Priority-Aware NoC DVFS under Power Caps

Yash Kodali
University of California, Berkeley
yash.kodali@berkeley.edu

Evan Li
University of California, Berkeley
evan.li@berkeley.edu

Nathan Sutherland
University of California, Berkeley
nathan.sutherland@berkeley.edu

***Abstract*—As warehouse-scale systems increasingly rely on power capping to manage energy costs and infrastructure limits, Network-on-Chip (NoC) resources must be throttled efficiently. However, conventional uniform Dynamic Voltage Frequency Scaling (DVFS) policies fail to distinguish between latency-sensitive control traffic and throughput-oriented batch workloads, leading to severe tail-latency violations. As such, there is a need for strategies that enable power-constrained NoCs to provide performance efficiency for traffic classes without violating global power caps.**

This paper presents a Priority-Aware NoC DVFS framework designed to enforce power caps while selectively preserving the performance of critical traffic. We utilize the SNIPER multi-core simulator to generate network traces from both batch and control class benchmarks from PARSEC and TailBench++ respectively. These traces are evaluated through a cycle-accurate BookSim 2.0 interconnection network simulation to compare three distinct control paradigms: a reactive hardware controller, a queue-theoretic PID controller, and a model-predictive performance-targeted controller.

Our experimental results show that class-aware DVFS reduces control-class `P99` latency under power caps relative to uniform throttling while complying to the power-budget. This work provides a path for managing NoC power without compromising the responsiveness of critical data center services.

***Index Terms*—Warehouse-scale computing, power capping, network-on-chip, DVFS, tail latency, quality of service**

I. Introduction

Warehouse-scale computers (WSCs) increasingly operate under *site-level* power caps imposed by electrical provisioning and cooling limits. When aggregate demand approaches these limits, operators enforce caps by throttling lower-priority work so that user-facing services continue to meet strict tail-latency service-level objectives (SLOs) [1]–[3]. At the processor level, the same constraints appear as tight power/thermal envelopes that motivate fine-grained power management inside the chip. As core counts increase, on-chip data movement and interconnect activity become a major contributor to system power and performance [5].

One straightforward way to enforce a chip power cap is to throttle components uniformly. In the context of NoCs, *uniform throttling* means reducing the frequency (and voltage) of all routers by the same factor to meet a power limit. While effective at capping power, this approach can needlessly degrade performance – especially for latency-critical traffic - because it slows down every network packet uniformly. In many applications, on-chip traffic is heterogeneous: certain control messages require low latency, whereas batch or bulk data transfers can better tolerate delays. Treating all network traffic the same under a power cap fails to account for these differing requirements. As a result, latency-sensitive transactions can suffer unnecessary slowdowns, potentially violating service-level objectives (SLOs), even if other parts of the network have slack.

We can intelligently throttle less critical traffic while preserving performance for critical flows, via Dynamic Voltage and Frequency Scaling (DVFS) on the NoC. DVFS allows modulating the supply voltage and clock frequency of router hardware, trading off performance for power savings in a controllable way [9]. By lowering frequency/voltage on parts of the NoC, dynamic power consumption can be reduced (roughly quadratically with voltage). Our approach is to leverage fine-grained DVFS domains in the NoC to balance network latency and saving enough power to meet the cap. We adjust the frequency of different region domains or individual routers, independently, based on their workload and traffic class. For example, routers primarily handling best-effort batch traffic can be throttled more when needed, freeing power headroom to keep the critical control traffic flowing with low latency.

We explore the question of how a priority-aware NoC DVFS policy can keep control-class packets within their SLO while meeting the power budget, using a trace-driven, cycle-accurate methodology: a 64-core Sniper configuration generates class-tagged NoC packet traces, and BookSim 2.0 simulates a mesh/flatfly NoC with multi-domain DVFS actuation [6], [7]. We compare a uniform-throttling baseline against three families of controllers: (i) a low-cost reactive hardware-style policy, (ii) a queue-theoretic feedback controller, and (iii) an optimization performance-targeted controller. We study how these control policies affect the latency and stability of network traffic across different topologies, packet injection rates, and power caps.

II. Problem Setting

A. Control vs. batch traffic and SLO definition

We model two packet classes. *Control-class* packets (class 0 in our traces) represent latency-sensitive messages whose delay can directly stall cores or elongate critical paths (e.g., coherence commands/acks, synchronization). *Batch-class* packets (class 1) represent throughput-oriented transfers (e.g., bulk data movement or cache-line payload responses) that can tolerate additional latency. This two-class abstraction matches WSC power-management practice, where batch work is typically slowed first during power emergencies [2]. Since absolute cycle targets vary by microarchitecture, we define the control SLO *relative to a practical baseline*: do no worse than uniform

NoC throttling at the same power cap. A priority-aware policy is successful if it stays power-safe while reducing control-class `P99` versus uniform DVFS, accepting some batch performance loss if needed.

