Sketch-Based Heavy-Hitter Detection for High-Rate Network Telemetry with Adaptive Precision Allocation

Andi Saputra[1] **and Fajar Santoso**[2]
[1]Universitas Negeri Makassar, Jalan Daeng Tata Raya, Makassar 90222, Indonesia
[2] Universitas Jenderal Achmad Yani Yogyakarta, Jalan Ringroad Barat, Sleman 55292, Indonesia

ABSTRACT High-rate network telemetry increasingly relies on streaming analytics that must operate under tight memory, latency, and power constraints while observing traffic whose statistical structure can shift rapidly. Heavy-hitter detection is a canonical task in this setting because it supports congestion localization, anomaly detection, capacity planning, and security monitoring, yet it is difficult to perform exactly when per-packet processing budgets are measured in a handful of nanoseconds and per-device memory is limited. Sketch-based algorithms provide an attractive alternative by offering approximate frequency estimates with probabilistic error guarantees and mergeability across devices, but practical deployments face an additional challenge: a fixed-precision sketch over-allocates resources to low-impact traffic while under-allocating to the few flows that dominate volume. This paper studies sketch-based heavy-hitter detection for high-rate telemetry with adaptive precision allocation, where memory, counter resolution, and update effort are redistributed online toward likely heavy contributors. The approach combines a coarse, fast path that maintains candidate evidence with a refinement path that selectively increases measurement precision for uncertain or high-value flows. A principled optimization model is developed to trade accuracy against resource budgets under heavy-tailed traffic, including multi-objective formulations for latency, energy, and memory. Error analyses link adaptive decisions to false positive and false negative rates, and system design considerations address pipeline constraints, hash design, merge protocols, and reproducibility. The resulting framework aims to maintain stable heavy-hitter recall under rapid workload shifts without requiring static overprovisioning.

KEYWORDS

1. Introduction

Network operators increasingly treat measurement as a first-class control surface: decisions about routing, congestion response, mitigation, and capacity expansion depend on telemetry that must be timely, accurate enough to be actionable, and cheap enough to run continuously [1]. The core difficulty is that packet streams are both high volume and high entropy. Even when keys are restricted to canonical notions such as five-tuples, host pairs, prefixes, or application identifiers, the number of distinct keys within a short window can be extremely large, and the distribution of frequencies is often heavy-tailed. Under such conditions, exact counting for all keys is impractical on commodity hardware, and it is often infeasible even in programmable data planes where state is scarce and per-packet operations are severely constrained.

Heavy-hitter detection is one of the most broadly useful streaming primitives in telemetry. Informally, a heavy hitter is a key whose traffic volume within an interval exceeds a specified fraction of the total. More generally, network operators care about top-k keys, hierarchical heavy hitters over prefixes, and approximate quantiles over flow sizes [2]. A heavy-hitter detector must preserve recall for truly large contributors while controlling false positives that would waste downstream processing or trigger unnecessary mitigations. The detector must also adapt to time-varying behavior, including flash crowds, DDoS bursts, routing changes, and diurnal trends, and it must do so under strict performance engineering realities such as cache miss penalties, memory bandwidth ceilings, and the update pipelines imposed by switches, NIC offloads, and kernel-bypass stacks.

Sketches provide a standard approach to approximate fre-

quency estimation. Data structures such as Count-Min, Count-Sketch, and related variants support fast updates via hash-indexed arrays and provide error bounds that scale with sketch width and depth. They are attractive because they can be mergeable across distributed measurement points, can be implemented with simple arithmetic, and naturally support streaming. However, classic sketches allocate uniform precision across the key space. Uniform allocation is mismatched to network telemetry where most keys are tiny and a small minority dominate volume [3]. Uniform allocation tends to waste resolution on the long tail while leaving heavy keys competing with an enormous number of small keys for the same counters, increasing collision-induced bias or variance. In practice, meeting strict recall targets under adversarial or worst-case traffic often forces overprovisioning memory, which is costly in switches and expensive in CPU cache footprint on servers.

This paper develops sketch-based heavy-hitter detection with adaptive precision allocation. The central idea is to preserve the constant-time update structure of sketches while redistributing measurement fidelity online. Precision can refer to memory devoted to particular regions of the sketch, counter bit-width, number of hash functions effectively used, sampling rate, or the degree of refinement applied to selected candidates. The approach emphasizes a two-path architecture: a fast, always-on coarse sketch that is cheap per packet and produces a small candidate set, and a refinement mechanism that increases precision for flows deemed likely to be heavy or whose current uncertainty materially affects decisions. Such refinement can be implemented via selective counter promotion, dynamic resizing of sketch partitions, secondary sketches over candidate keys, or adaptive sampling policies [4].

A major concern is that adaptivity must not destroy the simplicity that makes sketches deployable. Decisions must be computed with low overhead, and state transitions must be safe under concurrency and merging. The analysis must connect adaptive decisions to rigorous error control, including the probability of false negatives, which is the more operationally dangerous outcome in many telemetry tasks. Furthermore, adaptivity introduces new failure modes: a poorly tuned policy can overreact to noise, amplify transient bursts, or starve new heavy flows that have not yet accumulated evidence. The paper therefore develops an explicit mathematical model for adaptive allocation, including resource constraints and multi-objective trade-offs among accuracy, latency, and energy. The model incorporates heavy-tailed priors, backprop-friendly surrogate objectives for automated tuning, and conservative guards that preserve worst-case bounds when traffic behaves adversarially [5].

From a systems perspective, the proposed framework is designed to map cleanly to modern telemetry stacks. In programmable switches, it aligns with pipeline constraints by using a small constant number of memory accesses per packet on the fast path and deferring expensive work to control-plane intervals. In software, it aligns with vectorized hashing, cache-aware layouts, and per-core sharding. In distributed settings, it supports mergeable summaries and communication-efficient reporting, with coding-theoretic intuition for compressing candidate information relative to the entropy of the heavy-hitter identity distribution.

The remainder of the paper formalizes the telemetry model and the heavy-hitter problem, reviews sketching primitives and derives error behavior under collisions, introduces adaptive precision allocation as an optimization and control problem, describes implementation strategies in both hardware and software pipelines, and outlines an evaluation methodology emphasizing reproducibility and performance engineering. The focus is on the technical mechanics and trade-offs rather than on any single deployment environment, with the aim of providing a framework that can be specialized to diverse telemetry requirements.

2. Problem Formulation and Telemetry Model

Consider a packet stream observed over a measurement interval, modeled as a sequence of updates k_t, Δ_t for $t = 1, \ldots, T$ where $k_t \in \mathcal{K}$ is a key and $\Delta_t > 0$ is a weight. In simple byte-count telemetry, Δ_t is the packet size in bytes; in packet-count telemetry, $\Delta_t = 1$; in richer settings Δ_t can encode cost, priority, or sampled weight [6]. Let $f_k = \sum_{t:k_t=k} \Delta_t$ denote the true frequency of key k, and let $F = \sum_{k\in\mathcal{K}} f_k$ denote total traffic volume in the interval. A ϕ-heavy hitter is typically defined as a key satisfying $f_k \geq \phi F$ for a threshold $\phi \in 0, 1$. An approximate detector is often parameterized by $\phi, \varepsilon, \delta$, where it must report all keys with $f_k \geq \phi F$ with probability at least $1 - \delta$ while not reporting any key with $f_k \leq \phi - \varepsilon F$ with probability at least $1 - \delta$. Intermediate keys are allowed to be arbitrary due to the gap between ϕ and $\phi - \varepsilon$.

In network telemetry, the measurement interval can be time-based (for example, one second), packet-count-based (for example, every N packets), or event-based (for example, a congestion trigger). The interval definition interacts with burstiness: smaller windows reduce memory requirements for drift but make estimates noisier relative to variance in arrivals, while larger windows smooth fluctuations but increase the number of distinct keys. Many deployments also require sliding or tumbling windows, which add the problem of expiring old contributions. This paper focuses primarily on tumbling windows for clarity and then discusses how adaptive precision interacts with windowing [7].

A detector produces an estimate $\widehat{f}_k$ for any queried key and a reported set $\widehat{H}$ intended to approximate the true heavy-hitter set $H = \{k : f_k \geq \phi F\}$. The engineering requirement is that computing $\widehat{H}$ must not require enumerating all keys, because $|\mathcal{K}|$ may be huge. Instead, practical pipelines maintain a candidate set of manageable size and validate candidates, potentially via additional measurement or via control-plane queries.

The adversarial aspect of networking must also be acknowledged. Traffic can be influenced by attackers who might try to evade detection or trigger false positives. Worst-case guarantees are therefore relevant, but purely worst-case analysis can be overly pessimistic and can lead to excessive memory. A useful approach is to adopt a hybrid model: maintain baseline worst-case error bounds while optimizing expected performance under a stochastic model of traffic. Heavy-tailed models are empirically common, so it is natural to consider priors over fre-

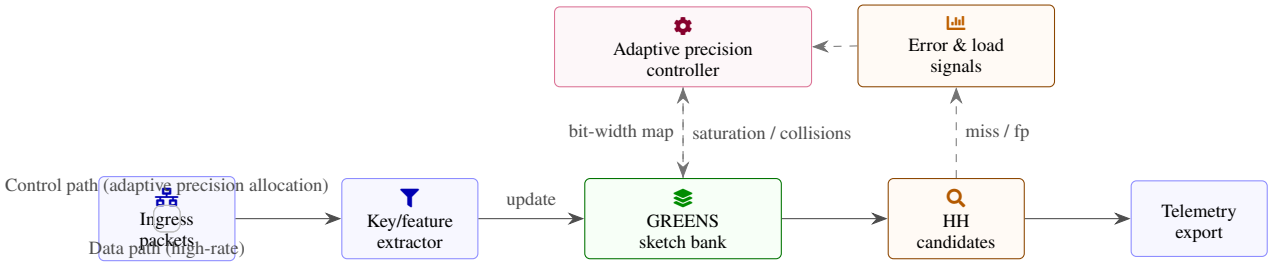

Figure 1 GREENS overview: packets are summarized in a sketch bank on the fast path, while a lightweight controller monitors sketch health (e.g., counter saturation and collision pressure) and reallocates precision to keep heavy-hitter detection accurate under changing load.

