.7(a) shows query latency as a function of datapoint volume for both simple metric retrieval and complex slice-level queries (*e.g.*, computing the 95th

Table 3.4: **Slice-level labeling reduces query latency.** Session-level labels require slice-scoped queries to aggregate over many network-function/session series, whereas MONARCH exposes pre-resolved slice-level labels.

Representation	Series touched	Query latency
Session-level (<code>pod</code> , <code>seid</code>)	≈ 5000	200–250 ms
Slice-level (<code>slice_type</code> , <code>slice_id</code>)	≈ 100	20–50 ms

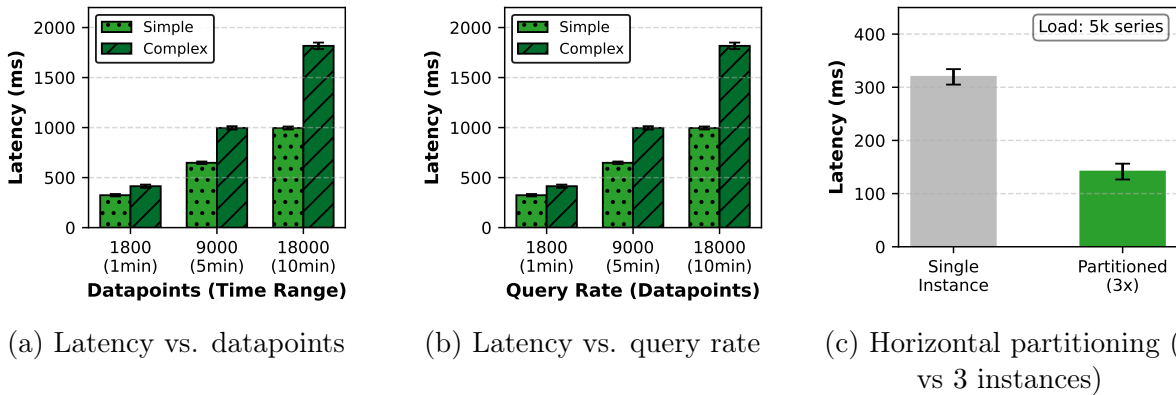


Figure 3.7: **Query latency grows predictably with workload.** Latency increases with datapoint volume and query arrival rate, particularly for complex slice-level KPIs. Horizontal partitioning of the aggregation layer reduces latency under fixed total load.

percentile of slice latency across all active slices). With a 5 s collection interval and 50 slices, a 5-minute query window produces approximately 9000 datapoints; extending this to 10 minutes produces 18000 datapoints. Latency scales proportionally: from the figure, we see that increasing datapoint volume by $2\times$ results in approximately $1.8\times$ higher latency, reflecting the additional aggregation overhead. Complex queries consistently exhibit higher latency than simple retrieval, as expected from the additional correlation logic needed.

Impact of Query Rate. Figure 3.7(b) shows that latency increases with query arrival rate for both query types. At higher arrival rates, complex queries approach approximately 1s latency, which is nearly double that of simple retrieval. This highlights that query complexity compounds under load. Pre-computing derived metrics (*e.g.*, percentiles) during telemetry ingestion could be a possible strategy to mitigate query-time overhead, albeit at the cost of losing raw metric granularity for short-term analysis.

Horizontal Partitioning. To mitigate latency under high load, MONARCH’s Data

Distribution module leverages Thanos [27] to support parallel querying across multiple NSSDC instances. Figure 3.7(c) compares a single-instance deployment against a three-instance partitioned deployment under fixed total load (5000 time series, corresponding to 50 slices). The figure shows that the partitioned deployment achieves substantially lower latency at equivalent resource costs. Partitioning by slice type is used in this case, though alternative strategies (*e.g.*, by geographic region or network segment) are also feasible.

Take-away. Enriching telemetry with slice-level labels at collection time, rather than leaving applications to resolve slice membership over fine-grained identifiers at query time, reduces slice-query latency by up to an order of magnitude in our experiments. Beyond this, centralized aggregation introduces predictable, approximately linear latency growth with datapoint volume and query rate, and horizontal partitioning reduces this growth under fixed load. Together these confirm that centralized slice-level aggregation (R3) does not introduce prohibitive bottlenecks within the evaluated workload range.

3.5.6 Adaptive Monitoring Trade-offs

We evaluate MONARCH’s adaptive monitoring mechanism (§3.3.4) using real-world 5G traffic traces [24], replayed through our testbed. MONARCH computes e2e slice throughput KPIs as described in §3.4. The experiment spans three hours; Table 3.5 reports the aggregate statistics over this duration, while Figure 3.8 illustrates a representative 300s snapshot. The adaptive policy evaluated here operates at the slice level and dynamically adjusts the ingestion interval based on observed throughput variation. The objective of this experiment is not to design a sophisticated controller, but to evaluate whether simple slice-level interval adaptation within MONARCH can improve the accuracy–overhead trade-off under realistic workload dynamics.

Table 3.5: **MONARCH’s adaptive monitoring achieves better accuracy–overhead trade-off compared to commonly used fixed-frequency.** Lower error (MAE, MAPE) and competitive data savings ratio (DSR) result in the highest expected utility (EU).

Strategy	MAE	MAPE	DSR	EU
Fixed 5s	0.0301	0.2351	0.80	3.40
Fixed 10s	0.0760	0.5899	0.90	1.53
Adaptive	0.0232	0.1742	0.76	4.34

Accuracy–Overhead Trade-off. Table 3.5 reports mean absolute error (MAE), mean absolute percentage error (MAPE), data savings ratio (DSR), and expected utility (EU)

for fixed and adaptive strategies. DSR measures reduction in telemetry volume relative to a 1s baseline, representing the highest supported sampling granularity in our deployment. The EU captures the efficiency of error reduction per unit of telemetry savings.

Fixed-interval strategies expose a clear trade-off. A 10s interval achieves the highest telemetry savings (DSR 0.90) but incurs substantially higher error (MAPE 0.5899). Reducing the interval to 5s lowers error (MAPE 0.2351) but increases telemetry volume (DSR 0.80). The adaptive policy achieves lower error than both fixed configurations (MAPE 0.1742) while maintaining substantial telemetry savings (DSR 0.76), resulting in the highest expected utility. These results indicate that static intervals cannot simultaneously balance fidelity and efficiency under bursty slice traffic. In contrast, slice-level interval adaptation enables monitoring precision during dynamic periods while reducing telemetry during stable intervals.

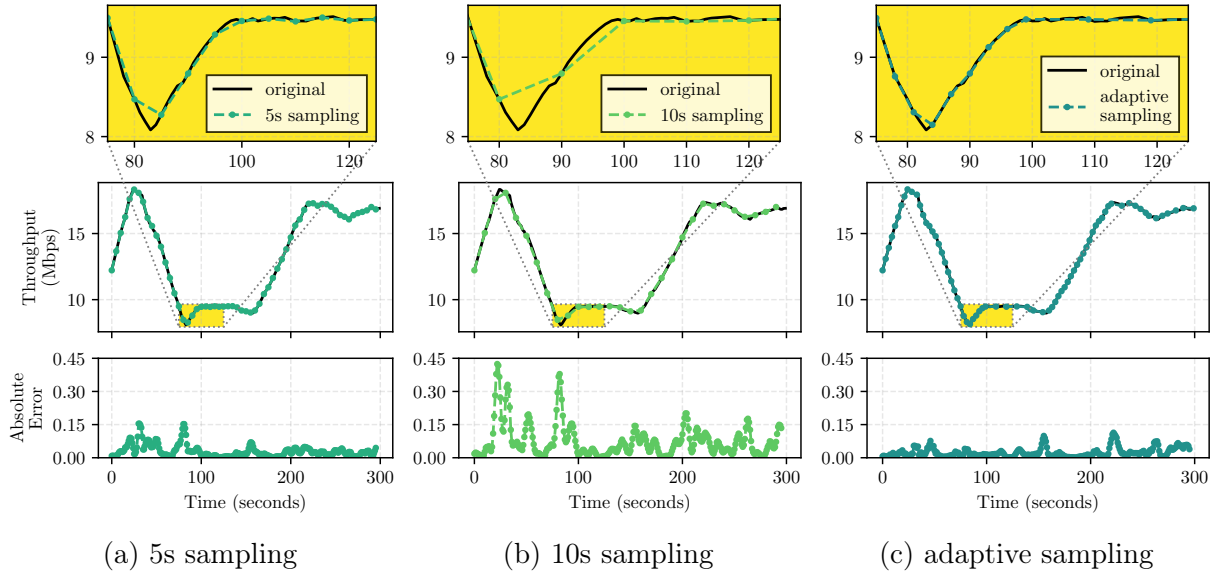


Figure 3.8: Adaptive monitoring preserves fidelity while reducing telemetry volume. Network slice throughput KPI under fixed 5s, fixed 10s, and adaptive intervals (top), with corresponding absolute error relative to 1s baseline (bottom). Adaptive monitoring increases sampling frequency during traffic bursts while reducing it during stable periods, maintaining low error with reduced monitoring overhead.

Temporal Adaptation. Figure 3.8 illustrates a representative 300s window of slice throughput under fixed and adaptive sampling intervals. The top row shows the monitored time series, while the bottom row reports absolute error relative to the 1s baseline. We

observe that fixed 10s sampling fails to capture short-lived throughput fluctuations, leading to pronounced instantaneous error during traffic bursts. Fixed 5s sampling improves temporal fidelity but maintains a constant telemetry cost irrespective of traffic dynamics.

In contrast, the adaptive policy increases sampling frequency during periods of rapid throughput variation and relaxes it during stable intervals. This behavior allows the monitored time series to more closely track the baseline during transient events while reducing telemetry volume during steady-state operation. The observed adaptation aligns with burst-driven traffic dynamics in real 5G workloads, where slice activity alternates between stable and highly variable phases.

Take-away. Adaptive monitoring provides the primary control lever for resource-efficient slice monitoring. Because ingestion frequency dominates CPU and overhead costs (§3.5.4), dynamically regulating this interval yields substantial savings without sacrificing much accuracy. These results validate MONARCH’s ability to satisfy R4 by balancing monitoring fidelity and operational efficiency under realistic 5G workloads.

3.5.7 Evaluation Summary

Slice abstraction is not a bottleneck. Some scaling is unavoidable: monitoring more slices means more telemetry streams and more processing work. The relevant question is whether the slice abstraction itself becomes a centralized bottleneck, and the results indicate it does not. Scaling from 1 to 50 slices ($50\times$) increases CPU usage by only $8.5\text{--}9\times$ and monitoring traffic by $2.6\text{--}3.2\times$, while memory remains essentially constant ($1.04\text{--}1.05\times$) and ingestion latency stays flat at $2.25\text{--}2.75\text{ ms}$ (§3.5.2, §3.5.3). The growth appears approximately linear because the *marginal* cost per slice is roughly constant; every *ratio*, however, sits far below $50\times$ because the fixed baseline, *i.e.*, shared NFs, inter-component communication, and shared ingestion structures, is amortized across slices. No measured dimension grows super-linearly within the evaluated range. MONARCH thus centralizes slice semantics while keeping telemetry execution distributed, without turning slice-level monitoring into a centralized bottleneck.

The cost is in telemetry execution. Where overhead does appear, it concentrates in telemetry extraction, export, and ingestion rather than in the slice abstraction. Figures 3.4(e) and 3.4(b) show that MDEs account for most monitoring traffic and NSSDCs for most CPU usage, and Figure 3.6 shows why placement matters: high-cardinality telemetry is expensive when concentrated behind a single endpoint but stable when distributed across endpoints. Consistent with this, Table 3.3 shows that increasing the ingestion interval reduces CPU usage by 38%, far more than reducing local metric generation (3–4%). MONARCH therefore

does not make slice visibility free; it makes its cost visible, localizable, and controllable, through where telemetry is placed, how it is scoped, and how often it is reported.

Centralized KPI queries remain tractable. A distinct concern is whether computing slice-level KPIs centrally creates a query-time bottleneck. In §3.5.5, Table 3.4 shows that exposing pre-resolved slice-level labels substantially reduces slice-query latency compared to querying over session-level labels. Figures 3.7(a) and 3.7(b) show query latency growing predictably with datapoint volume and query rate (a $2\times$ increase in datapoints yields $\approx 1.8\times$ latency), while Figure 3.7(c) shows that horizontal partitioning reduces latency under fixed total load. This supports computing slice-level KPIs inside the monitoring layer, rather than forcing each downstream control application to reconstruct slice state independently.

Reporting interval is one knob, not the whole problem. Finally, §3.5.6 shows that the reporting interval can serve as a runtime control knob: Table 3.5 shows that a simple adaptive policy improves the accuracy–overhead trade-off relative to fixed monitoring intervals. But this policy adapts only the reporting interval of a single monitored slice KPI. It does not decide how a shared monitoring budget should be allocated across multiple slices with heterogeneous SLAs; this fidelity-allocation problem is addressed by SLICESCOPE in Chapter 4.

3.6 Discussions and Limitations

Scaling beyond the evaluated workload. Our evaluation scales MONARCH to 50 slices (§3.5), whereas carrier deployments may host considerably more. Rather than extrapolating a single performance curve, we use the decomposition of resource cost in the evaluation to identify the scaling pressures most likely to dominate beyond the evaluated range.

First, the slice abstraction itself is not the dominant scaling cost. CPU usage and monitoring traffic are driven primarily by the NSSDCs and MDEs, rather than by the northbound API or the slice-level abstraction. Within the evaluated range, CPU usage and monitoring traffic grow sub-linearly with slice count, while memory consumption remains nearly constant. The cost of scaling therefore appears mainly in the telemetry execution tier, which can be placed and scaled independently, rather than in the centralized semantic layer.

Second, ingestion bottlenecks depend on where cardinality is concentrated. The experiments in §3.5.3 show that a single endpoint saturates only when it exposes more than approximately 10^3 time series, whereas the representative slice-level labeling experiment yields on the order of 10^2 series per slice (Table 3.4). This indicates that endpoint fan-in,

rather than global time-series cardinality alone, is the critical deployment concern. In the evaluated deployment, MDEs are placed per NF, which partitions telemetry across endpoints and keeps each endpoint below the observed saturation knee. In larger deployments, the same principle implies that visibility should be extended by distributing telemetry across additional MDE endpoints, rather than by concentrating more slice and session series behind a small number of exporters.

Third, the remaining costs, namely aggregate export and query-time aggregation, are exposed by MONARCH as deployment-level control points. Export traffic is governed primarily by the reporting interval, which under the push-based export model coincides with the effective ingestion interval. Table 3.3 shows that increasing this interval reduces CPU usage far more effectively than reducing local metric generation frequency. Query-time aggregation, in turn, can be reduced by partitioning the aggregation layer, as shown in Figure 3.7(c). Finally, the MDE interface allows lower-overhead collectors to replace scrape-based exporters without changing the slice-level abstraction.

While these observations are based on measurements at 50 slices, the underlying cost decomposition is structural, and we expect it to hold as the deployment scales. Validating this behavior at carrier scale, where per-slice cardinality, label dimensionality, and per-NF sharing patterns are deployment-dependent, remains future work.

NWDAF and MONARCH synergy. Network Data Analytics Function (NWDAF) [4] is a standardized 5G core function focused on analytics and network data exposure. While the NWDAF primarily targets core-network data sources, MONARCH extends monitoring visibility across all segments of an e2e slice, including transport and RAN segments. In hierarchical or distributed NWDAF deployment models (as in 3GPP Release 17), MONARCH’s NSSDC component aligns with edge-level data collectors, while its centralized KPI computation complements NWDAF’s aggregation and analytics functions. Thus, a combined deployment can leverage MONARCH for submission of monitoring requests, data collection, and slice-level KPI computation, while the NWDAF operates as a downstream consumer and produces network analytics.

Integrating efficient and heterogeneous collectors. MONARCH relies on its MDEs to extract monitoring data from the available instrumentation sources, and the MDE interface is the principal mechanism through which more efficient or domain-specific collectors are incorporated. In the prototype, MDEs are implemented using Prometheus-compatible exporters and OpenTelemetry collectors, as these integrate naturally with Kubernetes and cloud-native monitoring pipelines. The architecture does not, however, require scrape-based collection. A collector that extracts metrics through a lower-overhead mechanism, such as eBPF-based instrumentation in the RAN or 5G core, can be integrated behind the

same MDE interface without modification to the remainder of the architecture. In such a deployment, the collector improves the efficiency or scope of local metric extraction, while MONARCH continues to provide the slice-aware functions: monitoring-request translation, context enrichment, cross-domain correlation, and e2e slice KPI computation. Existing systems illustrate the range of collectors the interface can accommodate. 5GC-Observer [55] and Janus [38] demonstrate techniques for extracting core- and RAN-specific metrics, and xApps [77] can serve as RAN-side telemetry sources realized as RAN-specific MDEs. The sketch-based summaries developed in Chapter 5, for instance, provide a low-overhead source of fine-grained user-plane evidence that can be exposed to MONARCH through an MDE. Such collectors are therefore complementary to MONARCH rather than alternatives to it: they reduce the cost of extracting particular measurements, whereas MONARCH determines how those measurements are associated with slices and exposed to control applications.

Request translation and orchestration. MONARCH’s request translation assumes access to slice-to-instance bindings from an external slice orchestrator. In our prototype, this binding information is provided through explicit configuration due to the dearth of mature open-source slice orchestration platforms. This is not a limitation of MONARCH; rather, it reflects the current state of open-source slice orchestration: existing platforms such as ONAP [116] were designed primarily for VNF-based deployments, and their slice management capabilities are not well-studied for dynamic, cloud-native 5G environments. Therefore, automated resolution of slice context through orchestration APIs is an orthogonal open problem; we consider such integration with emerging cloud-native orchestration platforms as future work.

3.7 Chapter Summary

To realize the goal of efficient slice monitoring, this chapter addressed a key architectural gap: the disconnect between e2e network slice KPIs and the slice-agnostic telemetry produced by individual NFs. We presented MONARCH, a slice-native monitoring architecture that closes this gap by (i) elevating slices to first-class monitored entities, (ii) providing a unified northbound API to express slice-level monitoring intent, and (iii) centralizing cross-domain correlation, e.g., joining control plane context with user-plane metrics within the monitoring layer. We prototyped MONARCH using cloud-native technologies and integrated it with a 5G network slicing testbed (Appendix A). Based on this prototype, we validated MONARCH using concrete use-cases: computing network slice throughput KPI as well as NF-level session counts.

Through experiments on this prototype, we demonstrated that MONARCH scales well

with the number of slices, and is able to maintain bounded ingestion latency with increasing slice counts. Finally, based on real 5G traffic traces, we highlighted the benefits of adaptive monitoring for slice KPI monitoring. These results establish MONARCH as the architectural foundation for this dissertation’s subsequent contributions. In the next chapter, based on MONARCH’s slice-level abstractions, we show how SLA-aware control policies can be used to budget monitoring fidelity.

Chapter 4

SLICESCOPE: SLA-Aware Network Slice Monitoring

4.1 Introduction

Chapter 3 addressed the architectural abstraction gap in network slice monitoring by introducing MONARCH, which elevates slices to first-class monitored entities and enables slice-level Key Performance Indicator (KPI) computation. This architectural foundation establishes slice-level observability; we now build on it to address the *control gap* identified in §2.3: *how should monitoring fidelity be allocated across slices when telemetry resources are limited?*

As discussed in §1.1.2, modern networks may host dozens to hundreds of concurrently active slices [12, 13, 23], each governed by distinct Service Level Agreements (SLAs). For example, Ultra-Reliable Low-Latency Communications (URLLC) slices are expected to provide very low loss rates (e.g., $< 0.001\%$) and stringent end-to-end (e2e) latency (e.g., 1–5 ms), while Enhanced Mobile Broadband (eMBB) slices demand high throughput (e.g., up to 1000 Mbps) but tolerate higher loss (e.g., $< 1\%$) and latency (e.g., 10–50 ms) [7, 8, 10]. These contrasting requirements impose fundamentally different monitoring demands across slices. To ensure compliance with these SLAs, network operators must *continuously* monitor, in *real time*, the relevant SLA metrics for *every active slice*. Operators must know immediately if the per-packet latency of a latency-critical slice exceeds its acceptable threshold to trigger mitigation actions.

To see why this is challenging, consider a network with 500 active slices. While these slices may be broadly categorized into URLLC and eMBB types, each slice *instance* is

governed by a unique SLA, and the set of active slices may *dynamically change over time* due to slice churn and workload variation. The operators of the 5G network need to monitor each slice to ensure that its traffic, in aggregate, meets the unique performance requirements specified in that slice’s SLA. For instance, if 10 out of the 500 slices have very strict SLAs for e2e latency, the operator could decide, for example, to use a packet-level telemetry mechanism such as In-Band Network Telemetry (INT) [119], which embeds measurement data within packet headers, to monitor e2e latency. They could then collect INT information on every packet belonging to those 10 slices but less frequently for the rest, keeping the total overhead within budget.

Now, suppose the set of active slices changes such that 100 out of the 500 slices have very strict SLAs for e2e latency, and collecting telemetry information for every packet of all these 100 slices would exceed the total monitoring budget. In this case, the best strategy to allocate this limited budget could be to collect telemetry information from critical slices at a “base” frequency, but adjust it per slice if e2e latency fluctuations in a particular slice require more frequent INT collection. That is, in the presence of a dynamically changing set of slices, there is no “one-size-fits-all” monitoring strategy – as the set of active slices, their SLAs, and their traffic patterns change, the monitoring system needs to continuously revisit how it allocates its monitoring budget among the active slices in the network to effectively detect SLA violations in real time. This motivates Research Question RQ2:

How can monitoring fidelity be dynamically regulated across slices with heterogeneous and evolving SLAs, ensuring high-precision visibility for critical services while strictly adhering to global resource budgets?

Closed-loop control formulation. We approach this question by modeling the monitoring system as a *closed control loop* that continuously observes the set of active slices, their SLAs, and relevant traffic metrics, and *tunes* the data plane to collect the “right” amount of telemetry for each slice. Realizing this control loop in practice raises three key technical challenges.

The first key challenge is to formalize the underlying resource-allocation problem in a way that is *independent* of the specific data plane mechanism used for telemetry collection. This abstraction enables systematic reasoning about control plane design and helps identify the minimal properties that a data plane primitive must satisfy to enable slice- and SLA-aware closed-loop control of monitoring resources. At a high level, our formulation introduces a “tuning” variable $k_{s,m}$, defined per slice s and SLA metric m , along with two functions, $E(k_{s,m})$ and $\Gamma(k_{s,m})$, that capture the monitoring error and overhead, respectively, for a given choice of $k_{s,m}$. To make effective allocation decisions, the control plane must accurately

estimate E and Γ , which depend on both the monitoring primitive and the characteristics of the active traffic in the network.

This leads to the second challenge: estimating these functions accurately is difficult since their behavior varies with both the primitive and traffic dynamics. For example, the bandwidth overhead of change-triggered selective INT – which collects telemetry information only when metric fluctuations exceed a threshold [26, 102] – depends on the variability of the monitored metric, which in turn depends on the characteristics of interacting traffic streams in a network (e.g., extent and timing of bursts). Similarly, in sketch-based monitoring primitives, the collision probability that affects measurement accuracy depends on the active traffic streams [30]. As such, in practice, E and Γ must be continuously learned from data plane observations. This, however, is challenging since, at any moment, only one configuration of $k_{s,m}$ values is deployed in the data plane, while the control loop requires estimates of E and Γ across a *range* of configurations to choose the most effective operating point.

Finally, the third challenge concerns the design of the data plane primitive itself. Because E and Γ define a trade-off space between accuracy and overhead that the controller navigates per slice, the primitive must (1) expose a sufficiently wide range of operating points within this space, (2) support configurability at slice granularity, and (3) allow transitions between these operating points at runtime. As an extreme example, a primitive that always samples one out of ten packets would be unsuitable, since E and Γ would have the same value regardless of $k_{s,m}$, slice s , and SLA metric m . We discuss these properties in detail in §4.3.3.

SLICESCOPE overview. To demonstrate the feasibility of addressing these challenges and the benefits of slice-aware and SLA-aware closed-loop control, we design and implement an INT-based monitoring system, called SLICESCOPE, that realizes this abstraction in practice. We choose a INT-based data plane because slice monitoring in modern networks requires *continuous, real-time* measurement of per-packet performance metrics, rather than aggregate statistics such as percentiles or averages. Moreover, SLA metrics like latency, jitter, and loss are increasingly defined *end-to-end*, rather than per-hop [7, 8, 10].

INT enables network switches to add and update performance metrics to individual packets as they traverse the network [119], exposing per-packet e2e performance metrics. As such, it is well-suited for satisfying these requirements. INT, however, is known to incur non-negligible bandwidth overhead [15]. As such, SLICESCOPE opts for a change-triggered selective INT approach that enables trading off telemetry collection frequency (i.e., bandwidth overhead) against monitoring accuracy, while providing the flexibility and granularity required for slice-aware and SLA-aware closed-loop control. While we focus on a INT-based implementation in this chapter, the control framework is general and could be

extended to other data plane monitoring mechanisms, such as probabilistic INT or sketches, which we leave for future exploration.

Summary of contributions. This chapter makes the following contributions:

- We formalize SLA-aware slice monitoring as a closed-loop control problem (§4.3), by introducing an abstract optimization framework parameterized by tuning knobs $k_{s,m}$ and trade-off functions $E(k_{s,m})$ and $\Gamma(k_{s,m})$. This formulation is independent of the specific telemetry primitive and identifies the minimal requirements a data plane primitive must satisfy, which we term the Telemetry Primitive Contract (TPC).
- We present SLICESCOPE (§4.4), a system that realizes our framework by implementing the TPC with change-triggered INT. SLICESCOPE introduces: (1) per-slice state tracking for slice-level control, (2) composable e2e semantics for bounded error guarantees, and (3) mathematical models that enable the control plane to estimate the $E(k_{s,m})$ - $\Gamma(k_{s,m})$ trade-off for different knob configurations in the design space.
- We demonstrate the feasibility and benefits of slice-aware and SLA-aware closed-loop control by implementing SLICESCOPE on Intel Tofino switches and evaluating it on a testbed with emulated Fifth Generation (5G) traffic and through large-scale simulations with 300 slices exhibiting diverse SLA targets and traffic patterns (§5.6).

Chapter roadmap. The remainder of this chapter is organized as follows. Section 4.2 motivates the need for SLA-aware slice-level control by examining SLA diversity and resource constraints in modern sliced networks. Section 4.3 formalizes monitoring fidelity regulation as a closed-loop control problem and introduces our abstraction based on tuning knobs and error-overhead trade-offs. Section 4.4 presents SLICESCOPE, our INT-based realization of this framework and its Telemetry Primitive Contract (TPC). Section 4.5 evaluates the system on programmable switches and large-scale simulations. Finally, Section 4.6 discusses limitations and broader implications.

4.2 The Case for SLA-Aware Slice-Level Control

Before formalizing the control framework in detail, we first examine the operational context that motivates it. In particular, we highlight the diversity and dynamics of slice-level SLAs in modern networks (§4.2.1), and explain why these characteristics demand SLA-aware, slice-level monitoring (§4.2.2).

4.2.1 SLA Diversity in Modern Networks

Modern networks increasingly support logical slicing abstractions, where services are provisioned with distinct performance guarantees. We ground our discussion in 5G networks, where slicing is a core architectural primitive. 5G networks are expected to support dozens to hundreds of active slices [12, 13, 23], each with distinct SLAs. There are three broad slice categories: URLLC, eMBB, and Massive Machine-Type Communications (mMTC) (see Table 4.4). However, real-world deployments require more granularity. For example, within the broad URLLC category, discrete automation needs 10 ms latency and 10 Mbps throughput, while process automation needs 60 ms latency and just 1 Mbps throughput [10]. Therefore, operators typically provision many slices, as opposed to just one for each broad category, each with distinct, fine-grained SLAs that must be continuously monitored in real-time for effective service assurance [13, 62]. Other examples of slicing in modern networks include cloud providers providing service-specific connections with distinct performance guarantees (e.g., Azure ExpressRoute [72], Google Dedicated Interconnect [42]), and wide-area networks providing differentiated services [35].

4.2.2 Why Slice-level Monitoring?

From a network operator’s perspective, the natural monitoring unit is a *slice* – the level at which resources are provisioned and SLAs are defined. Slice-level monitoring offers two key advantages.

- *Alignment with SLAs*: Slices are the natural unit of SLA definition and resource provisioning. Operators provision resources at slice granularity and commit to slice-level Quality of Service (QoS) guarantees, making slice-level monitoring the natural level for SLA verification and control decisions. The responsibility of managing *individual flows within a slice* typically rests with the service provider using the slice.
- *Tractability*: Although a network may have millions of flows, the number of active slices is typically in the hundreds [12, 13, 23]. This makes dedicated and accurate (as opposed to approximate) per-slice state manageable in programmable switches. This also makes closed-loop optimization of resource allocation computationally tractable – our evaluations (§4.5.4) show that joint optimization of monitoring resources over 300 slices completes in < 1s, which enables real-time adaptation.

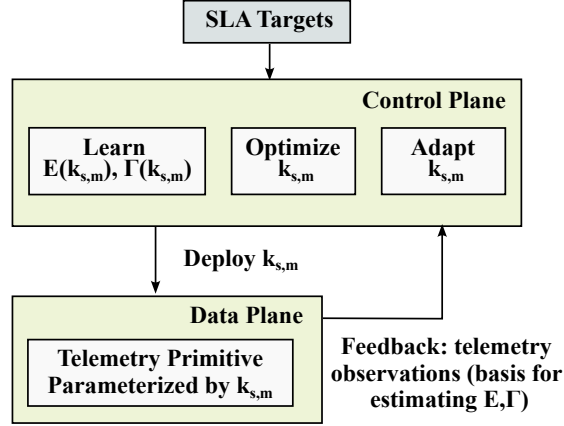


Figure 4.1: **Closed-loop formulation of SLA-aware slice monitoring.** The control plane continuously learns from telemetry observations that reflect slice-level performance, estimates monitoring error $E(k_{s,m})$ and overhead $\Gamma(k_{s,m})$, and tunes per-slice knobs $k_{s,m}$ of the data plane telemetry primitive to satisfy SLA objectives under resource constraints.

4.3 Formalizing Closed-loop SLA-aware Slice Monitoring

As shown in figure 4.1, we model the slice monitoring system as a closed control loop that continuously takes in the set of active slices, their SLAs, and relevant information about their traffic patterns and observed SLA metrics, and tunes the network data plane to collect the “right” amount of telemetry information for each slice. In this section, we describe how we formalize the resource allocation problem that the control loop must solve independent of the particular data plane monitoring mechanism used to collect telemetry information (§4.3.1), how it can adapt to changing network conditions (§4.3.2), and its implications on the properties needed from the data plane monitoring primitives (§4.3.3).

Unlike prior adaptive telemetry approaches that tune monitoring parameters in isolation, our formulation: (i) explicitly couples monitoring accuracy to slice SLAs via per-slice error tolerances, (ii) performs joint optimization across slices under a shared telemetry budget, and (iii) reasons over abstract accuracy–overhead trade-off space rather than mechanism-specific heuristics. These requirements motivate and define the TPC, a minimal abstraction that we introduce to capture the capabilities required for SLA-aware, slice-level closed-loop monitoring independent of any particular data plane realization.