B. Microarchitectural Considerations: Flit Reordering and FIFOs

a) Why this matters under NoC DVFS: Applying DVFS to routers and links changes the *service rate* of buffers and arbiters without changing the *arrival process* (packet injection). When DVFS is applied heterogeneously across domains (e.g., per-router or per-region), adjacent routers may operate at different effective speeds. This creates transient *rate mismatch* at domain boundaries, where an upstream router can deliver flits faster than the downstream router can drain them (or vice versa). The mismatch amplifies queueing and can manifest as head-of-line (HoL) blocking for latency-critical control traffic, even if average utilization is moderate. Therefore, DVFS-aware design must explicitly consider buffering, backpressure, and ordering constraints.

b) Flit-level Behavior and Ordering: In wormhole routers, packets are decomposed into flits that traverse the network subject to flow control. Many architectures preserve in-order delivery per flow/VC, but DVFS-induced stalls can cause flits of different packets and/or different classes to interleave differently at routers, especially when: (i) adaptive routing is enabled, (ii) VCs are shared across classes, or (iii) arbitration priorities change dynamically based on class-aware policies. If the model allows multiple paths or multiple outstanding packets per flow, differences in per-hop service rate can also create packet-level reordering at the destination unless additional constraints are enforced.

c) FIFO design points in the simulator/model.: To keep the evaluation faithful while avoiding artifacts, we adopt the following model assumptions:

- **Per-VC input FIFOs.** Each input port maintains per-VC FIFOs with credit-based backpressure; DVFS scales the dequeue/service rate (pipeline progress) but does not alter credit semantics.
- **Class separation.** Control (class 0) and batch (class 1) traffic are either assigned dedicated VCs or use a shared VC with strict-priority arbitration. Dedicated VCs eliminate cross-class HoL blocking at the expense of VC resources; shared VCs more closely mimic deployments where QoS is enforced by priority alone.
- **Boundary buffering.** At DVFS domain boundaries, we ensure sufficient buffering to absorb short rate mismatches. In practice, this is modeled via input FIFO depth and/or an explicit boundary FIFO.

d) Implication for control objectives: These microarchitectural effects directly tie into our control policy. A DVFS reduction in a congested region increases effective queuing delay superlinearly near saturation, so policies that rely purely on average occupancy can inadvertently violate control-class P99. Conversely, targeted DVFS increases in hot spots can "unlock" blocked FIFOs and reduce tail latency without increasing system-wide power, motivating domain-level actuation rather than uniform throttling.

III. Related Work

A. Priority-aware power capping in WSCs

WSC power capping must account for workload priority to protect user-facing tail latency. CapMaestro proposes a scalable, closed-loop, priority-aware control architecture that shifts power budget from low-priority to high-priority work across the power distribution hierarchy [1]. Thunderbolt targets cluster-scale power caps by throttling batch tasks "just enough" while preserving serving QoS, enabling safe power oversubscription [2]. At the single-node level, Rubik demonstrates that fast analytical models can guide power allocation to protect latency-critical work under a cap [4]. Bhattacharya *et al.* highlight that data center power demand can change faster than existing controllers can react, and that instability under power capping can cause dramatic latency degradation [3]. Rubik demonstrates that fast, model-based power control can preserve tail latency for colocated latency-critical services while improving efficiency [4].

B. NoC DVFS and power budgeting

NoC DVFS has been studied extensively, often using utilization or buffer occupancy driven reactive policies.Early closed-loop formulations for NoC power management treat DVFS as a control problem with estimation and feedback [8]. Hesse *et al.* argue that purely reactive DVFS is limited and improve prediction by leveraging coherence protocol information to anticipate NoC demand [9]. At the budgeting level, PEPON proposes hierarchical allocation of a chip-wide power budget to NoC routers to improve performance under a cap [10]. Optimization and control-inspired approaches for multi-VFI systems also exist [11]. Our work differs in focus: we study priority-aware NoC DVFS specifically through the lens of WSC-style control-vs-batch QoS under a hard power cap, with tail-latency as the primary metric.

C. NoC QoS and traffic differentiation

QoS mechanisms such as priority arbitration, virtual-channel partitioning, and bandwidth reservation are also widely used to isolate latency-sensitive traffic. Commercial on-chip fabrics may expose multiple traffic classes to reduce queuing for critical messages [12]. Our approach is complementary: QoS scheduling reduces contention *given a fixed service rate*, whereas DVFS changes the service rate itself. Under a power cap, priority-aware DVFS can be viewed as redistributing limited service capacity to better satisfy control-traffic tail-latency constraints.

IV. Understanding Power Caps in WSCs

Datacenter operators provision power delivery and cooling for a fixed envelope, then enforce *site-level* caps to avoid overload, reduce operating cost, and enable safe oversubscription. Real power demand can change rapidly, and naive throttling can destabilize latency-sensitive services [3]. As a

result, modern WSC power managers explicitly differentiate priorities, throttling batch work more aggressively than serving traffic during cap events [1], [2].