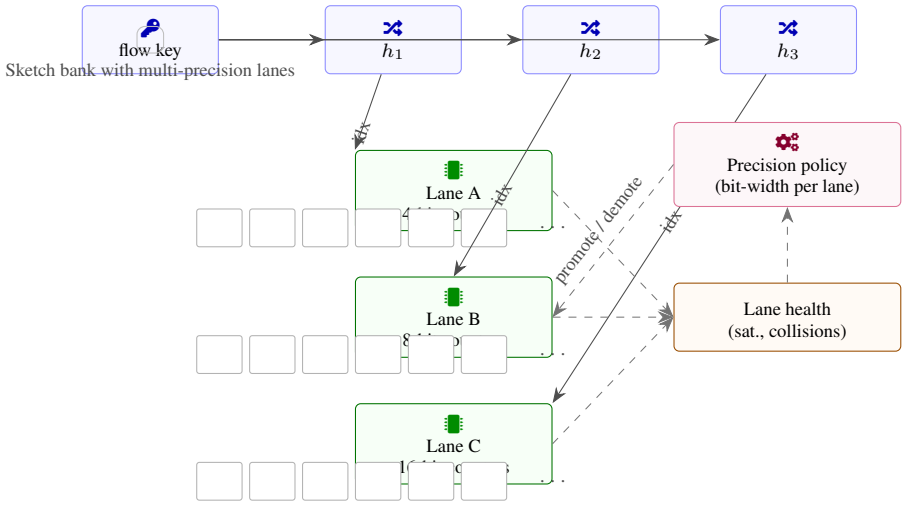

Figure 2 Sketch-bank internals: the same flow key is hashed into multiple lanes, each with a different counter precision. Lane health signals guide policy updates that reassign precision (e.g., promoting a stressed lane to higher bit-width) without disrupting the fast update path.

quencies such as Pareto-like distributions, lognormal mixtures, or Zipf-like rank-frequency laws, recognizing that the precise model is interval-dependent.

Let keys be ranked by frequency such that $f_1 \geq f_2 \geq \dots$. A heavy-tailed assumption can be expressed as an approximate power law $f_r \approx Cr^{-\alpha}$ for some $\alpha > 1$ over a range of ranks. In such a regime, a small set of top keys accounts for a large fraction of F [8]. For telemetry, the practical quantity is not only the tail index but also the stability of identities: a key may be heavy transiently and then disappear, which suggests that adaptivity must include mechanisms for exploration to avoid missing new heavy keys.

The computational model is a high-rate update stream where each update must be processed in amortized constant time with very small constants. In a switch data plane, the budget is typically a fixed number of pipeline stages with limited memory operations per stage. In a software collector, the budget is often constrained by per-core packet rate, cache residency, and vectorization. Adaptive precision allocation must therefore be expressible as either low-overhead decisions per packet or periodic adjustments at control-plane timescales.

A useful abstraction is to treat the measurement system as maintaining a state vector θ that parameterizes a family of estimators [9]. For sketches, θ includes hash seeds, array dimensions, counter widths, and any auxiliary state such as candidate lists. The update rule maps θ, k_t, Δ_t to a new state θ'. Classical sketches use a fixed θ throughout the interval. Adaptive precision allocation allows θ to evolve based on observed evidence, but it must do so while remaining stable and mergeable across distributed instances. Mergeability can be expressed as an associative operation $\oplus$ such that for disjoint substreams S_1, S_2, the summary of the union equals the merge of summaries: $\mathcal{S}S_1 \cup S_2 = \mathcal{S}S_1 \oplus \mathcal{S}S_2$. Maintaining exact mergeability under adaptivity is nontrivial; a central design goal is therefore to isolate adaptive components so that mergeability is either preserved or approximated with bounded error.

The notion of precision in this paper is intentionally broad. Memory precision includes the number of counters allocated to represent the stream, their arrangement, and how they are shared among keys via hashing [10]. Numerical precision includes counter bit-width and saturation behavior. Statistical precision includes sampling rate and estimator variance. Algorithmic precision includes the number of hash functions consulted per update, the number of candidate validations performed, and the fidelity of refinement for selected keys. Adaptive precision allocation is the process of choosing these degrees of freedom online to optimize accuracy under constraints.

The constraints are multidimensional. Memory is a hard constraint in many devices; latency per update is a hard constraint in switches and a soft but critical constraint in software; energy

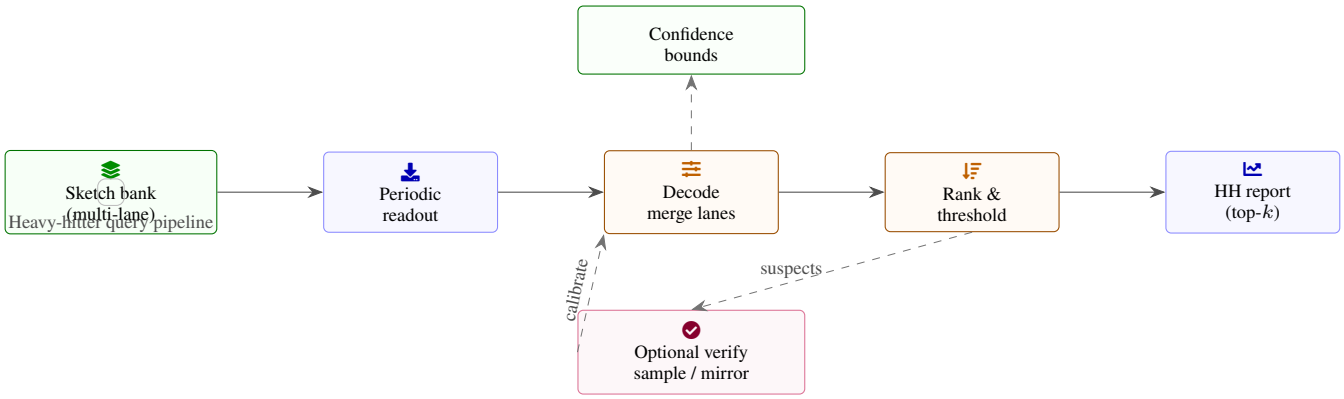

Figure 3 Detection pipeline: GREENS periodically reads sketch state, decodes multi-precision lanes into flow estimates, and ranks candidates against a heavy-hitter threshold. Optional verification (e.g., lightweight sampling or mirroring) can be used to calibrate decoding and confidence bounds.

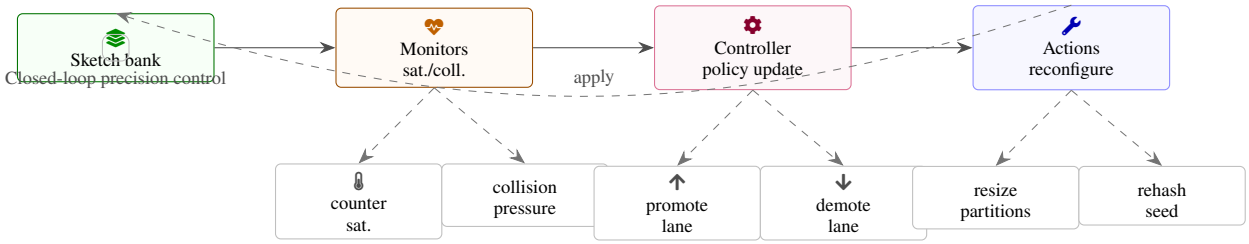

Figure 4 Closed-loop control: GREENS continuously monitors sketch stress (e.g., saturation and collision pressure), updates a precision policy, and applies targeted actions such as lane promotion/demotion, partition resizing, or rehashing to stabilize accuracy at high rates.

per update is increasingly relevant for high-throughput telemetry on programmable devices and SmartNICs. Let M denote the total memory budget in counters, L denote a maximum per-update latency budget measured in operations or cycles, and E denote an energy budget per update or per interval [11]. The adaptive allocation problem is to choose a policy π that maps observed sketch state to actions, where actions may include promoting counters, resizing partitions, or changing sampling rates, such that the resulting estimator minimizes an error objective while respecting M, L, E.

The error objective can be defined in several ways depending on operational goals. For heavy-hitter detection, the most relevant are false negative probability, false positive rate, and estimation error for the top keys. A convenient continuous surrogate is to define a loss over estimated frequencies, for example a weighted squared error over the top ranks, plus penalties for misclassification around the threshold. Let w_k weight the importance of key k (which can be uniform or can encode business value). A generic objective is

$$\mathcal{L}\theta = \mathbb{E}\Big[\sum_{k\in\mathcal{K}} w_k \, \ell\big(\widehat{f}_k\theta, f_k\big)\Big] \;\lambda_{\mathrm{FP}}\,\mathbb{E}\mathrm{FP} \;\lambda_{\mathrm{FN}}\,\mathbb{E}\mathrm{FN} \quad , \qquad (2.1)$$

where expectations are over hashing randomness and any stochastic model of traffic [12]. The adaptive problem then becomes a constrained stochastic control problem over θ.

Because the system must remain implementable, the policy class must be restricted. The framework developed later centers on a two-level decomposition where a base sketch provides uniform baseline guarantees, while an adaptive overlay reallocates a limited fraction of resources. This decomposition makes it possible to retain worst-case bounds for the baseline and concentrate adaptivity on improving typical-case performance.

3. Sketching Primitives, Estimators, and Error Behavior

Sketches approximate the frequency vector $f \in \mathbb{R}_{\geq 0}^{|\mathcal{K}|}$ by maintaining a compact array structure indexed by hashes. A canonical example is the Count-Min sketch with depth d and width w, which uses hash functions $h_j : \mathcal{K} \to \{1, \ldots, w\}$ for $j = 1, \ldots, d$ and maintains counters Cj, i. Upon an update k, Δ, it increments $Cj, h_j k \leftarrow Cj, h_j k \;\Delta$ for all j. The estimate is $\widehat{f}_k = \min_j Cj, h_j k$. Under standard assumptions of pairwise independent hashing, the estimate is biased upward due to collisions, and with appropriate parameters one obtains an additive error bound of roughly εF with probability at least $1 - \delta$ when $w = \lceil e\varepsilon \rceil$ and $d = \lceil \ln 1\delta \rceil$ [13].

The operational meaning of the Count-Min bound is that every estimate is within a fixed additive budget proportional to total traffic, independent of the key. For heavy-hitter detection with threshold ϕF, setting ε smaller than ϕ yields a gap that can be used to separate heavy from non-heavy keys. However, in high-cardinality streams, the collision term can be dominated by the aggregate mass of many small keys, and the additive error can be too loose when ϕ is small. Moreover, the bound treats all keys symmetrically, ignoring the fact that heavy hitters are a tiny subset. This mismatch motivates adaptivity: the system should

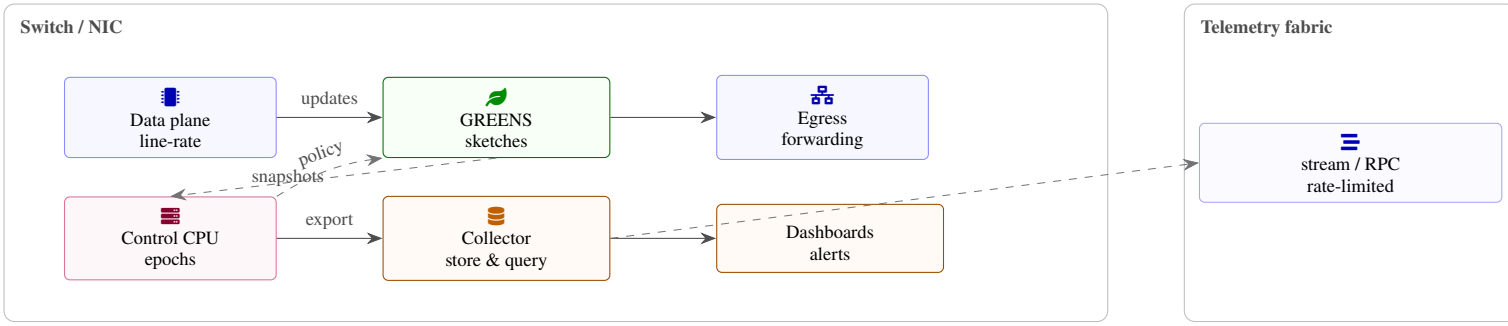

Figure 5 Deployment view: GREENS runs at line rate in the data plane with epoch-based control on a local CPU. Sketch snapshots are exported to a collector for storage and querying, while the control plane pushes updated precision policies back to the sketch bank through a lightweight channel.