4.3.1 Joint Multi-Slice Resource Allocation

SLA model and monitoring error tolerance. We model a slice SLA for a metric m by its target value α (e.g., max latency of 10 ms). Critical slices (e.g., URLLC) have stringent targets (e.g., $\alpha=5$ ms) while non-critical slices (e.g., mMTC) have more relaxed targets (e.g., $\alpha=40$ ms). Slice SLAs specify targets for e2e metrics but not acceptable *monitoring* error, despite its acknowledged impact [35].

Slice monitoring as a joint resource allocation problem. Each slice–metric pair operates within its own accuracy–overhead trade-off space: allocating more monitoring resources (e.g., computational resources in network devices or telemetry bandwidth) yields higher measurement accuracy for that metric in that slice. However, as we show in our evaluation (§4.5.2), independent per-slice optimization is suboptimal; *joint optimization* enables coordinated trade-offs where slices with error-budget slack can accept reduced fidelity while near-constrained slices receive tighter monitoring. As such, for a network with N active slices and M SLA metrics, the monitoring system must manage $N \times M$ “instances” of this trade-off space while balancing per-SLA monitoring accuracy with network-wide monitoring cost. Specifically, our formulation captures the accuracy-overhead trade-off space in a unified manner with two functions, $E(x)$ and $\Gamma(x)$, that are connected through the “*tuning*” parameter x . This provides our formulation with a unified way to explore the $N \times M$ instances of the trade-off space using dedicated tuning variables for each slice and metric. Concretely, we define a “tuning” variable $k_{s,m}$ for each slice s and SLA metric m , along with two functions, where $E(k_{s,m})$ and $\Gamma(k_{s,m})$ capture the monitoring error and overhead, respectively, for a given choice of $k_{s,m}$. The goal is to select configurations $\{k_{s,m}\}$ that minimize system-wide cost – sum of $\Gamma(k_{s,m})$ across all slices and metrics – while respecting all constraints with respect to SLA monitoring accuracy, i.e., $E(k_{s,m}) \leq \epsilon_{s,m}$.

We instantiate this joint resource allocation problem as an Integer Linear Program (ILP), summarized in Table 4.1. This formulation serves as a concrete realization of the multi-slice allocation challenge by explicitly coupling SLA-driven error constraints, per-slice tuning variables, and a shared telemetry budget. While we utilize an ILP here to accommodate the discrete nature of realizable hardware knobs (e.g., fixed-point registers), the underlying formulation captures a general joint resource allocation problem and admits alternative solution approaches (e.g., greedy or learning-based controllers).

Objective: balancing accuracy and overhead among acceptable joint solutions. The objective is to minimize both expected monitoring error and telemetry overhead across all slices and metrics. Note that, as we discuss below, our formulation enforces that for each slice s and metric m , the monitoring error $E(k_{s,m})$ should not exceed the acceptable threshold $\epsilon_{s,m}$ through dedicated constraints. The objective includes a tunable weight λ

		Symbol	Description
Objective	$\min_K \sum_{s,m} \lambda E(k_{s,m}) + (1 - \lambda) \Gamma(k_{s,m})$	$E(k_{s,m})$	Expected monitoring error for slice s and metric m given $k_{s,m}$ as tuning parameter
		$\Gamma(k_{s,m})$	Expected telemetry overhead for slice s and metric m given $k_{s,m}$ as tuning parameter
Constraints			
C1	$\forall s, m : E(k_{s,m}) \leq \epsilon_{s,m}$		
C2	$\forall s, m : \sum_c z_{s,m,c} = 1$	$\epsilon_{s,m}$	SLA-specific monitoring error tolerance for slice s and metric m
C3	$\forall s, m, c : z_{s,m,c} \in \{0, 1\}$		
C4	$\forall s, m : K_{s,m} = \sum_k z_{s,m,c} \cdot K_{s,m,k}$	λ	Trade-off between accuracy and overhead
		$K_{s,m,c}$	Candidate “knob” values
		$z_{s,m,c}$	Binary variable selecting $K_{s,m,c}$
(a) Optimization formulation		(b) Variable definitions	

Table 4.1: SLA-aware multi-slice joint resource allocation

that operators can adjust to express policy preferences. Specifically, λ allows operators to balance accuracy and overhead among all valid solutions: a larger λ favors more accurate solutions, whereas a smaller λ favors those with lower network-wide overhead. Moreover, including error in the objective (in addition to respecting per-slice tolerances) can prevent undesirable solutions, e.g., highly unbalanced ones that would otherwise just barely satisfy the monitoring requirements of critical slices.

Accuracy and $k_{s,m}$ selection constraints. Each slice s has an error tolerance $\epsilon_{s,m}$ for each metric m , with critical slices having lower tolerances than non-critical ones. Constraint C1 enforces that the chosen $k_{s,m}$ values respect these tolerances. Moreover, our formulation selects $k_{s,m}$ from a discrete set of predefined candidate values. Each $k_{s,m}$ represents a distinct operating point in the accuracy–overhead trade-off of the data plane monitoring primitive. While some primitives may conceptually support a continuous range of $k_{s,m}$ values (§4.4), real-world programmable switches (e.g., Intel Tofino [51]) typically implement them using fixed-point arithmetic with a configurable step size, yielding a finite set of realizable values. For example, with an 8-bit fixed-point representation and step size 0.05, $k_{s,m}$ can take on 256 possible values between 0 and 12.75.

To generalize across different data plane primitives and reflect these hardware constraints, we model the candidate values as $K_{s,m,c}$ and use binary variables $z_{s,m,c}$ to indicate which one is selected. Constraints C2–C4 ensure that exactly one candidate value is chosen per slice–metric pair (s, m) .

The challenge of learning trade-off functions. To make effective allocation decisions, the control plane must accurately estimate the “trade-off functions” E and Γ ,

which depend on both the monitoring primitive and the characteristics of the active traffic in the network. For example, the bandwidth overhead of change-triggered selective INT – which collects telemetry information only when metric fluctuations exceed a threshold [26, 102] – depends on the variability of the monitored metric, which in turn depends on the characteristics of interacting traffic streams in a network. Similarly, in sketch-based monitoring primitives, the collision probability that affects measurement accuracy depends on the active traffic streams [30]. As such, E and Γ must be continuously learned from the data plane observations.

What makes this challenging is that the control plane needs to construct models for predicting how *arbitrary candidate values* of $k_{s,m}$ affect monitoring accuracy E and overhead Γ from *limited observations* of a data plane that is configured with *one set of concrete $k_{s,m}$ values* at that point in time. We cannot simply try out different $k_{s,m}$ configurations. Trying suboptimal settings risks SLA violations during exploration. Nor can the control plane rely on simple heuristic adaptation rules (e.g., monotonic increase/decrease of $k_{s,m}$), which operate on individual slices in isolation and do not reason about the global accuracy–overhead trade-off under a shared telemetry budget. Moreover, with hundreds of slices, various metrics, and large parameter spaces, exhaustive exploration is impractical. As such, the control plane must generalize from limited observations without requiring deployment of every candidate in the data plane. As mentioned above, E and Γ also depend on the particular data plane monitoring primitive used in the control loop. We describe how SLICESCOPE, our realization of the closed-loop control with change-triggered INT as its data plane primitive, addresses this challenge and estimates these functions in §4.4.2.

4.3.2 Adapting to Network Dynamics

The formulation in §4.3.1 provides optimal $k_{s,m}$ values for each slice and metric. However, its effectiveness depends on E and Γ , which can vary over time with changing network conditions. As such, we divide time into *epochs of τ seconds*, and poll the data plane at the beginning of each epoch to update the learned estimates of E and Γ , and subsequently re-run the joint optimization (§4.3.1) to recompute $k_{s,m}$ values. The choice of epoch length τ is a trade-off: it must be short enough to capture network dynamics, yet long enough for the control plane to compute feasible thresholds. We evaluate the impact of τ in §4.5.4.

Update strategies. We need to update the $k_{s,m}$ values within bounded per-epoch computation time. What makes this challenging is that optimization may take a long time to converge, requiring approximate methods or fallbacks to ensure timely decisions (§4.3.2). Our formulation is an Integer Linear Program (ILP), which is NP-complete in the general

case. As such, exact optimal solutions may take exponential time in the worst case. To ensure robustness in the face of uncertain optimization times and enable timely adaptation, we use a two-tiered approach:

1. *ILP with early stopping.* We limit solver runtime to a fraction of the epoch length τ , to ensure that a solution is found in bounded time. Even if the global optimum is not reached, the best current solution is validated for feasibility before being deployed. Our evaluation shows that early stopping provides feasible solutions for τ as small as 5 seconds (§4.5.4).
2. *Greedy heuristic fallback.* If the ILP solver fails to find a feasible solution within the allotted time, we switch to a lightweight greedy heuristic. The heuristic iterates over slices in order of criticality (e.g., URLLC first). For each slice-metric pair, it selects the set of candidate $k_{s,m}$ values (out of the discrete set of possible values) that satisfy that pair’s monitoring error tolerance (based on E). If multiple candidates meet this criterion, the one with the lowest expected overhead is chosen. When no candidate satisfies the tolerance, the heuristic defaults to the most conservative $k_{s,m}$ value (i.e., smallest error) to ensure SLA safety.

This greedy approach prioritizes constraint satisfaction over the error-overhead trade-off: it ensures all $\epsilon_{s,m}$ bounds are met, then minimizes overhead among valid solutions, rather than jointly balancing the two objectives via λ . This preserves SLA guarantees while enabling timely decisions when full optimization is infeasible.

4.3.3 Requirements from the data plane telemetry primitive

For the joint multi-slice optimization to be effective, we need a data plane primitive with particular capabilities in the closed control loop. Inspired by our general formulation of the problem in §4.3.1 and §4.3.2, we derive a minimal set of such capabilities, which we call the *Telemetry Primitive Contract (TPC)* and describe in this section.

(R1) Per-slice per-metric tunability. The control plane optimization relies on navigating the accuracy-overhead trade-off space – through E and Γ – *for each slice*. As such, the data plane telemetry primitive should expose *per-slice, per-metric runtime knobs* that the control plane can adjust each epoch, enabling differentiation across slices with heterogeneous SLAs. Moreover, these knobs should offer a wide range of operating points in this trade-off space, i.e., with a wide variety of values for E and Γ . A primitive that, say, always samples one out of ten packets would be unsuitable, as E and Γ would have the same value regardless of $k_{s,m}$, slice s , and SLA metric m .

(R2) Run-time tunability. Not only should the data plane primitive offer a wide range of operating points in the accuracy-overhead trade-off space, but it should also allow transitions between them *at runtime*. That is, to be able to do closed-loop control, the control plane should be able to adjust data plane knobs, and in turn, monitoring and accuracy for each slice-metric pair without having to recompile the data plane.

(R3) Robust and composable e2e semantics: To allow the control plane to reason about slice-level SLA monitoring fidelity, we need a data plane primitive with telemetry semantics whose local measurements combine into a predictable end-to-end (e2e) accuracy (E) and overhead (Γ) view (through strict analytical bounds or calibrated proxies), allowing the control plane to reason about slice-level SLA monitoring fidelity. This is because modern SLAs are *real-time* and increasingly defined *end-to-end*, as opposed to per-hop [7, 8, 10]. As established in §2.3.2, per-hop primitives cannot reconstruct e2e behavior without per-packet correlation across hops. Tracking SLA metrics such as latency, jitter, and loss per-hop (as statistics or quantiles) is not effective for real-time compliance: without per-packet correlation across hops, we cannot tell whether observations at different hops affect the same or different packets. For example, suppose a discrete automation application with an e2e latency SLA of 15 ms runs in a URLLC slice instance, and we find out 50% of packets exceed 10 ms at switch A , and 50% exceed 5 ms at switch B . Without per-packet correlation, we cannot tell whether these are the *same* packets accumulating delay across hops, or distinct packets encountering independent issues. Only the former indicates a SLA violation. As such, fine-grained (e.g., per-packet) e2e visibility is key to providing robust and composable e2e semantics that can be used as part of continuous closed-loop control.

Summary. The TPC captures the minimal high-level requirements for a data plane telemetry primitive for SLA-aware network slice monitoring, without prescribing *how* a primitive satisfies these requirements. In SLICESCOPE (§4.4), we present one realization of the TPC using change-triggered selective INT; however, alternative primitives could satisfy it through different means.

4.4 SLICESCOPE: Realizing the Closed Loop with Change-Triggered INT

Building on §4.3, SLICESCOPE realizes the real-time SLA-aware closed-loop control using a change-triggered INT-based telemetry in the data plane. We first explain why change-triggered INT is uniquely well-suited for this framework (§4.4.1), then develop the mathematical models for estimating $E(k_{s,m})$ and $\Gamma(k_{s,m})$ (§4.4.2), describe the data plane

implementation (§4.4.3), and discuss additional mechanisms needed for practical deployment (§4.4.4).

Generalizability. Although we instantiate the TPC abstraction using change-triggered selective INT, the abstraction itself is not tied to any specific primitive. Other telemetry mechanisms, such as future probabilistic or hybrid designs, could satisfy the same contract if extended with equivalent slice-level control and e2e semantics. Together, our closed-loop control formulation (control plane) and TPC (data plane) provide a blueprint for any dynamic, SLA-aware monitoring system.

4.4.1 Why change-triggered In-Band-Network Telemetry (INT)?

Existing telemetry mechanisms fall short. Section §4.3.3 defined the abstract TPC in the form of three requirements R1-R3. These requirements rule out several classes of monitoring approaches. *Active probing* uses synthetic probes to collect telemetry for e2e paths. However, the measurements are for the probes themselves, not actual slice traffic. Probes can experience different treatment (e.g., equal-cost multi-path, queueing/scheduling, policing), so their telemetry is not representative of per-slice behavior; consequently, it fails to meet the TPC (R1–R3). *Sketch-based* approaches (e.g., [63, 131]) allow recording traffic summaries and memory-efficient estimates of statistics such as top-k heavy hitters or latency quantiles at a single switch. However, they cannot correlate observations across packets to reconstruct e2e behavior (R3); our evaluation (§4.5.3) shows this yields insufficient accuracy for SLA tracking. Moreover, they are typically not sufficiently tunable at run-time (R2). *Probabilistic sampling* [15] aims at reducing overhead by instrumenting a subset of packets according to a configured sampling probability. While it is possible to adapt the sampling rate per-slice, probabilistically spreading telemetry across packets breaks the packet-path continuity needed for strict e2e semantics (R3). As we show in §4.5.3, this loss of continuity leads to insufficient accuracy for SLA tracking.

INT-based approaches provide the necessary per-packet, e2e correlation. *Path-aggregated* INT [67] carries complete e2e metrics in fixed-size packet headers, ensuring full visibility but at fixed overhead with no run-time per-slice control (R1 and R2). *Change-triggered* INT [26, 102] is more promising: it inserts telemetry only when metrics deviate by more than threshold Δ , naturally exposing a control knob. However, existing designs apply Δ globally and statically, and for one concrete metric, providing no concrete pathway to per-slice run-time adaptation and for a range of e2e common SLA metrics. Thus, our data plane requirements are only partially satisfied ($\sim$ R1–R3). Moreover, the existing designs all lack any form of SLA-aware closed-loop control.

	Data Plane Requirements (§4.3.3)		SLA-aware Closed-Loop Control (§4.3.1 & §4.3.2)		
	Slice-Level Run- Time Tunability (R1-R2)	e2e Semantics (R3)	Trade-off Learning	Multi-Slice Optimization	Runtime Adaptation
No e2e per-packet visibility					
Active probing [16, 49]	✗	✗	✗	✗	✓
Sketches [61, 63, 131]	✗	✗	✗	✗	✗
Probabilistic [15, 113]	✓	✗	✗	✗	✓
e2e visibility, but lacking control					
Path-aggregated [67]	✗	✓	✗	✗	✗
Change-triggered [26, 102]	◐	◐	✗	✗	✗
SLICESCOPE (this chapter)	✓	✓	✓	✓	✓

Table 4.2: **Comparison of representative telemetry approaches against data plane requirements (§4.3.3) and closed-loop control challenges (§4.3.1 and §4.3.2).** Only SLICESCOPE satisfies all criteria. ✓ supported, ✗ not supported, ◐ partial support.

Table 4.2 summarizes these limitations against our data plane requirements (R1–R3), as well as various components of our closed-loop control addressed by SLICESCOPE. Prior work satisfies only a subset of these requirements.

SLICESCOPE extends change-triggered INT to address these gaps. We make Δ a per-slice, per-metric knob $\Delta_{s,m}$, add mechanisms for composable e2e semantics, and enable the control plane to learn trade-offs, allocate thresholds across slices, and adapt them at runtime. This provides the missing link: a controllable primitive that enables the closed loop of learning, resource allocation, and adaptation required for real-time, SLA-aware monitoring.

4.4.2 Mathematical Framework for Modeling E and Γ

Section §4.3.1 identified that the control plane must estimate $E(k_{s,m})$ and $\Gamma(k_{s,m})$ from limited observations, a challenge whose solution depends on the specific telemetry primitive. In SLICESCOPE, our telemetry primitive is changed-triggered INT. Specifically, we define per-slice, per-metric knob $\Delta_{s,m}$, and only add INT information to the packet only when metric m for slice s deviate by more than threshold $\Delta_{s,m}$ from previously collected value. We set $k_{s,m} \equiv \Delta_{s,m}$, and develop the mathematical models that enable this estimation for change-triggered INT. The core challenge is that only one threshold configuration, $\Delta_{s,m}$, is

deployed in the data plane at any given time, yet the optimizer needs to evaluate many candidates to select the best one (§4.3.1). We address this by modeling the relationship between $\Delta_{s,m}$ and telemetry insertion decisions, which in turn, determines both monitoring accuracy and overhead.

Data plane operation (preview). Before developing the models, we briefly preview how change-triggered INT works (full description in §4.4.3): Each switch maintains per-slice state, including the last reported metric value for each metric. When a packet arrives, the switch computes the accumulated difference $\delta_{s,m}$ between the current metric and the last reported value. If $\delta_{s,m} > \Delta_{s,m}$, the switch inserts telemetry and resets $\delta_{s,m}$; otherwise, it skips insertion. This mechanism creates the knob: a smaller $\Delta_{s,m}$ causes more insertions (higher accuracy, higher overhead), while a larger $\Delta_{s,m}$ suppresses insertions, as shown in Figure 4.2.

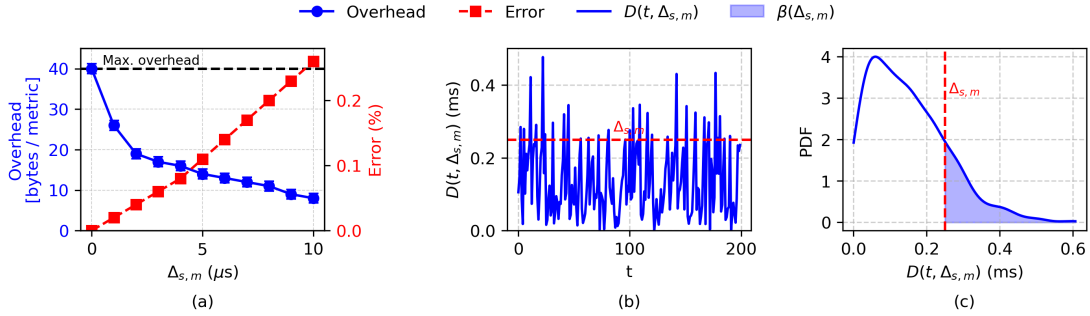


Figure 4.2: **Analytical framework for change-driven telemetry insertion.** (a) Accuracy-overhead trade-off as a function of the telemetry threshold $\Delta_{s,m}$: increasing $\Delta_{s,m}$ reduces telemetry overhead (blue) but increases monitoring error (red). (b) Example trace of the simulated cumulative metric difference $D(t, \Delta_{s,m})$, which mimics how $\delta_{s,m}$ accumulates in the data plane, and triggers telemetry insertion whenever it exceeds $\Delta_{s,m}$. (c) Probability density of $D(t, \Delta_{s,m})$, where the shaded region corresponds to the insertion probability $\beta(\Delta_{s,m})$.

Modeling telemetry insertion probability. Given this mechanism, the key is to quantify the relationship between $\Delta_{s,m}$ and insertion likelihood through the telemetry insertion probability: $\beta(\Delta_{s,m}) = \mathbb{P}(\delta_{s,m} > \Delta_{s,m})$. However, directly modeling $\delta_{s,m}$ creates a circular dependency: the accumulated difference $\delta_{s,m}$ depends on the last reported state, which itself depends on when telemetry was last inserted (i.e., when $\delta_{s,m}$ previously exceeded the threshold currently deployed). This circular dependency makes direct modeling of $\beta(\Delta_{s,m})$ impractical.

Learning underlying packet differences. Our key insight is that while we cannot directly model $\delta_{s,m}$, we can instead work with the underlying packet-to-packet differences that generate these deviations. Specifically, we learn the distribution of successive metric differences $d(t) = E_{curr}^t - E_{last}^t$ from the switch data plane, which captures how the metric changes between consecutive packets at time t . Unlike $\delta_{s,m}$, these packet differences are independent of the chosen threshold. Using this learned distribution, we can simulate, in the control plane, how different threshold values would affect telemetry insertion. We do this by computing the cumulative sum $D(t, \Delta_{s,m}) = |\sum d(t)|$, resetting it whenever it reaches $\Delta_{s,m}$. This mimics exactly how $\delta_{s,m}$ accumulates and resets in the data plane. Thus, the cumulative sum D effectively serves as a proxy for $\delta_{s,m}$. This simulation enables us to construct the probability function for arbitrary thresholds. Figure 4.2 illustrates this process, showing how the telemetry insertion probability $\beta(\Delta_{s,m})$ is computed as the area under the probability density of $D(t, \Delta_{s,m})$.

Having established how different thresholds determine telemetry insertion probabilities, we next examine how these probabilities affect monitoring accuracy along a path. Skipped insertions reduce overhead, but also allow errors to accumulate across switches. Quantifying and bounding this accumulation is critical to ensure SLA error tolerances are respected.

Modeling error accumulation. Consider packets from slice s traversing a path $\mathcal{P}$. Let η_i denote the absolute error between the true and measured value of metric m at switch i . At each hop $i-1$, telemetry is inserted with probability $\beta_{i-1}(\Delta_{s,m})$. The expected error therefore, evolves as:

$$\mathbb{E}[\eta_i] = \mathbb{E}[\eta_{i-1}] + (1 - \beta_{i-1}(\Delta_{s,m})) \cdot \Delta_{s,m}.$$

Intuitively, when telemetry is inserted (probability β_{i-1}), the current switch simply inherits the previous error η_{i-1} . On the other hand, when it is skipped (probability $1 - \beta_{i-1}$), the error grows by up to $\Delta_{s,m}$ in addition to η_{i-1} .

Bounding error growth. A naive worst-case argument assumes every hop skips insertion, yielding an upper bound of $(|\mathcal{P}| - 1) \cdot \Delta_{s,m}$ (one potential $\Delta_{s,m}$ contribution per hop). However, insertions occur with probability $\beta_j(\Delta_{s,m})$ and reset the error. Unrolling the recurrence above (by induction on path length) gives:

$$E(k_{s,m}) = E(\Delta_{s,m}) = \mathbb{E}[\eta]_{UB} = \left((|\mathcal{P}| - 1) - \sum_{j=1}^{|\mathcal{P}|-1} \beta_j(\Delta_{s,m}) \right) \Delta_{s,m}$$

This is a β -discounted (tighter) upper bound: it matches the naive bound exactly when $\beta_j = 0$ for all hops (no insertions), and is strictly smaller whenever any $\beta_j > 0$. Hence,

it avoids the overprovisioning implied by the naive bound. This is what we use for the monitoring error $E(k_{s,m})$ in the closed-loop control’s optimization problem in §4.3.1.

Estimating per-packet monitoring overhead. Having established how errors accumulate, we next quantify the cost of telemetry in terms of expected header overhead. Each monitored packet consists of: (i) a per-packet shim and bitmap (b_0 bits), (ii) per-hop metadata that is always present (b_h bits per hop), and (iii) metric payloads (b bits each) included only when $\delta_{s,m} > \Delta_{s,m}$, i.e., with probability $\beta_j(\Delta_{s,m})$ at hop j (see §4.4.3 for details). The expected per-packet overhead for metric m on slice s is therefore

$$\Gamma(k_{s,m}) = \Gamma(\Delta_{s,m}) = \mathbb{E}[\gamma] = (b_0 + b_h) \cdot |\mathcal{P}| + b \sum_{j=1}^{|\mathcal{P}|} \beta_j(\Delta_{s,m}).$$

The first term reflects the fixed cost that scales linearly with path length, while the second term captures the conditional metric payloads and grows in proportion to the insertion probabilities. We use this as the monitoring overhead $\Gamma(k_{s,m})$ in the closed-loop optimization (§4.3.1).

4.4.3 Telemetry Data Structures and Operations

Having developed models for $E(\Delta_{s,m})$ and $\Gamma(\Delta_{s,m})$ (§4.4.2), we now describe how the data plane realizes change-triggered INT and extends it to satisfy the Telemetry Primitive Contract (TPC) from §4.3.3.

Existing change-triggered INT designs apply thresholds globally and lack mechanisms for composable end-to-end semantics. SLICESCOPE addresses these gaps through: (1) per-slice state tracking that enables slice-specific thresholds $\Delta_{s,m}$ (R1, R2), and (2) a two-part telemetry header with conditional insertion that provides bounded e2e error and overhead guarantees (R3). We first describe the data structures, then walk through the four-step processing algorithm with examples for latency, jitter, and loss. Deployment mechanics (e.g., Maximum Transmission Unit (MTU) headroom, multipath disambiguation, recovery) are addressed in §4.4.4.

Per-slice state tracking. Each switch maintains d bucket arrays, each with w buckets, to track telemetry state for active slices (Figure 4.3). On packet arrival, a key x including the slice ID is extracted from the packet headers. Next, for each bucket array A_i , an independent hash function $h_i(x)$ is used to map this packet to a bucket A_{ij} (Figure 4.3). Each bucket stores: (1) the key x , (2) three e2e values for a given metric — E_{prev} (estimate from previous hops), E_{rep} (last reported), and E_{last} (value from most recent packet), (3)

auxiliary metrics V_{aux} needed for computation of some metrics (e.g., estimated flow size for computing packet loss), and (4) a 1-bit table miss flag F_{tm} to help recovery when downstream telemetry state is lost due to hash collisions. This per-slice state tracking enables slice-level customization of telemetry behavior (R1), as each slice can use its own threshold $\Delta_{s,m}$ to determine whether telemetry is inserted or skipped, at runtime (R2).

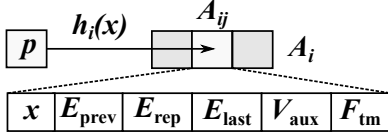


Figure 4.3: **Per-slice telemetry state stored in each bucket A_{ij} of the bucket arrays.** Each bucket maintains the slice key x , e2e estimates (E_{prev} , E_{rep} , E_{last}), auxiliary state V_{aux} (e.g., counters for packet loss), and a miss flag for recovery.

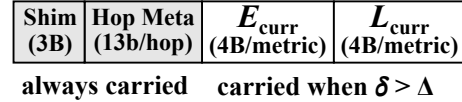


Figure 4.4: **Telemetry header format.** The fixed part (always present) includes a shim and per-hop metadata. The conditional part (inserted only when $\delta_{s,m} > \Delta_{s,m}$) carries updated e2e metric values E_{curr} and auxiliary state V_{aux} for computing metrics such as jitter or packet loss.

Telemetry header. The telemetry header (Figure 4.4) has two parts: a *fixed part* which is always carried, and a *conditional part* only carried when significant changes occur (only when $\delta > \Delta_{s,m}$). The fixed part consists of (1) a telemetry shim with a bitmap indicating which selective telemetry fields are present, and (2) per-hop metadata containing a 10-bit node ID and three 1-bit anomaly flags. The conditional part consists of (1) current e2e values E_{curr} for the slice metric, and (2) any auxiliary metrics V_{aux} needed to compute that slice metric. For example, to compute e2e packet loss, each switch appends the number of packets it forwarded for that slice (V_{aux}), allowing the next hop to estimate loss by comparing it with its own packet count.

Next, we show how the telemetry header and per-slice state are used to track per-slice SLAs in an end-to-end per-packet manner with bounded error (R3). SLICESCOPE’s data plane algorithm runs in the switch egress pipeline, and processes incoming packets in the following steps:

Step 1 (Bucket array lookup) The switch computes the key x based on the packet headers, and looks it up in the bucket arrays. If a match is found, the switch retrieves the relevant telemetry state from previous packets, including E_{prev} (estimate from previous hop) and E_{rep} (last reported).

Step 2 (Metric computation) The metric computation step involves three key operations:

- *Compute the current per-hop value (L_{curr}):* This can be derived using either internal device state (e.g., hop latency) or auxiliary metrics (e.g., hop loss). Note that this value is different from the e2e metric values E_{curr} , which are aggregated across the slice path.
- *Update the previous hop estimate of the e2e metric (E_{prev}):* If the packet carries a new update from the previous hop, it will be used and stored as the new E_{prev} .
- *Compute the current e2e value (E_{curr}):* This is done by combining the (potentially updated) previous hop estimate E_{prev} with L_{curr} using a simple aggregation operation (e.g., sum or max).

Step 3 (Selective telemetry insertion) The switch compares the current e2e value E_{curr} with the last reported value E_{rep} . If the difference δ exceeds a given threshold $\Delta_{s,m}$ for that slice’s metric, it is considered a significant change. The switch then inserts the current e2e value E_{curr} in the e2e metrics part of the packet header, and, when required, auxiliary information V_{aux} to support downstream metric computation. For example, to compute e2e packet loss across multiple hops, each switch inserts the number of packets it forwarded (V_{aux}), enabling the next hop to detect missing packets and estimate per-hop loss.

Step 4 (State update) If telemetry data is added to the packet, the switch updates the last reported e2e value E_{rep} in the corresponding bucket. We also update E_{last} to be E_{curr} .

Pkt	Sw	State Before	L_{curr}	E_{curr}	$ \delta $ vs Δ	Action	State After
p1	s_1	$E_{prev}=0, E_{rep}=0$	5	$0 + 5 = 5$	$ 5-0 = 5 \geq 2$	Add telemetry (5)	$E_{rep} \leftarrow 5$
p1	s_2	$E_{prev}=5, E_{rep}=0$	6	$5 + 6 = 11$	$ 11-0 = 11 \geq 2$	Add telemetry (11)	$E_{rep} \leftarrow 11$
p2	s_1	$E_{prev}=0, E_{rep}=5$	4	$0 + 4 = 4$	$ 4-5 = 1 < 2$	Skip telemetry	No change
p2	s_2	$E_{prev}=5, E_{rep}=11$	8	$5 + 8 = 13$	$ 13-11 = 2 \geq 2$	Add telemetry (13)	$E_{rep} \leftarrow 13$

Table 4.3: **Example execution of change-triggered telemetry on a two-hop path ($s_1 \rightarrow s_2$) with threshold $\Delta = 2$.** For each packet, the switch computes the local metric L_{curr} , updates the e2e estimate E_{curr} , and inserts telemetry only when the deviation $|E_{curr} - E_{rep}|$ exceeds Δ . The example illustrates selective insertion, state updates, and bounded per-hop error under skipped reports.