At the node level, site-level constraints translate into per-server or per-socket budgets. Hardware mechanisms and software control loops allocate these budgets across subsystems (cores, caches, memory, and fabrics). In this paper we focus on the NoC share of the node budget and treat it as a hard cap over the policy decision epoch. All policies are evaluated under the *same* NoC power cap to isolate the impact of allocation decisions on control-class tail latency.

V. NoC DVFS Background

Dynamic voltage and frequency scaling (DVFS) trades performance for power by adjusting supply voltage V and clock frequency f. Dynamic power scales approximately as $P \propto V^2 f$, so reducing both can substantially cut power at the cost of lower service rate. Applied to a NoC, DVFS controls each router's *service capacity*: lowering f increases per-hop serialization latency and can increase queueing delay when offered load approaches service capacity.

A. Actuation granularity

DVFS can be applied globally (one domain for the entire NoC) or in multiple voltage-frequency islands that scale independently. Finer granularity enables targeted throttling: domains that primarily carry batch traffic can be slowed while preserving frequency along control-critical routes. However, finer granularity also increases hardware cost and can complicate control. Due to a limitation of time, we focus our evaluation of router-level domain control to understand and quantify the returns of our control policies.

B. Transition latency and control stability

DVFS transitions are not instantaneous: voltage changes require regulator settling, and frequent toggling can waste time and harm latency. At WSC scale, demand variability and control-loop stability are central challenges for power capping [3]; similar issues appear on-chip when DVFS is driven by bursty injection and localized congestion.

a) Hysteresis, quantization, and actuation granularity.: NoC DVFS typically supports only a finite set of frequency points and may apply per-domain rather than per-router in real systems. Quantization can cause limit cycles when the operating point hovers near a threshold. To mitigate this, we incorporate:

- **Hysteresis** on occupancy/latency thresholds to prevent frequent toggling,
- **Rate limiting** (e.g., max Δf per epoch) to reduce oscillations,
- **Minimum dwell time** at a DVFS point to amortize transition overhead.

b) Mapping DVFS to NoC service.: In the model, DVFS scales the effective router pipeline rate and link traversal rate, thereby scaling the *service rate* μ of queues. From a queueing perspective, reducing frequency increases the utilization $\rho = \lambda/\mu$, and tail latency grows rapidly as $\rho \to 1$. This motivates policies that: (i) protect control traffic via an SLO-aware override (boost when control P99 approaches the SLO), and (ii) avoid globally reducing μ uniformly when congestion is localized.

C. DVFS as a NoC service control knob

From a queuing perspective, each DVFS decision changes the service rate μ of the router pipeline. When the offered load λ is low, reducing f primarily increases serialization latency. When λ approaches μ, queuing delay dominates and tail latency can rise sharply. Priority-aware DVFS aims to keep μ high where it most benefits control-class `P99` (hot spots and critical routes), while reducing μ in regions that primarily affect batch throughput.

VI. Infrastructure

A. Overview

The simulation framework framework follows a multi-stage process that separates DVFS policy development with trace generation. This separation allows us to test to tweak and experiment with policies quickly without having to re-run the entire benchmark every time. First, we feed control and batch class benchmarks into an x86 simulator, SNIPER to get network traces. Then, we feed those network traces as well as a custom DVFS policy into a fast network simulator, BookSim 2.0 that captures metrics.

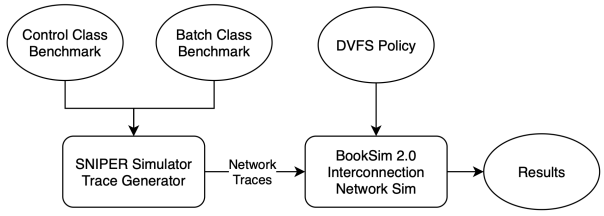

Fig. 1. Infrastructure Setup Flowgraph

B. Benchmarks

To evaluate how power management affects different types of data center traffic, we needed to simulate two specific types of workloads: Control-class and Batch-class. In modern data centers, these different tasks often run on the same hardware. However, they have very different goals. Control-class tasks, like web search or voice recognition, need to be extremely low latency. Batch-class tasks, like compressing files or processing images, need high throughput but the short-term time requirement to finish execution is not as important. By testing both together, we can see if aggressive power saving for the batch tasks affects latency-sensitive control jobs.

For our benchmarks, we chose TailBench++ for the control class and PARSEC for the batch class. From TailBench++, we used Xapian (search) and Sphinx (speech) because they act like real internet services that get bursts of user requests. From PARSEC, we used VIPS and Dedup because they move a lot of data around, which puts a heavy load on the Network-on-Chip (NoC). This setup lets us test if our power-saving policies can differentiate between "urgent" traffic and "background" traffic.