Symbol	Description	Default value
N	Number of distinct flows in the interval	10^6
ϕ	Heavy-hitter threshold (fraction of total volume)	0.01
w	Sketch width (counters per row)	2^{18}
d	Sketch depth (number of rows)	4
B	Bits per counter before allocation	8
M	Total memory budget for sketch	512 KB

Table 1 Notation used in the sketch-based heavy-hitter detection model.

spend disproportionate precision on the keys near or above the heavy threshold.

Count-Sketch is another common primitive, designed to reduce bias by using random signs. It maintains $Cj, h_j k \leftarrow Cj, h_j k \; s_j k \Delta$ where $s_j k \in \{-1, 1\}$ is a hash-based sign. The estimate is the median of $s_j k Cj, h_j k$ across rows [14]. Under assumptions of 4-wise independent hashing, the estimator is unbiased with variance tied to the ℓ_2 norm of the tail of the frequency vector. For heavy-hitter tasks, Count-Sketch can provide stronger separation when the tail has small ℓ_2 energy relative to the head. Yet its hardware implementation can be more complex due to signed arithmetic and potential underflow with limited counter widths.

A systems-aware analysis must consider counter saturation, limited bit widths, and the cost of hashing. In a switch, counters may be 32-bit or smaller, and saturating arithmetic can bias results if heavy hitters exceed the maximum representable value. In a CPU implementation, using 64-bit counters reduces overflow risk but increases memory bandwidth. Adaptive precision allocation can treat bit width as a tunable parameter: most counters associated with low-volume keys can be kept narrow, while counters likely to receive heavy contributions can be widened or stored in a higher-precision tier [15].

Collision error can be expressed explicitly. For Count-Min, for a fixed row j, define the collision mass for key k as

$$X_{j,k} = \sum_{k' \neq k} f_{k'} \, \mathbf{1}\{h_j k' = h_j k\} \quad . \tag{3.1}$$

Then $Cj, h_j k = f_k \; X_{j,k}$ and $\widehat{f}_k = f_k \; \min_j X_{j,k}$. Under uniform hashing, $\mathbb{E} X_{j,k} = F - f_k w$. A Markov bound yields $\Pr X_{j,k} \geq \varepsilon F \leq \mathbb{E} X_{j,k} \varepsilon F$, and taking a minimum over d independent rows yields a failure probability that decays exponentially in d. The resulting guarantee is worst-case in the sense that it bounds collisions by total mass. In heavy-tailed settings, a more informative analysis conditions on the head and tail of the distribution. Let H_m denote the set of top m keys and T_m the remainder. Then for non-heavy keys, collisions with the head dominate false positives, whereas for heavy keys, collisions with the tail dominate the bias in the minimum estimator [16]. Adaptive strategies can therefore attempt to isolate head contributions by dedicating sketch space to candidates, thereby reducing head-on-tail interference.

For Count-Sketch, an estimator for f_k in row j is $Z_{j,k} = s_j k Cj, h_j k = f_k \; \sum_{k' \neq k} f_{k'} s_j k s_j k' \mathbf{1}\{h_j k' = h_j k\}$. The cross terms have mean zero when signs are independent, and variance roughly $1w \sum_{k' \neq k} f_{k'}^2$ under simplifying assumptions. Thus, the tail ℓ_2 mass drives uncertainty. Heavy-hitter detection is then easier when the heavy hitters dominate ℓ_2 as well as ℓ_1, which is often true but not always; many moderate flows can inflate ℓ_2.

Many telemetry pipelines use additional refinements such

Trace	Duration (s)	Packets (M)	Avg rate (Gb/s)	Unique flows $\times 10^3$
CAIDA-1	60	95.3	9.7	320
CAIDA-2	60	88.1	8.9	295
ISP-Backbone	120	210.4	14.3	910
Datacenter-Core	30	75.8	20.5	640
Campus-Edge	90	40.2	3.6	180

Table 2 Summary of packet traces used to evaluate heavy-hitter detection.

Sketch	Memory (KB)	Depth d	Width w $\times 10^3$	Counter type
Count-Min	512	4	256	Fixed-width integer
Count-Sketch	512	5	205	Signed integer
UnivMon	512	6	128	Hierarchical counters
APA-Sketch	512	4	256	Adaptive precision
APA-Sketch+TCM	512	4	192	Adaptive precision + sharing

Table 3 Sketch configurations compared in the experiments under a fixed memory budget.

as conservative update for Count-Min, where on update one increments only those counters that equal the current minimum estimate, reducing overcounting by avoiding unnecessary increments in rows where collisions have already inflated the counter. Another refinement is to combine a sketch with a small exact table for top candidates, using the sketch to propose candidates and the table to count precisely. Such hybrid approaches are particularly compatible with adaptive precision allocation because they naturally define tiers: a low-cost approximate tier and a high-precision exact tier.

The heavy-hitter detection problem also admits specialized algorithms such as Misra–Gries and Space-Saving, which maintain a table of $O1\varepsilon$ keys with approximate counts and provide deterministic guarantees for insertion-only streams [17]. These algorithms are attractive because they directly maintain candidate identities, but they require key storage and per-update table operations that can be costly at high rates, particularly when keys are long. Sketches avoid key storage and replace it with hashing, at the expense of requiring a separate mechanism to recover identities. Adaptive precision allocation can be viewed as bridging these worlds: it uses sketches for broad coverage and low cost, while allocating a limited key-aware memory budget to maintain a candidate dictionary whose size adapts to observed traffic.

Identity recovery is itself a bottleneck. A sketch can estimate frequencies of queried keys, but to list heavy hitters one needs candidate keys. Typical solutions include maintaining a small cache of recently seen keys, sampling packets to extract keys, using a hierarchical sketch over prefixes, or using invertible data structures. For example, an invertible bloom lookup table-like structure can support peeling to recover keys when the structure is sparse [18]. However, invertibility can fail under high load and can be adversarially attacked by creating collisions. Adaptive precision allocation suggests a more conservative approach: maintain an always-valid candidate set of bounded size and use the sketch to prioritize which keys enter and remain in that set.

The merging property is critical in distributed telemetry. If multiple devices observe disjoint substreams, their sketches can be merged by elementwise addition when hashes and dimensions match. For Count-Min and Count-Sketch, this yields exact equivalence to processing the union stream. Adaptive schemes must therefore ensure that any adaptive changes either occur in a synchronized manner across devices or are encapsulated in summaries that can still be merged. One approach is to keep the base sketch fixed and mergeable while allowing only the overlay candidate dictionary to vary; the overlay can be merged by set union with conflict resolution [19]. Another approach is to maintain multiple fixed sketches at different precisions and treat adaptivity as choosing how to combine them, which preserves mergeability by keeping the underlying structures fixed.

To understand the role of precision, it is helpful to express memory and error scalings explicitly. A Count-Min sketch with depth d and width w uses dw counters. If each counter uses b bits, the memory is $M = dwb$. The standard additive error scales as OFw, so increasing width reduces error linearly, while increasing depth reduces failure probability exponentially. In high-rate telemetry, depth is often kept small due to the per-packet cost of multiple memory accesses, so width dominates [**?**]. Adaptive precision allocation can be interpreted as dynamically reshaping the effective width experienced by candidate keys. If a heavy hitter can be isolated in a dedicated region with fewer competitors, its collision mass decreases without

Stage	Flow volume percentile	Counter precision (bits)	Allocation share (%)
Background	$0, 80$	4	25
Mid-volume	$80, 95$	6	25
High-volume	$95, 99$	8	25
Candidate heavy-hitters	$99, 100$	12	25

Table 4 Example adaptive precision allocation policy over flow volume percentiles.

Method	Precision (%)	Recall (%)	F1-score (%)
Count-Min	86.4	78.1	82.0
Count-Sketch	88.7	80.3	84.3
UnivMon	91.9	83.5	87.5
Static 8-bit sketch	92.6	84.1	88.1
APA-Sketch (proposed)	96.8	91.7	94.2

Table 5 Heavy-hitter detection accuracy at $\phi = 0.01$ on the ISP-Backbone trace.

requiring a global increase in w.

Similarly, numerical precision interacts with probabilistic error. Suppose counters are quantized with step size q or are probabilistically updated via sampling, such that expected increments equal Δ but actual stored increments are noisy. Then the estimator has an additional quantization or sampling variance term. If quantization is uniform, the resulting mean squared error for a counter after n updates has a term proportional to nq^2 under basic assumptions. Adaptive allocation can reduce total error by using small q (high precision) only for counters expected to accumulate large mass or that are near a decision boundary, while using larger q elsewhere [20].

A central technical challenge is to integrate these error sources into a unified allocation policy that is stable and efficient. The next section models adaptivity as an optimization over precision budgets, incorporating both collision error from hashing and numerical error from limited representation, while explicitly respecting latency and energy constraints.

4. Adaptive Precision Allocation: Optimization, Control, and Learning-Compatible Surrogates

Adaptive precision allocation is modeled here as choosing, over the course of a window, how to distribute measurement resources across keys or regions of the sketch. The difficulty is that keys are not explicitly tracked in a pure sketch, so allocation decisions must be expressed in terms of sketch-accessible signals, such as counter magnitudes, estimated uncertainties, or candidate scores derived from auxiliary sampling.

A useful starting point is a two-tier architecture. Let the base tier be a fixed sketch $\mathcal{S}_0$ with parameters d_0, w_0, b_0 that processes every packet. It provides baseline estimates with known error bounds. Let the adaptive tier be an overlay $\mathcal{S}_1$ that consumes a subset of updates, either by filtering on candidate membership or by probabilistic sampling, and that has higher precision, for example larger width, larger counters, or key-aware exact counting. The goal is to define a policy π that chooses which updates go to $\mathcal{S}_1$ and how $\mathcal{S}_1$ allocates its internal resources.