Walkthrough Example. Consider the simple case of a linear two-hop path with switches s_1 and s_2 , where $\Delta = 2$ for both switches. Table 4.3 walks through how two packets (p_1 , p_2) are processed at each hop. For packet p_1 : At s_1 , the local observation is $L_{curr} = 5$, giving an e2e estimate $E_{curr} = 0 + 5 = 5$. Since $\delta = |5 - 0| = 5 \geq 2$, telemetry is added and E_{rep} updates to 5. At s_2 , the local observation is 6, updating E_{curr} to $5 + 6 = 11$. Since $\delta = |11 - 0| = 11 \geq 2$, telemetry is again inserted. For packet p_2 : At s_1 , $L_{curr} = 4$ gives $E_{curr} = 0 + 4 = 4$, but $\delta = |4 - 5| = 1 < 2$, so no telemetry is added. At s_2 , $L_{curr} = 8$ updates E_{curr} to $5 + 8 = 13$. Since $\delta = |13 - 11| = 2 \geq 2$, telemetry is inserted.

The actual e2e latency for packet p_2 is $4 + 8 = 12$, but the reported estimate is 13 (using the stale $E_{rep} = 5$ from s_1). This creates an error of $|12 - 13| = 1$ at switch s_1 , which is bounded by $\Delta_{s,m} = 2$. This example shows how selective insertion reduces overhead (one skipped report) while maintaining per-switch error bounds. Section 4.4.2 describes how these per-switch bounds can be combined to derive e2e error guarantees.

Metric-specific instantiations. The same four-step logic generalizes to different SLA metrics by adapting the definition of the local observation L_{curr} and any auxiliary state V_{aux} . For example:

- **e2e latency.** Each switch computes L_{curr} as the difference between its egress and ingress timestamps ($t_{eg} - t_{ig}$). The e2e estimate is updated additively as $E_{curr} = E_{prev} + L_{curr}$, requiring no auxiliary state.
- **e2e jitter.** Here L_{curr} is the change in per-hop latency relative to the previous packet. This requires storing the prior packet’s latency as V_{aux} ; the switch then computes $L_{curr} = L_{curr}^{latency} - V_{aux}$ and updates the e2e jitter estimate additively.
- **e2e packet loss.** Each switch maintains a per-slice packet counter and, when telemetry is inserted, exports its current counter value V_{aux} in the packet header (*cf.* Figure 4.4). The downstream hop computes L_{curr} as the difference between the upstream counter and its own local count, and aggregates this into the cumulative e2e loss estimate ($E_{curr} = E_{prev} + L_{curr}$).

In all cases, the dataplane executes the same four-step loop; only the definitions of L_{curr} and V_{aux} differ by metric. These examples show how SLICESCOPE’s threshold-based logic enables *end-to-end*, per-slice monitoring of *diverse SLA metrics* (latency, jitter, loss) with bounded error. We next address deployment aspects that preserve these semantics in practice (§4.4.4).

4.4.4 Design and Implementation Challenges

The data plane design (§4.4.3) enables adaptive per-slice telemetry with bounded error. Realizing it on a programmable switch and making it deployable in practice, however, surfaces four challenges: (1) executing the per-packet logic within the stateful-processing limits of the switch Application-Specific Integrated Circuit (ASIC), (2) ensuring telemetry headers fit within MTU constraints, (3) maintaining correctness under multipath routing, and (4) recovering from missing telemetry state. SLICESCOPE includes mechanisms to address each, which together make its data plane deployable in practice.

1. Line-rate processing under ASIC constraints. The most fundamental constraint is that the design must execute within the stateful-processing limits of a line-rate programmable ASIC (*cf.* §2.1.3). Register state in a given bucket array can be read, modified, and written only once per packet within a single pipeline stage, and each bucket array consumes a dedicated stateful Arithmetic Logic Unit (ALU), which bounds the number of arrays that can be maintained per pipeline. SLICESCOPE accommodates these limits by keeping the per-packet operation sequence minimal, namely hash, bucket lookup, comparison against the slice threshold $\Delta_{s,m}$, conditional state update, and conditional telemetry insertion. Intermediate values are carried across pipeline stages in fixed-width metadata rather than recomputed, and the design avoids packet recirculation and multi-pass processing. These constraints are the reason SLICESCOPE adopts a compact change-triggered primitive rather than a more expressive mechanism that would require per-flow state or recirculation, and they determine the resource operating point analyzed in §4.6.

2. Bounded telemetry header. A key deployability challenge is ensuring *predictable header size*. Because sources cannot know in advance whether a packet will carry conditional telemetry, operators need a bounded worst-case header size to avoid fragmentation. SLICESCOPE’s telemetry header (Figure 4.4) is designed with this in mind: it has (i) a fixed part that grows with the hop count H , and (ii) conditional fields that grow with the number of reported metrics M but are independent of H . The fixed part consists of a 3B shim plus 13 bits per hop (10b node ID + 3b flags), rounded to bytes. The conditional part, carried only when $\delta_{s,m} > \Delta_{s,m}$, adds at most 8B per metric (4B E_{curr} and, when needed, 4B V_{aux}). The maximum header size is therefore $3 + \lceil \frac{13H}{8} \rceil + 8M$ bytes.

Crucially, the *heavier conditional fields do not scale with path length, unlike in traditional INT*. Even for three metrics (latency, jitter, loss) and $H=8$ hops, the total is only $3+13+24 = 40$ B. For longer paths ($H=16$) this grows to just 53 B. In practice, operator 5G transport networks are typically engineered with fewer than ~ 15 hops [58], so provisioning a small constant headroom (e.g., 64B via TCP MTU settings) ensures no fragmentation even in worst-case conditions.

3. Multi-path support. Slice traffic often traverses multiple paths, i.e., distinct sequences of switches between the same endpoints. Aggregating telemetry state across paths can obscure performance variations and compromise accuracy. For example, suppose packets p_1 and p_3 of slice s traverse $s_1 \rightarrow s_2 \rightarrow s_3$, while p_2 takes $s_1 \rightarrow s_4 \rightarrow s_3$. If switch s_3 maintains a single telemetry state for slice s , then p_2 updates E_{rep} based on its higher-latency path. When p_3 arrives via the shorter path, its δ will be computed against p_2 's state, potentially suppressing telemetry that should have been reported.

To avoid this, SLICESCOPE keys its telemetry state not only by slice ID but also by *path identifier* and output port. The path ID distinguishes between upstream routes that converge at the same switch, while the output port disambiguates diverging downstream paths. To disambiguate paths without incurring large header overheads, SLICESCOPE uses a compact 16-bit path identifier, pre-assigned by the control plane based on known slice routes. The path ID is constant-size and avoids carrying an explicit list of switch IDs. In practice, a 16-bit field suffices: with ~ 300 slices and a few disjoint paths per slice (~ 1000 paths total), the probability of collision remains below 0.05%.¹

4. Recovering from state loss. Finally, state in the bucket arrays may be lost due to collisions or selective INT suppressing upstream updates. If a packet from slice k arrives at s_3 but no valid E_{prev} is found, accurate e2e computation would be impossible. SLICESCOPE recovers by using the programmable packet replication engine to send a *table miss notification* upstream. Upon receiving such a notification, the upstream switch sets a “table miss” bit for slice k , forcing the next packet to carry full telemetry regardless of its δ relative to Δ_k . If that upstream switch has also lost state, the notification propagates recursively until it reaches the ingress of the slice path, where the state can be reinitialized. This ensures that downstream visibility is restored quickly after any loss. We evaluate the additional overhead of this mechanism in §4.5.4.

Summary. SLICESCOPE implements the closed-loop framework from §4.3 using change-triggered INT with per-slice thresholds $\Delta_{s,m}$. We developed mathematical models to estimate monitoring accuracy $E(\Delta_{s,m})$ and overhead $\Gamma(\Delta_{s,m})$ (§4.4.2), designed a data plane that provides slice-level control and e2e semantics (§4.4.3), and developed mechanisms to enable practical deployment of SLICESCOPE (§4.4.4). We now evaluate whether this design achieves the framework’s goals of SLA-aware monitoring with controlled overhead.

¹By the birthday bound with $M = 2^{16}$ identifiers and $N \approx 1000$ paths: $P_{\text{collision}} \approx 1 - e^{-N(N-1)/(2M)} \approx 0.0005$.

4.5 Evaluation

We demonstrate the feasibility and benefits of slice-aware and SLA-aware closed-loop control by evaluating SLICESCOPE along three main aspects:

- **The benefit of per-slice closed-loop control (§4.5.2).** Unlike prior work (e.g., [26, 102]) that relies on a single, static threshold, SLICESCOPE continuously repositions each slice within the monitoring accuracy–overhead trade-off space. We demonstrate the benefit of this dynamic, SLA-aware adaptation by comparing against two baselines: a *static slice-agnostic* control plane (same threshold for all slices) and a *static slice-aware* control plane (different thresholds for each slice type, fixed over time).
- **Choice of data plane monitoring primitive (§4.5.3).** In §4.4.1, we discuss why we choose change-triggered selective INT over per-hop sketches or probabilistic selective INT. In this section, we further justify this choice by empirically comparing SLICESCOPE against *LightGuardian* [131] and *PINT* [15] in their ability to provide *accurate end-to-end* tracking of per-slice SLA metrics.
- **Microbenchmarks (§4.5.4) and testbed evaluation (§4.5.5).** We evaluate the impact of the number of slices on the control plane’s decision time, the effect of epoch length on monitoring performance, and the impact of bucket array size on reporting frequency and other resource overheads. We also demonstrate SLICESCOPE’s effectiveness and hardware resource overheads over a testbed with Intel Tofino switches and emulated 5G traffic.

4.5.1 Experimental Setup, Workloads, and Implementation

While our work applies broadly to any network supporting slicing, we have grounded our motivating example and experiments in 5G networks, where slicing is a core architectural primitive and networks are expected to support hundreds of active slices [12, 13, 23], each with distinct SLAs (§4.2.1).

5G-Inspired topology and slices. Our simulation topology is based on Telecom Italia’s regional transport network [87]: with 25 Gbps access links, 40 Gbps aggregation links, and a 100 Gbps core. We use 300 network slices of three main types: URLLC, eMBB, and mMTC (Table 4.4). Network switches do weighted round-robin scheduling with weights decided based on the criticality of the main slice type and 22 MB of buffer per-port [123]. See §4.5.4 for SLICESCOPE’s default parameters.

Main Slice Type	Range of performance metrics for SLA purposes and expected traffic pattern [7, 8, 10]					Composition of Evaluated Workloads (% of slice instances of each main type)		
	e2e Lat. (ms)	Loss (%)	Jitter (ms)	Pkt. Size (B)	BW (Mbps/user)	More small pkts (SP)	Balanced (BAL)	More large pkts (LP)
uRLLC	1-5	<0.001	<1	20-250	1-10	60%	33%	20%
eMBB	10-50	<1	5-30	1K-1.5K	15-50	20%	33%	60%
mMTC	50-100	1-10	50-100	20-125	0.001-0.1	20%	33%	20%
Total # Slices in Workload						300	300	300

Table 4.4: **Summary of slice types and workloads used in the evaluation.** Each slice type reflects a class of applications with similar SLA characteristics; many instances of each slice type are generated, each with its own SLA within the specified range. Bandwidth values refer to typical per-user usage; total slice bandwidth is computed as: (App. BW) $\times$ (#users per slice), which is 3 – 10 (uRLLC), 10 – 20 (eMBB), and $\sim 10,000$ (mMTC). Representative applications include industrial automation (uRLLC), UHD video streaming (eMBB), and sensor telemetry (mMTC). Details on workload generation are discussed in §4.5.1.

Workloads and SLAs. The impact of monitoring overhead varies with packet size – small packets are more affected by INT headers. Moreover, the “right” point in the per-slice accuracy-overhead trade-off depends on the SLAs of active slices – fewer critical slices means the control plane can use lower Δ values more freely, as less-critical slices require fewer INT reports. As such, we generate three different workloads with different compositions of slice types and packet sizes by changing the fraction of active slice instances that are of each of the three main slice types.

Each main slice type has a distinct range of expected performance metrics and packet sizes in the 5G literature [7, 8, 10]. For example, uRLLC and mMTC slices have much smaller packets than eMBB, and uRLLC has stricter latency requirements than eMBB, which in turn is stricter than mMTC (Table 4.4). Accordingly, the compositions of our three workloads are: (1) *More small (and highly-critical) packets (SP)* with 60% of slices of the uRLLC type and 20% for each of eMBB and mMTC, (2) *Balanced (BAL)* with equal fraction of slices from each main type, and (3) *More large (and less highly-critical) packets (LP)* with 60% eMBB, 20% uRLLC, and 20% mMTC. For each workload, we generate traffic by creating the set of active slices, each with randomly assigned SLAs, flow counts, packet sizes, and per-flow rates from expected ranges of these values for the corresponding main slice type. We set the monitoring error tolerance to 5% of each metric’s SLA value.

These parameterized synthetic workloads allow us to systematically evaluate the closed-loop monitoring controller across a wide range of operating conditions. Given SLICESCOPE’s

focus on monitoring control dynamics, we identified the following as key properties for evaluating the closed-loop controller: (i) diversity of slice SLAs, (ii) variability in metric dynamics (e.g., latency fluctuations and burstiness), and (iii) scale in the number of concurrently active slices. While these properties are difficult to isolate independently using real traffic traces, they can be *explicitly controlled* using synthetic workloads. The simulated slice types, SLA targets, and traffic characteristics are grounded in 3rd Generation Partnership Project (3GPP) specifications and operator reports (Table 4.4), ensuring that the evaluated scenarios reflect realistic operational regimes.

Implementation. We have implemented SLICESCOPE’s data plane pipeline (§4.4) using P4 [118] ($\approx 2K$ lines of code) in the egress pipeline of an Intel Tofino switch [51]. The telemetry data is carried in an INT shim header and metadata stack between the TCP/UDP headers and the payload. In our Tofino implementation, we added 3 pad bits to the 13-bit per-hop metadata header to meet byte-alignment requirements. We use the Tunnel Endpoint Identifier (TEID) from the GPRS Tunneling Protocol (GTP) protocol used in 5G as the `slice_id`. In networks using Segment Routing over IPv6 (SRv6), `slice_id` can be embedded in the SRv6 header [39]. In other networks, such as Multiprotocol Label Switching (MPLS), the MPLS labels, added by the ingress or provider edge router can be used instead. SLICESCOPE’s control plane is implemented in Python, using the proprietary Gurobi library [45] to solve the ILP. To estimate the telemetry insertion probabilities $\beta(\Delta_{s,m})$, we fit a Laplace distribution to packet differences $d(t)$ after learning the distribution parameters from the data plane. The control plane runs on a server equipped with an Intel i9 5.7 GHz processor and 32 GB RAM. For large-scale experiments, we developed a discrete-event simulator using SimPy [104] that replicates the behavior of our P4 implementation.

4.5.2 The Benefit of Per-Slice Closed-Loop Control

SLICESCOPE’s control plane continuously revisits the per-slice $\Delta_{s,m}$ values based on the set of active slices and network conditions to reposition each slice within the monitoring accuracy–overhead trade-off space. To show the benefit of such per-slice closed-loop control, this section compares it against (1) a *Static slice-agnostic* control plane (same threshold for all slices, similar to prior work [26, 102]), and (2) a *Static slice-aware* control plane (different but fixed thresholds for each main slice type that does not change over time). In our experiments, we set the monitoring error tolerance set to 5% of the metric’s SLA value. Our goal is to show which control plane strategy better navigates the trade-off between tolerance violations (i.e., accuracy) and bandwidth overhead.

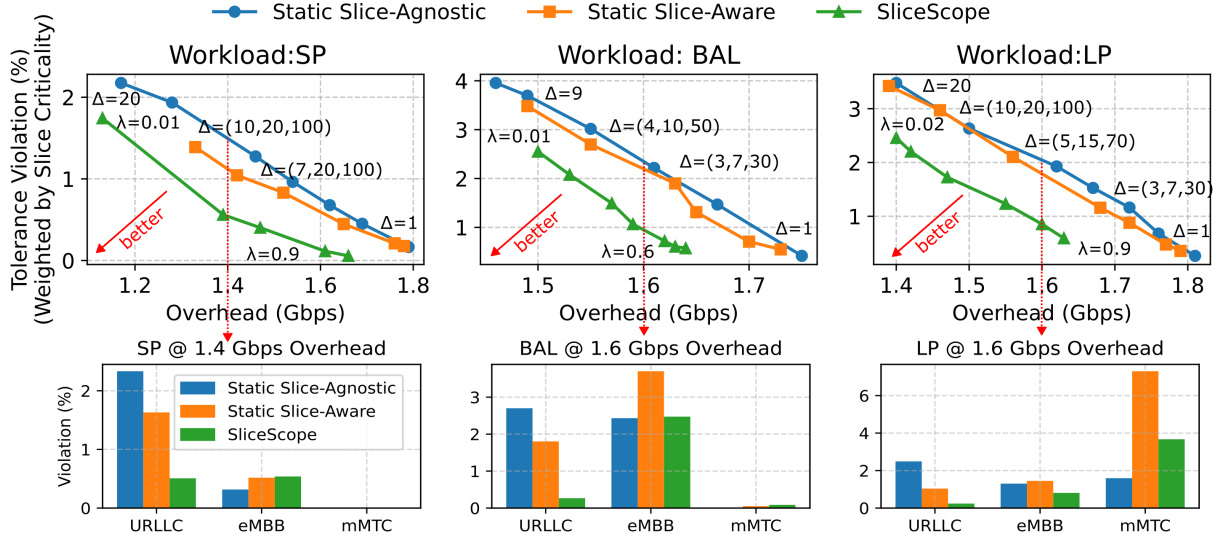


Figure 4.5: **Pareto frontiers comparing per-packet overhead and tolerance violations across different workload mixes (SP, BAL, and LP).** SLICESCOPE “moves” monitoring resources from less critical slices to more critical ones as needed. For *Static slice-agnostic*, points represent varying Δ values (1-20), for *Static slice-aware*, different combinations of Δ values for (URLLC, eMBB, mMTC) slices, and for SLICESCOPE, different values of tuning parameter λ .

Figure 4.5 depicts the results. The Y axis represents the percentage of violations of the monitoring error tolerance across all packets, and the X axis is the total bandwidth overhead (lower and to the left is better). For each control plane strategy, we plot the *Pareto frontier* representing how it navigates this trade-off space. For *Static slice-agnostic*, each point is a different Δ value, which is fixed for all slices and over time. For *Static slice-aware*, each point represents a combination of three specific Δ values, one for each of the three slice types. For SLICESCOPE, each point represents a different value of the tuning parameter λ in the optimization objective.

The results demonstrate that SLICESCOPE achieves the best balance between overhead and tolerance violations across all workloads. While *Static slice-agnostic* with higher Δ values achieves lower overhead, it does so at the cost of increased tolerance violations, which can be particularly problematic for slices with tight SLAs. *Static slice-aware* does better, showing the benefit of customizing Δ even just based on the main slice type, but fails to adjust to changing network conditions and active slices over time. Specifically, for the BAL workload, SLICESCOPE with $\lambda = 0.6$ reduces tolerance violations by $\sim 2.3\times$ compared

to *Static slice-agnostic* and *Static slice-aware*, while maintaining similar or slightly lower bandwidth overhead.

A closer look per slice type. To better understand where SLICESCOPE offers improvements, we break down the tolerance violations by slice type for each workload at a fixed bandwidth overhead (Figure 4.5(bottom)). SLICESCOPE consistently reduces violations for latency-sensitive URLLC slices compared to both *Static slice-agnostic* and *Static slice-aware* across all workloads. For example, in the SP workload, SLICESCOPE reduces the tolerance violations by $3\times$ compared to the best performing static scheme. In the BAL workload, the reduction is up to $4.5\times$. This can come at the expense of less-critical slices. For example, in the LPS workload, SLICESCOPE exhibits higher violation rates for mMTC slices. This behavior reflects its design: rather than uniformly allocating monitoring bandwidth to all traffic, SLICESCOPE prioritizes critical slices where SLA violations are more costly, while reducing monitoring fidelity for less sensitive slices. In doing so, it allocates limited monitoring bandwidth resources where they matter the most.

Take-away. The results underscore the importance of optimizing Δ values per slice. SLICESCOPE effectively tracks critical slices with high accuracy, while achieving savings in overhead by assigning higher thresholds, i.e., fewer monitoring resources, to non-critical slices or in scenarios when the measured metric exhibits less variation. This adaptive approach enables SLICESCOPE to balance the efficiency of selective INT insertion with the need for fine-grained monitoring of specific slices.

4.5.3 Data Plane Primitives for Slice SLA Monitoring

To validate our choice of threshold-based selective insertion for the data plane of SLA-aware closed-loop control, we empirically compare SLICESCOPE against two prominent alternative data plane primitives: *probabilistic INT* (PINT [15]) and *sketch-based monitoring* (LightGuardian [131]). We evaluate their ability to provide accurate e2e tracking of per-slice SLA metrics, focusing on a scenario where an URLLC slice exhibits natural latency variation under our balanced (BAL) workload, as shown by the ground truth line in Figure 4.6(a). This comparison demonstrates why threshold-based insertion is superior for slice SLA monitoring compared to approaches that sacrifice per-packet end-to-end visibility. Similar trends are observed under the SP and LP workloads.

We compare the reported e2e latency from SLICESCOPE and PINT at the per-packet level (Figure 4.6(a)), and evaluate 90th percentile (P90) tracking accuracy across all three schemes – SLICESCOPE, PINT, and LightGuardian – over 500 ms intervals (Figure 4.6(b)) and using the Cumulative Distribution Function (CDF) of P90 error in ms (Figure 4.6(c)).

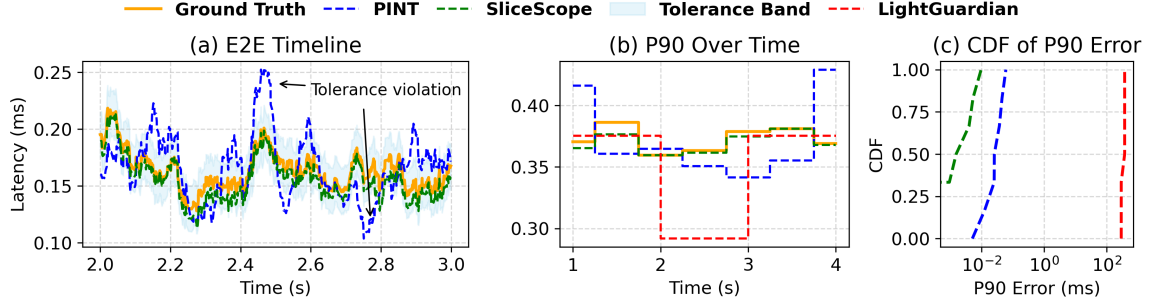


Figure 4.6: Comparison of different data plane monitoring primitives for per-packet e2e slice monitoring

LightGuardian does not provide per-packet tracking and is therefore omitted from the timeline in Figure 4.6(a). All schemes operate under comparable or higher bandwidth overhead budgets than SLICESCOPE.

We observe that SLICESCOPE consistently provides accurate tracking of both per-packet and aggregate (P90) latency, closely matching the ground truth. In contrast, PINT and LightGuardian show larger deviations: PINT has spikes where it violates the allowable error tolerances, and LightGuardian exhibits noticeable P90 error in e2e latency.

Comparison with PINT [15]. PINT implements selective INT for latency by a distributed sampling process where each packet probabilistically carries latency from one or more hops randomly chosen along its path. This allows lightweight tracking of per-hop latency quantiles under a given per-packet overhead budget. However, unlike SLICESCOPE, it does not natively support per-slice e2e latency monitoring (§4.4.1). To enable such comparison, we adapted PINT with a slice-aware reconstruction process at the collector: for each slice, the collector maintains the most recently observed latency for every hop, and reconstructs e2e latency by summing these values across the slice’s path. We compare the reported e2e latency from SLICESCOPE and PINT at the per-packet level (Figure 4.6(a)). PINT only reports per-hop latency quantiles. Due to the lack of per-packet alignment in PINT’s reconstruction of e2e latency (i.e., it may combine hop measurements from different packets in the same slice), its reported e2e latencies deviate significantly from the ground truth, as indicated by the two spikes crossing the allowable monitoring error tolerance.

Comparison with LightGuardian [131]. Sketch-based approaches such as LightGuardian are designed to track per-hop distributions using compact, approximate data structures. LightGuardian uses the SuMax sketch — a variant of the Count-Min Sketch (CMS), combined with binning, to track the distribution of per-hop latencies. It segments

the latency range into multiple bins and uses CMS to count the number of packets in each bin. The sketches are then split into “sketchlets” and embedded into packets, allowing the collector to gradually reconstruct per-hop latency distributions.

To enable a fair comparison in our setting, we simulate LightGuardian with idealized conditions: we assume accurate per-hop sketches are received at the collector every 500 ms, allowing the collector to reconstruct per-hop latency distributions for each slice. Next, to compute e2e P90 latency for a slice, the collector convolves the per-hop latency distributions along the slice path to obtain the combined distribution, then extracts the 90th percentile. This convolution is necessary because LightGuardian does not track e2e metrics. We also increase the number of bins from 4 to 10, compared to LightGuardian’s original setup, to improve accuracy.

Despite these favorable assumptions, LightGuardian’s reconstructed slice-level e2e latency still deviates significantly from the ground truth (Figure 4.6(b)), due to both the coarseness of its update interval and the inherent inaccuracy of deriving e2e metrics from per-hop metrics. In contrast, SLICESCOPE leverages the fact that the number of active slices is much smaller than the number of active flows to use a data plane primitive that stores exact telemetry information (as opposed to approximate statistics) in the data plane and report that when there is a significant enough change, and it can directly provide accurate per-packet e2e latency.

Take away. These experiments highlight the limitations of primitives such as per-hop sketches or probabilistic sampling approaches that do not natively support per-packet, e2e monitoring for SLA-aware closed-loop control of slice monitoring resources. Whether due to unaligned per-hop sampling or sketch-based summaries, these approaches struggle to provide accurate visibility into slice-level performance metrics. SLICESCOPE provides a better alternative in navigating the accuracy-overhead trade-off space in the presence of network slices with well-defined SLAs, where real-time and accurate tracking of e2e metrics is required.

4.5.4 Microbenchmarks

We conduct a series of microbenchmarks to assess the performance of our control plane and guide parameter selection in the data plane.

Control plane scalability (optimizer and heuristic). We evaluate the runtime of our control plane as the number of slices increases from 50 to 300. Recall from §4.3.2 that SLICESCOPE employs an ILP optimizer for threshold selection, with a greedy heuristic as a bounded-time fallback. Figure 4.7(a) shows how both strategies scale as the slice count

increases. Each control epoch involves two stages: (1) computing expected monitoring overhead and error across the candidate knobs $\Delta_{s,m}$ (§4.4.2), and (2) selecting the final configuration via the optimizer or heuristic (§4.3.1).

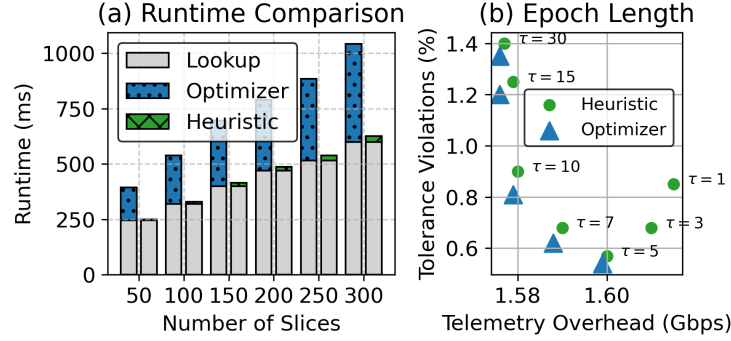


Figure 4.7: **Control plane scalability.** (a) Runtime scaling with slice count for the ILP optimizer and heuristic fallback. (b) Monitoring quality across epoch lengths τ ; the heuristic achieves comparable overhead–violation trade-offs while running $\sim 12\times$ faster.

To efficiently evaluate overhead and error across many candidate thresholds $\Delta_{s,m}$ (§4.3.1), SLICESCOPE pre-computes and caches them in lookup tables. This avoids repeatedly estimating $\beta(\Delta_{s,m})$ and $E(\Delta_{s,m})$ inside the optimizer: each candidate $\Delta_{s,m}$ would otherwise require a full recomputation of reporting probability and expected error, which the ILP solver (and even our heuristic) would invoke many times per iteration. This lookup phase accounts for a significant portion of the total runtime and grows linearly with slice count, taking ~ 250 ms at 100 slices and ~ 500 ms at 300 slices on a 16-core machine. Although lookup cost scales linearly, it is inherently parallelizable since lookups for slice–metric pairs are independent and easily distributed across CPU cores.

Once lookup results are available, we select thresholds using either the ILP optimizer with early stopping or our greedy heuristic (§4.3.2). The ILP solver’s timeout is set proportional to the epoch length τ to ensure the control plane adapts to network changes in a bounded time. For $\tau > 5$ s, the optimizer consistently converges to high-quality solutions, while shorter epochs ($\tau < 5$ s) more frequently trigger fallback to the heuristic. As shown in Figure 4.7(a), the optimizer adds ~ 600 ms at 300 slices, while the heuristic incurs only ~ 50 ms of additional latency. *Despite its $\sim 12\times$ lower runtime, the heuristic achieves comparable overhead–violation trade-offs to the ILP in regimes where both complete ($\tau \geq 5$ s) (Figure 4.7(b)).* This is because enforcing all monitoring-error constraints and minimizing overhead closely approximates the ILP’s joint objective when constraints are tight, making the heuristic a practical, ILP-safe fallback under tighter time budgets.

Epoch length. We also evaluate the effect of epoch length τ on bandwidth overhead and SLA tolerance violations (Figure 4.7(b)). We observe that $\tau = 5$ s achieves the best overall trade-off, with $\tau = 7$ s a close second. Shorter intervals (e.g., 1 – 3 s) provide insufficient time to learn stable metric variation patterns, leading to noisier $\Delta_{s,m}$ selection and higher overhead and violation rates. Longer intervals (e.g., 10 – 15 s) reduce overhead by choosing more informed thresholds, but react too slowly to network changes, causing more frequent SLA violations. A 5 – 7 s control interval offers a sweet spot: it is short enough to adapt to network dynamics, yet long enough to support accurate learning.