Initially, we planned to use a suite called DCPerf, a set of benchmarks from Meta that closely mimics datacenter workloads. While DCPerf is very realistic, it is designed to run on a "full-system" simulator, which simulated benchmarks too slow for our schedule. Instead, we switched to the SNIPER multi-core simulator. SNIPER allowed us to skip the slow parts of full-system simulation and generate network traces much faster.

C. Trace Generation

In order to simulate a NoC, we need to first generate traces from the benchmarks. We initially explored the gem5 simulator to perform full-system cycle-accurate simulations; however, due to significant time constraints and the immense computational overhead required for many-core configurations, we transitioned to the SNIPER multi-core simulator for the trace generation phase.

SNIPER is a a parallel multi-core simulator with the detail of cycle-level simulation that uses interval simulation. It groups instructions into intervals between major events like branch mispredictions or cache misses, which greatly saves simulation time. In comparison tests, Sniper has been shown to be the fastest and most accurate among contemporary x86 simulators like gem5 and PTLsim, though it is slightly less flexible for modeling entirely new hardware features. [6] This limitation is not a concern for our study. Since our primary goal is to generate benchmark network traces, SNIPER's high speed and validated accuracy on x86 architectures make it the most effective tool for our infrastructure.

D. Network Simulator

The second half of our infrastructure uses BookSim 2.0, a cycle-accurate simulator specifically designed for the Network-on-Chip (NoC). Because we use pre-generated traces from SNIPER, we can run the same traffic patterns through BookSim over and over again while changing only the policies that modify power and frequency settings. This allows us to see precisely how a site-level power cap affects the latency of important data packets.

In addition to standard BookSim metrics, we model DVFS actuation either *globally* (single DVFS domain) or *spatially* (multiple domains, up to per-router granularity). Each router maintains a continuous `freq_scale` that controls how many internal pipeline steps it can advance per BookSim cycle. Lower `freq_scale` therefore reduces the effective service rate of routing, VC allocation, switch allocation, and crossbar traversal, increasing queueing delay and tail latency under load.

Policies execute at fixed DVFS epochs. During an epoch, the simulator collects telemetry including total NoC power, per-router queue occupancies, injection/stall rates, and per-class latency percentiles. At the epoch boundary, a policy computes the next frequency scale(s), clamped by `dvfs_min_scale` and `dvfs_max_scale`, and applied to the configured `router_domains`. This mirrors a hardware/software DVFS governor where sensors are sampled periodically and new operating points are programmed with bounded actuation granularity.

E. ORION 3.0 power model integration

We integrate the ORION 3.0 power model directly into BookSim's IQRouter implementation. When `use_orion=1`, each router initializes the ORION model with architectural parameters from the configuration (e.g., `Vdd`, `Orion_Freq`, flit width, VC count, buffer sizes, arbitration models, crossbar model, and wire length). The router attaches buffer and switch monitors to track ingress/egress activity and initializes an ORION link bus model to estimate link energy. This gives us an activity-driven power estimate that reflects actual routing, buffering, and crossbar usage under the current DVFS setting.

At each DVFS epoch, the traffic manager queries each router for its ORION-based power using the current `freq_scale`. Internally, ORION computes average energy per cycle from the observed buffer writes/reads, converts that to power using the scaled frequency, and adds a simple link power term based on output flit activity. These per-router values are summed into `total_power` and stored in `PowerTelemetry`, along with headroom under the cap. The monitors are reset at the end of each epoch so that the next update reflects only the new interval. This ORION-derived telemetry is what drives our power-capped DVFS policies and is logged in the energy/epoch CSVs for analysis.

When ORION is disabled, we fall back to a lightweight analytic power model that uses `power_dyn_base`, `power_leak_base`, and the configured `dvfs_freqs` / `dvfs_voltages` (or per-domain lists) to compute dynamic power proportional to $V^2 f$ and leakage proportional to V. This fallback is useful for fast design-space sweeps, while ORION provides the more detailed per-router power accounting used in our main experiments.

F. Class-aware scheduling and the DVFS control loop

Our simulator makes policies class-aware end-to-end. Each injected packet carries a *class ID* (synthetic traffic, or trace-defined via `node_types` in netrace). A pluggable `ClassAssigner` interface can override or infer classes based on source/destination or runtime signals; in our current experiments we use static, trace-provided class labels. Per class, `ClassConfig` specifies a base priority and an optional latency SLO in cycles. When an SLO is present, we track that class's tail latency (`P99`) and treat it as the *control class*.

Class awareness also appears in router scheduling. A `StaticPriorityPolicy` assigns each flit the class's base priority at enqueue time, and a `DeadlineBoostPolicy`

can temporarily increase a class's priority when its measured `P99` exceeds its SLO. This creates a fast path for urgent traffic at arbitration time, while DVFS adjusts the *service rate* of the fabric.