Let $a_t \in \{0, 1\}$ indicate whether update t is processed by the overlay, with $a_t = 1$ meaning it is. If the overlay is a key-aware table, then it processes updates only for keys in its dictionary; if it is a sketch, it can process all updates but with a higher cost [21]. A general resource constraint is

$$\sum_{t=1}^{T} a_t \leq B \quad , \tag{4.1}$$

where B is a budget of overlay updates per window, reflecting CPU cycles, memory bandwidth, or pipeline slots. One can also allow weighted budgets when Δ_t varies.

The error objective for heavy-hitter detection can be connected to classification around the threshold. For each key k, define a decision score $s_k = \widehat{f}_k - \phi\widehat{F}$, where $\widehat{F}$ is an estimate of total volume. A key is reported heavy if $s_k \geq 0$. Errors occur when s_k has the wrong sign relative to $f_k - \phi F$. If estimation error for f_k is modeled as a random variable with mean bias b_k

Method	Throughput (Mpps)	Median latency (μs)	99th percentile (μs)	CPU utilization (%)
Count-Min	82.3	4.1	8.7	63
Count-Sketch	76.5	4.8	9.9	68
UnivMon	59.2	6.7	13.4	79
Static 8-bit sketch	88.0	3.9	8.1	66
APA-Sketch (proposed)	94.6	3.5	7.3	61

Table 6 Throughput and latency under synthetic line-rate traffic at 40 Gb/s.

Component	Memory share (%)	Description
Sketch counters	64	Core counting array for packet volume
Hash seeds	4	Parameters for the d independent hash functions
Allocation metadata	12	Per-stage precision configuration and thresholds
Candidate cache	15	Small table of top flows tracked explicitly
Runtime statistics	5	Monitoring and reconfiguration state

Table 7 Memory footprint breakdown for APA-Sketch with a 512 KB budget.

and variance σ_k^2, then a Gaussian-like approximation yields [22]

$$\text{Prfalse negative for } k \approx \Pr\widehat{f}_k < \phi\widehat{F} \mid f_k \geq \phi F \tag{4.2}$$

$$\approx \Phi\left(\frac{\phi F - f_k \;\; b_k \;\; \phi F - \widehat{F}}{\sigma_k}\right) \quad , \tag{4.3}$$

where Φ is the standard normal CDF. While sketch errors are not truly Gaussian, this form motivates focusing resources on keys with small margin $m_k = f_k - \phi F$ and large uncertainty σ_k. Adaptive precision allocation aims to reduce σ_k and bias for those keys most likely to influence decisions.

Because f_k is unknown, the policy must estimate margin and uncertainty from sketch state. For Count-Min, a conservative uncertainty proxy is the spread between row counters for a key, because collisions vary across rows. For Count-Sketch, the sample variance across rows is more directly meaningful [23]. Another signal is temporal persistence: a key whose estimate remains large across multiple sub-intervals is more likely to be truly heavy than a key that spikes briefly due to collisions. Adaptive policies can exploit such signals without tracking all keys.

A tractable optimization model is to allocate an effective precision $p_k \geq 0$ to each key in a candidate universe $\mathcal{C}$, where $\mathcal{C}$ is a manageable set derived from sampling or a cache. Precision p_k can represent the number of overlay counters or the number of overlay updates devoted to k. Suppose the estimator variance for k scales as $\sigma_k^2 p_k \approx \sigma_{k,0}^2 1 \;\gamma p_k$ for some $\gamma > 0$, capturing diminishing returns. Suppose also that the bias decreases similarly or is controlled by conservative update. Then a weighted risk objective can be approximated as

$$\min_{\{p_k\}} \quad {}_{k\in\mathcal{C}}\, w_k\, \Psi\left(\frac{\widehat{m}_k}{\sigma_k p_k}\right) \tag{4.4}$$

$$\text{s.t.} \quad {}_{k\in\mathcal{C}}\, c_k p_k \leq P \quad , \quad p_k \geq 0 \quad , \tag{4.5}$$

where $\widehat{m}_k$ is an estimated margin from the base sketch, c_k is a cost per unit precision for k, P is the overlay precision budget, and Ψ is a decreasing function that proxies error probability, such as $\Psi z = \exp -z^2 2$ or $\Psi z = \Phi - z$. This continuous relaxation yields a water-filling style solution when Ψ is convex in $1\sigma_k$ and σ_k is a rational function of p_k. The corresponding KKT conditions can be expressed via a Lagrangian [24]

$$\mathcal{J} = {}_{k\in\mathcal{C}}\, w_k\, \Psi\left(\frac{\widehat{m}_k}{\sigma_k p_k}\right) \;\lambda\left({}_{k\in\mathcal{C}}\, c_k p_k - P\right) - {}_{k\in\mathcal{C}}\, \mu_k p_k \quad , \tag{4.6}$$

with $\lambda \geq 0$ and $\mu_k \geq 0$. For differentiable Ψ and σ_k, stationarity gives

$$\frac{\partial \mathcal{J}}{\partial p_k} = w_k\, \Psi'\left(\frac{\widehat{m}_k}{\sigma_k p_k}\right) \cdot \left(-\widehat{m}_k \frac{\sigma_k' p_k}{\sigma_k^2 p_k}\right) \;\lambda c_k - \mu_k = 0 \quad . \tag{4.7}$$

When $\sigma_k p_k = \sigma_{k,0}\sqrt{1\;\gamma p_k}$, then $\sigma_k' p_k = -\frac{\gamma}{2}\sigma_{k,0} 1\;\gamma p_k{}^{-32}$ and the marginal benefit of precision decreases as $1\;\gamma p_k{}^{-1}$. The resulting allocation assigns more p_k to keys with large w_k, small estimated margin magnitude, and high baseline uncertainty.

This continuous view must be mapped to discrete actions. In practice, p_k may correspond to allocating one of a limited number of slots in a candidate table, promoting a key to a higher-precision sketch, or increasing its sampling rate among a small set of allowed levels. Discretization introduces combinatorial

Variant	Adaptive precision	F1-score (%)	Throughput (Mpps)
Baseline sketch	No	88.1	88.0
+ Adaptive precision only	Yes	93.6	90.2
+ Candidate cache only	No	90.4	86.9
+ Both (APA-Sketch)	Yes	94.2	94.6
+ Both + TCM sharing	Yes	93.9	96.1

Table 8 Ablation study of design components in the proposed scheme.

Heavy-hitter threshold ϕ	Precision (%)	Recall (%)	F1-score (%)
0.0025	91.8	94.5	93.1
0.0050	94.2	93.9	94.1
0.0075	95.6	92.7	94.1
0.0100	96.8	91.7	94.2
0.0150	97.9	88.3	92.9
0.0200	98.4	84.2	90.7

Table 9 Sensitivity of APA-Sketch accuracy to the heavy-hitter threshold ϕ.

complexity. A representative discrete formulation is [25]

$$\min_{\{x_{k,r}\}} \quad \sum_{k\in\mathcal{C}} w_k \, \Psi\left(\frac{\widehat{m}_k}{\sigma_{k,r}}\right) \tag{4.8}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{C}} \sum_{r\in\mathcal{R}} c_{k,r} x_{k,r} \le P \quad , \quad \sum_{r\in\mathcal{R}} x_{k,r} = 1 \quad , \quad x_{k,r} \in \{0,1\} \quad , \tag{4.9}$$

where r indexes discrete precision regimes, such as counter widths or overlay depths. This resembles a multiple-choice knapsack problem. A worst-case optimal solution is NP-hard by reduction from knapsack: one can encode items as keys, regimes as selecting an item or not, costs as weights, and utilities as error reduction. The implication is not that adaptivity is infeasible, but that one should rely on heuristics with predictable runtime, such as greedy selection based on marginal error reduction per cost, possibly with periodic re-optimization.

A greedy heuristic can be derived from the Lagrangian interpretation by computing for each key a marginal value

$$\Delta_k r \to r' = w_k \left[\Psi\left(\frac{\widehat{m}_k}{\sigma_{k,r'}}\right) - \Psi\left(\frac{\widehat{m}_k}{\sigma_{k,r}}\right) \right] \quad , \tag{4.10}$$

and selecting upgrades with the best ratio $\Delta_k \Delta c_k$ until the budget is exhausted. Stability can be improved by adding hysteresis, so that keys are not frequently promoted and demoted due to noise [26]. Hysteresis can be modeled as a switching cost term $\eta \cdot \mathbf{1}\{r \neq r_{\text{prev}}\}$ in the objective, which discourages oscillation.

Multi-objective optimization is intrinsic because latency, memory, and energy do not collapse to a single scalar naturally. A standard approach is a weighted sum:

$$\min_{\pi} \quad \mathbb{E}\text{Err}\pi \;\; \alpha\, \mathbb{E}\text{Lat}\pi \;\; \beta\, \mathbb{E}\text{Eng}\pi \tag{4.11}$$

$$\text{s.t.} \quad \text{Mem}\pi \le M \quad , \tag{4.12}$$

where Err could be a proxy for false negatives and false positives, and Lat and Eng represent per-update or per-window costs. Varying α, β traces a Pareto frontier of policies. In practice, one often imposes hard constraints on latency and memory and optimizes error within those constraints, because exceeding latency can break line rate. That yields a constrained formulation with Lagrange multipliers that can be tuned offline and then fixed online.

The above optimization requires estimates of $\sigma_{k,r}$ or $\sigma_k p_k$, which depend on collision structure and on traffic distribution. A Bayesian-ish approach can provide uncertainty estimates that are more robust to nonstationarity [**?**]. Suppose the true frequency f_k is treated as a latent variable with a prior distribution reflecting heavy-tailed behavior, for example $f_k \sim \text{LogNormal}\mu, \tau^2$ or a Pareto mixture for large values. Observations are sketch-derived estimates $\widehat{f}_{k,0}$ from the base tier and possibly additional noisy

measurements from sampling. One can model

$$\widehat{f}_{k,0} = f_k \ \epsilon_{k,0} \quad , \quad \epsilon_{k,0} \sim \mathrm{NoiseModel}_0 \tag{4.13}$$

$$\widehat{f}_{k,1} = f_k \ \epsilon_{k,1} \quad , \quad \epsilon_{k,1} \sim \mathrm{NoiseModel}_1 p_k \quad , \tag{4.14}$$

where $\mathrm{NoiseModel}_1$ has smaller variance when more precision is allocated. While exact likelihoods for sketch noise are complex, a pragmatic approach is to use variance proxies derived from empirical row spreads and treat them as heteroskedastic Gaussian noise in a working model. The posterior variance $\mathrm{Var} f_k \mid \widehat{f}$ then acts as an uncertainty score. Adaptation becomes a form of Bayesian experimental design: allocate precision to reduce posterior uncertainty for keys whose heavy-hitter classification is uncertain.