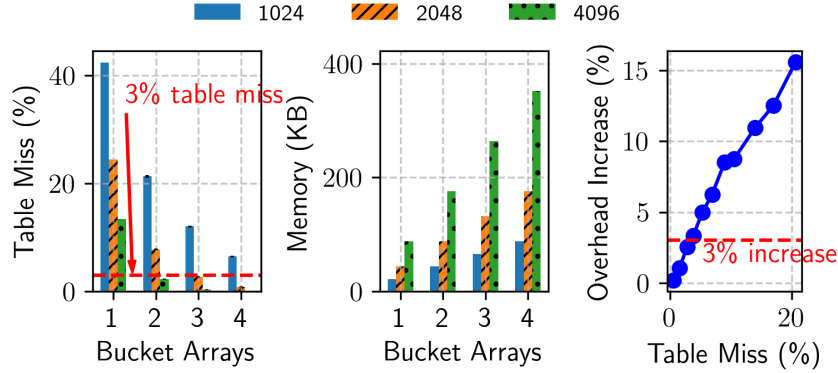


Figure 4.8: **Bucket array sizing.** Table-miss rate, memory usage, and overhead for different (d, w) configurations. A configuration of $d=2$, $w=4096$ achieves $< 3\%$ miss rate with minimal extra bandwidth overhead.

Bucket array sizing (Figure 4.8.) SLICESCOPE uses bucket arrays in the data plane to track per-slice metrics. When a lookup fails to find a match, i.e., a table miss, the upstream switch always emits the full telemetry in the next packet. This fallback ensures correctness but can increase overhead (§4.4.4). We vary the number of arrays (d) from 1 to 4 and number of buckets (w) from 1024 to 2048, and report, in Figure 4.8, the percentage of table miss (left), memory usage (middle), and associated increase in bandwidth overhead (right) with different configurations of d and w . We observe diminishing returns as d increases: from $d = 1$ to $d = 2$, table misses reduce significantly, but further increases yield only marginal gains while incurring higher memory cost. A configuration of $d = 2$, $w = 4096$ strikes a good balance, achieving $< 3\%$ table miss rate and only $\sim 3\%$ additional bandwidth overhead. Fewer arrays are preferable, as increasing d consumes more data plane resources (Table 4.5).

Summary. These microbenchmarks inform our system parameter choices: we set

$\tau = 5s$ for optimal overhead-violation trade-off and use the heuristic for scalability, and configure bucket arrays in the data plane with 4096 buckets and 2 arrays to meet our $< 3\%$ miss rate target.

4.5.5 Testbed Evaluation

We evaluate our P4 implementation of SLICESCOPE on a 5G hardware testbed comprising a radio access network based on OpenAirInterface [81] and Software-Defined Radios (SDRs), a 5G core based on Open5GS [80], and an Intel Tofino switch (UfiSpace S9180-32X) as transport. The radio access network consists of a single SDR-backed Next Generation Node B (gNB) serving three commercial User Equipments (UEs) (Google Pixel 7 Pro smartphones), each driving a distinct slice, enabling us to replicate real-world 5G traffic patterns. To further ensure realism, we replay commercial 5G traffic traces [24] that include applications such as cloud gaming and live streaming. User-plane traffic is carried over GTP-encapsulated flows, representative of slice traffic. To emulate multi-hop topologies within hardware limits, we apply external loopbacks on the Tofino, following prior work [15]. Additional details of the testbed configuration are provided in Appendix A.

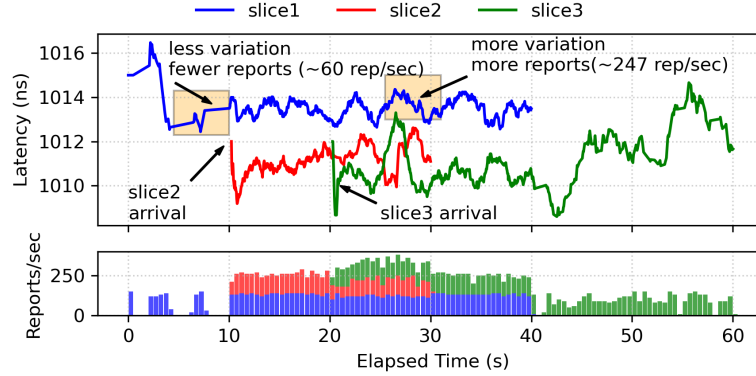


Figure 4.9: **Real-time tracking of slice metrics on Tofino.** SLICESCOPE adapts telemetry reporting rates as slice activity and latency variation change over time.

Real-time tracking. Figure 4.9 presents real-time monitoring of slice e2e latency (top) and telemetry reporting rate (bottom) across three distinct slices. Traffic from slices 2 and 3 are started at seconds 10 and 20, respectively. On slice arrival, SLICESCOPE immediately starts reporting metrics for the new slice, demonstrating real-time per-slice tracking. The figure also shows that SLICESCOPE adapts its reporting frequency in response

to variation in the observed metric. While only slice 1 is active, and latency is stable, the system reports at a low rate (~ 60 reports/sec), conserving bandwidth. After slices 2 and 3 join the network, latency variations increase, and SLICESCOPE correspondingly raises its telemetry rate to ~ 247 reports/sec. This behavior exemplifies the core design goal of SLICESCOPE: providing real-time, selective telemetry that adjusts telemetry bandwidth overhead according to SLA-relevant dynamics.

Bucket Array (d, w)	Hash	SALU	SRAM
(1, 2048)	7.2%	27.1%	9.2%
(1, 4096)	8.4%	27.1%	9.2%
(2, 2048)	11.1%	27.1%	9.4%
(2, 4096)	12.5%	27.1%	9.4%

Table 4.5: **Tofino hardware resource utilization under different bucket-array configurations.** For the selected configuration ($d = 2, w = 4096$), SLICESCOPE consumes 12.5% of hash units, 27.1% of SALUs, and 9.4% of SRAM.

Switch hardware overhead. Our prototype leverages the three key hardware resources introduced in §2.1.3. *Hash units* compute bucket indices for each row of the hash table, *SRAM* stores the per-bucket telemetry state, and *stateful ALUs (SALUs)* support one read/write per register array, with each row in the bucket array requiring one SALU. SLICESCOPE occupies 11 pipeline stages, introducing only ≈ 110 ns of additional processing delay per switch compared to simple forwarding. Resource usage for different bucket array sizes (d, w) is summarized in Table 4.5.

4.6 Discussions and Limitations

Resource cost and the deployment sweet spot. Characterizing SLICESCOPE’s scaling limit requires distinguishing its hardware cost, measured on the Tofino, from the collision behavior of its per-slice state, swept in simulation. On the hardware side (Table 4.5), stateful ALU usage is fixed at 27.1%, dominated by the per-packet metric-computation logic rather than array lookups, while hash-unit usage grows with the array count to 12.5% at $d=2, w=4096$ and SRAM stays below 10%. The design is thus not resource-bound at the evaluated scale.

The binding constraint is the table-miss rate. Because the bucket arrays are hash-indexed, the slice count sets the miss rate for a given size (d, w), and a miss triggers the

full-telemetry fallback that inflates bandwidth (§4.4.4). Using the simulator to instantiate all 300 slices, we find $d=2, w=4096$ holds the miss rate below 3% (Figure 4.8). Supporting more slices requires a larger (d, w) to hold this target, which the hardware accommodates. The practical limit is not resource exhaustion but headroom: a production switch must also run forwarding, queueing, and other transport functions, so the ceiling is reached when array growth would encroach on the budget reserved for them, at which point telemetry must be partitioned across switches or restricted to fewer metrics. Within the evaluated range, SLICESCOPE monitors 300 slices at a sub-3% miss rate while leaving substantial headroom for co-resident functions.

Deployability Beyond Programmable Switches. SLICESCOPE instantiates the Telemetry Primitive Contract (TPC) using change-triggered selective INT, which provides fine-grained, per-packet e2e visibility and tunable accuracy–overhead trade-offs on programmable switches. However, the TPC itself is an abstraction: it specifies the capabilities required by the control loop for SLA-aware network slice monitoring, rather than mandating a particular telemetry mechanism or hardware platform (§4.3.3). As discussed in §4.4.1, programmable data planes enable an expressive realization of the TPC using change-triggered INT.

Commodity switching ASICs typically lack support for programmable packet headers and per-packet stateful processing, making an exact realization of the TPC using change-triggered INT challenging. Such platforms can expose per-hop telemetry signals that are correlated at the control plane to approximate coarse end-to-end behavior, but do not natively preserve the end-to-end semantics required by the TPC. As shown empirically in §4.5.3, per-hop reduction followed by control plane composition weakens end-to-end SLA monitoring accuracy. Consequently, different hardware platforms correspond to different achievable accuracy–overhead trade-off regions. While commodity ASICs offer a coarser operating envelope, the closed-loop formulation remains applicable, with the control plane learning platform-specific trade-offs and allocating monitoring resources accordingly. Exploring concrete approximations of TPC requirements on specific ASIC platforms, and quantifying the resulting trade-offs, is an important direction for future work.

Epoch length and behavior between epochs. SLICESCOPE adapts monitoring parameters at discrete control epochs rather than continuously. As shown in §4.5.4, very short epochs ($\tau=1\text{--}3\text{s}$) do not improve accuracy and can reduce stability, whereas moderate epochs ($5\text{--}7\text{s}$) permit learning more robust accuracy–overhead trade-offs under varying traffic. Frequent re-optimization in response to short-term variability is therefore unnecessary, and stable adaptation at moderate timescales suffices for SLA-aware control. Changes in the active slice set are incorporated at epoch boundaries by modifying the decision variables; such lifecycle events are not expected at sub-second timescales relative to the adaptation

loop [33, 106]. To preserve scalability as the slice count grows, SLICESCOPE also provides a bounded-time greedy heuristic (§4.3.2) with accuracy–overhead trends comparable to the ILP. More broadly, the formulation in §4.3.1 is solver-agnostic and admits alternative strategies, including learning-based optimizers, without changing the abstraction.

The epoch length governs how often thresholds are *re-optimized*, not when telemetry is *emitted*. Within an epoch the data plane continues to apply the installed thresholds $\Delta_{s,m}$ per packet, so if a slice experiences a burst or a sudden increase in latency variation between epochs, the affected packets still trigger telemetry immediately whenever $\delta > \Delta_{s,m}$. Visibility is thus preserved by the change-triggered primitive even before the next optimization runs. The trade-off is that the telemetry budget is enforced at the control-loop timescale rather than as a strict instantaneous cap: an unexpected within-epoch increase in metric variation may produce more telemetry than the prior estimate of $\beta(\Delta_{s,m})$ predicted. This temporary inflation is diagnostically useful, as it exposes the affected slices precisely when their behavior changes, and the next epoch revises the threshold–overhead estimate and reallocates to restore the target trade-off.

4.7 Chapter Summary

This chapter addressed Research Question RQ2: how to regulate monitoring fidelity across slices with heterogeneous SLAs and evolving network conditions under explicit global budgets. We argued that meeting stringent, real-time slice guarantees requires treating telemetry as a *shared* and *allocatable* resource: critical slices demand high-fidelity, per-packet visibility, while less critical slices can tolerate coarser monitoring, but the “right” allocation must adapt as slice mixes and traffic dynamics change. To enable principled, runtime allocation, we formalized SLA-aware slice monitoring as a closed-loop control problem that jointly optimizes per-slice monitoring knobs under a global resource budget. We further distilled the minimal data plane capabilities required by this control loop into the Telemetry Primitive Contract (TPC), separating *what the controller needs* from *how telemetry is realized*.

We then presented SLICESCOPE, an INT-based realization of this framework using change-triggered selective telemetry. SLICESCOPE exposes per-slice, per-metric knobs, provides composable e2e semantics with bounded error, and enables the control plane to learn and navigate the accuracy–overhead trade-off space online. Our prototype on Intel Tofino and large-scale experiments demonstrate that closed-loop, SLA-aware control can substantially improve monitoring quality for stringent slices while respecting budget

constraints, achieving up to $4\times$ better SLA tracking for critical slices than static and slice-agnostic baselines.

This chapter contributes a general control abstraction (closed-loop optimization and TPC) and a concrete system design (SLICESCOPE) showing that high-precision slice visibility can be delivered selectively, in proportion to SLAs, within realistic telemetry budgets. Building on this budgeted control perspective, the next chapter shifts from slice-level allocation to fine-grained anomaly attribution in the user plane, while observing the same principles of efficient and high-precision telemetry for network slices.

Chapter 5

KESTREL: Lightweight Telemetry for Network Slice User-Plane Monitoring

5.1 Introduction

The previous chapters in this dissertation have thus far addressed two distinct layers of the monitoring stack (*cf.* Figure 1.3) towards achieving the goal of efficient, high-precision monitoring. Chapter 3 addressed the architectural abstraction gap by exposing slices as first-class monitored entities. With monitoring at the slice-level possible, Chapter 4 subsequently introduced Service Level Agreement (SLA)-aware control of monitoring fidelity, enabling each slice to dynamically adjust monitoring according to its SLA requirements and overall resource constraints.

While these mechanisms make significant progress towards slice assurance, they operate primarily at the granularity of service-level aggregates. This leaves a critical gap: fine-grained attribution within the data plane. In particular, while SLICESCOPE can determine whether a slice is violating its end-to-end (e2e) SLA, it is not designed to attribute that violation to the individual traffic flows or sessions that comprise the slice’s total traffic. This fine-grained, user-plane visibility gap, surveyed in §2.4, is the subject of this chapter.

This distinction is fundamental to effective closed-loop management of network slices. Detecting slice-level performance degradation is only the first step; without isolating the precise source of an anomaly among thousands of concurrent traffic streams, mitigation actions such as scaling, rerouting, or rate limiting must be applied coarsely. Consider a slice that exhibits transient latency spikes due to a small subset of flows. In the absence of

fine-grained attribution, mitigation must operate coarsely, at the slice level, for example, scaling the entire slice (wasting resources) or imposing slice-wide rate limiting, which degrades unaffected traffic. Yet neither approach isolates the true source of performance degradation. Achieving the *high-precision monitoring* therefore, requires reconciling fine-grained attribution capabilities with the strict memory and bandwidth constraints of the data plane. This problem becomes particularly acute in data plane components such as the User Plane Function (UPF) and Centralized Unit (CU), which typically aggregate traffic from multiple slices onto hardware-accelerated platforms (*e.g.*, programmable switches, SmartNICs; *cf.* §2.1.3) [66, 88, 105, 124].

The monitoring dilemma. Detecting Quality of Service (QoS) degradations in these data planes requires fine-grained, real-time telemetry. However, because thousands of flows share limited queues and on-chip memory, these systems typically only expose coarse per-class counters [2], or selective reports based on sampling or threshold triggers [78], which obscure transient or per-flow anomalies. Programmable hardware can, in principle, support full per-packet visibility through *postcard telemetry* that records queue depth, latency, and QoS identifiers for each packet. However, postcard export is prohibitively expensive: even at a modest 10 Gbps with 200 B packets, line-rate export would generate over 3 Gbps of telemetry (>30% overhead), and production systems operate at hundreds of Gbps.

To make this challenge concrete, we focus on the UPF – a critical Fifth Generation (5G) data plane function that forwards user traffic over GPRS Tunneling Protocol – User Plane (GTP-U) tunnels identified by *Tunnel Endpoint Identifiers (TEIDs)* (*cf.* §2.1.1). Each such tunnel may carry multiple *QoS Flow Identifiers (QFIs)* mapped to standardized service classes and enforced by a small set of shared hardware queues. When multiple TEIDs share a queue, the standard 3rd Generation Partnership Project (3GPP) per-QFI counters [2, 80] collapse fine-grained behavior into QFI-level averages. For instance, counters polled every second may report a healthy 2 ms average delay at QFI-level, yet per-TEID histograms can reveal a brief spike in TEID A, with 6% of packets delayed >20 ms. Postcards would expose this, but at an unsustainable export cost.

This exposes a fundamental tension: aggregate counters hide transient, per-flow anomalies, yet per-packet telemetry is prohibitively expensive. This tension motivates Research Question RQ3:

How can we achieve high-fidelity visibility into transient QoS anomalies (e.g., microbursts), and attribute these anomalies to specific slices, without incurring the prohibitive overhead of per-packet monitoring?

We address this question by designing KESTREL, a sketch-based telemetry system

for detecting transient QoS anomalies in programmable data planes under strict memory and bandwidth constraints. The key insight guiding our design emerges from empirical analysis. Our experiments (§5.2.2) show that anomalies at the UPF manifest as distinctive distributional signatures: latency tail inflation, sub-second delay spikes, per-TEID throughput violations, and oscillatory Inter-Arrival Time (IAT) patterns. Detecting such phenomena, therefore, requires visibility into per-flow distributions rather than simple aggregates. However, maintaining explicit per-TEID histograms is infeasible: each histogram requires multiple bins of state, which can quickly exhaust the limited memory available on hardware-accelerated data planes. These constraints require a principled rethinking of telemetry design to enable fine-grained anomaly detection in programmable data planes.

KESTREL overview. KESTREL is a telemetry system that provides per-TEID distributional visibility through a principled extension of Count-Min Sketch (CMS) tailored to UPF anomaly detection. Unlike prior sketch-based systems [63, 127] that focus on volume metrics (packet counts, heavy hitters), KESTREL augments sketch buckets to capture compact histograms of latency and inter-arrival times – the distributional signals that distinguish different anomalies. To make distribution tracking robust, we develop a formal detectability framework that quantifies how sketch collisions and normal-traffic variation affect anomaly visibility. This analysis yields three practical design principles: (1) per-Queue Identifier (QID) sketch partitioning to limit cross-class interference, (2) adaptive binning that preserves sensitivity to rare events under normal load, and (3) principled sizing rules derived from anomaly detection requirements.

Our analysis shows that modest sketch dimensions ($w=512$, $d=3$) are sufficient to effectively detect diverse anomalies, including transient events as short as ≈ 50 ms. KESTREL exports compact summaries ($\approx 0.15\%$ of user-plane traffic), achieving over $10\times$ *lower telemetry volume* than existing selective-postcard schemes [102] while improving anomaly detection accuracy by 10% on mixed workloads.

Summary of contributions. This chapter makes the following contributions:

- We characterize the telemetry requirements for UPF anomaly detection, showing that representative anomalies require per-TEID distributional visibility such as latency tails, inter-arrival patterns, and sub-second resolution (§5.2).
- We design a histogram-augmented sketch data structure that tracks per-TEID distributions using shared buckets rather than per-flow state, enabling operation within the memory constraints of programmable switches (§5.3).
- We develop formal detectability guarantees for sketch-based anomaly detection, and introduce a detectability-driven parameterization framework that maintains anomaly

visibility under measurement noise, traffic distribution drift, and constrained memory budgets (§5.4).

- We implement KESTREL on Intel Tofino switches, and demonstrate that it achieves 10% better anomaly detection accuracy than selective postcard sampling while reducing export bandwidth by $10\times$ (§5.6).

Chapter roadmap.

5.2 Characterizing the Fine-Grained Telemetry Problem in 5G Data Planes

Modern 5G networks increasingly implement key data plane functions, including the UPF and CU, on hardware-accelerated platforms [66, 88, 105, 124]. These functions aggregate traffic from multiple network slices, which makes them critical points for detecting QoS degradations that threaten SLA compliance. We define any data plane event that threatens SLA compliance as an *anomaly*, whether it stems from malicious activity, misconfiguration, or disruptive traffic patterns.

The central challenge in detecting such anomalies is a fundamental *visibility gap*: these systems typically expose only coarse-grained counters or heavily sampled telemetry, which fail to capture the fine-grained, transient signals required for reliable detection. In UPFs, this arises from aggregation of per-session statistics; in CUs, it manifests as coarse event reports that obscure per-User Equipment (UE) dynamics [53].

We focus on the UPF as a representative case, developing and evaluating our approach on a programmable UPF in our 5G testbed. However, the visibility gap and resulting telemetry requirements apply broadly across 5G data plane components. We first characterize the architectural and resource constraints that create this gap (§5.2.1), then demonstrate empirically what telemetry granularity is required for reliable anomaly detection (§5.2.2).

5.2.1 The Visibility Gap in UPF Telemetry

The visibility gap stems from a fundamental mismatch between QoS enforcement requirements and hardware constraints. UPFs forward traffic using GTP-U tunnels identified by TEIDs, with service classes distinguished by QFI and enforced by QoS Enforcement Rules (QERs). To sustain line-rate throughput, UPFs are increasingly offloaded to programmable

hardware (e.g., programmable switches, SmartNICs) that provide typically 8 – 100 egress queues per port [66, 105, 124]. Thousands of active TEIDs must therefore multiplex onto these shared queues, creating QoS inter-dependencies: a single aggressive TEID can degrade performance for others sharing its queue, making per-TEID monitoring essential for anomaly attribution.

However, tracking per-TEID metrics explicitly is infeasible: programmable data planes provide only a few MB of Static Random-Access Memory (SRAM) (*cf.* §2.1.3), insufficient for maintaining counters across thousands of concurrent TEIDs. Existing telemetry systems work around this memory constraint as follows:

- **Aggregate counters:** Standard 3GPP performance metrics [2], implemented in systems like Open5GS [80], report per-QFI packet/byte counts, loss, and average delay every few seconds. By aggregating across TEIDs, they reduce state requirements but collapse per-packet dynamics into coarse aggregates that mask TEID-level variation.
- **Selective telemetry:** Programmable data planes can generate stateless per-packet metadata [47, 52, 78] containing queue depth, timestamps, and identifiers. Postcards avoid maintaining per-TEID state, but impose prohibitive export bandwidth: at 10 Gbps with 200 B packets, line-rate export exceeds 3 Gbps (>30% overhead). Recent systems adopt change-triggered sampling (Δ -SMP) [26, 102], exporting telemetry only when the change in a metric exceeds a threshold Δ , but events that remain below thresholds or overlap with others are missed.

Neither approach provides what anomaly detection requires: *temporal granularity* to capture transient events, *spatial granularity* to isolate per-TEID behavior, and *distributional visibility* into queue dynamics beyond simple averages. We demonstrate these requirements empirically next.

5.2.2 Empirical Characterization of Anomaly Signatures

To validate these requirements and understand what telemetry signals enable reliable anomaly detection, we inject controlled anomalies into our Tofino-based UPF testbed carrying a mix of application traffic: cloud gaming, live streaming, video conferencing, and Internet of Things (IoT). Each application maps to standardized QFIs, with multiple TEIDs per application sharing queues. We implement eight queues¹ with strict-priority and

¹Our Tofino-1 testbed operates at 10 Gbps with 8 egress queues per port.

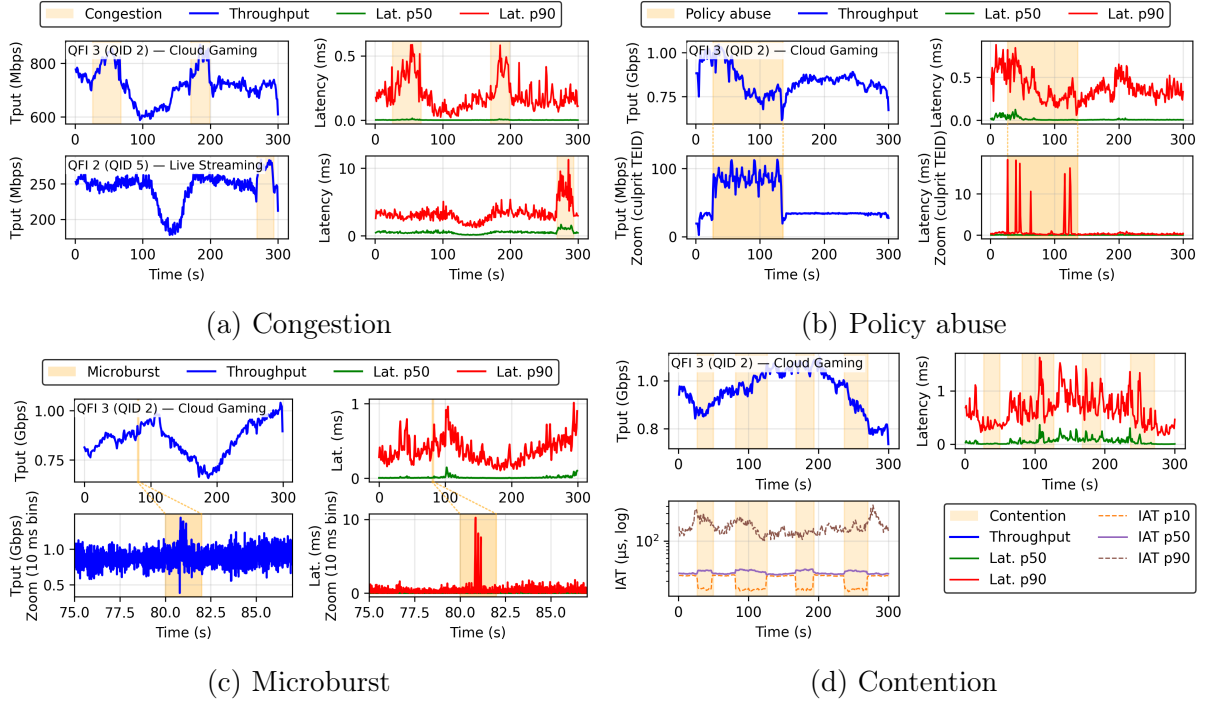


Figure 5.1: **Anomaly signatures require appropriate monitoring granularity.** (a) Congestion produces sustained tail latency on specific QFIs. (b) Policy abuse appears benign in QFI aggregates but is visible at the TEID-level. (c) Microbursts vanish in second-level averages yet emerge with sub-second resolution. (d) Contention induces distributed oscillations in IAT across flows.

Weighted Round Robin (WRR) scheduling, plus Two-Rate Three-Color Marker (trTCM)-based policing [66, 124].

Figure 5.1 shows four representative anomaly types and their detection requirements. *Congestion* from sustained surges produces long-tail latencies concentrated on specific QFIs. (Figure 5.1(a)). *Policy abuse*, where TEIDs exceed QER allocations (e.g., by remapping to higher-priority classes), appears benign at the QFI level but reveals throughput violations and latency spikes when zoomed in to TEID level (Figure 5.1(b)). *Microbursts* from bursty video streams manifest as millisecond-scale spikes that vanish in one-second QFI aggregates but emerge at finer temporal resolution (Figure 5.1(c)). *Contention* from competing backhaul traffic induces oscillatory Inter-Arrival Time (IAT) patterns distributed across flows (Figure 5.1(d)).

These anomalies share a critical property: *their signatures become visible only with appropriate measurement granularity*. Specifically, effective detection requires:

1. *Sub-second temporal resolution* to capture transient bursts.
2. *Per-TEID spatial resolution* to attribute misbehavior within shared queues.
3. *Distributional features* such as latency tails and IAT patterns rather than simple averages.

The challenge is designing a telemetry primitive that preserves these signals while remaining lightweight enough for carrier-scale deployment.

5.3 Design of KESTREL

5.3.1 Design Objectives

Section 5.2 established that reliable anomaly detection requires per-TEID distributions at sub-second resolution without the memory overhead of explicit tracking or the bandwidth cost of continuous export. Addressing this challenge requires meeting three objectives simultaneously:

1. **Bounded export cost.** Telemetry overhead must remain fixed even as traffic rates and active TEIDs grow, eliminating the bandwidth explosion that makes postcard export untenable at carrier scale.
2. **Per-TEID distributional fidelity.** Telemetry must retain the anomaly signatures such as latency tails, inter-arrival irregularities, and color counts, for each TEID, rather than collapsing behavior into slice-level averages (§5.2.2).
3. **Pipeline-constrained efficiency.** The telemetry primitive must fit comfortably within switch memory and execute within a single pipeline pass, avoiding recirculation, external memory, or coordination across stages. This ensures deployability on commodity programmable hardware such as SmartNICs and Tofino switches, while leaving room for other 5G data plane functions.

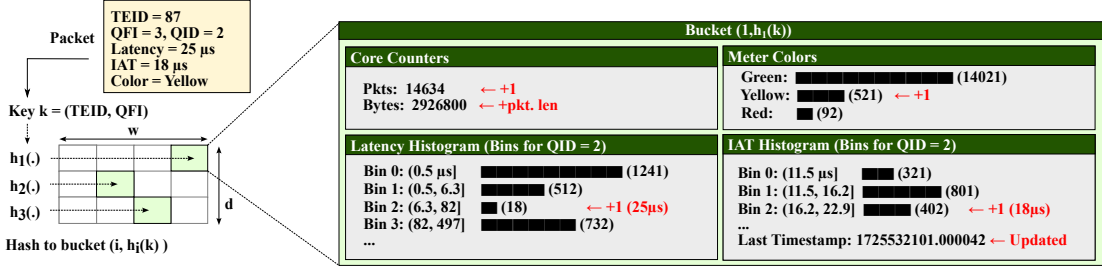


Figure 5.2: **Data structure and operations of KESTREL.** Buckets extend CMS to record packet/byte totals, latency histograms, IAT histograms, and policer colors. Example shows an arriving packet (TEID=87, QFI=3, QID=2, latency=25 μ s, IAT=18 μ s, color=yellow) being hashed to a bucket: totals and color counters are incremented, latency/IAT bins are updated using QID-specific edges, and the last is timestamp refreshed (updates in red).

5.3.2 Design Challenges and Solutions

Meeting the objectives in §5.3.1 is not straightforward. One natural alternative is to optimize selective postcard sampling, for example, with adaptive thresholds or smarter triggers. But such schemes fundamentally violate Objective (1): their export cost remains traffic dependent, flooding collectors precisely during bursts when visibility is most critical. We instead pursue sketch-based summaries that bound export volume regardless of conditions. For this, we identify two main challenges:

Challenge 1: From volumes to distributions. State-of-the-art sketches such as ElasticSketch [127] and NitroSketch [63] excel at estimating flow volumes and identifying heavy hitters. They achieve high accuracy at scale, but remain tied to *volume metrics*: packet counts, byte counts, and flow sizes. What they cannot capture is how packets are distributed across latency or inter-arrival bins, the very signals that distinguish anomalies such as microbursts, contention, or policy abuse.

KESTREL addresses this gap by *extending Count-Min Sketch (CMS)* to preserve distribution signals. CMS is an attractive foundation because it offers (i) clean analytical error bounds, (ii) a simple hash-and-increment update model that maps directly to Application-Specific Integrated Circuit (ASIC) pipelines, and (iii) a lightweight footprint that leaves memory and Arithmetic Logic Unit (ALU) budget for essential UPF functions such as policing, queuing, and scheduling. While CMS does not optimize overestimation bias, its simplicity makes it the right starting point.

To further mitigate bias, KESTREL maintains *per-QID sketches* rather than a single global structure (§5.3.3). Since QIDs group flows with similar QoS targets, partitioning

reduces cross-class collisions and makes error bounds less pessimistic in practice. Each CMS bucket is *augmented* to hold not only packet and byte counters but also compact histograms of latency, IATs, and policing colors. These enriched buckets yield one-second summaries that preserve the distribution signals required for anomaly detection. The challenge lies not in augmenting the sketch buckets but in *configuring them correctly*: bin edges must cap diagnostic occupancy, and sketch width w and depth d must be sized to ensure anomalies remain visible despite collisions. We formalize these requirements in §5.4, deriving detectability guarantees and sizing rules that make histogram-augmented sketches practical.