Finally, DVFS actuation is applied through a `NetworkControl` adapter. Policies output either a single domain scale (`SetDomainSpeed`) or per-router scales (`SetRouterSpeed`). Each router tracks a continuous `freq_scale`; lowering this value reduces the router pipeline progress executed per simulator cycle, increasing effective per-hop service time and queueing. Domain and per-router bounds are enforced via `dvfs_min_scale`, `dvfs_max_scale`, and a configurable `router_domains` partitioning. At each DVFS epoch, the simulator summarizes telemetry (power, occupancy, injection/stall rates, and per-class latency percentiles) and the controller computes new scale(s) for the next epoch.

VII. Data

A. Trace Format

```
20 1 0 78 4 0
14 52 51 14 1 1
24 4 0 120 6 0
15 51 50 14 1 1
30 0 63 14 1 0
16 50 49 14 1 1
31 63 62 14 1 0
17 49 48 14 1 1
```

Fig. 2. Example Trace Snippet

This is an example SNIPER-generated network trace that includes both batch-class and control-class workloads in a line network. This interleaving provides a pseudo-realistic representation of the system operating with both workload types simultaneously. The formatting of these traces is shown below.

index 0	cycle #
index 1	sender ID
index 2	receiver ID
index 3	packet size
index 4	latency
index 5	control (0) / batch (1)

B. Metrics

We collect both power and QoS-oriented performance telemetry at DVFS epoch boundaries: (i) total NoC power under the current frequency configuration, (ii) per-router input-buffer occupancy (and derived congestion signals), (iii) per-router injection and stall rates, (iv) per-class delivered-packet latency percentiles (P50/P95/P99), and (v) per-class injection and throughput (delivered packets/flits per epoch). These signals are sufficient to implement purely power-driven baselines (uniform throttling) as well as class-aware controllers that explicitly protect control-class `P99` under a hard cap.

VIII. DVFS Control Policies

All DVFS controllers run in discrete *DVFS epochs*. During an epoch, the NoC executes at the currently programmed frequency scale(s). At the epoch boundary, the simulator summarizes telemetry and the controller computes new scale(s) for the next epoch, subject to a hard NoC power budget. We study two baseline policies and three increasingly class-aware controllers.

A. Baselines

Static DVFS (no control). This baseline fixes the NoC frequency scale (typically 1.0) and never adapts. It serves as a reference point for latency and power when the control loop is disabled.

Uniform throttling (power-cap baseline). Uniform throttling enforces the NoC power cap by applying a *single global* frequency scale to all routers. If total NoC power exceeds the cap, the policy reduces this global scale (e.g., linearly) until the estimated power meets the budget; if the cap is not exceeded, it runs at the maximum scale. Uniform throttling is intentionally *not* class-aware: it reacts only to total power and cannot protect control-class tail latency.

B. HWReactive: threshold governor with SLO override

HWReactive models a lightweight hardware DVFS governor. Each epoch it measures a configurable congestion signal X (e.g., average queue occupancy, injection rate, stall rate, or observed latency) and compares it to low/high thresholds $(T_{\text{low}}, T_{\text{high}})$ with hysteresis. In single-domain mode it toggles between $(f_{\text{low}}, f_{\text{high}})$:

$$f[n+1] = \begin{cases} f_{\max} & \text{if } L_{99}^{\text{ctrl}}[n] > \text{SLO} \\ f_{\max} & \text{if } L_{99}^{\text{ctrl}}[n] \approx \text{SLO} \\ f_{\text{high}} & \text{if } X[n] > T_{\text{high}} \\ f_{\text{low}} & \text{if } X[n] < T_{\text{low}} \\ f[n] & \text{otherwise (hold)} \end{cases} \quad (1)$$

The key class-aware feature is the *SLO override*: if the control class P99 latency exceeds (or is close to) its target, the controller forces a high-frequency state to preserve slack.

In multi-domain (per-router) mode, HWReactive computes a per-router metric M_i and normalizes it to a weight w_i (one simple choice is $w_i = \frac{M_i}{\sum_j M_j}$). It then interpolates each router's scale:

$$f_i = f_{\text{low}} + w_i \cdot (f_{\text{high}} - f_{\text{low}}), \qquad f_i \in [f_{\text{low}}, f_{\text{high}}]. \quad (2)$$

Busy routers receive higher frequency, while idle routers are throttled more aggressively, concentrating power where it most

reduces queueing delay. The SLO override can still bias the overall allocation upward when control-class tail latency is at risk.