A key difficulty is that heavy-hitter detection is inherently about extreme values and threshold crossings, not about minimizing average squared error over all keys. Therefore, the policy should be sensitive to the top tail and to the threshold neighborhood. One way to encode this is to weight keys by a function of their estimated rank or margin, for example $w_k = \rho \widehat{f}_{k,0} \widehat{F}$ where ρ increases sharply near ϕ. Another way is to minimize a differentiable surrogate of the misclassification indicator. A convenient surrogate is the logistic loss: [27]

$$\ell_k = \log\left(1 \ \exp\left(-y_k \cdot \frac{\widehat{f}_k - \phi \widehat{F}}{\tau}\right)\right) \quad , \tag{4.15}$$

where $y_k \in \{1, -1\}$ indicates whether the key is truly heavy, which is unknown online but can be used offline during training or tuning, and τ controls smoothness. Online, one replaces y_k with a soft label derived from the posterior probability of heaviness. This yields a backprop-friendly objective for learning policy parameters, such as thresholds for promotion, hysteresis constants, or sampling rates.

To connect to embeddings and feature vectors, one can view each key k as having a feature representation φk derived from metadata, such as prefix structure, application class, or historical behavior. The policy can then generalize adaptivity across keys: rather than learning a separate parameter per key, it learns a function of features. For example, define a score

$$u_k = \langle \mathbf{w}, \mathbf{x}_k \rangle \quad , \quad \mathbf{x}_k = \begin{bmatrix} \widehat{f}_{k,0} \widehat{F} \\ \widehat{\sigma}_{k,0} \\ \mathrm{age}_k \\ \mathrm{persist}_k \\ \langle \mathbf{e}, \varphi k \rangle \end{bmatrix} \quad , \tag{4.16}$$

where $\mathbf{e}$ is an embedding vector for key attributes and age_k and $\mathrm{persist}_k$ are temporal features capturing how long the key has remained a candidate and how stable its estimate is. A promotion probability can be $p_k = \sigma u_k$ where σ is the logistic function. This is compatible with gradient descent on offline traces [28]. If one wants dimensionality reduction for feature engineering, one can apply PCA or SVD to a matrix of observed key features to obtain a low-rank projection $\mathbf{z}_k = \mathbf{U}_r^\top \mathbf{x}_k$ that preserves the principal directions of variation, reducing compute overhead in the online scorer. Such projections can be computed offline and then compiled into a small matrix multiplication, which can be implemented efficiently in SIMD for software collectors or approximated with fixed-point arithmetic in hardware.

In addition to allocating precision to keys, one can allocate precision to sketch regions. Consider partitioning the width of a sketch into G groups, each with its own width w_g and counter bit width b_g, with $\sum_{g=1}^{G} w_g b_g \leq M'$. A key is assigned to a group based on a routing hash or based on a learned classifier that predicts whether the key will be heavy. The expected collision mass depends on the group width. If group g is reserved for likely heavy keys, it should be wide enough to minimize collisions among them, while groups for tail keys can be narrower. This resembles a mixture-of-experts allocation problem, where the gating function assigns keys to experts (groups) and experts have different capacities [29]. A differentiable relaxation assigns each key a probability $q_{k,g}$ of belonging to group g, with $\sum_g q_{k,g} = 1$, and the expected error becomes a function of q and w_g. Constrained optimization can then be performed offline, and the learned gating can be approximated online using a small set of hash-based features.

Approximate query processing considerations arise when heavy-hitter queries must be answered frequently during the interval, not only at the end. If a controller queries heavy hitters every Δ milliseconds, the sketch must provide intermediate answers. Adaptive precision allocation then becomes a dynamic programming problem over time: one must decide when to spend overlay budget early to improve interim accuracy versus saving budget to handle later changes. A simplified formulation partitions the window into S epochs. Let $p_{k,s}$ be precision allocated to key k in epoch s, with per-epoch budget constraints. The objective aggregates expected error across epochs. One can express a Bellman-like recursion over the overlay state, but exact dynamic programming is infeasible at scale [30]. A practical approximation is receding horizon control: periodically recompute a greedy allocation based on current state and a predicted traffic persistence model, then apply it for the next epoch.

Because adaptivity can be attacked, it is important to preserve a baseline guarantee. A robust design uses the base sketch to ensure that any truly heavy key will, with high probability, appear among the top candidates according to base estimates, even if base estimates are noisy. The overlay then refines within that candidate set. The candidate set size C must be large enough to cover the uncertainty of base ranking. Order statistics of sketch errors can be analyzed to choose C as a function of ε and traffic distribution. Under a heavy-tailed model, the gap between the k-th and k 1-th frequencies can be large, allowing a small C [**?**]. Under flatter distributions, C must be larger. Adaptive precision can adjust C itself, increasing it when the base sketch indicates that many keys are near the threshold, and decreasing it when a few keys dominate.

Finally, precision allocation interacts with communication efficiency in distributed telemetry. If devices report candidate sets and refined counts to a collector, communication can be reduced by encoding only the identities and counts of candidates

whose posterior probability of heaviness exceeds a threshold. The entropy of candidate identities is often much smaller than that of all keys because only a few are plausible heavy hitters. This yields a coding-theoretic motivation for adaptivity: by concentrating precision on a small candidate set, one also concentrates communication on a small set, reducing bandwidth. If counts are quantized, one can allocate more bits to heavier keys, analogous to rate-distortion allocation where the distortion penalty is larger for heavy keys [**?**]. A simple rate-distortion proxy allocates bits r_k to key k to minimize $_k\, w_k 2^{-2r_k}$ subject to $_k\, r_k \leq R$, yielding a water-filling allocation $r_k \propto \frac{1}{2}\log_2 w_k \lambda$ for a Lagrange multiplier λ. While this is idealized, it provides intuition for allocating counter bit width or report precision.

5. Algorithmic Design: Adaptive Sketch Structures, Hashing, and Approximation Guarantees

This section describes concrete adaptive sketch mechanisms that instantiate the optimization ideas while remaining implementable at high rate. The emphasis is on maintaining a simple per-packet update path, bounding worst-case error, and enabling mergeability.

A baseline design begins with a fixed Count-Min sketch $\mathcal{S}_0$ with small depth d_0 to control memory accesses. For each packet update k, Δ, the system updates $\mathcal{S}_0$ and optionally updates an auxiliary structure that supports candidate discovery. Candidate discovery can be achieved by combining three signals: a small reservoir sample of keys, a cache of recently seen keys, and periodic scanning of a small subset of sketch counters to find unusually large counters. The goal is to obtain a candidate universe $\mathcal{C}$ whose size is bounded, for example on the order of a few thousand keys, even when the number of distinct keys is much larger.

Given $\mathcal{C}$, the system maintains an overlay that provides higher precision for candidates. One approach is a key-aware table storing exact counts for candidates, updated only when packets match a candidate key. The challenge is that matching requires a lookup, which can be expensive [31]. In software, a hash table lookup per packet may be too slow at very high packet rates. In hardware, comparing full keys is infeasible. Therefore, a practical approach uses fingerprinting: store a short fingerprint of each candidate key and use it to probabilistically match packets. A fingerprint of r bits yields a false match probability of about 2^{-r}. False matches add noise to candidate counts, but this noise can be bounded and reduced by choosing r appropriately. Adaptive precision allocation can increase r for keys that are near the threshold or are especially valuable, while keeping r small for others.

Another approach uses a secondary sketch $\mathcal{S}_1$ dedicated to candidates. Instead of storing keys, the overlay sketch uses a different set of hash functions and a width tuned to the candidate set size. The key insight is that when only candidates are updated, the effective load factor is much lower, reducing collisions [32]. This preserves mergeability because sketches merge by addition. The system can implement $\mathcal{S}_1$ as a set of small arrays that can be activated for a subset of keys. Activation can be controlled by a gating hash: only keys whose gating hash falls into an active region are refined. The active region can be resized by changing a threshold on the gating hash, which effectively adjusts sampling without changing hash functions.

Selective refinement can also be implemented via hierarchical sketches. For example, maintain a sketch over prefixes, and when a prefix appears heavy, allocate more precision to its sub-prefixes or to individual flows within it. This is useful for hierarchical heavy hitters where the goal is to detect heavy prefixes at multiple aggregation levels. Adaptive precision then becomes a tree allocation problem: allocate measurement budget across nodes in a prefix tree to minimize misclassification [33]. The collision structure differs because keys within a prefix share structure. A sketch can be maintained per tree level, but that increases memory. A more efficient adaptive design maintains a base sketch for all keys and a small set of per-prefix overlays only for heavy prefixes. This resembles sparse refinement in compressed sensing, where one refines only where signal energy concentrates.

The hashing choice matters for both error and performance. In high-rate telemetry, computing multiple strong hashes per packet can be expensive. Many implementations use tabulation hashing or multiply-shift hashing that provides good practical dispersion with low cost [34]. The error guarantees of sketches often assume pairwise or 4-wise independence. Tabulation hashing can provide sufficient independence for many practical bounds, but the exact guarantee depends on the variant. A safe design is to use two base hashes and derive multiple row hashes by linear combinations modulo w, reducing compute while preserving some independence. Let $h_a k$ and $h_b k$ be base hashes. Define row hashes as

$$h_j k = h_a k \;\; j \cdot h_b k \bmod w \quad 35. \tag{5.1}$$

This double-hashing technique reduces computation but introduces correlations that can worsen worst-case behavior. Adaptive precision can mitigate this by using stronger hashing only for candidate refinement, where the volume is lower and cost is acceptable.

Approximation guarantees under adaptivity require careful reasoning. If the base sketch remains unchanged, its standard guarantee holds regardless of adaptivity. The overlay can only improve estimates when combined properly. Suppose the final estimate is a weighted combination $\widehat{f}_k = \alpha_k \widehat{f}_{k,1} \;\; 1 - \alpha_k \widehat{f}_{k,0}$, where $\widehat{f}_{k,1}$ is the overlay estimate when available. If $\widehat{f}_{k,1}$ is unbiased with variance $\sigma^2_{k,1}$ and $\widehat{f}_{k,0}$ has bias $b_{k,0}$ and variance $\sigma^2_{k,0}$, then choosing α_k to minimize mean squared error yields

$$\alpha_k^\star = \frac{\sigma^2_{k,0}}{\sigma^2_{k,0} \;\; \sigma^2_{k,1}} \quad , \tag{5.2}$$

when biases are negligible or corrected. In practice, Count-Min bias is positive and depends on collision mass [36]. One can correct bias by estimating expected collision mass from counter statistics, but that can be fragile. A safer approach is to use the overlay to produce a lower-variance estimate and then use $\widehat{f}_k = \min \widehat{f}_{k,0}, \widehat{f}_{k,1} \;\; \beta_k$ for some guard β_k that accounts for overlay

underestimation due to sampling or fingerprint misses. This preserves the one-sided nature of Count-Min while incorporating refined evidence.