Challenge 2: Capturing timing without per-flow state. Objective (2) also requires preserving IAT patterns, which are crucial for distinguishing contention from congestion. A naïve design would timestamp each active flow, requiring $O(F)$ state for F TEIDs, far beyond ASIC budgets and violating Objective (3). KESTREL instead timestamps *buckets*, not flows. Each CMS bucket stores the arrival time of its last packet. When the next packet (possibly from a colliding flow) arrives, the IAT is computed relative to this stored timestamp and recorded into the bucket’s histogram. This reduces the state to $O(d \cdot w)$ rather than $O(F)$. While collisions introduce some noise, our analysis (§5.4) shows that the diagnostic bins remain sparse under baseline traffic, so anomalies still stand out reliably.

Therefore, adapting sketches for UPF anomaly detection requires rethinking both the *signal model* (distribution vs. volume) and the *state model* (bucket vs. per-flow). We next present KESTREL’s data structure that realizes this design.

5.3.3 Data Structure and Operations

Architecture overview. KESTREL combines two design choices that make sketching effective in the UPF setting.

First, *per-QID partitioning*: rather than one global sketch, KESTREL maintains a separate sketch per QID (8 in our prototype). Packets mapped to the same queue by UPF scheduling policies typically share QoS targets (e.g., low-latency cloud gaming vs. best-effort bulk). Per-QID partitioning reduces cross-class collisions and enables queue-specific binning (a 1 ms latency threshold is meaningful for gaming but irrelevant for best-effort). While this increases the number of sketches, each sketch can be kept small: as we show in §5.4 and §5.5, sketches with modest dimensions (e.g., $w=512$, $d=3$ in our prototype) achieve strong detection performance while fitting within hardware budgets. This makes per-QID partitioning practical, since the total memory cost remains well within hardware budgets.

Second, *CMS with enriched buckets* to capture distributional signals. KESTREL maintains d rows and w buckets per row, with independent hash functions $h_1, \dots, h_d$ over QoS flow keys $k = (\text{TEID}, \text{QFI})$. As shown in Figure 5.2, each bucket $j_i = h_i(k)$ stores four components: (a) *core counters* for packets and bytes, (b) *meter colors* tallying trTCM outcomes (green, yellow, red), (c) a *latency histogram* updated from per-packet sojourn time, and (d) an *IAT histogram* updated from the bucket’s timestamp register. These enriched buckets allow KESTREL to summarize traffic distributions compactly.

Packet updates. On each arriving packet with key $k = (\text{TEID}, \text{QFI})$ and latency t_s , KESTREL computes the bucket indices $j_i = h_i(k)$ for each row i . For each corresponding bucket (i, j_i) , it then: 1) increments the packet and byte counters, 2) maps the latency t_s to a bin using QID-specific edges, 3) computes the IAT as the difference from the bucket’s last-seen timestamp, updates the corresponding IAT bin, and overwrites the timestamp, and 4) increments the color counter corresponding to the packet’s trTCM marking.

Export and query model. At the end of each 1 second window, the control plane collects one compact record per bucket. To estimate the feature vector for a specific flow with key k , it queries all d buckets $h_i(k)$ to which the flow was mapped. The estimate for each feature (*e.g.*, the count in latency bin #3) is the minimum across the d candidate buckets, following CMS query semantics. The control plane, which already maintains TEID–QFI mappings for bearer/session management, uses these mappings to reconstruct per-flow distributions suitable for Artificial Intelligence/Machine Learning (AI/ML)-based anomaly detection. This model keeps the switch pipeline lightweight while ensuring anomaly-relevant signals are exposed at second-level cadence.

Illustrative Example (Figure 5.2). Consider a GTP-U packet arriving at the UPF with TEID=87 and QFI=2, experiencing a sojourn time of $25\mu\text{s}$ and an inter-arrival gap of $18\mu\text{s}$. The UPF maps this flow to QID=3 according to its scheduling policies. KESTREL processes this packet as follows: First, it forms the sketch key k from TEID+QFI and applies d hash functions to map k to buckets (i, j_i) across all rows. Second, it retrieves the QID=3-specific bin edges; for latency ($\{0.5, 6.3, 82, \dots, 4970\} \mu\text{s}$) and IAT ($\{11.5, 16.2, 22.9, \dots, 2.7 \times 10^6\} \mu\text{s}$). Finally, for each bucket (i, j_i) , it: increments packet/byte counters; assigns the $25\mu\text{s}$ latency to bin 2 (as shown in Figure 5.2); computes and records the $18\mu\text{s}$ IAT (updating bin 2 and refreshing the timestamp); and updates the color counter based on meters.

5.4 Parameterizing KESTREL

KESTREL addresses the visibility gap identified in §5.2 by exporting per-TEID distributional features such as latency and inter-arrival time histograms, meter colors, and traffic counters from compact, per-QID sketches (§5.3.3). These enriched sketches preserve the fine-grained signals shown in §5.2.2 to distinguish congestion, microbursts, contention, and policy abuse, while respecting the memory and pipeline constraints of programmable data planes.

The remaining challenge lies in *parameterization*: how to choose sketch width and depth, bin boundaries, and adaptation policies so that anomaly-induced feature deviations remain *detectable* despite hash collisions, distribution drift, and heterogeneous queue loads. Although anomalies manifest as structured changes in the exported feature distributions, their visibility ultimately depends on how collision noise interacts with background traffic within each sketch bucket.

Figure 5.3 summarizes our *detectability-driven parameterization strategy*, which proceeds in three logical steps. First, we derive the *ideal sketch dimensions* required to guarantee detectability under sparse diagnostic regions (§5.4.1–§5.4.2). Second, we introduce a *temporal adaptation mechanism* that preserves these guarantees as traffic distributions drift over time (§5.4.3). Finally, we present a *spatial allocation policy* that maps these ideal dimensions to limited on-chip memory, balancing detectability across heterogeneous queues (§5.4.4).

5.4.1 Detectability Guarantees

We first establish when anomalies become visible in KESTREL sketches. As discussed in §5.2, many anomalies induce concentrated changes in specific sub-regions of the distributional features maintained by KESTREL, such as latency tails, IAT heads, or abnormal meter color patterns. We refer to such sub-regions as the *diagnostic region* T .

Diagnostic sparsity. KESTREL explicitly designs the diagnostic region to remain sparse under normal traffic. Let N denote the total packet mass in a window and N_T the mass falling into T . Bin boundaries are chosen so that $N_T \leq \rho N$, where ρ is a configurable *diagnostic occupancy target* (typically 1 – 2%). This sparsity ensures that anomaly-induced increases in T are not drowned out by background traffic.

Sources of noise. Two effects can obscure anomaly signals. First, only a fraction of anomaly packets may land in T ; we denote the spillover fraction by β . Second, sketch collisions introduce overestimation noise that scales with εN_T , where $\varepsilon \approx e/w$ is the CMS error rate for width w .

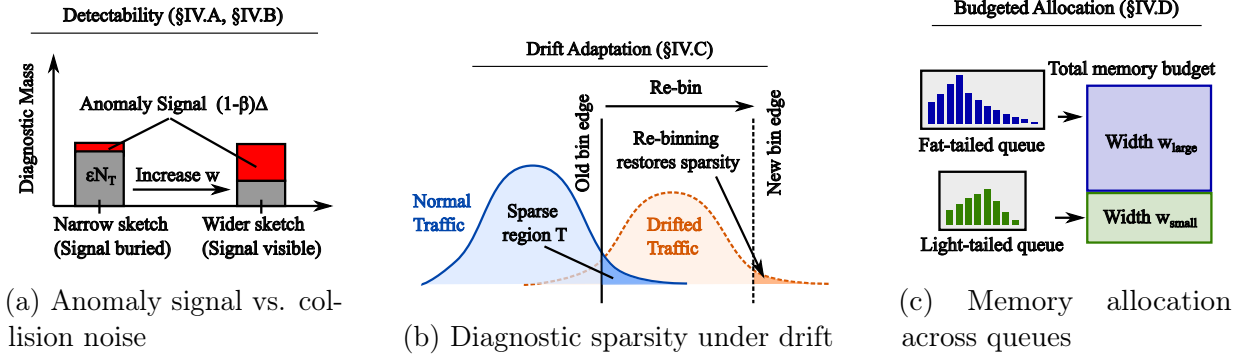


Figure 5.3: **Detectability-driven parameterization of KESTREL.** (a) Anomalies are detectable when the diagnostic-region signal exceeds sketch collision noise εN_T ; increasing sketch width w lowers this noise floor. (b) Traffic drift increases baseline diagnostic occupancy N_T , violating the target $N_T \leq \rho N$ and necessitating re-binning. (c) Under a fixed on-chip memory budget W_{total} , per-queue widths w_q are allocated to equalize worst-case detectability (minimax) across heterogeneous queues.

Detectability Condition. For a flow k , consider an anomaly that injects Δ packets within a window, with spillover fraction β , so that $(1 - \beta)\Delta$ packets land in the diagnostic region T .

$$(1 - \beta)\Delta > \varepsilon N_T, \quad (5.1)$$

where N_T is the baseline packet mass in T and $\varepsilon \approx e/w$ is the CMS error rate. A full derivation accounting for background mass interactions and CMS error bounds is provided in Appendix B.

Design Implication. The condition above captures a simple design principle: anomaly detection succeeds when the anomaly-induced lift in the diagnostic region dominates the collision noise contributed by background traffic. By enforcing diagnostic sparsity via careful binning (§5.4.3), this noise floor is kept intentionally low.

Consider a traffic rate of $N=10^6$ pkt/s and a diagnostic occupancy target of $\rho=1\%$, so that only $N_T=10^4$ packets fall into the diagnostic region under normal operation. With a sketch width of $w=512$, the collision noise is $\varepsilon N_T \approx 53$ packets. Even allowing for substantial spillover ($\beta=0.3$), detectability requires only a total anomaly size of $\Delta \gtrsim 76$ packets.

Even a short 50 ms microburst at 50 kpps injects approximately 2500 packets. Even with substantial spillover, this results in hundreds to thousands of packets falling into T , exceeding the detectability threshold above by more than an order of magnitude. Consequently, even

brief, low-volume anomalies remain clearly visible.

5.4.2 Sketch Sizing under Detection Targets

We now translate the detectability condition from §5.4.1 into sizing rules for the sketch parameters: width w and depth d .

Width Requirement. To detect all anomalies that inject at least $\Delta^{\min}$ packets within a window, the sketch width must suppress collision noise below the anomaly-induced signal that reaches the diagnostic region T . Under a conservative spillover bound $\beta_{\max}$, at least $(1 - \beta_{\max})\Delta^{\min}$ packets land in T .

Using $N_T^{\max} = \rho N^{\max}$, where $N^{\max}$ denotes the maximum total packet mass in a window, and $\varepsilon \approx e/w$, the detectability condition yields

$$w \geq \frac{e N_T^{\max}}{(1 - \beta_{\max}) \Delta^{\min}}, \quad (5.2)$$

where $N_T^{\max}$ is the maximum diagnostic occupancy enforced by binning and $\beta_{\max}$ is a conservative upper bound on the anomaly spillover fraction.

For example, with $N_T^{\max} = 10^4$ and $\beta_{\max} = 0.3$, detecting anomalies with total size $\Delta^{\min} \approx 80$ packets requires $w \geq 485$. This inequality establishes an analytical lower bound on the sketch width.

Depth Requirement. The sketch depth d determines the probability that CMS error bounds hold simultaneously across all diagnostic bins. Let $K = |T|$ denote the number of diagnostic bins. By applying the union bound over these K bins plus the total count estimator, we require:

$$d \geq \left\lceil \ln \frac{K + 1}{\zeta} \right\rceil. \quad (5.3)$$

This ensures that with probability at least $1 - \zeta$, all CMS bounds required for detection hold within a window. In practice, K is small (typically 2–3 bins for latency tails and IAT heads), so $d = 3$ suffices for >99% confidence per window. Since most anomalies persist across multiple windows, the resulting miss probability is negligible.

Limitations of Static Sizing. The sizing rules above assume that the diagnostic occupancy $N_T^{\max}$ remains stable. In practice, traffic distributions evolve over time, increasing background mass in the diagnostic region and invalidating this assumption. We next show how KESTREL maintains diagnostic sparsity under such distribution drift.

5.4.3 Maintaining Diagnostic Sparsity Under Drift

KESTREL’s detectability guarantees rely on a simple but critical property: *under normal traffic, the diagnostic region should remain nearly empty*. If a large fraction of baseline packets fall into diagnostic bins, collision noise grows proportionally, and increasingly stronger anomalies are required to stand out. As a result, bin placement is as important as sketch width: even a wide sketch cannot compensate for diagnostic regions that are frequently occupied by normal traffic.

Target-occupancy binning. Rather than fixing bin boundaries a priori, KESTREL explicitly controls diagnostic sparsity through *target-occupancy binning*. Bin edges are chosen so that, under baseline traffic, at most a small fraction ρ of packets (typically 1–2%) fall into the diagnostic region. Intuitively, this makes diagnostic bins act as *quiet sentinels*: they remain mostly empty during normal operation, so even modest anomaly-induced increases appear as sharp signals. In practice, this is achieved by placing latency-tail and IAT-head boundaries at appropriate quantiles of the baseline distribution. We show in §5.6.4 that this strategy outperforms fixed and log-spaced binning alternatives.

Why drift breaks sparsity. Traffic characteristics are not stationary. Changes in application mix, background load, and diurnal effects gradually shift latency and inter-arrival distributions. If bin boundaries remain fixed, these shifts cause increasing amounts of normal traffic to spill into the diagnostic region, eroding sparsity. For example, with $\rho=1\%$ and a packet rate of 10^6 pkt/s, the diagnostic region initially contains $\approx 10^4$ pkt/s. If drift doubles this occupancy, collision noise doubles as well, requiring proportionally stronger anomalies for detection. Re-binning is therefore essential to restore the system to its intended operating regime.

Conservative re-binning strategy. Re-binning is inherently asymmetric. Re-binning during anomalous traffic is dangerous: abnormal structures may be absorbed into the baseline and treated as normal until the next re-bin, suppressing future detection. In contrast, delaying a safe re-bin merely allows drift to accumulate gradually, which degrades performance slowly rather than catastrophically. KESTREL therefore adopts a conservative policy that prioritizes baseline integrity over responsiveness. Re-binning is triggered only when all three of the following conditions hold:

1. **Sustained normality.** KESTREL leverages its own control plane detection output, which labels each window as normal or anomalous (§5.5). To avoid re-binning during transient misclassifications or short-term jitter, a window is admissible only after $H=4$ consecutive normal decisions. We derive H from detector thresholds calibrated for

high precision ($\approx 90\%$), ensuring that the probability of erroneous re-binning during an anomaly is minimal.

2. **Confirmed drift.** The estimated diagnostic occupancy $\hat{N}_T/N$ must exceed the target ρ , indicating that the current binning no longer preserves sparsity. In our experiments, we consider drift confirmed only when this ratio exceeds the target by a safety margin of $2\times$.
3. **Cool-down interval.** A minimum interval M must elapse between successive re-bins. Each re-bin resets the baseline, and the controller requires sufficient *fresh, normal* evidence before recomputing bin edges. We choose M based on the sample complexity needed to estimate diagnostic occupancy with high confidence under sparse events. In our evaluation, this yields $M=60$ windows, which proved sufficient across QIDs with varying traffic rates.

Having established *when* to re-bin, we now address *how* to estimate the necessary parameters without incurring heavy telemetry costs.

Bounded control plane overhead. Drift *detection* relies entirely on sketch-derived diagnostic occupancy and incurs no continuous telemetry export. However, *recomputing bin edges* requires estimating quantiles of the underlying feature distribution, which cannot be recovered from sketches alone. To support re-binning, the switch maintains a small, fixed-size *ring buffer* of recent feature samples in the data plane for each QID. The buffer is populated via lightweight sub-sampling, ensuring that it spans a representative recent time horizon while using fixed memory ($n=2000$ samples). By the Dvoretzky–Kiefer–Wolfowitz (DKW) inequality, this sample size bounds the worst-case Cumulative Distribution Function (CDF) estimation error to $\approx 3.5\%$ with 99% confidence [68]. In practice, errors are smaller for smooth traffic distributions, and any residual misplacement is corrected by the closed-loop policy, which directly observes the achieved diagnostic occupancy and can trigger re-binning again if needed.

Crucially, this mechanism is *decoupled from steady-state telemetry export*. The reservoir resides entirely in the data plane and is accessed by the controller only when drift necessitates reconfiguration. Unlike postcard-based approaches, it consumes zero export bandwidth during steady-state operation and does not scale with traffic volume.

Re-binning preserves diagnostic sparsity within each QID over time. However, sparsity alone does not equalize detectability across traffic classes: QIDs can differ substantially in volume and baseline diagnostic behavior. We next address this remaining source of imbalance through budget-aware width allocation.

5.4.4 Maintaining Collision Bounds under Traffic Heterogeneity

KESTREL maintains a separate CMS per QID to avoid cross-class collisions. However, QIDs differ significantly in both traffic volume and diagnostic-region behavior: interactive and gaming traffic often exhibits persistent tail mass, while background or IoT traffic may be extremely light. Assigning a uniform sketch width across QIDs therefore forces collision-dominated operation for heavy classes while wasting scarce SRAM on light ones. Our goal is to allocate a per-QID width w_q that preserves anomaly detectability for each traffic class, while respecting the UPF’s global memory budget.

Detectability Constraint. As shown in §5.4, anomaly detectability in KESTREL is governed by a simple principle: anomalies are visible only when their contribution to the diagnostic region exceeds the collision noise introduced by background traffic. For a given QID, this collision noise grows with both (i) the total traffic volume and (ii) the fraction of traffic that naturally falls into the diagnostic region, and decreases with the sketch width allocated to that QID. Concretely, under normal operation, the expected collision noise in the diagnostic region of QID q scales as

$$\text{noise}_T(q) \propto \frac{r_q N_q}{w_q}, \quad (5.4)$$

where N_q is the QID’s traffic volume, r_q is its diagnostic-region occupancy, and w_q is the sketch width. To reason about worst-case behavior, we conservatively characterize each QID by stable high-percentile envelopes $r_q^{\max}$ and $N_q^{\max}$.

Requiring the anomaly-induced signal to exceed this collision floor by a fixed safety margin yields a minimum width requirement of the form

$$w_q^{\text{req}} \propto r_q^{\max} N_q^{\max}. \quad (5.5)$$

This relationship above captures the key intuition: QIDs that exhibit heavier baseline tail mass require proportionally more sketch width to preserve anomaly detectability.

Baseline Envelope Estimation. The quantities $r_q^{\max}$ and $N_q^{\max}$ capture long-term structural properties of each QID under normal operation. During an initial bootstrap period, KESTREL monitors per-window estimates of total QID load and diagnostic occupancy derived from sketch summaries. It tracks high-percentile (*e.g.*, 95th) envelopes of these signals and declares them stable once the envelopes converge within a small tolerance (*e.g.*, 5%) over a sliding history window. In practice, several minutes of representative traffic are sufficient for stabilization in our traces.

Analytical Width Assignment. When the UPF’s available memory suffices to satisfy all analytical requirements, KESTREL assigns $w_q = w_q^{\text{req}}$ for each QID, which provides strong, per-QID detectability guarantees. As shown in Figure 5.2, each bucket stores packet and byte counters, three meter color counters, latency and IAT histograms, and a timestamp register. Under a conservative implementation with 32-bit registers and fixed 8-bin latency and IAT histograms, this corresponds to 22 32-bit words, or 88 bytes per bucket. With depth $d = 3$, a QID with width w_q therefore requires $264 w_q$ bytes.

In our prototype, the analytically derived widths (up to $w_q^{\text{req}} \approx 2,600$) result in an aggregate footprint of approximately 2.16 MiB across all QIDs, well within the $O(10)$ MiB on-chip SRAM budgets of modern programmable switches [25, 65]. When such headroom exists, assigning $w_q = w_q^{\text{req}}$ for all QIDs is both practical and sufficient.

Budget-Limited Allocation. As deployments scale, factors such as an increased number of QIDs and other forwarding and telemetry state may lead to scenarios where available memory no longer satisfies the analytical width requirements of all QIDs simultaneously, i.e., $\sum_q w_q^{\text{req}} > W_{\text{total}}$. To handle this regime, KESTREL allocates sketch width in a *detectability-aware* manner. Recall from §5.4 that anomaly detectability is governed by the ratio between the anomaly-induced signal in the diagnostic region and the collision noise introduced by background traffic. Under worst-case normal operation for QID q , this noise scales proportionally to $r_q^{\text{max}} N_q^{\text{max}}$ and inversely with the allocated width w_q . Consequently, the smallest anomaly that can be reliably detected in QID q scales as

$$\Delta T_q^{\min}(w_q) \propto \frac{r_q^{\text{max}} N_q^{\text{max}}}{w_q}, \quad (5.6)$$

up to constant factors introduced in §5.4.

When memory is limited, KESTREL seeks to prevent any single QID from becoming the detectability bottleneck. Intuitively, this corresponds to equalizing the worst-case detectable anomaly magnitude across QIDs, while allowing operators to bias protection toward higher-priority traffic classes. Formally, KESTREL allocates widths by solving the following budget-constrained minimax problem:

$$\begin{aligned} \min_{\{w_q\}} \quad & \max_q \text{priority}_q \frac{r_q^{\text{max}} N_q^{\text{max}}}{w_q} \\ \text{s.t.} \quad & \sum_q w_q \leq W_{\text{total}}, \quad w_q \geq w_{\min}. \end{aligned} \quad (5.7)$$

This objective directly follows from the detectability condition in §5.4: reducing the maximum collision-dominated noise floor across QIDs maximizes the smallest anomaly that can be reliably detected anywhere in the system.

To produce deployable configurations, KESTREL solves this problem as a discrete, budget-constrained Integer Linear Program (ILP) that enforces hardware-supported width granularities and per-QID bounds. The resulting optimization involves only tens of decision variables for realistic numbers of QIDs and is executed infrequently on the control plane. In practice, it can be solved efficiently using standard ILP solvers; we use Gurobi [45] in our implementation.

Take-away. Our analysis yields a detectability-driven parameterization of KESTREL. Detectability is preserved when the anomaly-induced signal exceeds sketch collision noise in the diagnostic region. In practice, this is achieved by maintaining diagnostic sparsity through target-occupancy binning with adaptive re-binning under drift, and by allocating sketch width in a QID-aware manner across heterogeneous traffic classes. In our evaluation, these principles translate to small diagnostic regions (2 – 3 bins), shallow sketches ($d=3-4$), and per-QID widths on the order of a few thousand buckets, resulting in compact, anomaly-revealing summaries suitable for deployment in programmable UPFs.

5.5 Prototype Realization

5.5.1 Prototype and Experimental Setup

Testbed and Data plane. We prototype KESTREL on a UfiSpace S9180-32X switch equipped with an Intel Tofino ASIC. The switch implements UPF functions including queuing, scheduling, and trTCM metering [48], and connects over 10 GbE NICs to four Ubuntu 22.04 servers. These servers act as: (i) a GTP-U traffic generator that allows fine-grained control over TEID/QFI mappings, (ii) a control plane host that programs the data plane and configures runtime telemetry, (iii) a collector for postcards and sketch registers, and (iv) an AI/ML pipeline server for anomaly detection.

The data plane implementation consists of $\sim 2K$ lines of P4 [17, 118] code in the egress pipeline. The design supports all three telemetry modes: 3GPP-PM counters, postcards (via packet mirroring), and sketches. It fits within a single pipeline pass (~ 7 stages) without recirculation. Sketches use $d=3$ rows and $w=512$ buckets (§5.4); each bucket stores counters, 8-bin latency and IAT histograms, and trTCM color tallies. Per-QID bin tables are runtime-programmable, enabling dynamic reconfiguration. Queues implement strict-priority and WRR scheduling across QIDs to emulate realistic UPF service classes.

Workloads and Anomalies. Traffic workloads combine a 5G operator trace [24] (cloud gaming, live streaming, buffered streaming) with additional traffic types we collected

for IoT, Voice over Internet Protocol (VoIP), and best-effort applications. The generator sustains 1 – 4 Gbps aggregate load across up to 100 UEs $\times$ 9 QFIs (≈ 900 flows)². We inject four representative anomaly scenarios: *microbursts* (short high-rate bursts), *congestion* (sustained overload), *contention* (shared backhaul bottlenecks), and *policy abuse* (QFI remapping), as well as mixed cases. Table 5.1 summarizes their scope, durations, and injection frequencies. Each anomaly ultimately degrades QoS for affected UEs.

Table 5.1: Anomaly scenarios used in our evaluation. Single anomalies (1–4) and mixed anomalies (5).

Scenario	Anomaly	Target	Dur.	Freq.
1	Microburst	One/more TEIDs	0.3–1.0 s	10–30 s
2	Congestion	One/more QFIs	25–40 s	60–120 s
3	Contention	Many QFIs/TEIDs	12–30 s	90–150 s
4	Policy abuse	Selected TEIDs	30–60 s	150–300 s
5	Mixed	Multiple	As above	As above

Telemetry export and processing. Sketches are exported every second via the Intel P4Studio runtime Application Programming Interface (API) [83], while postcards are captured by a dedicated collector with a memory-mapped ring buffer for high-throughput, zero-copy capture. All telemetry modes are aligned to one-second windows for fair comparison. Feature extraction proceeds according to the intrinsic visibility of each mode. Sketches yield per-(TEID,QFI) distributions by querying the d rows for each key and applying CMS query semantics. Postcards provide per-(TEID,QFI) distributions directly from In-Band Network Telemetry (INT) metadata (latency, IAT, color). Counters expose only QFI-level aggregates. From these, we construct feature vectors including traffic volume, latency percentiles and tail fractions, inter-arrival head fractions, policer color ratios, and contextual metrics such as TEID counts per QFI.

Anomaly detection pipeline. We implement an anomaly detection pipeline incorporating four dedicated detectors that target congestion, contention, microbursts, and policy abuse. Each detector is implemented as an eXtreme Gradient Boosting (XGBoost) [21] model trained on baseline and anomaly-injected traces, enabling robust classification between normal and abnormal traffic patterns. As a result, each detector is allowed to focus on anomaly-specific behaviors. For instance, the microburst detector leverages sub-second

²Our prototype generator sustains a few Gbps; higher loads are feasible with a DPDK-based [31] design, which we leave to future work.

temporal spikes, while the contention detector uses distributional shifts across flows. To ensure robustness, models are validated using temporally blocked cross-validation, preventing leakage between training and test windows.

5.5.2 Design and Implementation Challenges

Realizing KESTREL on a line-rate ASIC, and evaluating it at scale, surfaces five challenges. The first four concern fitting per-flow distributional state within the switch pipeline; the fifth concerns driving the user plane at representative flow counts.

1. Per-flow histograms under one-shot stateful updates. Conceptually each bucket is a single 88-byte record of counters, colors, two histograms, and a timestamp, but a stateful ALU performs only one bounded-width read-modify-write per packet, so the record cannot be updated atomically. Histograms are harder still: incrementing the bin a packet falls into requires a data-dependent choice the pipeline cannot make within a single stateful word. KESTREL decomposes the bucket across register arrays and externalizes each histogram into a flat register, selecting the target bin through a match-action table and a computed index. The logical bucket is thus realized over several physical arrays, preserving the per-flow abstraction while respecting the one-shot, no-data-dependent-indexing constraints, KESTREL’s analogue of “one stateful update per packet.”

2. Fitting the decomposed state in a single pass. This decomposition yields several register arrays per row, each consuming one ALU and accessed once per packet with no recirculation. KESTREL fits the budget by keeping each update minimal and ordering them as a straight-line sequence, with the resulting footprint (§5.6.4) leaving headroom for the forwarding, policing, and queuing logic a programmable UPF runs in the same pipeline.

3. Per-QID partitioning and value quantization. KESTREL partitions a shared register array by offsetting each QID into a fixed block of buckets, so per-QID sketches cost no additional arrays. Because the bin-assignment tables match on bounded-width keys, latency and inter-arrival values are quantized before binning, capping the representable range, which is acceptable since the diagnostic regions concern bounded tails and short inter-arrivals.

4. Runtime reconfiguration without recompilation. Re-binning under drift (§5.4.3) requires changing bin boundaries at runtime. Because bin assignments are match-action table entries rather than compiled constants, the control plane updates per-QID edges by rewriting entries, with no recompilation; sketch state is exported each window through the runtime API [83].

5. Driving the user plane at representative scale. Open-source radio stacks could not generate the required load: our Software-Defined Radio (SDR)-based Next Generation Node B (gNB) sustained only ~ 150 Mbps at 106 Physical Resource Blocks (PRBs), far below the multi-gigabit aggregate needed, with no way to inject anomalies at per-TEID granularity. Since this limit lies in the radio front-end, orthogonal to the UPF pipeline under test, we drive the user plane with a GTP-U traffic generator [95] that replays real application traces [24] under controlled TEID/QFI assignments and injects the evaluated anomalies (Table 5.1). This exercises the pipeline at up to 900 concurrent (TEID, QFI) flows ($100 \text{ tunnels} \times 9 \text{ QFIs}$) with real traffic content, reproducing user-plane patterns.

5.6 Evaluation

We evaluate KESTREL along four complementary dimensions: detection accuracy, cost-effectiveness, responsiveness under dynamic traffic, and deployment practicality. Together, these dimensions assess not only how well KESTREL detects anomalies, but also whether it remains efficient, stable, and deployable under realistic operating conditions. Specifically, we ask:

1. How *accurately* can KESTREL detect diverse anomalies?
2. How *cost-effective* is it compared to existing approaches?
3. How *responsive and stable* is it under dynamic traffic conditions?
4. Is it *practical* for deployment on programmable switches?

Table 5.2: Telemetry approaches evaluated in our testbed.

Baseline	Switch	Collector	Overhead
3GPP-PM	Per-QFI counters	Poll every 1s	<i>Low</i> , $O(\#\text{QFIs})$
Kestrel	Per-QID sketches	Query every 1s	<i>Medium</i> , $O(wd)$
Δ-SMP	Postcard sampling	Collect & aggregate every 1s	<i>High</i> , $O(\#\text{pkts})$

Baselines. We compare KESTREL against two representative telemetry approaches: (i) *3GPP-PM*, the standards-defined per-QFI performance measurements [2, 80], which

expose per-QFI packet and byte counts, average delay, and loss; and (ii) Δ -SMP, selective postcard sampling that exports telemetry only when monitored metrics change beyond a configured Δ threshold [26, 102], thereby reducing postcard volume relative to per-packet export. Table 5.2 presents the key characteristics of these baselines alongside KESTREL, highlighting their switch operations, collection mechanisms, and asymptotic overhead. These baselines capture the design spectrum – from coarse-grained low-overhead ($3GPP$ -PM) to fine-grained high-overhead (Δ -SMP).

We structure the evaluation in two parts. First, we evaluate KESTREL’s core behavior: detection accuracy (§5.6.1), cost-effectiveness (§5.6.2), responsiveness (§5.6.3), and deployment practicality via microbenchmarks (§5.6.4). We then evaluate KESTREL under extended operating regimes: maintaining detectability under sustained traffic drift (§5.6.5) and budget-aware sketch allocation under constrained memory (§5.6.6).