C. QueuePID: occupancy-regulating feedback loop with budget normalization

QueuePID treats DVFS as service-rate control and regulates router input-buffer occupancy around a target. Let $O_i[n]$ be the measured occupancy of router/domain i at epoch n and $T_i[n]$ be its target. The error is $e_i[n] = O_i[n] - T_i[n]$. A per-router PID loop computes a frequency update:

$$P_i[n] = K_P e_i[n] \tag{3}$$
$$I_i[n] = I_i[n-1] + K_I e_i[n] \tag{4}$$
$$D_i[n] = K_D(e_i[n] - e_i[n-1]) \tag{5}$$
$$\Delta f_i[n] = P_i[n] + I_i[n] + D_i[n] \tag{6}$$
$$f_i[n{+}1] = \text{clamp}(f_i[n] + \Delta f_i[n], f_{\min}, f_{\max}). \tag{7}$$

To bias power toward hot spots, QueuePID uses *adaptive occupancy targets*: routers with higher relative load are assigned lower T_i (forcing speed-up), while lightly loaded routers are assigned higher T_i (allowing slowdown). After computing candidate f_i, the controller enforces the global power cap by a normalization step that preserves relative differences:

$$\text{if } P_{\text{NoC}} > P_{\text{cap}},\ \alpha = \frac{P_{\text{cap}}}{P_{\text{NoC}}},\quad \forall i:\ f_i \leftarrow \alpha f_i. \tag{8}$$

Class awareness enters through the same control-class P99 feedback used by HWReactive: when L_{99}^{ctrl} violates (or approaches) its SLO, QueuePID biases the update upward and can force $f_{\max}$ in the single-domain case. Conceptually, occupancy regulation becomes subordinate to SLO protection whenever they conflict.

D. PerfTarget: direct P99 targeting

PerfTarget closes the loop around the metric of interest: control-class tail latency. Let $L_{99}^{\text{ctrl}}[n]$ be the measured control P99 at epoch n and L_{target} be the setpoint. Define latency error $e_L[n] = L_{99}^{\text{ctrl}}[n] - L_{\text{target}}$. A simple proportional form is:

$$f[n{+}1] = \text{clamp}\Big(f[n] + K_L \cdot \frac{e_L[n]}{L_{\text{target}}},\ f_{\min}, f_{\max}\Big), \tag{9}$$

implemented with step sizes and hysteresis in practice. If the control class has no completed packets in an epoch (no P99 sample), the controller holds its previous decision rather than making a blind adjustment. When the control class has ample slack and power headroom, PerfTarget gently reduces frequency to reclaim power.

In per-router mode, PerfTarget uses occupancy as a spatial hint to *localize* DVFS action under a tight cap: when P99 is high, it boosts high-occupancy routers (likely on the critical path) while throttling cold routers to remain within the budget. This approximates a constrained optimization of "meet L_{target} subject to P_{cap}" without an expensive solver.

E. Why multi-domain, class-aware DVFS helps

Across all three controllers, multi-domain actuation is the key feature that uniform throttling is missing: it allows the policy to *reallocate* limited power toward routers that most influence queueing delay for latency-critical traffic, while extracting power from regions that primarily affect batch throughput. Combined with class-aware telemetry (control P99) and, optionally, priority boosting at arbitration, these policies aim to preserve control-class tail latency under site-level caps where uniform DVFS would degrade all traffic equally.

EVALUATION

In this section, we evaluate whether priority-aware NoC DVFS controllers can (i) satisfy a hard NoC power budget derived from a site-level cap, (ii) keep *control-class* tail latency (P99) within a tight SLO, and (iii) outperform the conventional baseline of *uniform* throttling under the same budget. I focus on two families of metrics: **(a) actuation behavior** (the frequency scales selected under each power cap) and **(b) user-visible performance** (control-class P99 latency).

F. Experimental Setup and Metrics

a) Topology, workload, and classes: All results in this section use the **flatfly** topology. Traffic is split into two classes: **control (class 0)** and **batch (class 1)**. Control traffic represents latency-critical RPCs and coordination; batch represents throughput-oriented background work. The simulator tracks per-class latency distributions and reports **P99 control latency** as the primary SLO metric.

b) Power caps and DVFS actuation: Each experiment is run under a **hard NoC power cap** (x-axis in W). Controllers output a per-domain frequency scale $f \in (0, 1]$; the plots report the **average frequency scale** across time and domains unless otherwise noted. All policies are constrained to remain within the specified budget.

c) Policies compared:

- **Uniform**: A baseline that applies the same frequency scale to all routers/domains to meet the cap.
- **HW Reactive**: A threshold-based reactive controller driven by queue occupancy.
- **Queue PID**: A feedback controller that uses PID control on queue occupancy relative to a target.
- **Perf Target**: A performance/SLO-oriented controller that explicitly biases actuation to meet control-class tail latency.

d) Load levels: To expose behavior across congestion regimes, I sweep multiple offered loads (legend "inj"), which correspond to increasing injection intensity. Conceptually, these runs move the network from lightly loaded (delay dominated by base hop count) to heavily loaded (delay dominated by queuing and contention).

G. Actuation Behavior: Frequency Scale vs. Power Cap

Figure 3 shows the *average* frequency scale each policy selects as the power cap relaxes.