When the overlay uses sampling, unbiasedness depends on Horvitz–Thompson style correction. Suppose updates are sampled with probability p_k for key k. Then an unbiased estimator for f_k is $\widehat{f}_{k,1} = {}_{t:k_t=k}\, \Delta_t \mathbf{1}\{a_t = 1\} p_k$. The variance is approximately $1 - p_k f_k^2 p_k$ where f_k^2 is a second-moment-like term depending on update sizes. This suggests allocating higher sampling probability to keys with larger f_k and to keys near the threshold, aligning with the earlier optimization. However, per-key sampling probabilities are not directly implementable without key tracking. Therefore, a practical method uses a small set of sampling levels and assigns keys to levels based on candidate scores, updating assignments periodically [37]. This yields piecewise-constant p_k.

Adaptive counter width is another concrete mechanism. Consider storing counters in a compact format such as 16-bit for most cells and promoting to 32-bit or 64-bit when a counter exceeds a threshold. Promotion can be implemented by maintaining an overflow table keyed by cell index, storing high bits separately. This reduces memory while avoiding saturation. The promotion threshold can be adapted based on observed counter growth rates. If the stream is heavy-tailed, only a small fraction of counters overflow, so the overflow table remains small [**?**]. The error introduced by saturation is avoided. The overhead is in the overflow lookup on increment, but this occurs only when counters are near the threshold. A careful design uses a two-stage check: increment in 16-bit, detect overflow via a comparison, and only then perform a secondary lookup. In a switch, such conditional operations may be difficult, but in software they are feasible and can be optimized via branch prediction when overflows are rare.

Adaptive width partitioning can be implemented without changing hash functions by using a power-of-two width and masking. Suppose w is a power of two and the index is $hk\&w - 1$ [38]. If one wants to allocate more width to a subset of keys, one can maintain multiple arrays with different widths and route keys to arrays. For example, maintain a small, wide array for candidates and a larger, narrower array for the tail. Routing can be based on a candidate membership test using a bloom filter of candidate fingerprints. A bloom filter membership test is cheaper than a full hash table and can be implemented in hardware. False positives cause some tail keys to be routed to the candidate array, increasing collisions there, but this can be bounded by bloom filter parameters. Adaptive precision can adjust bloom filter size and number of hashes to keep false positives low when candidate array load is high.

The probability of missing a true heavy hitter depends on whether it becomes a candidate and whether refinement preserves it [**?**]. If the base sketch is Count-Min, then a true heavy key k has $\widehat{f}_{k,0} \geq f_k$, so it cannot be underestimated, which helps recall if the decision rule is based on exceeding a threshold. The main risk is that the threshold uses $\widehat{F}$ and that collisions inflate many keys, causing the candidate set to be polluted. If the candidate set is limited to size C, one might miss a heavy key if too many other keys have inflated estimates above it. Bounding this event requires analyzing the number of keys whose collision mass pushes them above f_k. A conservative bound can be obtained by noting that a key's estimate is at most $f_k\ \varepsilon F$ with high probability. Thus, if f_k is significantly larger than $\phi - \varepsilon F$, it will remain above most non-heavy keys. More refined bounds use the heavy-tailed gap between ranks. Under a power law $f_r \approx Cr^{-\alpha}$, the ratio $f_r f_{r1} \approx 1\ 1r^{\alpha} \approx 1\ \alpha r$, so gaps shrink slowly with rank. This suggests that candidate set size should scale with the number of near-threshold keys, which can be estimated from the sketch distribution.

An alternative to a fixed-size candidate set is to use a score threshold rather than a top-C truncation, allowing candidate size to expand when the stream is flatter [39]. This is consistent with adaptive precision allocation: when many keys are near the threshold, the system can allocate more overlay budget to exploration, at the expense of slightly reduced precision per candidate. In a constrained environment, one can cap candidate size but adjust the threshold to keep within the cap.

Approximation algorithms also appear in query planning for telemetry. Suppose a collector receives summaries from multiple devices and must answer heavy-hitter queries per tenant, per prefix, or per application class. It may be infeasible to materialize all aggregates. A dynamic programming approach can decide which intermediate aggregates to compute based on reuse and cost [40]. Let queries correspond to a set of group-by keys, and let computing an aggregate has cost proportional to the number of candidate keys involved. A DP can approximate an optimal plan by exploiting overlap among groupings, similar to classic query optimization, but with approximate summaries instead of exact tables. Sketch merge operations are linear and cheap, but candidate identity reconciliation can be costly. Therefore, the plan should reuse candidate dictionaries when possible. This motivates storing candidate identities in a canonical form and using similarity metrics between candidate sets to decide reuse. If candidate sets are represented as feature vectors of hashed fingerprints, one can compute Jaccard similarity via MinHash sketches and cluster similar devices or intervals, enabling amortized reuse of refinement structures.

Similarity metrics also matter for adaptive policies that track persistence [**?**]. If a key is heavy in one window, it is likely but not guaranteed to be heavy in the next. A policy can allocate initial precision based on previous window candidates. The risk is that stale candidates waste resources when traffic shifts. A compromise is to maintain a low-rank model of historical heavy-hitter patterns. Let $\mathbf{y}_t$ be a vector indicating candidate counts or scores at time t over a universe of recurring keys. One can approximate $\mathbf{y}_t \approx \mathbf{U}\mathbf{z}_t$ where $\mathbf{U}$ is a learned basis and $\mathbf{z}_t$ are coefficients, using SVD or incremental PCA. This yields a compact predictor of which keys are likely to be heavy, allowing pre-allocation. Online, one updates $\mathbf{z}_t$ via gradient descent on observed scores, and one triggers exploration when prediction residual is large.

Because telemetry must run in real time, any learning component must be stable and computationally light. One can precompute the basis $\mathbf{U}$ and use a small number of components r so that projection costs Or per candidate, which is acceptable given that the candidate set is small. The policy can then be

expressed as a simple linear score function with thresholds and hysteresis, which is implementable in both software and control-plane logic [41].

6. Systems Implementation: Data Structures, Storage Internals, and Distributed Execution Mechanics

Implementing adaptive precision allocation requires careful attention to performance engineering. The dominant costs in sketch-based telemetry are memory accesses, hashing, and branch behavior. At high rates, the system is often memory-bound rather than compute-bound, so reducing cache misses and improving locality can matter more than optimizing arithmetic.

A fundamental layout decision is row-major versus column-major storage for sketch counters. Row-major storage, where counters for a row are contiguous, allows sequential scanning of a row for diagnostics but causes updates for a key to touch d different rows, potentially far apart in memory. Column-major, where counters across rows for the same column are contiguous, can improve locality when using the same index across rows, but indices differ across rows due to hashing. A practical compromise is to store each row in a separate contiguous array and to choose w so that each row fits in a level of cache when possible [42]. For example, in a software collector, one might size w so that a row fits in L2 cache per core. Adaptive precision allocation can reduce effective footprint by keeping the base sketch small and placing overlay structures in a separate cache region accessed less frequently.

Hash computation should be vectorized when possible. In software, using SIMD-friendly hash functions allows processing batches of packets. If packets are processed one by one due to I/O constraints, one can still leverage instruction-level parallelism by computing multiple hashes in parallel and unrolling loops. Adaptive overlay updates should be structured to avoid unpredictable branches [43]. For example, instead of an if-statement per packet for whether to update the overlay, one can compute a mask based on a gating hash and use conditional moves or masked increments when supported. This reduces branch mispredict penalties.

Candidate dictionaries can be implemented with compact open-addressing tables storing fingerprints and counts. To avoid storing full keys, store a 64-bit fingerprint of the key, derived from a hash. When reporting heavy hitters, the system must map fingerprints back to keys; this can be done by maintaining a separate key store for sampled packets or by querying an external key database. In many telemetry use cases, the heavy-hitter identity is needed at least at the granularity of the key, so some key recovery mechanism is required. One practical approach is to store a small LRU cache mapping fingerprints to full keys for recently seen candidates [44]. Because the number of heavy hitters is small, this cache can be small. Fingerprint collisions remain possible; to reduce risk, one can use a 128-bit fingerprint stored as two 64-bit words for promoted candidates only, while keeping 64-bit for most. This is an instance of adaptive numerical precision in identity space.

Overflow handling for adaptive counter width should be designed to minimize overhead on the common path. Suppose base sketch counters are 16-bit and an overflow table stores 48 additional bits keyed by the cell index. Each increment performs a 16-bit add; if it does not overflow, it completes. If it overflows, it sets the 16-bit counter to max and adds the excess to the overflow table [45]. To keep overflow lookups cheap, use a fixed-size cuckoo hash table or a small array of buckets indexed by a second hash of the cell index. The overflow rate is expected to be low under heavy-tailed traffic if the base sketch is sufficiently wide. Adaptive allocation can raise the overflow threshold for counters that grow quickly by promoting entire regions to 32-bit, which reduces overflow events in hot regions.

In programmable switches, the constraints are different. Memory is often organized as register arrays with limited read-modify-write per stage. Depth d is constrained by pipeline stages, and hash computation is limited. Therefore, the most feasible adaptivity is coarse-grained and control-plane-driven [46]. A realistic design maintains a base sketch in the data plane and periodically exports partial state to the control plane. The control plane computes which keys or prefixes are candidates and installs match-action rules or bloom filters to steer candidate packets into a second measurement path, such as a set of per-candidate counters or a small candidate sketch. Precision allocation decisions occur at the control-plane timescale, for example every 100 ms or 1 s. This introduces delay but can still be effective because heavy hitters often persist long enough to be captured, and the base sketch provides immediate coarse evidence.

To avoid excessive rule churn in switches, adaptive policies should update candidate filters with hysteresis and should cap the number of changes per interval. A policy can maintain a stable candidate set of size C and replace only a fraction per update interval based on scores [47]. This is analogous to maintaining a stable working set in caching. The control plane can also compute similarity between consecutive candidate sets and avoid updates when similarity is high, reducing churn. Such similarity can be measured by set overlap or by comparing hashed sketches of candidate identities.

Distributed execution introduces additional mechanics. Suppose multiple measurement points observe traffic and must report heavy hitters globally or per segment. A naive approach merges all sketches centrally and then computes heavy hitters. However, identity recovery and candidate refinement complicate this because candidate dictionaries may differ across devices [**?**]. A robust design uses a two-phase protocol. In phase one, each device reports a small summary: base sketch $\mathcal{S}_0$ (or a compressed version) and a candidate list with approximate counts. The collector merges base sketches to obtain global estimates and forms a global candidate set. In phase two, the collector requests refined counts for only the global candidates from devices, which can be obtained either from overlay sketches or from per-candidate tables. This reduces communication because refinement is requested only for plausible heavy hitters. Adaptive precision allocation at devices can be guided by predicted global candidates, for example by pre-allocating overlay precision to keys that were global candidates recently.

Communication efficiency can be improved by compressing sketches and candidate reports. Sketch arrays often contain many

small values and a few large ones [48]. One can delta-encode counters relative to a baseline or use variable-length coding. If counters are nonnegative integers, entropy coding such as Huffman-like coding over counter magnitudes can reduce size. However, in a telemetry system, encoding and decoding costs must be low. Therefore, a practical approach is to quantize counters and compress using simple run-length encoding over zeros or small values, exploiting the fact that many counters may remain small. Adaptive precision allocation can increase quantization resolution for hot regions where accuracy matters and keep coarse quantization elsewhere, aligning with a rate-distortion view. The collector must be aware of quantization to interpret errors correctly.