We evaluate KESTREL under a set of predefined workload and anomaly scenarios. Each scenario corresponds to a fixed workload and anomaly configuration, with traffic mixes and parameter ranges specified in Table 5.1. For each scenario, anomaly parameters are sampled uniformly within the listed ranges, and the *frequency* column denotes the inter-arrival time between consecutive anomaly events. To ensure statistical robustness, we evaluate each scenario over 10 independent runs using different random seeds, and report metrics aggregated across these runs. Where shown, error bars represent 95% confidence intervals.

5.6.1 Anomaly Detection Accuracy

We first investigate how accurately KESTREL detects anomalies in both single and mixed anomaly scenarios (Table 5.1).

Single anomaly scenarios. Figure 5.4(a) compares the Area Under the Precision–Recall Curve (AUPRC) across the three telemetry approaches for Scenarios 1 – 4. Detection accuracy varies significantly by anomaly type and telemetry granularity. For *congestion*, all three approaches perform well: $3GPP$ -PM achieves 0.88, while both Δ -SMP and KESTREL reach near-perfect accuracy (AUPRC ≈ 1.0). Congestion’s clear QFI-level signature makes it detectable even with coarse counters.

Contention presents the starkest contrast: $3GPP$ -PM collapses to 0.22, Δ -SMP improves to 0.37, while KESTREL achieves 0.75, more than $3\times$ better than $3GPP$ -PM and $2\times$ better than Δ -SMP. This gap stems from contention’s anomaly signature: subtle distortions that evade both aggregate counters and threshold-based sampling but are captured by KESTREL’s sketches. For *microbursts*, the gap is narrower: $3GPP$ -PM reaches 0.48, while Δ -SMP (0.86) and KESTREL (0.87) perform comparably, both nearly doubling counter

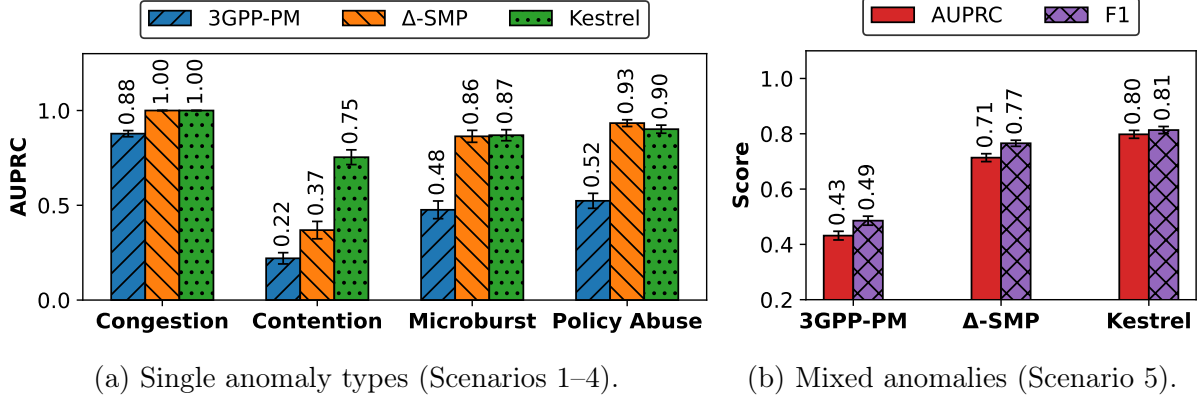


Figure 5.4: **Detection accuracy across telemetry approaches.** (a) In single-anomaly scenarios, KESTREL achieves higher AUPRC than coarse counters and selective sampling. (b) Under mixed anomaly workloads, KESTREL maintains strong detection performance, demonstrating robustness to concurrent anomaly types.

accuracy. For *policy abuse*, Δ -SMP slightly edges out KESTREL (0.93 vs. 0.90), although both outperform counters (0.52).

Mixed anomalies. We next consider Scenario 5, where multiple anomaly types may overlap in the same window. This is a strictly harder setting: anomaly signatures dilute one another, and overlap increases the likelihood of false positives. Figure 5.4(b) shows that 3GPP-PM fails entirely (AUPRC 0.43, F1 0.49), while Δ -SMP achieves moderate accuracy (0.71/0.77). KESTREL maintains the highest performance (0.80/0.81), which is about 13% higher AUPRC and 5% higher F1 than Δ -SMP, and nearly 2 $\times$ better than 3GPP-PM. KESTREL’s advantage stems from continuous summarization of per-packet information, which allows it to capture even very subtle changes in telemetry patterns.

Take-away. KESTREL maintains high accuracy across anomaly types and remains robust to overlapping anomalies, outperforming baselines especially on subtle cases like contention.

5.6.2 Cost-Effectiveness

We next turn from accuracy to cost, quantifying the telemetry overhead required to achieve the results above.

Accuracy-cost trade-off. Figure 5.5 plots the accuracy-cost trade-off, with ideal

operating points in the top-left region. *3GPP-PM* occupies the lower-left: minimal overhead (~ 1 Mbps) but poor accuracy (AUPRC ~ 0.4). Δ -*SMP* achieves moderate accuracy ($0.7 - 0.76$) but at very high cost: $60 - 80$ Mbps. KESTREL defines the Pareto frontier, achieving AUPRC $0.7 - 0.82$ with only $5 - 10$ Mbps export overhead. The sweet spot at $w=512, d=3$ delivers AUPRC 0.81 which is $2\times$ higher accuracy than *3GPP-PM* and $10\times$ cheaper than Δ -*SMP*. Larger configurations show diminishing returns, confirming our parameter analysis from §5.4.

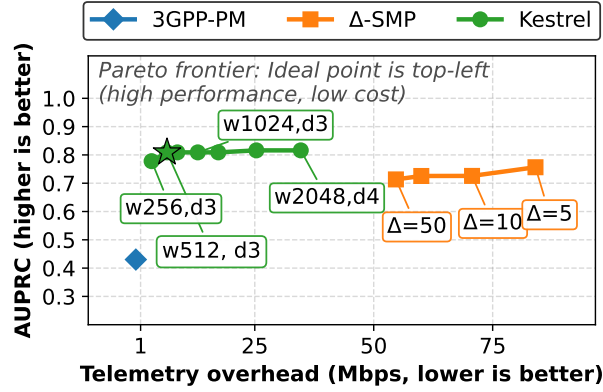


Figure 5.5: **Detection accuracy vs telemetry cost (Pareto tradeoff).** KESTREL with $w=512, d=3$ achieves AUPRC 0.81 at approximately 6 Mbps export rate, providing roughly $2\times$ higher accuracy than *3GPP-PM* and an order-of-magnitude lower telemetry volume than Δ -*SMP*.

Predictable and bounded cost. At the tested loads of $1 - 4$ Gbps (§5.5), KESTREL’s export rate remains stable at ~ 6 Mbps ($0.15 - 0.6\%$ of traffic rate). Overhead depends only on configuration parameters (w , d , bin count, and number of QIDs) rather than the instantaneous packet rate, making telemetry export cost deterministic and controllable. In contrast, Δ -*SMP*’s event-driven postcards scale unpredictably with traffic burstiness and anomaly severity, producing spikes during busy periods. In our evaluation, Δ -*SMP* consumed $60 - 80$ Mbps ($1.5 - 8\%$ of traffic rate), an order of magnitude higher despite lower accuracy.

Take-away. KESTREL provides high accuracy with predictable, configuration-bounded overhead, simplifying telemetry planning for operator-scale UPFs.

5.6.3 Responsiveness

We next evaluate responsiveness, measured as the *Time to First Detection (TTFD)*: the median delay between anomaly onset and the first alarm. Figure 5.6 summarizes results across telemetry approaches (left) and per anomaly type (right), using the tuned thresholds from §5.6.1.

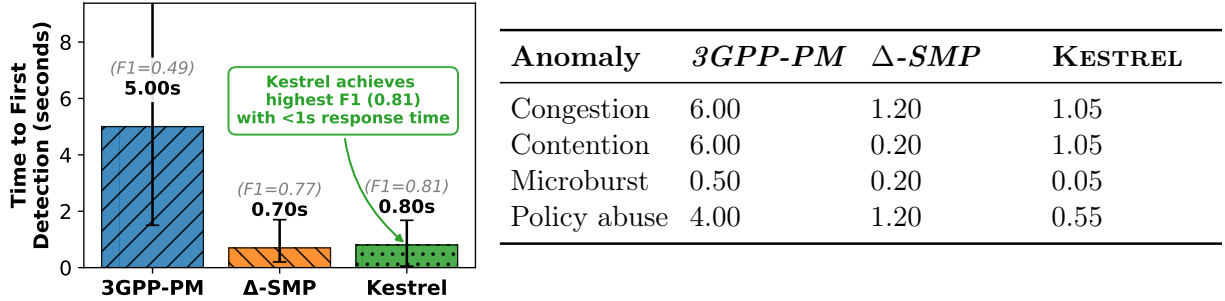


Figure 5.6: **Responsiveness measured by median time-to-first-detection (TTFD).** KESTREL achieves sub-second detection latency while maintaining the highest F1 score among evaluated schemes. The table reports per-anomaly breakdowns, showing that KESTREL detects short-lived events (e.g., microbursts) rapidly and consistently outperforms coarse counters.

Aggregate performance. Figure 5.6 (left) shows that KESTREL responds quickly (median 0.80s) with the highest detection performance (F1 0.81). *Δ-SMP* responds slightly faster (0.70s) but with lower performance (F1 0.77). *3GPP-PM* is much slower (5.00s) and far less accurate (F1 0.49), making it unsuitable for timely anomaly detection.

Per-anomaly breakdown. For *microbursts*, KESTREL responds immediately (0.05s median) while maintaining high F1 (0.92). *Δ-SMP* also reacts quickly (0.20s) but with lower accuracy, missing bursts when metric changes remain below the threshold. *3GPP-PM* responds reasonably fast (0.50s) but miss smaller events. *Δ-SMP* exhibits slightly lower latency in some cases (e.g., 0.20s for contention vs. 1.05s for KESTREL). This is because postcards are exported immediately upon threshold crossings, while KESTREL waits for the end of each window to query sketches. However, this speed advantage comes at the cost of detection fidelity: for *contention*, *Δ-SMP*’s AUPRC is only 0.37 (recall §5.6.1) compared to KESTREL’s 0.75, demonstrating that faster alarms are only valuable when reliable. For *congestion*, both achieve similar latency (1.05 – 1.20s) with high accuracy. For *policy abuse*, sketches detect anomalies within 0.55s, roughly 7× faster than *3GPP-PM* (4.00s) and 2× faster than *Δ-SMP* (1.20s).

Take-away. KESTREL balances responsiveness and accuracy: while per-packet export can react instantly to anomaly events, KESTREL’s windowed summarization achieves comparable latency with higher detection fidelity.

5.6.4 Microbenchmarks and Design Insights

We complement the accuracy and cost evaluation with microbenchmarks that validate KESTREL’s practicality. We discuss three aspects: (a) per-window processing latency, (b) sensitivity to binning strategies, and (c) hardware resource usage on the Tofino platform.

Scheme	p50 (ms)	IQR ($\pm$ ms)	I/F/ML (%)
3GPP-PM	1.83	0.04	37 / 29 / 33
Kestrel	53.3	0.45	27 / 72 / 1
Δ -SMP	200.4	7.3	33 / 66 / 1

Figure 5.7: **Per-window pipeline latency.** Median (p50) and interquartile range (IQR) of e2e processing time per 1 s window, with time breakdown across ingestion (I), feature extraction (F), and ML detection (ML).

Pipeline latency. We measure e2e processing time per window (every one second), broken down into ingestion (I), feature extraction (F), and ML detection (ML). Figure 5.7 shows that all methods finish well within the deadline: counters in ~ 2 ms, KESTREL in 53 ms, and Δ -SMP in 200 ms (median). Counters are trivially cheap (~ 2 ms) but have poor anomaly detection performance (§5.6.1). KESTREL spends most of its budget (72%) on feature extraction, as sketch registers must be queried and aggregated, yet the absolute cost remains modest: ~ 53 ms per window, independent of packet rate or number of flows. In contrast, Δ -SMP spends two-thirds of its budget on feature extraction because each sampled packet is processed individually, causing latency to grow with traffic volume. KESTREL also exhibits low variance (Interquartile Range (IQR) ± 0.45 ms), confirming predictable performance. Therefore, it strikes a practical middle ground: richer than counters, yet an order of magnitude faster and more stable than postcards.

Binning strategies. A key design choice in KESTREL is how to discretize latency and IAT distributions into bins. We evaluate four approaches that represent different philosophies for capturing distributional behavior: (1) *Target*, KESTREL’s target-occupancy method (§5.4.3), which caps baseline occupancy in diagnostic bins at $\rho\%$ to make anomalies appear as sharp spikes; (2) *P90*, which sets boundaries at the 90th and 99.8th percentiles

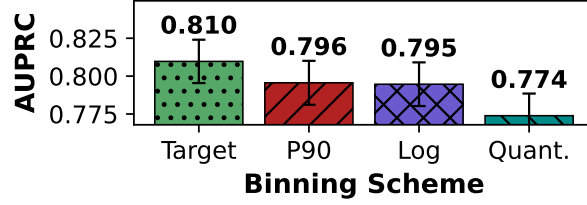


Figure 5.8: **Accuracy across binning strategies.** Adaptive target-occupancy binning yields consistently higher AUPRC across anomaly types compared to other binning strategies.

with log spacing to emphasize the “knee” of latency/IAT distributions; (3) *Log*, which applies uniformly log-spaced bins across a fixed global range, capturing both short and long tails without per-QID tuning; and (4) *Quantile*, which divides samples into equal-frequency bins for uniform sensitivity across the range. Figure 5.8 shows that target-occupancy achieves the highest AUPRC (0.810) compared to P90 (0.796), Log (0.795), and Quantile (0.774). Although these improvements seem modest, they have a practical impact on anomaly detection: target-occupancy reduces contention false positives by $\sim 20\%$ at 90% recall, compared to other binning approaches.

Hardware usage. KESTREL’s resource footprint on Tofino consumes less than 20% of SRAM, about 25% of hash units, and 35% of ALUs, leaving enough headroom for other UPF functions. These results demonstrate that KESTREL fits comfortably within the constraints of a commercial programmable switch.

Take-away. KESTREL’s principled, distribution-aware design translates into practical gains: bounded latency, efficient resource use, and reliable anomaly detection on real hardware.

5.6.5 Maintaining Detectability Under Distribution Drift

The results above establish that KESTREL achieves high detection accuracy, predictable overhead, and efficient hardware utilization under realistic UPF workloads.

We now turn to a complementary question: how KESTREL behaves under sustained distribution drift, where background traffic gradually shifts and increases occupancy in diagnostic regions. Such drift can erode detectability if left uncorrected. We therefore evaluate whether KESTREL’s adaptive re-binning mechanisms (§5.4.3) preserve diagnostic

sparsity under sustained drift, ensuring that the detectability conditions derived in §5.4 continue to hold in long-running deployments.

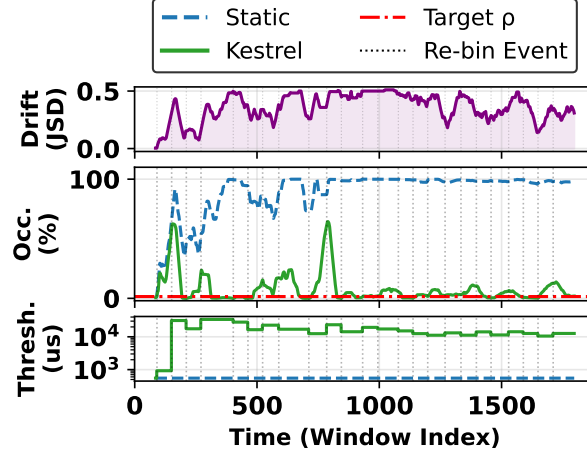


Figure 5.9: **Transient response under sustained distribution drift (QID 6).** Adaptive re-binning events (vertical dotted lines) counteract drift-induced accumulation in the diagnostic region, restoring occupancy toward the target ρ . Static binning leads to sustained saturation under the same drift episode.

Drift-aware re-binning restores sparsity. Figure 5.9 illustrates a representative sustained drift episode for QID 6. The top panel shows persistent divergence from the bootstrap baseline, measured via Jensen–Shannon Divergence (JSD) [60], confirming a long-lived shift in background traffic rather than transient noise. The middle panel shows the resulting diagnostic-region occupancy. Under static binning, background traffic accumulates in the diagnostic region, driving occupancy toward saturation and violating the sparsity assumptions required for detectability. In contrast, KESTREL maintains occupancy close to the target level ρ throughout the drift episode. The vertical re-bin markers indicate when KESTREL triggers adaptive re-partitioning: each re-binning event coincides with a corrective drop in occupancy, counteracting drift-induced accumulation. The bottom panel shows the corresponding evolution of the diagnostic threshold. While static binning uses a fixed threshold despite changing traffic conditions, KESTREL updates the threshold at re-bin events and adapts to distributional shifts. Despite operating on sketch summaries and bounded control plane samples, the thresholds selected by KESTREL closely track an oracle boundary computed from raw packet traces.

Together, these results show that KESTREL’s sketch-driven re-binning responds to sustained drift, restoring diagnostic sparsity while tracking the oracle threshold with

reasonable accuracy. This prevents drift from inflating the collision floor that undermines anomaly detectability, without requiring full traffic visibility (§5.4).

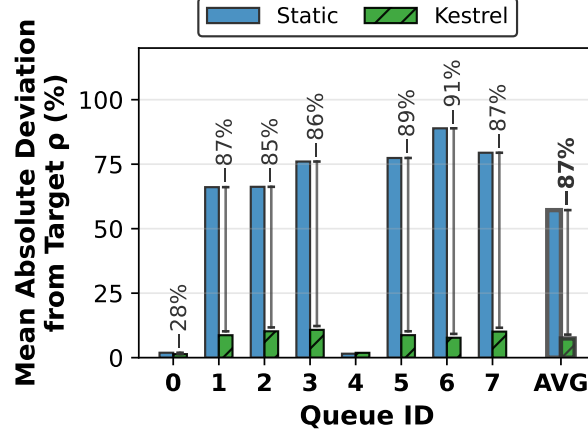


Figure 5.10: **Aggregate control accuracy across queues.** Mean absolute deviation from the target occupancy ρ under static and adaptive binning. Adaptive re-binning reduces deviation by an average of 87% across 8 queues.

System-wide reduction in occupancy error. Figure 5.10 shows the Mean Absolute Deviation (MAD) of diagnostic occupancy from the target ρ for each queue under sustained drift, along with the relative reduction achieved by KESTREL compared to static binning. Across most queues, adaptive re-binning substantially reduces deviation from the target, with improvements ranging from 28% to 91%. Averaged system-wide, KESTREL reduces MAD by 87% relative to static binning, indicating a strong overall tightening of occupancy control under drift. In one case, *i.e.*, QID 4, KESTREL introduces a slight increase in deviation. This is due to the fact that this queue experiences limited drift and remains close to the target even under static binning. In this case, re-binning introduces small periodic adjustments that marginally increase MAD.

These results show that while adaptive re-binning does not strictly dominate static binning on every queue, it delivers consistent and substantial reductions in occupancy deviation at the system level, preserving diagnostic sparsity across heterogeneous queues under sustained drift (§5.4).

Avoiding blindness under adaptive re-binning. A potential failure mode of occupancy-based control is *blindness*. A naïve controller could trivially satisfy the target ρ by pushing the diagnostic boundary deep into the tail, so that almost no traffic ever enters the diagnostic region. This would lead to the suppression of any anomaly signals altogether.

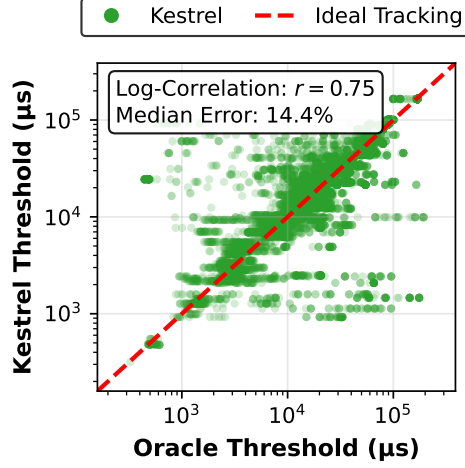


Figure 5.11: **Threshold sensitivity analysis.** Online diagnostic thresholds selected by KESTREL compared to oracle thresholds computed from raw traces (log–log scale). The strong correlation ($r = 0.75$) indicates that adaptive re-binning preserves sparsity without collapsing the diagnostic region into the tail.

Figure 5.11 addresses this concern by directly comparing the diagnostic thresholds selected by KESTREL against oracle thresholds computed from raw packet traces, across all windows and queues. Thresholds are plotted on a log–log scale to capture the wide dynamic range induced by heterogeneous traffic and sustained drift. Despite operating on sketch summaries and bounded control plane samples, KESTREL’s selected thresholds closely track the oracle, exhibiting strong log-correlation ($r = 0.75$) and a median absolute percentage error of 14.4%. Points concentrate around the ideal-tracking diagonal, indicating that threshold adjustments scale proportionally with the underlying distribution rather than collapsing into the tail.

Together, these results show that KESTREL does not maintain sparsity by making the diagnostic region empty. Rather, the adaptive re-binning mechanism (§5.4.3) preserves diagnostic sparsity while tracking the oracle thresholds. This maintains sensitivity to tail events and prevents blindness caused by drift.

Take-away. Sustained distribution drift can erode the sparsity assumptions required for sketch-based detectability if left uncorrected. These results show that KESTREL’s adaptive re-binning preserves diagnostic sparsity over time without inducing blindness: using only sketch-derived signals, the controller maintains occupancy near the target ρ and tracks oracle tail boundaries closely. Consequently, the detectability guarantees derived in

§5.4 continue to hold in long-running deployments without raw packet access or manual reconfiguration.

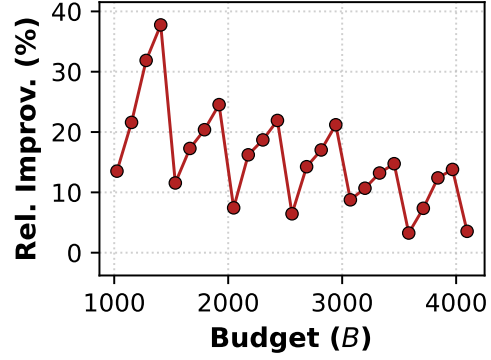
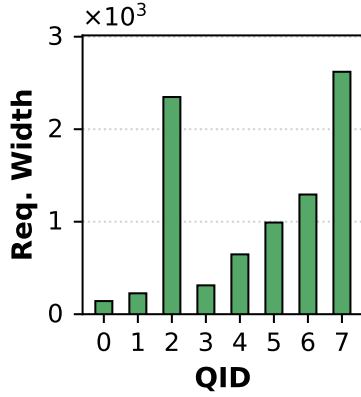
5.6.6 Budget-Aware Width Allocation Across QIDs

We now evaluate how KESTREL allocates sketch memory across heterogeneous traffic classes when the available memory budget is limited. While previous results used a fixed per-QID width, practical deployments must distribute a shared memory budget across QIDs whose traffic volumes and diagnostic occupancies may differ substantially. We begin by examining how detectability requirements vary across QIDs under realistic traffic conditions. Using the analytical model from §5.4.4, we compute the per-QID sketch width required to meet a common anomaly detectability target, based on measured traffic envelopes.

Heterogeneous detectability requirements. Figure 5.12(a) shows the required sketch width w_q^{req} for each QID. The results show substantial heterogeneity in required sketch widths to achieve similar detectability guarantees: the smallest QID requires fewer than 150 buckets, while the largest requires over 2600 buckets. This nearly $18\times$ spread reflects differences in both traffic volume and diagnostic-region occupancy across QIDs. As a result, a uniform-width allocation is inherently mismatched to the workload: light QIDs are over-provisioned, while heavy QIDs become detectability bottlenecks. This motivates budget-aware width allocation when total sketch memory is constrained.

Detectability under memory budgets. Figure 5.12(b) evaluates how budget-aware width allocation affects anomaly detectability under different fixed global sketch budgets B . We compare KESTREL’s ILP-based allocation against a uniform-width baseline, using the detectability metric from §5.4.4, which captures the collision-induced noise floor that limits the smallest detectable anomaly.

Across all budgets, optimized allocation consistently reduces the detectability threshold relative to uniform allocation. Under tight budgets, the improvement is significant. As shown in Figure 5.12(b), at a total width of 1408 buckets, optimized allocation lowers the collision floor by 38%, which allows detection of significantly weaker anomalies. The gap narrows with larger budgets since uniform allocation eventually provides sufficient width for all QIDs. However, even at larger budgets, optimized allocation provides measurable improvements by allocating more memory for traffic classes with higher diagnostic occupancy and volume. The sawtooth pattern reflects discrete width granularity ($w_{\text{min}}=64$, step size=64), enforced to minimize SRAM fragmentation on the switch hardware. This discreteness causes uniform allocation to improve only when all QIDs cross the next width step, while optimized allocation can benefit from incremental budget earlier.



(a) Per-QID sketch widths required to satisfy the detectability bound. (b) Detectability improvement of optimized allocation over uniform.

Figure 5.12: **Benefit of budget-aware width allocation.** (a) QIDs exhibit highly heterogeneous width requirements due to differences in traffic volume and diagnostic occupancy. (b) Under constrained budgets, optimized allocation significantly lowers the collision-induced noise floor (up to 38%), while the gap diminishes once sufficient memory is available.

Sufficient-memory regime. Throughout §5.6.1–§5.6.4, we use a uniform width $w=512$ ($B=4096$). While the analytical model in Figure 5.12(a) prescribes larger widths (e.g., for QID 7) to guarantee detectability under worst-case traffic envelopes, Figure 5.12(b) confirms that at $B=4096$ the system enters a regime of diminishing returns. Here, optimization improves detectability by only $\approx 3.5\%$, indicating that $w=512$ effectively masks heterogeneity for the evaluated workloads. Budget-aware allocation becomes critical only as memory tightens ($B < 3000$). Thus, our uniform baseline reflects a realistic, well-provisioned deployment, while the optimization in §5.4.4 extends KESTREL to memory-constrained environments.

Take-away. Traffic heterogeneity leads to widely varying memory requirements across QIDs. KESTREL’s budget-aware width allocation directly accounts for this heterogeneity by aligning sketch width with each QID’s traffic volume and diagnostic occupancy. As a result, KESTREL preserves robust anomaly detectability across operating regimes: converging to uniform allocation when memory is plentiful, and enforcing efficient, QID-aware allocation when resources become constrained.

5.7 Discussions and Limitations

Why Count-Min Sketch, and not more accurate sketches? A natural question is why KESTREL builds on CMS rather than on more accurate sketches such as ElasticSketch [127], NitroSketch [63], or CocoSketch [130]. These sketches optimize a different objective than KESTREL requires: they target *volume and frequency* queries (flow sizes, heavy hitters, partial-key sums), whereas KESTREL must preserve *distributional* features per (TEID, QFI), such as latency-tail occupancy and inter-arrival-time shifts, which these sketches do not track at all. A more accurate heavy-hitter estimate does not help detect a latency-tail anomaly. The relevant question is therefore not which sketch estimates volume most accurately, but which structure can be *extended* to carry compact per-flow distributions under data-plane constraints.

CMS is the right foundation for this because its simplicity is what makes it both analyzable and deployable: its error behavior admits the clean bounds underlying KESTREL’s detectability analysis (§5.4), and its hash-and-increment update maps directly onto ASIC pipelines while leaving ALU and SRAM budget for co-resident UPF forwarding, policing, and queuing. The more elaborate logic of higher-accuracy sketches is difficult to sustain at line rate and, more fundamentally, reduces a bias KESTREL does not depend on: its diagnostic signal is the *sparsity* of a designed diagnostic region, not the precision of a volume estimate. Residual collision noise is instead addressed by per-QID partitioning, which is feasible precisely because QIDs map to a small, architecturally bounded set of egress queues and 3GPP service classes [105, 124], while the unbounded per-TEID key is absorbed within each sketch through bucket sharing.

Matching the primitive to the monitoring task. KESTREL adopts sketches as its telemetry primitive, whereas Chapter 4 argued that sketches cannot provide the bounded-error, per-packet e2e fidelity required for slice SLA tracking. This apparent inconsistency reflects a difference in monitoring objectives rather than a contradiction. SLICESCOPE must reconstruct per-packet e2e metrics such as latency, jitter, and loss; for that task per-hop sketch summaries are insufficient, since they lose the packet alignment needed to determine whether observations at different hops belong to the same packet. KESTREL’s objective is different: it does not reconstruct each packet’s e2e trajectory, but preserves distributional evidence predictive of QoS anomalies within shared UPF resources. For that, compact per-(TEID, QFI) summaries, capturing tail-latency occupancy, inter-arrival-time shifts, and color fractions over a short window, are exactly the right representation. The same reasoning that leads SLICESCOPE to reject sketches for packet-aligned e2e tracking thus leads KESTREL to adopt them for local user-plane attribution.

Generality beyond QoS anomalies. The anomalies examined in this chapter, congestion,

microbursts, contention, and policy abuse, are representative of fine-grained user-plane anomalies, which have received comparatively little attention in the literature relative to control-plane and management-plane detection (§2.4). They leave no signature in control-plane state and are not visible in aggregate counters, surfacing only in per-flow latency, inter-arrival, and policing behavior.

KESTREL’s primitive is not tied to these specific scenarios. The question for generality is representational: it applies to any anomaly whose symptoms appear as sparse changes in bounded per-flow or per-tunnel summaries, and extends naturally to related effects on packet timing, queue occupancy, or traffic concentration. Anomalies whose evidence is absent from per-flow user-plane distributions, such as those requiring payload inspection, protocol-state reconstruction, or cross-domain causal reasoning, fall outside the current design. KESTREL’s applicability is therefore determined by whether an anomaly admits a compact data-plane signature, not by the specific anomaly types evaluated here.

Portability across dataplane targets. Our evaluation focuses on a Tofino-based UPF; however, KESTREL’s design is not specific to a single hardware target. Its sketch-based abstraction and fixed-rate export model naturally extend to other programmable data planes, including SmartNIC-based UPFs and software pipelines built on Extended Berkeley Packet Filter (eBPF) [32] or eXpress Data Path (XDP) [50]. Similarly, the same principles apply to other 5G data plane components such as the Centralized Unit (CU) in the Radio Access Network (RAN), where queueing and scheduling effects give rise to analogous distributional anomaly signatures. We do not evaluate these targets in this chapter; portability beyond Tofino remains an open question that warrants further investigation.