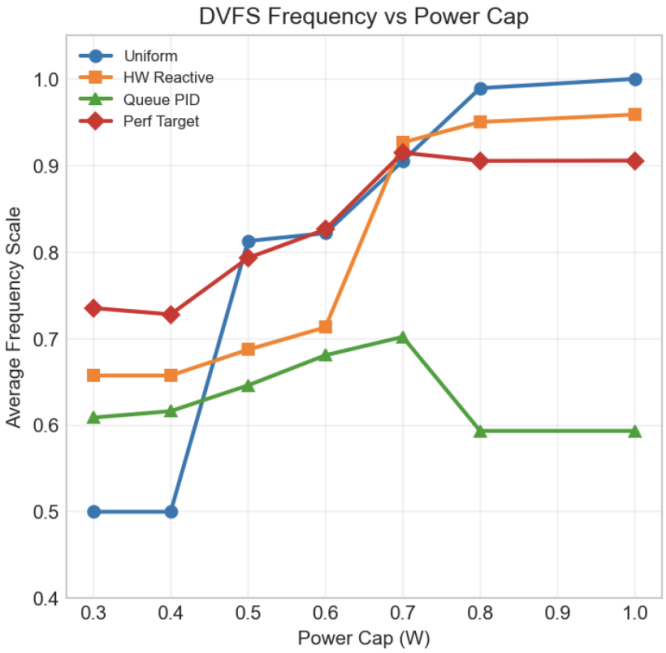

Fig. 3. Average DVFS frequency scale selected by each policy vs. the NoC power cap (flatfly).

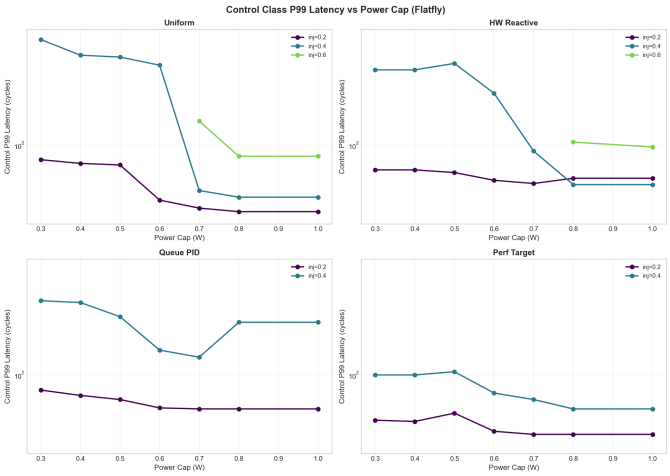

Fig. 4. Control-class P99 latency vs. power cap for multiple injection levels (flatfly). Each panel corresponds to one DVFS policy.

a) Uniform actuation behavior: At the tightest caps (0.3–0.4W), Uniform drops to a very low average frequency (around 0.5). This is expected as the uniform policy has only one degree of freedom (a single global scale), so meeting the cap forces it to slow *everywhere*, including routers that are not on the critical congestion path. As the cap relaxes to 0.5–0.7W, Uniform jumps upward quickly (to ∼0.8–0.9), then approaches 1.0 by 0.8–1.0W.

b) HW Reactive spends more frequency early to avoid instability: HW Reactive stays notably higher than Uniform at the tightest caps (roughly mid-0.6s vs. 0.5). The key mechanism is that occupancy thresholds effectively act as an

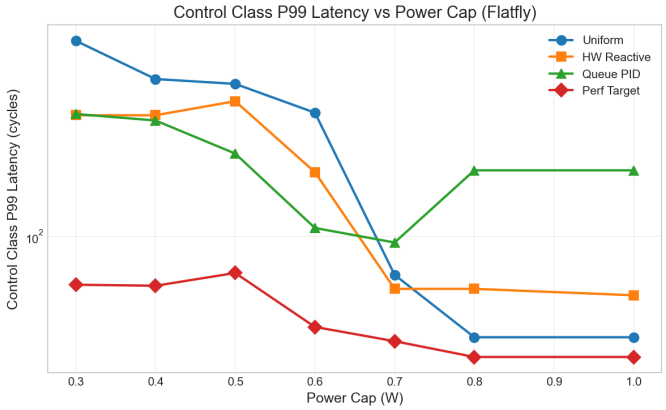

Fig. 5. Aggregate view: control-class P99 latency vs. power cap (flatfly). Lower is better.

early-warning signal: when queues start to build, HW Reactive boosts frequency before the network enters a runaway queuing regime. It effectively uses some of the limited power budget to prevent the system from crossing the high-utilization cliff where tail latency explodes.

c) Queue PID can become conservative at high caps: Queue PID rises gradually with cap up to about 0.7W, but then *drops* its average frequency at 0.8–1.0W. This is an issue due to over-trusting its queue occupancy signal. When average occupancy falls below the target (which is common once congestion eases), the PID loop drives the actuation downward to "save power," even when the cap would permit more headroom. We can see that *average occupancy is not a tail-latency guarantee*. Under bursty control traffic, it is possible to have low average occupancy but still experience transient queue spikes that dominate P99. That mismatch is also what shows up in the latency plots later.

d) Perf Target opts for saving power after achieving target tail latency: Perf Target stays relatively high at low caps (low/mid-0.7s) and then flattens around ∼0.9 beyond 0.7W. This pattern is consistent with an SLO-oriented goal: once the controller has achieved the desired tail-latency region, it stops "chasing" maximum frequency and instead maintains a stable operating point with margin for bursts. This policy is the most *stable performance* under the cap.