Mergeability under adaptive overlays depends on the overlay type [49]. If the overlay is a sketch updated on a subset of packets, merging remains elementwise addition provided that sampling decisions are independent across devices or that sampling probabilities are known for correction. If the overlay is a key-aware table keyed by fingerprints, merging requires combining entries with the same fingerprint, which is straightforward, but fingerprint collisions across devices can create false merges. Using larger fingerprints for promoted candidates reduces this risk. If devices use different candidate sets, the union can be taken at the collector. The collector may then request missing candidates from devices if needed. This resembles distributed top-k computation where local top-k lists are merged to approximate global top-k [50].

Windowing and expiration are important in practice. Tumbling windows allow resetting sketches each interval, which simplifies. Sliding windows require expiring old updates. Exact expiration is hard for sketches, but approximate methods exist such as maintaining multiple sub-window sketches and subtracting, or using exponential decay where older contributions are discounted. Adaptive precision allocation interacts with this because overlays may be promoted based on recent evidence. In an exponential decay model, the frequency evolves according to

$$f_k t = \sum_{t_i : k_{t_i} = k} \Delta_{t_i} \exp\left(-\frac{t - t_i}{\tau}\right) \quad , \tag{6.1}$$

where τ is a decay constant [**?**]. Implementing decay in sketches can be done by periodically scaling counters by a factor, but scaling is expensive. Instead, one can use time-partitioned sketches and combine them with weights. Adaptive precision allocation can focus refinement on the most recent sub-windows to track emerging heavy hitters quickly while relying on coarse summaries for older sub-windows.

Performance engineering requires careful accounting. Let c_h be the cost of computing hashes, c_m the cost of a memory increment, and c_o the overhead of overlay gating and potential lookup. The per-packet cost for a base sketch of depth d_0 is approximately $d_0 c_h \; c_m$ plus overhead. If the overlay is updated for a fraction ρ of packets, the expected additional cost is $\rho c_{h,1} \; c_{m,1} \; c_o$. To sustain line rate, this sum must fit within the per-packet budget [51]. Therefore, adaptivity must ensure ρ remains small or that overlay costs are minimal. This justifies designs where overlay updates are limited to candidates and where candidate membership tests are cheap.

Time complexity in Big-O terms is straightforward: sketch updates are Od per packet, and candidate maintenance is typically $O1$ amortized when using fixed-size structures. The more relevant metric is constant factors. A policy that saves a hash computation per packet can be more valuable than a marginal reduction in asymptotic error. Adaptive precision allocation should therefore be designed with micro-architectural awareness, and evaluation should report cycles per packet or packets per second, not only algorithmic complexity [52].

7. Evaluation Methodology, Error Analysis Under Approximate Queries, and Reproducibility Considerations

Evaluating adaptive heavy-hitter telemetry requires measuring both statistical accuracy and systems performance under realistic workloads. Accuracy must be reported in terms of heavy-hitter recall and precision, false positive and false negative rates, and estimation error for top keys. Performance must be reported in terms of throughput, latency, CPU utilization, cache behavior, and memory footprint. In distributed settings, communication overhead and merge latency must be reported as well.

A key methodological issue is that heavy-hitter ground truth depends on the definition of keys and windows. The evaluation should therefore consider multiple key granularities, such as five-tuples, source addresses, destination prefixes, and application identifiers, because the distribution and cardinality differ. It should also consider multiple window lengths and both tumbling and sliding windows, because adaptivity may behave differently when the heavy hitters change quickly [53].

Traffic traces exhibit nonstationarity, so evaluation must include scenarios with abrupt changes, such as flash crowds and attack bursts. A useful practice is to construct composite traces by concatenating segments with different distributions or by injecting synthetic heavy hitters into real background traffic. This allows controlled experiments where the true heavy hitter changes at known times. The policy should be evaluated on how quickly it adapts, measured by time-to-detect and time-to-forget, and on whether it overshoots and causes oscillations.

Approximate query workloads matter. Some systems query heavy hitters continuously, while others query only at the end of a window. If intermediate queries are frequent, then adaptivity must maintain stable interim accuracy [54]. Evaluation should therefore measure accuracy as a function of query time within the window. This can be expressed as a curve of recall versus time since the start of the window. A policy that achieves high recall only at the end may be insufficient for operational tasks requiring quick reaction.

Error analysis should separate collision-induced error, sampling-induced error, and numerical precision error. For collision error in Count-Min, one can empirically measure the distribution of overestimation $\widehat{f_k} - f_k$ across keys and compare it to the theoretical bound εF. Adaptive policies should reduce the tail of this distribution for keys near the threshold. For sampling error, one can measure variance across repeated runs with different random seeds or across bootstrap resamples. For numerical error, one can compare results with different counter

widths and quantify the impact of saturation or quantization [55].

When overlays use fingerprints, false matches introduce additional noise. Let r be the fingerprint bit length. If the overlay table has C entries, and packets are matched by comparing fingerprints, then a non-candidate key will match a candidate with probability approximately $C2^{-r}$ under a uniform fingerprint model. The expected false increment mass over the window is then about $F - {}_{k\in\mathcal{C}}\, f_k C2^{-r}$, which can be bounded by choosing r such that $C2^{-r}$ is small. Adaptive allocation can increase r when C grows or when the tail mass is large. Evaluation should measure the realized false match rate and its effect on candidate counts, especially for keys near the threshold.

Distributed evaluation must consider skew across devices. Heavy hitters may be localized to a subset of devices or may be global [**?**]. A policy that allocates precision locally might miss globally heavy keys that are moderate on each device. Therefore, evaluation should include both localized and distributed heavy-hitter patterns. The two-phase protocol described earlier should be evaluated for communication overhead, number of refinement requests, and end-to-end detection latency. Communication overhead can be reported as bytes per window and as a fraction of baseline telemetry bandwidth.

Reproducibility requires controlling randomness in hashing and sampling. Experiments should report results averaged over multiple seeds and should quantify variability. In addition, implementations should expose configuration parameters clearly: sketch width and depth, counter widths, candidate set size, promotion thresholds, hysteresis constants, sampling rates, and query intervals [56]. Because adaptivity introduces many knobs, it is easy to overfit to a specific trace. A reproducible evaluation should therefore separate traces used for tuning from traces used for testing and should report sensitivity to parameter variation.

A useful approach to summarizing adaptivity benefits is to compare policies at equal resource budgets. For example, fix total memory and per-packet operation count and compare a uniform sketch baseline to an adaptive policy that uses a smaller base sketch plus an overlay. Then report how recall and precision change. Alternatively, fix accuracy targets and compare the required memory and compute. This yields a resource savings view [**?**]. When using multi-objective trade-offs, one can report approximate Pareto frontiers by varying a small number of policy parameters and plotting accuracy versus throughput or accuracy versus memory.

Because micro-architectural effects can dominate, evaluation should include low-level measurements such as cache miss rates, branch mispredict rates, and memory bandwidth utilization for software implementations. For switch-like implementations, evaluation should include the number of pipeline stages used, memory accesses per packet, and update contention. While not all environments allow direct measurement of energy, one can approximate energy by counting memory operations and arithmetic operations weighted by cost models, recognizing that this is an approximation.

It is also important to evaluate robustness. Adversarial patterns can be constructed to cause hash collisions, inflate many keys, or trigger frequent candidate churn. While worst-case adversaries may be unrealistic in benign settings, telemetry used for security should anticipate them [57]. A robust evaluation includes stress tests where many keys are crafted to collide under particular hashes or where the key distribution is flattened to reduce head-tail separation. Adaptive policies should not catastrophically fail in these cases; at minimum, they should degrade gracefully to baseline sketch behavior. This motivates keeping the baseline sketch always active and treating adaptivity as an enhancement rather than a replacement.

Approximate query processing introduces additional error propagation. Suppose the system answers a query for top-k heavy hitters using candidate estimates that are themselves approximate. The top-k set can be unstable when many keys have similar estimated counts [58]. One can model this instability by considering the probability that the estimated rank order differs from the true order. If errors are independent with standard deviation σ_k, then the probability that two keys i and j are misordered depends on the gap $\Delta_{ij} = f_i - f_j$ relative to $\sqrt{\sigma_i^2\ \sigma_j^2}$. Adaptive precision should increase precision for keys whose estimated ranks are near the cutoff, reducing misorder probability. Evaluation can measure rank stability via Kendall tau correlation between estimated and true rankings among the top candidates.

Finally, reproducibility in adaptive telemetry benefits from deterministic fallbacks. While sampling and hashing are random, one can choose fixed seeds and deterministic promotion rules to make runs repeatable. When learning-based scoring is used, one should report the learned parameters and the training procedure. Even when offline tuning is used, one should provide the parameter values and the objective used for tuning [59]. The ultimate goal is that another implementation can replicate observed trade-offs given the same traces and budgets.

8. Conclusion

Sketch-based heavy-hitter detection remains a central primitive for high-rate network telemetry because it offers compact summaries, fast updates, and mergeability. However, uniform-precision sketches are structurally mismatched to the heavy-tailed, time-varying distributions that arise in operational networks. Adaptive precision allocation addresses this mismatch by concentrating measurement fidelity where it most affects decisions, while preserving a low-cost baseline that maintains coverage and worst-case safety.

This paper developed a framework for adaptive precision allocation that spans algorithmic and systems considerations. The modeling view treats precision as a resource distributed across candidates or sketch regions under memory, latency, and energy constraints, with objectives tied to heavy-hitter misclassification risk. Continuous relaxations yield water-filling style allocations, while discrete instantiations expose NP-hardness and motivate greedy, hysteresis-stabilized heuristics [60]. Bayesian-ish uncertainty estimates and differentiable surrogates provide a path toward learning-compatible policies that can be tuned on traces without requiring complex online computation. Connections to embeddings and low-rank projections support generalization across keys and reduce the overhead of scoring and gating.

On the systems side, implementable designs emphasize a fixed, mergeable base sketch augmented by overlays that refine only a small fraction of traffic, using candidate dictionaries, secondary sketches, selective sampling, and adaptive counter width. Performance engineering considerations such as hashing cost, cache locality, overflow handling, and pipeline constraints shape which adaptive mechanisms are feasible in software collectors versus programmable data planes. Distributed execution benefits from two-phase reporting and from communication-efficient encoding that leverages the low entropy of heavy-hitter identities relative to the full key space. The overall picture is that adaptivity is most effective when it is constrained, stable, and layered: a simple baseline provides predictable behavior, and adaptive overlays improve typical-case accuracy without introducing fragile dependencies. Future work in this direction can further integrate windowing and decay, refine robustness under adversarial patterns, and develop standardized evaluation protocols that jointly report accuracy, throughput, and resource efficiency for adaptive telemetry systems [61].