Integration with 5G analytics interfaces. While KESTREL focuses on exposing anomaly-relevant signals from the user plane, these signals naturally fit within existing 3GPP analytics frameworks. In particular, anomaly alerts produced from KESTREL’s sketch summaries can be encoded as standardized analytics outputs conforming to the service-based interface defined in TS 29.520 [4]. Concretely, KESTREL’s control plane detection logic can emit per-QID anomaly events using a vendor-specific **AnalyticsId** (e.g., **USER_PLANE_ANOMALY**), along with the affected QID, anomaly type, and confidence. This allows downstream consumers such as a central Network Data Analytics Function (NWDAF) or Session Management Function (SMF) to ingest fine-grained, distribution-aware user-plane insights using existing 3GPP interfaces, without requiring changes to the data plane or increasing telemetry export rates. Viewed this way, KESTREL acts as a domain-specific analytics producer for the user plane by distilling high-frequency sketch data into compact, actionable signals. These signals facilitate correlation with control- and management-plane analytics, enabling closed-loop monitoring and optimization without violating strict resource bounds.

In-switch Machine Learning (ML). Recent in-switch ML systems [11, 103, 126] show that simple learning or decision logic can execute directly within the switch pipeline, providing immediate reactions but limited flexibility. Such designs are restricted to local, low-dimensional features that fit within a few match-action stages and cannot represent distributional behavior. KESTREL takes a complementary path: it maintains minimal per-packet state while preserving rich per-flow distributions that support arbitrary analytics in the control plane. This decoupling enables multiple anomaly detectors or ML models to operate over the same exported sketches without modifying the data plane. These sketches could themselves serve as inputs to lightweight in-switch inference, and enable early anomaly detection without reconstructing full histograms. Hybrid designs that combine localized inference with sketch-based summaries represent a natural extension of this architecture, but are beyond the scope of this chapter.

5.8 Chapter Summary

This chapter addressed Research Question RQ3: how to achieve high-fidelity visibility into transient QoS anomalies within shared user-plane functions, under strict memory and bandwidth constraints. We argued that anomaly detection in programmable data planes is inherently distributional and per-flow in nature, yet naïve per-packet telemetry is prohibitively expensive. Bridging this gap requires rethinking telemetry design around detectability rather than raw visibility.

To this end, we developed KESTREL, a detectability-driven telemetry architecture that extends sketch-based measurement with compact distributional summaries. By capturing latency tails, inter-arrival patterns, and per-TEID deviations within bounded on-chip state, KESTREL enables fine-grained anomaly detection without incurring per-packet export overhead. We further introduced a formal detectability framework that characterizes how sketch collisions and drift affect anomaly visibility, and derived practical design principles for partitioning, binning, and parameter sizing.

Our prototype based on the Intel Tofino programmable switch demonstrates that this approach is both effective and practical: KESTREL achieves a) higher anomaly detection accuracy than selective postcard sampling while reducing telemetry volume by an order of magnitude, b) maintains sub-second responsiveness, and c) fits comfortably within hardware resource budgets. Moreover, adaptive re-binning preserves diagnostic sparsity under sustained distribution drift, ensuring that the detectability guarantees continue to hold over the course of long-running deployments.

Together, these results show that fine-grained anomaly attribution in the user plane can be realized efficiently through principled telemetry abstractions. With architectural slice visibility (Chapter 3), budget-aware slice control (Chapter 4), and detectability-driven user-plane attribution (KESTREL), this dissertation establishes a framework for efficient, high-precision monitoring of network slices in 5G and beyond networks. The final chapter synthesizes these contributions and outlines broader implications and future research directions.

Chapter 6

Conclusion & Future Work

This dissertation addressed a fundamental problem in Fifth Generation (5G) and beyond networks: monitoring of network slices, which is both efficient and high-precision. In Chapter 1, we argued that this problem is structural, arising from (i) architectural fragmentation across domains, (ii) rigid and slice-agnostic monitoring policies, and (iii) memory and bandwidth limitations in the data plane, which limit flexible monitoring primitives.

Rather than seeking a monolithic solution, this thesis decomposed the problem along three complementary dimensions of the monitoring stack — architectural abstraction, budgeted Service Level Agreement (SLA)-aware control, and fine-grained attribution in the user-plane, and subsequently addressed each dimension systematically.

This final chapter synthesizes the contributions of this research, discusses its limitations, and outlines future directions for slice-centric monitoring.

6.1 Summary of Contributions

MONARCH: Slice-Native Monitoring Architecture (Chapter 3). At the architectural layer, we addressed the abstraction gap between slice Key Performance Indicators (KPIs), which are defined end-to-end (e2e), and the telemetry collected by existing monitoring architectures, which is typically slice-agnostic. MONARCH elevates slices to first-class monitored entities, and provides a unified northbound Application Programming Interface (API) for expressing slice-level monitoring intent. By centralizing cross-domain correlation within the monitoring layer, MONARCH decouples slice-level KPI computation from

management applications, reducing coupling between monitoring and control. Our prototype implementation scales linearly with the number of slices while maintaining bounded ingestion latency, and establishes the architectural foundation for slice-centric observability.

SLICESCOPE: SLA-Aware Monitoring Control (Chapter 4). At the monitoring-control layer, we formalized network slice monitoring as a closed-loop resource allocation problem. By treating telemetry fidelity as a shared and allocatable resource, SLICESCOPE enables monitoring precision to be distributed according to slice SLA under explicit global budgets. The introduction of the Telemetry Primitive Contract (TPC) separates the control objectives from their realization in the data plane, providing a general abstraction for budget-aware telemetry. Our evaluations show that such SLA-aware control of monitoring can significantly improve accuracy for critical slices while adhering to global resource constraints.

KESTREL: Detectability-Driven User-Plane Telemetry (Chapter 5). At the user-plane layer, we addressed the challenge of fine-grained anomaly attribution within data plane resources shared across slices. KESTREL reframes telemetry design around detectability rather than raw visibility, and uses compact distributional summaries to preserve per-(Tunnel Endpoint Identifier (TEID), QoS Flow Identifier (QFI)) anomaly signals with bounded memory. Our analysis provides formal detectability conditions for KESTREL’s design, and our prototype on programmable hardware demonstrates that fine-grained attribution can be achieved at an order-of-magnitude lower export cost than commonly used approaches.

6.2 Cross-Layer Synthesis

Viewed collectively, these contributions establish a cross-layer framework for achieving efficient and high-precision slice monitoring by directly addressing the three structural constraints identified in Chapter 1: cross-domain fragmentation, slice-agnostic fidelity policies, and data plane resource limits.

At the architectural layer, MONARCH defines visibility at slice scope by enabling the expression and computation of e2e slice KPIs from heterogeneous telemetry sources. At the monitoring-control layer, SLICESCOPE regulates precision by allocating monitoring fidelity in proportion to slice SLAs under explicit budgets. At the user-plane layer, KESTREL refines visibility to per-tunnel or per-flow granularity when attribution is required, using detectability-preserving summaries rather than uniform high-rate telemetry.

This hierarchy shows that efficient and high-precision monitoring does not require

uniformly high-fidelity telemetry across the network. High precision arises from slice-scoped semantics and progressive refinement of visibility, while efficiency is preserved through explicit budget allocation and detectability-aware user-plane summaries. The following scenario illustrates how these principles jointly enable precise diagnosis of slice-level violations without incurring prohibitive telemetry overhead.

6.2.1 Slice SLA Violation and Progressive Attribution

To illustrate the hierarchical refinement enabled by the contributions of this dissertation, consider an enterprise slice identified by S-NSSAI = 1-000001, provisioned for industrial automation. The slice enforces an SLA of $p99$ e2e latency ≤ 15 ms and jitter ≤ 3 ms [5]. Within the slice, traffic from industrial devices (e.g., robots and cameras) is carried over multiple Quality of Services (QoS) flows, associated with robot control (QFI=5), camera feed (QFI=7), and sensor telemetry (QFI=9). User sessions are anchored at two User Plane Functions (UPFs) and traverse disjoint paths in the transport network.

Now, consider a scenario in which transient congestion within one UPF inflates tail latencies for a subset of user sessions. This raises a fundamental monitoring question: *how can we determine whether the slice SLA has been violated, and if so, progressively localize and attribute the degradation under strict telemetry budgets?*

This question is addressed through a progressive refinement of visibility across the monitoring stack, as follows:

Step 1: Slice-Scoped SLA Detection. Leveraging the slice abstractions provided by MONARCH, a monitoring application submits a slice latency KPI monitoring request. MONARCH can materialize this request using a telemetry backend such as SLICESCOPE, which provides fine-timescale, path-aware latency measurement under explicit monitoring budgets. As the monitoring fidelity is allocated in proportion to slice SLA criticality, the enterprise slice can receive sufficient measurement precision; this allows the system to expose transient tail-latency inflation, without requiring uniform high-fidelity monitoring across all slices.

The resulting $p99$ latency timeseries rises to 27 ms, exceeding the 15 ms SLA threshold. At this stage, visibility is strictly slice-level: the system detects that the slice SLA is violated, but not yet why.

Step 2: Fault-Domain Narrowing. Although the degradation is observed at the slice level, the slice is realized across multiple UPF instances and transport paths. Localization can therefore proceed by refining the monitoring scope.

Using MONARCH’s unified monitoring interface, the monitoring application can issue subsequent queries at a narrower scope (*e.g.*, per-UPF or per-path) while preserving the same slice context. For example, it can request the same latency KPI broken down by UPF instance. In this example, consider it observes that only UPF-2 exhibits elevated tail latency, while UPF-1 remains normal. Further, if the transport telemetry further distinguishes forwarding context, the degradation is additionally associated with path-3.

The fault domain can thereby be narrowed from *the entire slice* to $\text{slice} \cap \text{UPF-2} \cap \text{path-3}$. This refinement supports targeted investigation and informs the next step of fine-grained attribution, avoiding coarse actions such as scaling or rate-limiting the entire slice.

Step 3: Fine-Grained User-Plane Attribution. Within the implicated UPF, KESTREL continuously maintains lightweight, sketch-based summaries at $(TEID, QFI)$ granularity under strict memory constraints. These always-on summaries preserve distributional visibility without per-packet export, but do not by themselves monitor e2e slice SLA.

Following the slice-level detection (Step 1) and fault-domain narrowing (Step 2), the monitoring application can consult KESTREL’s view and identifies a small set of offending $(TEID, QFI)$ pairs. In this scenario, the tail-latency inflation is concentrated in a small number of TEIDs carrying vision uplink traffic ($QFI=7$), while TEIDs dominated by robot-control traffic ($QFI=5$) remain nominal. The slice-level $p99$ inflation therefore originates from a subset of user sessions, rather than the entire slice workload.

Mitigation strategies. The fine-grained attribution as discussed above enables targeted operational actions by management applications. Possible mitigation strategies can include (i) rate-limiting the offending TEIDs, (ii) adjusting QoS policies for the concerned QFI, or (iii) migrating the affected sessions to a different UPF instance. Because it is possible to localize the degradation precisely, such mitigation actions can be applied selectively, without penalizing the entire slice.

Takeaway. This scenario illustrates a structured refinement of visibility from slice-level SLA detection (SLICESCOPE), to cross-domain scoping (MONARCH), and finally fine-grained attribution (KESTREL). Each layer consists of independently evaluated mechanisms, and the example above demonstrates how their abstractions compose together conceptually. As we can see, the high-precision visibility does not stem from high-fidelity telemetry applied uniformly across layers. Rather, it emerges from slice-level abstractions, SLA-aware budget allocation, and detectability-driven user-plane summaries working in concert.

6.3 Limitations and Design Tradeoffs

This dissertation demonstrates that efficient and high-precision slice monitoring is achievable through cross-layer design. However, several limitations and design tradeoffs remain.

Integration scope. The systems developed in this thesis, namely slice-level monitoring abstractions (MONARCH), SLA-aware monitoring control (SLICESCOPE), and detectability-driven user-plane telemetry (KESTREL), are intended to compose through a common slice abstraction and aligned monitoring interfaces. Together, they support a progressive monitoring workflow: slice-level KPI visibility can identify when a slice is at risk, SLA-aware control can allocate monitoring fidelity according to slice criticality and observed dynamics, and user-plane telemetry can provide finer-grained evidence for attribution when coarse slice-level KPIs are insufficient. The correctness, efficiency, and scalability claims of these systems are evaluated independently in their respective chapters, because each contribution operates at a different layer of the monitoring stack and is best evaluated against different baselines and constraints. §6.2.1 shows how they can work together conceptually. However, a fully unified runtime integrating these systems that automatically orchestrates cross-layer refinement from slice-level detection to fine-grained attribution is out of scope of this dissertation.

Such a system would require (i) explicit policies for refining monitoring scope and (ii) automated decision engines that can reason across and coordinate different telemetry views. These concerns extend beyond monitoring design into control plane orchestration and policy engineering. Moreover, realizing such a runtime does not require fundamentally new monitoring primitives; it would build upon the abstractions and mechanisms developed in this dissertation.

Domain focus and generalization. The contributions of this dissertation are developed and validated within the context of *network slicing in 5G networks*, with particular focus on cloud-native core network and programmable data planes. Although the underlying principles are not inherently specific to 5G, their application to other sliced or service-differentiated environments requires further study. Examples of such environments include cloud providers offering service-specific connectivity with distinct performance guarantees (*e.g.*, Azure ExpressRoute [72], Google Dedicated Interconnect [42]), as well as wide-area networks supporting differentiated services [35]. Although these systems share a similar notion of *slicing*, they may include different constraints, control plane assumptions, or traffic characteristics than those studied in this dissertation. Thus, adapting the developed mechanisms to these environments may require further modifications.

Design tradeoffs. The monitoring mechanisms developed in this dissertation are

mainly associated with two fundamental design tradeoffs: (i) the amount of telemetry detail that can be sustained under strict resource constraints, and (ii) the degree to which slice and SLA semantics are embedded within the monitoring layer.

- *Detail versus affordability.* High-precision monitoring improves the detectability of transient slice-level anomalies, but increases bandwidth, memory, and processing overhead in both control and data planes. This dissertation does not eliminate this tension; rather, the mechanisms developed manage this trade-off explicitly. SLICESCOPE regulates telemetry fidelity under slice-level budgets, allocating measurement precision in proportion to SLA criticality. Next, KESTREL further constrains telemetry volume by replacing per-packet export with compact, detectability-aware distributional summaries. Consequently, the monitoring detail is governed by operator-configured budgets and hardware constraints.
- *Raw export versus semantic structuring.* A second architectural choice concerns the role of the monitoring layer: should it simply export metrics for external analysis, or should it embed slice awareness and SLA context to support more systematic diagnosis? In operational environments, traditional infrastructure-centric monitoring often triggers high volumes of loosely related telemetry whose service-level impact is not immediately clear. This, in turn, increases time-to-resolution.

In contrast, this dissertation embeds slice semantics and SLA-awareness directly within the monitoring layer. MONARCH elevates slices to first-class monitored entities and provides a unified interface for monitoring at different scopes, while SLICESCOPE and KESTREL enable structured telemetry that supports progressive localization from slice-level detection to user-plane attribution. By structuring telemetry around slice abstractions rather than raw metrics export, this dissertation improves scalability and precision without requiring uniform high-fidelity instrumentation.

We next consider the threats to validity that arise from the evaluation methodology.

6.4 Threats to Validity

The mechanisms in this dissertation are evaluated through prototype implementations, programmable-switch deployments, emulation, and trace-driven experiments. This methodology supports controlled, reproducible study, but it also shapes how far the results generalize.

We discuss the principal threats to validity and, for each, the extent to which the conclusions hold despite it.

Testbed scale and production generality. The experimental testbed (*cf.* Appendix A) captures key architectural properties of 5G slice deployments, including a cloud-native 5G core, user-plane forwarding, programmable data-plane telemetry, and slice-specific monitoring pipelines. However, it is smaller than a production mobile network in terms of the number of physical sites, gNBs, users, and concurrently active services. The goal of the testbed is therefore not to reproduce the full scale of a carrier network, but to validate the mechanisms under realistic protocol, deployment, and hardware constraints. To partially address this limitation, the chapters combine testbed measurements with scalability experiments, trace-driven workloads, and parameter sweeps. Nevertheless, absolute overhead values and control-loop timescales may vary in larger operational deployments.

Workload and anomaly realism. The evaluation combines real traces, emulated 5G traffic, and injected workload dynamics to obtain controlled ground truth for repeatable comparison. This design allows us to isolate the effect of telemetry granularity on anomaly detection, but it does not capture the full variability of production mobile networks. Operational deployments may exhibit richer correlations induced by mobility, radio conditions, tenant behavior, diurnal load, handovers, and orchestration actions. As a result, the absolute detection gains measured in our testbed may vary under production traffic mixes, even though the evaluation captures the telemetry properties required by the targeted anomaly class.

KESTREL also makes a deliberate choice about anomaly scope. It targets fine-grained user-plane QoS anomalies, such as congestion, contention, policy abuse, and microbursts, that can evade aggregate counters and need not leave an explicit control-plane signature. Our characterization shows that this class of anomalies requires per-flow distributional visibility (§5.2), and the evaluation demonstrates that KESTREL’s telemetry primitives preserve the corresponding latency, inter-arrival, drop/color, and per-flow contribution signals. The results therefore do not generalize to anomalies whose primary evidence lies outside these signals, such as attacks requiring payload inspection, protocol-state reconstruction, or cross-layer radio diagnostics. Supporting such cases would require augmenting KESTREL’s sketch summaries with additional telemetry sources rather than changing the core argument that fine-grained user-plane evidence is necessary for this anomaly class.

Hardware and implementation assumptions. SLICESCOPE and KESTREL are evaluated using programmable data planes, with the Intel Tofino [51] serving as the primary hardware target. This is a realistic platform for studying line-rate telemetry under strict memory and pipeline constraints, but the exact resource trade-offs depend on the target

Application-Specific Integrated Circuit (ASIC), switch generation, compiler, and co-located forwarding logic. Designs that fit on Tofino are likely to be feasible on comparable or more capable programmable targets, but the specific limits on stateful ALUs, register accesses, hash units, and metadata width are hardware-dependent. Similarly, software data planes such as eBPF [32], VPP [36], or SmartNIC-based UPFs may support different trade-offs between expressiveness, update rate, and resource isolation.

Modeling and optimization assumptions. SLICESCOPE models the relationship between telemetry thresholds, monitoring error, and overhead using observations collected over control epochs. This assumes that recent metric-variation patterns are informative for the near future. Sudden changes within an epoch are still visible to the data plane because change-triggered insertion continues to emit telemetry when metric deviations exceed the installed thresholds, but the global budget allocation is not re-optimized until the next epoch. Thus, abrupt workload shifts may cause temporary overhead inflation or suboptimal threshold allocation. This behavior is inherent to any closed-loop controller operating at discrete epochs, and the epoch-length evaluation (§4.5.4) characterizes the resulting responsiveness–stability trade-off.

Composition of the three contributions. As discussed in §6.3, the three systems are evaluated individually and their composition (§6.2.1) is demonstrated conceptually rather than in a unified runtime. It is therefore possible that effects arising only from their interaction would not appear in the individual evaluations, for example contention for a shared monitoring budget, or interactions between SLICESCOPE’s epoch-based re-optimization and KESTREL’s always-on summaries operating at different timescales. Such effects could be characterized by implementing a unified runtime and evaluating the three systems jointly under shared workloads, which we leave to future work.

Metric and SLA assumptions. The thesis focuses on monitoring metrics that are central to slice assurance, including throughput, latency, loss, jitter, and QoS-related user-plane signals. The exact SLA targets, monitoring tolerances, and anomaly thresholds used in the evaluations are derived from representative 5G service classes and experimental design choices. Operators may choose different SLA contracts, aggregation windows, or violation semantics in practice. The proposed mechanisms are intended to be parameterized by such operator policies, but the measured numerical improvements should be interpreted relative to the workload and SLA assumptions used in the evaluation.

Several of these threats mark the boundary of what the dissertation establishes: the conceptual treatment of composition, the scope of the targeted anomaly class, and the detection latency imposed by epoch-based and control-plane analysis each point to open problems beyond the current design. We turn to these next.

6.5 Future Research Directions

The scenario in §6.2.1 illustrates the boundary of this dissertation: once the degradation is localized to a set of $(TEID, QFI)$ pairs at UPF-2, the framework is not designed to determine *why* those tunnels are degraded. In this direction, we identify three open research problems that follow directly: *what caused the violation?* (§6.5.1), *how to look efficiently?* (§6.5.2), and *how early can detection occur?* (§6.5.3).

6.5.1 Causal Attribution Across the Slice–Infrastructure Boundary

The first natural question to ask is: *what caused the violation?* The contributions in this dissertation close the observability gap: in the case of SLA violations, we can identify *where* the impact occurs and which slice entities are involved. However, the causal attribution step, which involves explaining *why* the violation emerged, remains open.

Recent Root Cause Analysis (RCA) frameworks for cloud-native systems [46, 129], perform joint reasoning over heterogeneous signals (*e.g.*, traces, metrics, logs) and model dependencies among infrastructure entities using graph-based learning. These approaches have shown good causal localization capabilities within cloud-native infrastructures.

However, they assume that causality is structured over infrastructure components (*e.g.*, services, pods, nodes). In next-generation networks, the primary entities of interest (slices) are logical overlays that share physical resources. Slice identifiers (S-NSSAI, QFI, TEID) are not independent components in the infrastructure graph; rather, they are mappings onto shared queues, schedulers, and processing pipelines. Therefore, simply replacing the infrastructure metrics with slice-level KPIs does not resolve this gap. Causal attribution in sliced networks requires a model that explicitly captures the relationships between logical slice entities and shared physical resources, as well as the possibility of cross-slice interference.

This attribution problem is further complicated by the fact that observability is not uniformly available: SLICESCOPE allocates monitoring fidelity in proportion to SLA criticality, which means the causal record is sparser by design for lower-priority slices. Causal attribution must therefore operate under structured, policy-based incompleteness, which is a setting that existing RCA frameworks do not address.

This raises a concrete design question: should causal reasoning in sliced networks be structured over infrastructure components, over slice-level entities, or over hierarchical graphs that explicitly capture their relationships? The S-NSSAI $\rightarrow$ QoS flow $\rightarrow$ TEID

mapping used in this dissertation provides a structural foundation for such graphs, but the design of general slice-aware causal reasoning remains open.

6.5.2 Principled Scope Refinement Policies

The second open problem is related to the traversal of the monitoring hierarchy. This dissertation demonstrates that progressive refinement of visibility is possible, starting from slice-level KPIs to per-UPF views and finally to per-TEID attribution (*cf.* §6.2.1), but delegates the reasoning logic to external applications. The question of *when* and *how* to initiate each refinement step is left open.

This challenge is conceptual rather than merely being an engineering problem. Existing RCA systems that perform multi-level localization determine traversal decisions from learned or statistically-inferred dependency structures [129]. In the 5G data plane, the S-NSSAI $\rightarrow$ UPF $\rightarrow$ TEID hierarchy is not learned but structurally defined by the data plane architecture and 3rd Generation Partnership Project (3GPP) specifications [1].

This hierarchy does not preclude a statistical learning process, rather it defines which transitions between monitoring views are structurally valid, independent of any observed data. Therefore, such structural constraints and statistical or learned evidence can play complementary roles: the protocol hierarchy prunes the space of candidate refinement steps, while anomaly signals can determine whether and when descent into a narrower scope is warranted. In this context, neither approach is sufficient alone. On one hand, statistical methods without structural grounding face a cold-start problem in inferring which refinement steps are meaningful. On the other hand, structural traversal without statistical evidence cannot determine when a deeper view is necessary for diagnosis.

The key difficulty in the refinement process is that the steps require selecting among multiple monitoring views with heterogeneous cost and fidelity properties. Descending from slice scope to UPF scope may involve, for example, monitoring e2e paths using SLICESCOPE, infrastructure Network Function (NF) metrics/state monitoring through polling, and transitioning to per-TEID attribution leveraging summaries maintained by KESTREL.

A principled refinement policy must therefore reason jointly about (i) the diagnostic value of a candidate view *i.e.*, how much ambiguity it reduces in the fault domain, and (ii) the operational cost of obtaining or processing that view, which may involve telemetry bandwidth, coordination costs, as well as processing overhead. Formulating refinement as a structured selection problem over heterogeneous monitoring mechanisms, and designing

policies that exploit the structure of the 5G protocol stack to constrain the search space, is an interesting direction this work brings into focus.

6.5.3 Data and Control Plane Boundary in Anomaly Detection

The monitoring architecture developed in this dissertation draws a deliberate boundary between the data- and the control plane: KESTREL maintains compact sketch-based summaries at $(TEID, QFI)$ granularity within the data plane, while the anomaly detection and attribution logic execute in the control plane over exported sketches. This separation keeps the data plane simple and allows multiple analytics applications to operate over the same exported summaries without modifying forwarding logic.

However, this boundary has a cost: anomaly detection latency is dependent on the sketch export interval, which may be insufficient for use-cases that require very-low latency violation detection. Recent work on in-network machine learning demonstrates that lightweight inference can execute directly within programmable switch pipelines [11, 103, 126], providing immediate reactions to local, low-dimensional signals. Such designs are complementary to KESTREL’s approach: in-switch inference is restricted to features that fit within match-action stages, while KESTREL’s sketches preserve rich distributional summaries that support arbitrary control plane analytics.

This raises a research question specific to slice-aware monitoring: how should the boundary between data plane summarization and control plane inference be positioned, and how does moving that boundary affect the detectability guarantees that KESTREL’s design provides? Pushing inference closer to the data plane reduces detection latency, but requires approximating the distributional summaries that full sketch export preserves; pulling it further into the control plane preserves analytical richness but increases reaction time. Characterizing this tradeoff formally, and designing hybrid architectures that exploit both in-network early detection and control plane attribution, is a concrete direction that follows directly from the monitoring primitives developed in this dissertation.

References

- [1] 3GPP. Management and Orchestration; 5G end to end Key Performance Indicators (KPI). Technical Specification (TS) 28.554, 3GPP, 2020. Version 17.0.0.
- [2] 3GPP. Management and Orchestration; 5G Performance Measurements. Technical Specification (TS) 28.552, 3GPP, 2020. Version 17.0.0.
- [3] 3GPP. System architecture for the 5G system; Stage 2. Technical Specification (TS) 23.501, 3GPP, 2020. Version 16.5.1.
- [4] 3GPP. 5G System; Network Data Analytics Services; Stage 3. Technical Specification (TS) 29.520, 3GPP, 2023. Version 17.12.0.
- [5] 3GPP. Service requirements for cyber-physical control applications in vertical domains. Technical Specification (TS) 22.104, 3rd Generation Partnership Project (3GPP), 2023.
- [6] 3GPP. Architecture enhancements for 5G System (5GS) to support network data analytics services. Technical Specification (TS) 23.288, 3GPP, 2024.
- [7] 3GPP. Service requirements for the 5G system. Technical Specification (TS) 22.261, 3GPP, 2024. Version 18.14.0.
- [8] 5G Americas. 5G Services and Use Cases. White paper, 5G Americas, 2017.
- [9] 5G Americas. New Services & Applications with 5G Ultra-Reliable Low Latency Communications. White paper, 5G Americas, 2018.
- [10] 5G Americas. Vehicular Connectivity: CV2X and 5G. White paper, 5G Americas, 2021.

- [11] Aristide Tanyi-Jong Akem, Guillaume Fraysse, and Marco Fiore. Encrypted traffic classification at line rate in programmable switches with machine learning. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9, 2024.
- [12] Alliance for Telecommunications Industry Solutions (ATIS). IOT Categorization: Exploring the Need for Standardizing Additional Network Slices, 2019. <https://access.atis.org/higherlogic/ws/public/download/51129/ATIS-I-0000075.pdf>.
- [13] Arjun Balasingam, Manikanta Kotaru, and Paramvir Bahl. Application-Level Service Assurance with 5G RAN Slicing. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 841–857, 2024.
- [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. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–7, 2020.
- [15] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. PINT: Probabilistic In-band Network Telemetry. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 662–680, 2020.
- [16] Deval Bhamare, Andreas Kasser, Jonathan Vestin, Mohammad Ali Khoshkholghi, Javid Taheri, Toktam Mahmoodi, Peter Öhlén, and Calin Curescu. IntOpt: In-band Network Telemetry Optimization Framework to Monitor Network Slices Using P4. *Computer Networks*, 216:109214, 2022.
- [17] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: programming protocol-independent packet processors. *ACM Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review*, 44(3):87–95, July 2014.
- [18] Badre Bousalem, Vinicius F. Silva, Rami Langar, and Sylvain Cherrier. DDoS attacks detection and mitigation in 5G and beyond networks: A deep learning-based approach. In *Proc. IEEE Global Communications Conference (GLOBECOM)*, pages 1259–1264, 2022.
- [19] F. Brockners, S. Bhandari, and T. Mizrahi. Data Fields for In Situ Operations, Administration, and Maintenance (IOAM). RFC 9197, May 2022.

- [20] Ashima Chawla, Anne-Marie Bosneag, and Anestis Dalgkitsis. Graph-based interpretable anomaly detection framework for network slice management in beyond 5G networks. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–6, 2023.
- [21] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.
- [22] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. BeauCoup: Answering Many Network Traffic Queries, One Memory Update at a Time. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 226–239, 2020.
- [23] Yongzhou Chen, Ruihao Yao, Haitham Hassanieh, and Radhika Mittal. Channel-Aware 5G RAN Slicing with Customizable Schedulers. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1767–1782, 2023.
- [24] Yong-Hoon Choi, Daegyeom Kim, and Myeongjin Ko. 5G Traffic Datasets. <https://dx.doi.org/10.21227/ewhk-n061>, 2023. DOI: 10.21227/ewhk-n061.
- [25] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, et al. dRMT: Disaggregated Programmable Switching. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 1–14, 2017.
- [26] Shihabur Rahman Chowdhury, Raouf Boutaba, and Jérôme François. LINT: Accuracy-adaptive and Lightweight In-band Network Telemetry. In *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 349–357, 2021.
- [27] Cloud Native Computing Foundation (CNCF). Thanos: Open source, highly available Prometheus setup with long term storage capabilities. <https://thanos.io/>, 2022. Accessed: 2022-01-13.
- [28] CNCF Telecom User Group. Cloud native thinking for telecommunications. Technical report, Cloud Native Computing Foundation, 2019.
- [29] Marc Cohn. The Case for Private Network SLAs: A Real-World View. <https://www.spirent.com/blogs/the-case-for-private-network-slas-a-real-world-view>, 2023. Accessed: 2025-01-16.

- [30] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [31] DPDK Project. Data Plane Development Kit (DPDK). <https://www.dpdk.org/>, 2024.
- [32] eBPF Foundation. eBPF: Introduction, tutorials & community resources. <https://ebpf.io/>, 2024.
- [33] Ericsson. Ericsson and Vodafone create UK’s first on-demand 5G network slice. <https://www.ericsson.com/en/news/3/2022/ericsson-and-vodafone-create-uks-first-on-demand-5g-network-slice>, March 2022. Technical Press Release. Accessed: 2025-12-23.
- [34] ETSI. Network functions virtualisation (nfv) release 3; architecture; report on the enhancements of the nfv architecture towards “cloud-native” and “paas”. Technical Report GR NFV-IFA 029 V3.3.1, European Telecommunications Standards Institute, November 2019.
- [35] Adrian Farrel, John Drake, Reza Rokui, Shunsuke Homma, Kiran Makhijani, Luis M. Contreras, and Jeff Tantsura. A Framework for Network Slices in Networks Built from IETF Technologies. RFC 9543, Mar 2024.
- [36] FD.io. VPP: Vector Packet Processing. <https://fd.io/>, 2024. Accessed: 2026-06-27.
- [37] Xenofon Foukas, Navid Nikaein, Mohamed M. Kassem, Mahesh K. Marina, and Kimon Kontovasilis. FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks. In *Proc. ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 427–441, 2016.
- [38] Xenofon Foukas, Bozidar Radunovic, Matthew Balkwill, and Zhihua Lai. Taking 5G RAN Analytics and Control to a New Level. In *Proc. ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 1–16, 2023.
- [39] Xuesong Geng, Luis M. Contreras, Reza Rokui, Jie Dong, and Ivan Bykov. IETF Network Slice Application in 3GPP 5G End-to-End Network Slice. IETF Internet-Draft, draft-ietf-teas-5g-network-slice-application-06, 2026. Work in Progress. Accessed: 2026-03-05.
- [40] Dimitris Giannopoulos, Panagiotis Papaioannou, Lydia Ntzogani, Christos Tranoris, and Spyros Denazis. A Holistic Approach for 5G Network Slice Monitoring. In *Proc. IEEE MeditCom*, pages 240–245, 2021.

- [41] Google. cAdvisor: Analyzes Resource Usage and Performance Characteristics of Running Containers. <https://github.com/google/cadvisor>, 2025. Accessed: 2026-02-16.
- [42] Google Cloud. Dedicated Interconnect overview, 2024. Accessed: 2025-05-30.
- [43] Grafana Labs. Grafana Cloud. <https://grafana.com/products/cloud/>, 2025. Accessed: 2025-02-20.
- [44] Ali Gungor. UERANSIM: Open Source 5G UE and RAN Simulator. <https://github.com/aligungr/UERANSIM>, 2022. Accessed: 2022-01-13.
- [45] Gurobi Optimizer. The world’s fastest solver. <https://www.gurobi.com/solutions/gurobi-optimizer/>.
- [46] Yongqi Han, Qingfeng Du, Ying Huang, Pengsheng Li, Xiaonan Shi, Jiaqi Wu, Pei Fang, Fulong Tian, and Cheng He. Holistic root cause analysis for failures in cloud-native systems through observability data. *IEEE Transactions on Services Computing*, 17(6):3789–3802, 2024.
- [47] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 71–85, 2014.
- [48] Juha Heinanen and Roch Guerin. A Two Rate Three Color Marker (trTCM). RFC 2698, IETF, 1999. Available at <https://www.rfc-editor.org/rfc/rfc2698.txt>.
- [49] Rumenigwe Hohemberger, Ariel G. Castro, Francisco G. Vogt, Rodrigo B. Mansilha, Arthur F. Lorenzon, Fabio D. Rossi, and Marcelo C. Luizelli. Orchestrating In-Band Data Plane Telemetry With Machine Learning. *IEEE Communications Letters*, 23(12):2247–2251, 2019.
- [50] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, Magnus Fazekas, Ahern Lawson, Enrico Venz, Jonathan Corbet, David Lebrun, Tom Herbert, and David S. Miller. The express data path: fast programmable packet processing in the operating system kernel. In *Proc. ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 54–66. ACM, 2018.
- [51] Intel. P4-programmable Ethernet switch ASICs. <https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processor/s/tofino.html>.

- [52] Intel Corporation. In-band Network Telemetry Detects Network Performance Issues. White Paper, 2020. Available at <https://builders.intel.com/docs/networkbuilders/in-band-network-telemetry-detects-network-performance-issues.pdf>.
- [53] Mikel Irazabal and Navid Nikaein. TC-RAN: A Programmable Traffic Control Service Model for 5G/6G SD-RAN. *IEEE Journal on Selected Areas in Communications*, 42(2):406–419, 2024.
- [54] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *Proc. ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*, pages 1–8, 2018.
- [55] Abderaouf Khichane, Ilhem Fajjari, Nadjib Aitsaadi, and Mourad Gueroui. 5GC-Observer Demonstrator: A Non-intrusive Observability Prototype for Cloud Native 5G System. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–3, 2023.
- [56] Kubernetes Network Plumbing Working Group. Multus cni: A cni meta-plugin for multi-homed pods in kubernetes. <https://github.com/k8snetworkplumbingwg/multus-cni>, 2026. Accessed: 2026-03-04.
- [57] Kubernetes Network Plumbing Working Group. ovs-cni: Open vswitch cni plugin. <https://github.com/k8snetworkplumbingwg/ovs-cni>, 2026. Accessed: 2026-03-04.
- [58] Han Li. 5G Transport Network Requirements, Architecture and Key Technologies. In *ITU-T Workshop on IMT-2020 Network Requirements and Standardization*, 2017. China Mobile.
- [59] Yuliang Li, Rui Miao, Hongqiang Harry Liu, et al. HPCC: High Precision Congestion Control. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 44–58, 2019.
- [60] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37:145–151, 1991.
- [61] Kate Ching-Ju Lin and Wei-Lun Lai. MC-Sketch: Enabling Heterogeneous Network Monitoring Resolutions with Multi-Class Sketch. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pages 220–229, 2022.

- [62] Qiang Liu, Nakjung Choi, and Tao Han. Atlas: Automate Online Service Configuration in Network Slicing. In *Proc. ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 140–155, 2022.
- [63] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. NitroSketch: Robust and General Sketch-Based Monitoring in Software Switches. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 334–350, 2019.
- [64] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, page 101–114, 2016.
- [65] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, et al. Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks. In *Proc. USENIX Security*, pages 3829–3846, 2021.
- [66] Robert MacDavid, Carmelo Cascone, Pingping Lin, et al. A P4-based 5G User Plane Function. In *Proc. ACM Symposium on SDN Research (SOSR)*, pages 162–168, 2021.
- [67] Jonatas Marques, Kirill Levchenko, and Luciano Gaspary. IntSight: Diagnosing SLO Violations with in-Band Network Telemetry. In *Proc. ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 421–434, 2020.
- [68] P. Massart. The Tight Constant in the Dvoretzky–Kiefer–Wolfowitz Inequality. *The Annals of Probability*, 18(3):1269–1283, 1990.
- [69] Mohamed Mekki, Sagar Arora, and Adlen Ksentini. A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks. *IEEE Transactions on Network and Service Management*, pages 1–1, 2021.
- [70] Abdelkader Mekrache, Karim Boutiba, and Adlen Ksentini. Combining network data analytics function and machine learning for abnormal traffic detection in beyond 5G. In *Proc. IEEE Global Communications Conference (GLOBECOM)*, pages 1204–1209, 2023.
- [71] Foivos Michelinakis et al. AI anomaly detection for cloudified mobile core architectures. *IEEE Transactions on Network Service Management*, 20(2):1976–1992, 2023.

- [72] Microsoft Azure. What is Azure ExpressRoute?, 2024. Accessed: 2025-05-30.
- [73] MinIO, Inc. MinIO: High Performance, Kubernetes Native Object Storage. <https://min.io/>, 2026. Accessed: 2026-02-16.
- [74] Nagios Enterprises. Nagios: The Industry Standard in Open Source IT Monitoring, 2021.
- [75] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-Directed Hardware Design for Network Performance Monitoring. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 85–98, 2017.
- [76] Netdata Inc. Netdata: The Open-Source Observability Platform. <https://github.com/netdata/netdata>, 2026. Accessed: 2026-02-26.
- [77] O-RAN Alliance. O-RAN Architecture Description v07.00. Technical specification, O-RAN Alliance, 2022.
- [78] ONF. In-band Network Telemetry (INT). <https://docs.sd-fabric.org/sdfabric-1.0/advanced/int.html>, 2022. Accessed: 2025-06-07.
- [79] Open Networking Foundation. ONOS: Open Network Operating System. <https://github.com/opennetworkinglab/onos>, 2026. Accessed: 2026-03-04.
- [80] Open5GS. Open5GS GitHub Repository. <https://github.com/open5gs/open5gs>, 2024. Accessed: 2024-12-13.
- [81] OpenAirInterface. OpenAirInterface 5G RAN Project. <https://openairinterface.org/oai-5g-ran-project/>, 2024. Accessed: 2024-08-13.
- [82] OpenStack. Ceilometer: Data Collection Service, 2025.
- [83] P4 Language Consortium. Open P4 Studio: Intel P4 Studio Software Development Environment. <https://github.com/p4lang/open-p4studio>, 2024.
- [84] R. Perez, J. Garcia-Reinoso, A. Zabala, P. Serrano, and A. Banchs. A Monitoring Framework for Multi-Site 5G Platforms. In *Proc. IEEE European Conference on Networks and Communications (EuCNC)*, pages 52–56, 2020.
- [85] A. Plascinkas, X. Foukas, and M. K. Marina. Towards Efficient and Adaptable Monitoring of Softwarized Mobile Networks. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–6, 2020.

- [86] Prometheus Community. Avalanche. <https://github.com/prometheus-community/avalanche>, 2024. Accessed: 2026-03-05.
- [87] Emilio Riccardi. Selection of metro node architectures and optical technology options. METRO-HAUL project, Deliverable D3.1, 2018. <https://zenodo.org/records/2586698>.
- [88] Mohamed Rouili and Raouf Boutaba. Blink: A P4-Based 5G Centralized Unit. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9, 2024.
- [89] Niloy Saha. 5G-Monarch Repository on GitHub. <https://github.com/niloysh/5g-monarch>, 2024. Accessed: 2022-01-14.
- [90] Niloy Saha. 5g-traffic-generator: Configurable GTP-U traffic generator. <https://github.com/niloysh/5g-traffic-generator>, 2024. Accessed: 2024-11-01.
- [91] Niloy Saha. Kestrel: Sketch-based user-plane telemetry for 5g networks. <https://github.com/nrg-uw/Kestrel>, 2024. Accessed: 2024-11-01.
- [92] Niloy Saha. open5gs-k8s: Kubernetes-based deployment of Open5GS. <https://github.com/niloysh/open5gs-k8s>, 2024. Accessed: 2024-11-01.
- [93] Niloy Saha. SliceScope: SLA-aware monitoring control for 5g network slices. <https://github.com/nrg-uw/SliceScope>, 2024. Accessed: 2024-11-01.
- [94] Niloy Saha. testbed-automator: Automation scripts for 5g testbed provisioning. <https://github.com/niloysh/testbed-automator>, 2024. Accessed: 2024-11-01.
- [95] Niloy Saha. 5G Traffic Generator: Tools for Sending GTP-U Packets with Configurable TEID and QFI Values. <https://github.com/niloysh/5g-traffic-generator>, 2025. Accessed: 2026-06-27.
- [96] Niloy Saha, Mina Tahmasbi Arashloo, Nashid Shahriar, and Raouf Boutaba. Dynamic SLA-aware Network Slice Monitoring. In *Proc. ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS)*, 2026.
- [97] Niloy Saha, Noura Limam, Yang Xiao, and Raouf Boutaba. Rethinking Telemetry Design for Fine-Grained Anomaly Detection in 5G User Plane. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9, 2026.

- [98] Niloy Saha, Nashid Shahriar, Raouf Boutaba, and Aladdin Saleh. MonArch: Network Slice Monitoring Architecture for Cloud Native 5G Deployments. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–7, 2023.
- [99] Niloy Saha, Nashid Shahriar, Muhammad Sulaiman, Noura Limam, Raouf Boutaba, and Aladdin Saleh. MonArch: Monitoring Architecture for 5G and Beyond Network Slices. *IEEE Transactions on Network and Service Management*, 22(1):777–790, 2025.
- [100] Vincenzo Sciancalepore, Konstantinos Samdanis, Xavier Costa-Perez, Dario Bega, Marco Gramaglia, and Albert Banchs. Mobile traffic forecasting for maximizing 5g network slicing resource utilization. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pages 1–9, 2017.
- [101] Vincenzo Sciancalepore, Faqir Zarrar Yousaf, and Xavier Costa-Perez. Z-TORCH: An Automated NFV Orchestration and Monitoring Solution. *IEEE Transactions on Network and Service Management*, 15(4):1292–1306, 2018.
- [102] Siyuan Sheng, Qun Huang, and Patrick P. C. Lee. DeltaINT: Toward General In-band Network Telemetry with Extremely Low Bandwidth Overhead. In *Proc. IEEE International Conference on Network Protocols (ICNP)*, pages 1–11, 2021.
- [103] Kyle A. Simpson and Dimitrios P. Pazaros. Revisiting the classics: Online RL in the programmable dataplane. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–10, 2022.
- [104] SimPy. Discrete event simulation for Python. <https://simpy.readthedocs.io/en/latest/>.
- [105] Suneet Kumar Singh, Christian Esteve Rothenberg, Jonatan Langlet, Andreas Kassler, Péter Vörös, Sándor Laki, and Gergely Pongrácz. Hybrid P4 Programmable Pipelines for 5G gNodeB and User Plane Functions. *IEEE Transactions on Mobile Computing*, 22(12):6921–6937, 2023.
- [106] SoftFIRE Project. 5g mobile core network slicing on an orchestrated and virtualised infrastructure: Deployment of virtual mobile core network slices on the softfire infrastructure. Whitepaper/technical report, FIRE+ Initiative, December 2017.
- [107] srsRAN Project. srsRAN: Open Source 4G and 5G Radio Access Network Software. <https://www.srsran.com>, 2024. Accessed: 2024-11-01.

- [108] Muhammad Sulaiman, Arash Moayyedi, Mahdiah Ahmadi, Mohammad A. Salahuddin, Raouf Boutaba, and Aladdin Saleh. Coordinated Slicing and Admission Control Using Multi-Agent Deep Reinforcement Learning. *IEEE Transactions on Network and Service Management*, 20(2):1110–1124, 2023.
- [109] Muhammad Sulaiman, Bo Sun, Mohammad Ali Salahuddin, Raouf Boutaba, and Aladdin Saleh. Data-driven Online Slice Admission Control and Resource Allocation for 5G and Beyond Networks, 2025. arXiv:2501.10285.
- [110] Chuanhao Sun, Ujjwal Pawar, Molham Khoja, Xenofon Foukas, Mahesh K. Marina, and Bozidar Radunovic. SpotLight: Accurate, Explainable and Efficient Anomaly Detection for Open RAN. In *Proc. ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 923–937, 2024.
- [111] Azadeh Tabiban, Hyame Assem Alameddine, Mohammad A. Salahuddin, and Raouf Boutaba. Signaling storm in O-RAN: Challenges and research opportunities. *IEEE Communications Magazine*, 62(6):58–64, 2024.
- [112] Yawen Tan, Jiajia Liu, Yuanhao Li, and Jiadai Wang. Deep learning based proactive anomaly detection for 5G core control plane network function interactions. *IEEE Transactions on Cognitive Communications and Networking*, pages 1–1, 2025.
- [113] Shaofei Tang, Sicheng Zhao, Xiaoqin Pan, and Zuqing Zhu. How to Use In-Band Network Telemetry Wisely: Network-Wise Orchestration of Sel-INT. *IEEE/ACM Transactions on Networking*, pages 1–15, 2022.
- [114] The Kubernetes Authors. Kubernetes: Production-Grade Container Orchestration. <https://kubernetes.io/>, 2025. Accessed: 2026-02-16.
- [115] The Linux Foundation. Prometheus: An Open-Source Monitoring System and Time Series Database. <https://prometheus.io/>, 2022. Accessed: 2022-01-14.
- [116] The Linux Foundation. Open Network Automation Platform (ONAP), 2025. Accessed: 2025-02-12.
- [117] The OpenTelemetry Authors. OpenTelemetry: High-Quality, Ubiquitous, and Portable Telemetry To Enable Effective Observability. <https://opentelemetry.io/>, 2026. Accessed: 2026-02-16.
- [118] The P4 Language Consortium. P4 16 Language Specification. <https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html>.

- [119] The P4.org Application Working Group. In-band Network Telemetry (INT) Dataplane Specification, 2024. https://p4.org/p4-spec/docs/INT_v2_1.pdf.
- [120] UfiSpace. S9180-32X 100GE Cloud and Data Center TOR Switch Datasheet. <https://hardwarenation.com/wp-content/uploads/2021/11/S9180-32X.pdf>, 2019. Accessed: 2026-03-04.
- [121] 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 *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–7, 2020.
- [122] P. Veitch, J. Browne, and J. Krogell. An Integrated Instrumentation and Insights Framework for Holistic 5G Slice Assurance. In *Proc. IEEE International Conference on Network Softwarization (NetSoft)*, pages 247–251, 2020.
- [123] Jim Warner. Switch Buffers. <https://people.ucsc.edu/~warner/buffer.html>.
- [124] Zhixin Wen and Guanhua Yan. HiP4-UPF: Towards High-Performance Comprehensive 5G User Plane Function on P4 Programmable Switches. In *Proc. USENIX Annual Technical Conference (ATC)*, pages 303–320, 2024.
- [125] 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. IEEE IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–7, 2019.
- [126] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, et al. Brain-on-switch: Towards Advanced Intelligent Network Data Plane via NN-driven Traffic Analysis at Line-speed. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1–22, 2024.
- [127] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 561–575, 2018.
- [128] Minlan Yu, Lavanya Jose, and Rui Miao. Software Defined Traffic Measurement with OpenSketch. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 29–42, 2013.

- [129] Y. Zhang, Z. Guan, H. Qian, L. Xu, H. Liu, Q. Wen, L. Sun, J. Jiang, L. Fan, and M. Ke. Cloudrca: A root cause analysis framework for cloud computing platforms. In *Proc. ACM International Conference on Information and Knowledge Management (CIKM)*, pages 4373–4382, 2021.
- [130] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, et al. CocoSketch: High-Performance Sketch-based Measurement over Arbitrary Partial Key Query. In *Proc. ACM SIGCOMM*, pages 207–222, 2021.
- [131] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, page 21, 2021.
- [132] Hao Zheng, Chen Tian, Tong Yang, Huiping Lin, Chang Liu, Zhaochen Zhang, Wanchun Dou, and Guihai Chen. FlyMon: Enabling on-the-Fly Task Reconfiguration for Network Measurement. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 486–502, 2022.
- [133] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, Pengcheng Zhang, Dennis Cai, Ming Zhang, and Mingwei Xu. Flow Event Telemetry on Programmable Data Plane. In *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 76–89, 2020.

APPENDICES

Appendix A

Experimental Testbed

This appendix describes the experimental testbed used throughout the dissertation. All experiments in Chapters 3–5 are conducted on a common infrastructure to ensure consistency, though each chapter activates a different subset of the platform.

Beyond serving as a shared evaluation environment, the testbed itself is an important enabling artifact of this dissertation. Its design was informed by both research requirements and discussions with industry partners, which helped identify practical constraints around cloud-native 5G deployment, slice-level monitoring, telemetry overhead, and deployability in operational environments. At the start of this work, open-source 5G platforms provided useful building blocks, but did not directly support the slice-aware telemetry, cross-layer state correlation, programmable transport integration, and reproducible traffic control required by the systems developed in Chapters 3–5. We therefore built an experimental substrate that connects cloud-native core functions, RAN simulation/emulation, programmable transport, monitoring infrastructure, and trace-driven traffic generation. This substrate makes it possible to evaluate slice-level KPI reconstruction, SLA-aware telemetry control, and fine-grained user-plane anomaly detection under a common set of deployment assumptions while activating only the components required by each chapter.

Figure A.1 illustrates the e2e platform used throughout this dissertation. The programmable transport network is implemented on a UfiSpace S9180-32X Tofino switch [120], where the same hardware pipeline is reused across chapters with different telemetry configurations. In Chapter 3, the switch operates as a transparent L3 forwarder. Chapter 4 enables In-Band Network Telemetry (INT)-based telemetry, allowing the SLICESCOPE controller (*cf.* §4.5.1) running on the compute server to dynamically adjust telemetry parameters. In Chapter 5, the switch additionally hosts a simplified User Plane Function (UPF) with

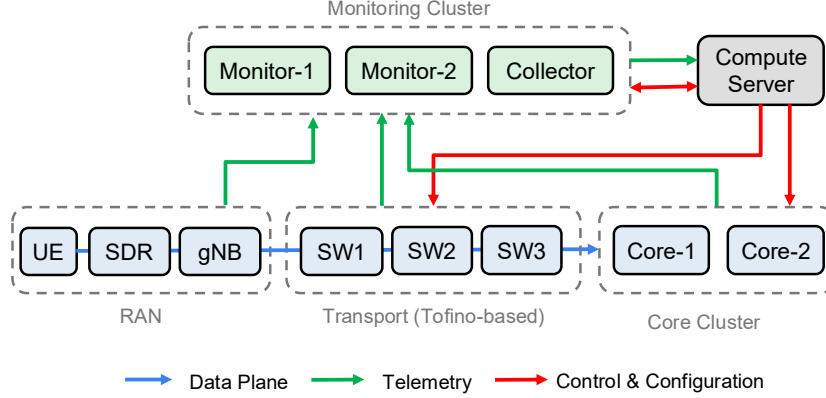


Figure A.1: **end-to-end (e2e) 5G experimental testbed**. The platform spans a Radio Access Network (RAN), a Tofino-based programmable transport network, and a cloud-native 5G core. Telemetry from the RAN, transport, and core is aggregated in the monitoring cluster, while the compute server runs the telemetry optimizer (SLICESCOPE, Chapter 4) and anomaly detection pipeline (KESTREL, Chapter 5). The multi-hop transport topology is emulated on a UfiSpace S9180-32X Tofino switch [120] using external loopbacks.

sketch-based telemetry used by the anomaly detection pipeline (*cf.* §5.5).

A.1 Hardware and Deployment

Table A.1 summarizes the hardware components. The testbed comprises six servers and a programmable switch, interconnected over 1 GbE management and 10 GbE data plane links. The Next Generation Node B (gNB) node is a Dell Precision 5820 workstation equipped with an 18-core Intel i9-10980XE processor (which can boost up to 4.6 GHz) and 32 GB of RAM, and hosts a USRP X310 software-defined radio connected over 10 GbE. The higher clock speed of the gNB node is needed to meet the real-time baseband processing demands of the gNB software stack; this node also runs a low-latency kernel (5.15.0-110-lowlatency) to minimize scheduling jitter. The remaining five servers are 1U servers, fitted with 8-core Intel Xeon E3-1230 v3 (3.7 GHz) and 16 GB of RAM, running Ubuntu 22.04. The testbed also includes a Compute server, with a 32-core Intel i9-14900K and 64 GB of RAM, fitted with an NVIDIA RTX 4090 GPU for ML workloads. The programmable transport is implemented on a UfiSpace S9180-32X switch [120] equipped with an Intel Tofino ASIC [51], connected to the servers over 10 GbE.

Role	CPU	RAM	Special Hardware	Count
gNB node	i9-10980XE (18c, boost 4.6 GHz)	32 GB	USRP X310 (10 GbE)	1
Core cluster	Xeon E3-1230 v3 (8c, 3.7 GHz)	16 GB	—	2
Monitoring cluster	Xeon E3-1230 v3 (8c, 3.7 GHz)	16 GB	—	3
Compute server	i9-14900K (32c, 3.2 GHz)	64 GB	RTX 4090 GPU	1
Transport	—	—	UfiSpace S9180-32X	1

Table A.1: **Hardware components of the experimental testbed.** The monitoring cluster comprises two monitor nodes and one collector node. The transport switch implements a multi-hop topology via external loopbacks. All server nodes run Ubuntu 22.04. Chapters 3–5 are evaluated on this shared infrastructure.

A.2 5G Network and Slice Implementation

Radio Access Network. The testbed supports two interchangeable software RAN stacks, srsRAN 5G (release 24.10) [107] and OpenAirInterface [81], both running on the gNB node (§A.1) and connecting to the same core. For experiments requiring real radio propagation, either stack drives the USRP X310 Software-Defined Radio (SDR) to serve commercial User Equipments (UEs) over the air. For scalability experiments involving many UEs and slices, the over-the-air interface is replaced by UERANSIM [44], which emulates UEs and gNBs in software while preserving 3rd Generation Partnership Project (3GPP) protocol semantics. The specific stack used in each chapter is noted in the corresponding evaluation section.

Core Network. The 5G core is implemented using Open5GS [80], deployed as containerized network functions on a comprising nodes **core-1** and **core-2** (Figure A.1). The core functions (*e.g.*, Access and Mobility Management Function (AMF), Session Management Function (SMF), and others) are deployed as microservices orchestrated via Kubernetes deployment scripts, which are publicly available at [92]. Cluster provisioning and network configuration are automated via scripts available at [94].

In our deployments, network slices are realized by deploying dedicated SMF and UPF instances per slice, while control plane functions such as the AMF are shared across slices. This configuration enables independent slice-level state while preserving realistic resource sharing across the infrastructure. Multi-interface support for core network function pods (*e.g.*, the UPF) is provided by Multus Container Network Interface (CNI) [56], which allows attaching dedicated secondary interfaces to pods. This allows critical interfaces (*e.g.*, N2 and N3 [3]) to operate over independent, isolated connections, ensuring that control plane

signaling and user-plane traffic do not interfere. Connectivity between distributed network functions across compute nodes is established via a Virtual eXtensible Local Area Network (VXLAN)-based overlay network constructed using OvS-CNI [57]. The overlay is managed by an ONOS SDN controller [79], which handles forwarding configuration and collects traffic statistics from the transport network, making them available to the monitoring cluster.

Transport. The transport network interconnecting the RAN and core is based on the VXLAN-based overlay described above. On the data path between the RAN and core is a UfiSpace S9180-32X switch [120] equipped with an Intel Tofino ASIC [51]. We implemented a multi-hop topology on this switch via external loopbacks. The role of this Tofino switch evolves across chapters. In Chapter 3, it acts as a transparent L3 forwarder. In Chapter 4, we additionally implement INT, and export per-packet measurements to the Collector, and subsequent processing by the Compute server. In Chapter 5, we implement a simplified programmable UPF, which includes queuing, scheduling, and Two-Rate Three-Color Marker (trTCM)-based metering alongside sketch-based telemetry. Slice differentiation in this stage is realized through queue-based scheduling policies implemented in the programmable data plane.

The data plane logic uses the Programming Protocol-Independent Packet Processors (P4) language [118], and is implemented in the Tofino switch’s ingress and egress pipelines. A more detailed description of the data plane design, telemetry mechanisms, and parameterization is provided in Chapters 4 and 5.

A.3 Traffic Generation

The testbed supports four traffic generation modes depending on the experiment.

- **Real UE trace replay.** For experiments requiring e2e realism, a commercial UE connected to the SDR-based RAN replays traces derived from a real-world Fifth Generation (5G) dataset [24], spanning cloud gaming, VR streaming, live video, and best-effort traffic. This preserves realistic transport dynamics such as burstiness, congestion interaction, and application pacing, while enabling slice-level Key Performance Indicator (KPI) computation.
- **Large-scale emulation.** For scalability experiments involving large numbers of slices, UERANSIM [44] is used to emulate multiple UEs and gNBs, enabling repeatable slice instantiation and controlled scaling of traffic.

- **Trace-driven traffic generation** Experiments requiring explicit control over user-plane identifiers (Tunnel Endpoint Identifier (TEID), QoS Flow Identifier (QFI)) use a custom C-based traffic generator that directly constructs GPRS Tunneling Protocol – User Plane (GTP-U) packets with specified $(TEID, QFI)$ values [90], but based on the real-world 5G dataset [24]. This enables precise mapping between logical flows and the user plane identifiers, which is not achievable with commodity UEs or standard emulators.
- **Synthetic workloads.** For controlled evaluation of SLA-aware monitoring policies, workloads are generated according to representative 5G service classes. Full details of the workload parameterization and slice compositions are described in Chapter 4.

Experimental scale across chapters. Because each contribution stresses a different part of the platform, the testbed configuration varies by chapter. MONARCH’s scaling experiments use UERANSIM to emulate a single gNB with up to 50 UEs/slices, while its adaptive-monitoring evaluation replays real 5G traces through one SDR-backed gNB and a commercial UE. SLICESCOPE is validated on the Intel Tofino programmable-switch path at small scale for hardware resource measurement, and in simulation at 300 slices for Service Level Agreement (SLA)-aware allocation. KESTREL drives the user plane with trace-replayed GTP-U traffic across 900 Quality of Service (QoS) flows ($100 \text{ TEIDs} \times 9 \text{ QFIs}$) without a physical RAN, to control $(TEID, QFI)$ mappings precisely. Physical radio components are used where radio realism matters; emulation, simulation, and trace-driven generation are used where the objective is to scale the slice count or fix user-plane identifiers.

Code availability. The software developed for this dissertation is released as six public repositories spanning the full experimental stack. Testbed provisioning and core-network deployment are automated in [92, 94]; identifier-controlled traffic generation is provided by [95]; and the three system prototypes, MONARCH, SLICESCOPE, and KESTREL, are available at [89], [93], and [91] respectively.

Appendix B

Proof of the Detectability Condition in KESTREL

We prove a sufficient condition for anomaly detectability by comparing the anomaly-induced increase in diagnostic bins to the worst-case collision noise introduced by Count-Min Sketch (CMS) estimation.

Consider a flow k during a single measurement window. Let the feature distributions maintained in KESTREL's sketch-based data structures be partitioned into bins indexed by j . As defined in §5.4.1, let T denote the set of *diagnostic bins*.

True masses. Let $x_{k,j}$ be the true number of packets of flow k that fall into bin j during the window, and define the true diagnostic mass of the flow as

$$x_{k,T} \triangleq \sum_{j \in T} x_{k,j}.$$

Let N_j denote the total number of packets (across all flows) that fall into bin j , and define the total background mass in the diagnostic region as

$$N_T \triangleq \sum_{j \in T} N_j.$$

CMS estimates and collision noise. Let $\hat{x}_{k,j}$ denote the CMS estimate of $x_{k,j}$. By the CMS guarantee, for each bin j ,

$$x_{k,j} \leq \hat{x}_{k,j} \leq x_{k,j} + \varepsilon N_j$$

with probability at least $1 - \delta$, where $\varepsilon \approx e/w$ for sketch width w . Summing these bounds over bins in T yields

$$x_{k,T} \leq \hat{x}_{k,T} \leq x_{k,T} + \varepsilon N_T, \quad \text{where } \hat{x}_{k,T} \triangleq \sum_{j \in T} \hat{x}_{k,j}.$$

Thus, under baseline traffic, the diagnostic mass estimate for flow k may be overestimated by at most εN_T due to sketch collisions.

Detectability condition. Now consider an anomaly for flow k that injects a total of Δ packets during the window. Let β denote the spillover fraction, meaning that only a $(1 - \beta)$ fraction of the anomaly packets land in diagnostic bins T . The anomaly, therefore, increases the true diagnostic mass to

$$x'_{k,T} = x_{k,T} + (1 - \beta)\Delta.$$

A sufficient condition for the anomaly to be visible in the diagnostic region is that the anomaly-induced increase in diagnostic mass exceeds the worst-case baseline collision noise in that region:

$$x_{k,T} + (1 - \beta)\Delta > x_{k,T} + \varepsilon N_T.$$

Canceling $x_{k,T}$ from both sides yields the detectability condition

$$(1 - \beta)\Delta > \varepsilon N_T$$