Acknowledgments

We would like to thank the reviewers of this research for their helpful comments and suggestions.

References

[1] A. Pratinav and S. Mishra, "Outbox pattern for reliable distributed systems," *ISCSITR-INTERNATIONAL JOURNAL OF CLOUD COMPUTING*, vol. 6, pp. 18–25, 12 2025.

[2] Q. Liu, D. Du, Y. Xia, P. Zhang, and H. Chen, "The gap between serverless research and real-world systems," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pp. 475–485, ACM, 10 2023.

[3] D. D. Gaudio, B. Ariguib, A. Bartenbach, and G. Solakis, "A live context model for semantic reasoning in iot applications," in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 322–327, IEEE, 3 2022.

[4] C. Herbert, V. Marschin, B. Erb, D. Meißner, M. Aufheimer, and C. Bösch, "Are you willing to self-disclose for science? effects of privacy awareness and trust in privacy on self-disclosure of personal and health data in online scientific studies-an experimental study.," *Frontiers in big data*, vol. 4, pp. 763196–, 12 2021.

[5] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*, pp. 167–184, Springer, 2009.

[6] M. Singh, "Optimization methods for high-load distributed systems," *International Journal of Latest Engineering and Management Research (IJLEMR)*, vol. 10, pp. 32–36, 3 2025.

[7] L. Hunhold, "Streamlining simd isa extensions with takum arithmetic: A case study on intel avx10.2," in *2025 14th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–6, IEEE, 6 2025.

[8] S. Zhu, "Chat application with distributed system," 5 2020.

[9] Y. Tytarchuk, S. Pakhomov, D. Beirak, V. Sydorchuk, and S. V. Zaitseva, "The impact of distributed systems on the architecture and design of computer systems: advantages and challenges," *Data and Metadata*, vol. 3, 12 2024.

[10] N. F. Hasani, "Load balancing in distributed system," *International Journal of Basic and Applied Sciences*, vol. 14, pp. 144–148, 8 2025.

[11] K. M. Goeschka, R. Oliveira, P. Pietzuch, and G. Russello, "Session details: Theme: Distributed systems: Dads - dependable, adaptive, and secure distributed systems track," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ACM, 3 2020.

[12] N. Desai, "Modeling reliability for distributed systems," 1 2020.

[13] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*, pp. 44–44, IEEE, 2006.

[14] V. K. Yadav, "Improve the scalability of system through distributed system," 3 2021.

[15] M. Itmi and A. E. Hami, "Distributed system approach to experiment regional competitiveness," *Computer Science & Information Technology*, pp. 103–109, 2 2018.

[16] R. Faiyaz and M. K. Abdul, "Strategic role of distributed systems in enhancing organisational agility," *Journal of Policies and Recommendations*, vol. 4, pp. 1–, 12 2025.

[17] K. K. Horvath, D. Kimovski, B. Spiess, O. Hohlfeld, and R. Prodan, "Seal-cc: Scalable latency evaluation methodology for internet-of-things services," in *Proceedings of the 14th International Conference on the Internet of Things*, pp. 117–126, ACM, 11 2024.

[18] M. Hauswirth, D. L. Phuoc, and J. X. Parreira, *Semantic Streams*, pp. 3419–3425. Springer New York, 12 2018.

[19] H. Li, H. Liu, and O. Marin, "Sigcse - demystifying distributed systems with a storytelling method," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pp. 1386–1386, ACM, 3 2021.

[20] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, IEEE, 2006.

[21] M. Qiu and S.-Y. Kung, "Special issue on 'smart computing and communication' in international journal of parallel, emergent and distributed systems," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 35, pp. 217–218, 5 2020.

[22] G. A. Qadir and S. R. M. Zeebaree, "Evaluation of qos in distributed systems: A review," 5 2021.

[23] N. V. Mokrova, A. M. Mokrov, and A. V. Safonova, "The distributed systems of engineering supervision of permanent structures," *IOP Conference Series: Materials Science and Engineering*, vol. 365, pp. 062036–, 6 2018.

[24] D. Lindsay, S. S. Gill, D. Smirnova, and P. Garraghan, "The evolution of distributed computing systems: from fundamental to new frontiers," *Computing*, vol. 103, pp. 1859–1878, 1 2021.

[25] L. Sorokin, "Towards auditable distributed systems," 1 2022.

[26] R. Achar, P. Dawn, and C. V. Lopes, "Gotcha: An interactive debugger for got-based distributed systems," 1 2019.

[27] J. Chen, C. Du, P. Han, and X. Du, "Real-time digital simulator for distributed systems:," *SIMULATION*, vol. 97, pp. 003754972098686–309, 1 2021.

[28] Y. Hu and Y. Bo, "Reliability on distributed system considering transmission lines," in *2022 5th International Conference on Power and Energy Applications (ICPEA)*, pp. 459–463, IEEE, 11 2022.

[29] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Resource management for latency-sensitive iot applications with satisfiability," *IEEE Transactions on Services Computing*, vol. 15, pp. 2982–2993, 9 2022.

[30] Z. Zhao, M. Wu, H. Chen, and B. Zang, "Characterization and reclamation of frozen garbage in managed faas workloads," in *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 281–297, ACM, 4 2024.

[31] O. Okusanya, "Consensus in distributed systems: Raft vs crdts," 5 2019.

[32] S. Grant, H. Cech, and I. Beschastnikh, "Icse - inferring and asserting distributed system invariants," in *Proceedings of the 40th International Conference on Software Engineering*, pp. 1149–1159, ACM, 5 2018.

[33] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2701–2706, 2007.

[34] R. A. Al-Saphory and N. J. Al-Jawari, "Sensor and regional gradient detectability of distributed systems," *Journal of Physics: Conference Series*, vol. 1999, pp. 012084–, 9 2021.

[35] G. B. C, A. J. S, S. S, V. M, and R. M, "Water theft and leakage identification in distributed system," in *2022 Smart Technologies, Communication and Robotics (STCR)*, pp. 1–4, IEEE, 12 2022.

[36] M. Krogh-Jespersen, A. Timany, M. E. Ohlenbusch, S. O. Gregersen, and L. Birkedal, *ESOP - Aneris: A Mechanised Logic for Modular Reasoning about Distributed Systems*, pp. 336–365. Germany: Springer International Publishing, 4 2020.

[37] A. Ba, F. O'Donncha, J. Ploennigs, and M. Azmat, "Efficient extraction of insights at the edges of distributed systems," in *2023 IEEE International Conference on Big Data (BigData)*, pp. 1610–1619, IEEE, 12 2023.

[38] A. Thakur, S. Verma, N. Sindhwani*, and R. Vashisth, "Applications of optimized distributed systems in healthcare," 3 2024.

[39] A. Dawra, K. Ramachandran, D. Mohanty, J. Gowrabhathini, B. Goswami, D. S. Ross, and S. M. Basha, "12enhancing business development, ethics, and governance with the adoption of distributed systems," 3 2024.

[40] I. Schagaev, *Distributed Systems: Maximizing Resilience*, pp. 249–266. Springer International Publishing, 7 2019.

[41] R. Laidig, F. Shibli, B. Tufekci, F. Dürr, and C. Tunc, "Improving drone communication qos through adaptive redundancy," in *2025 34th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–9, IEEE, 8 2025.

[42] D. Efimov, A. Polyakov, and A. Aleksandrov, "Proofs for "discretization of homogeneous systems using euler method with a state-dependent step"," 6 2019.

[43] M. Jančič, J. Slak, and G. Kosec, "Mipro - gpu accelerated rbf-fd solution of poisson's equation," in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 214–218, IEEE, 9 2020.

[44] R. Kuznets, "Communication modalities," 5 2024.

[45] I. Moura, A. Teles, D. Viana, J. Marques, L. Coutinho, and F. Silva, "Digital phenotyping of mental health using multimodal sensing of multiple situations of interest: A systematic literature review.," *Journal of biomedical informatics*, vol. 138, pp. 104278–104278, 12 2022.

[46] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*, pp. 628–629, IEEE, 2006.

[47] A.-M. Căilean, C. Beguni, and M. Dimian, "A novel approach for improved precision ranging based on vehicle-to-vehicle visible light communications: Concept and

experimental demonstration," in *2025 International Conference on Applied Electronics (AE)*, pp. 1–4, IEEE, 9 2025.

[48] L. Pick, A. Desai, and A. Gupta, "Psym: Efficient symbolic exploration of distributed systems," *Proceedings of the ACM on Programming Languages*, vol. 7, pp. 660–685, 6 2023.

[49] S. Gupta, R. Verma, and N. Dhanda, "Introduction to next-generation internet and distributed systems," 7 2024.

[50] N. S. Chatharajupalli, R. Rotta, R. Karnapke, and J. Nolte, "Probing considered harmful: Leveraging rssi for link quality prediction," in *2025 IEEE 50th Conference on Local Computer Networks (LCN)*, pp. 1–9, IEEE, 10 2025.

[51] N. B. Bhat, D. Madurasinghe, I. Ozcelik, R. R. Brooks, G. K. Venayagamoorthy, and A. Skjellum, *Evaluation and Design of Performable Distributed Systems*, pp. 211–227. Springer International Publishing, 11 2020.

[52] F. Zhao, Y. Teng, Z. Yang, Y. Xie, J. Liu, and J. Qi, *SeCPlat: A Secure Computation Platform Based on Homomorphic Encryption in Cloud*, pp. 513–518. Germany: Springer Nature Singapore, 5 2024.

[53] H. T. Nguyen, M. Usman, and R. Buyya, "Drlq: A deep reinforcement learning-based task placement for quantum cloud computing," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pp. 475–481, IEEE, 7 2024.

[54] V. N. S. Kiran, "Chaos engineering for building resilient distributed systems," *International Journal of Science and Research (IJSR)*, vol. 9, pp. 1678–1689, 3 2020.

[55] I. Argyroulis, "Recent advancements in distributed system communications," 1 2021.

[56] R. S. Chowhan, *Evolution and Paradigm Shift in Distributed System Architecture*. IntechOpen, 4 2019.

[57] A. Iosup, L. Versluis, A. Trivedi, E. van Eyk, L. Toader, V. van Beek, G. Frascaria, A. Musaafir, and S. Talluri, "The atlarge vision on the design of distributed systems and ecosystems.," 2 2019.

[58] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in *IMMERSCOM*, p. 18, 2009.

[59] S. Devismes, "Versatility and efficiency in self-stabilizing distributed systems," 12 2020.

[60] A. Bolfing, *Distributed Systems*, pp. 143–198. Oxford University PressOxford, 9 2020.

[61] M. Dahlmanns, F. Heidenreich, J. Lohmöller, J. Pennekamp, K. Wehrle, and M. Henze, "Unconsidered installations: Discovering iot deployments in the ipv6 internet," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pp. 1–8, IEEE, 5 2024.