H. Primary Result: Control-Class P99 Latency vs. Power Cap

Figure 5 summarizes control P99 latency as the cap increases.

a) Uniform has poor tail latency under tight caps: Uniform throttling shows extremely high P99 at low caps and then a dramatic drop around 0.7W.

- Under tight caps, uniform scaling reduces service rate μ everywhere. Even if offered load λ is unchanged, utilization $\rho = \lambda/\mu$ increases. As $\rho \to 1$, queues become very sensitive to bursts, and P99 can grow by multiples even when averages look stable.

- Once the cap is relaxed enough to push ρ away from 1 on the critical links/routers, queueing delay collapses rapidly, producing the sharp improvement around the knee.

We can see that the uniform throttling policy is a brittle strategy as it behaves reasonably only once the cap is high enough that it's not truly constrained.

b) HW Reactive improves over Uniform where it matters: The HW Reactive policy yields substantially lower P99 than Uniform at the most constrained caps. This happens because it prevents persistent queue buildup by boosting frequency in response to congestion signals. The policy is still not perfect—P99 remains elevated at very low caps—but compared to Uniform it avoids the worst-case runaway queuing that dominates tail latency.

c) Queue PID shows an instability region at higher caps: Queue PID improves P99 as caps increase up to about 0.7W, but then *gets worse* beyond that (its P99 rises at 0.8–1.0W). This aligns with the counterintuitive frequency drop in Figure 3. The most plausible explanation we have is that it is a control-theoretic mismatch:

- The PID loop is stabilizing *average occupancy* around a target.
- When the system is lightly loaded, average occupancy naturally falls and the controller responds by reducing frequency.
- That reduction removes burst headroom and increases the probability of transient queue spikes, which disproportionately affect P99.

So Queue PID can be "locally optimal" on its chosen signal (occupancy) while being globally suboptimal on the actual goal (tail latency).

d) Perf Target dominates P99 across caps: Perf Target achieves the lowest control-class P99 latency across nearly all caps. The implication is that explicit performance targeting is more robust than trying to infer tail behavior from occupancy alone. In particular, Perf Target seems to provide:

- **Burst tolerance:** it retains enough frequency headroom to absorb short control bursts.
- **Stability:** it avoids oscillations in actuation that would otherwise create periodic queuing spikes.
- **Cap efficiency:** it does not need to run at full frequency to keep P99 low; it finds a stable mid/high point.

I. Sensitivity to Load: P99 vs. Cap at Multiple Injection Levels

Figure 4 breaks the story down by injection level (offered load), which is essential because tail latency behavior is highly non-linear near saturation.

a) Uniform throttling: In the Uniform panel:

- At low injection (inj=0.2), P99 is comparatively modest and decreases smoothly with cap. This is expected: the network is not near saturation, so DVFS mainly adds a small service-rate penalty.
- At medium injection (inj=0.4), P99 is enormous at 0.3–0.6W and then collapses around 0.7W. This indicates that inj=0.4 sits near the saturation boundary for the flatfly under uniform scaling; tight caps push it into the unstable regime.
- At high injection (inj=0.6), the network remains stressed even at higher caps, producing elevated P99 despite more power headroom.

This is the clearest evidence that uniform DVFS is not graceful under pressure as it fails exactly when there is a high load + tight cap.

b) HW Reactive policy: HW Reactive reduces P99 substantially at medium injection compared to Uniform, especially in the constrained regime. Instead of needing the cap to reach the knee before improving, it begins improving earlier (because it responds to congestion proactively). However, the curves still show a load-dependent region where P99 is high at low caps. If the cap is simply too low, there isn't enough power budget to prevent queues from growing during sustained contention.

c) Queue PID policy: Queue PID improves with cap up to about 0.7W for inj=0.4, but then P99 rises again at 0.8–1.0W. This again points to the limitation of controlling occupancy as a proxy for tail latency. Under bursty traffic, average occupancy can be low while P99 is dominated by rare spikes. A controller that reduces frequency in response to low average occupancy can unintentionally *increase* spike amplitude and frequency.

d) Performance Target policy: Perf Target is the most consistent across injection levels:

- For inj=0.2, P99 stays low and largely insensitive to cap once beyond modest headroom.
- For inj=0.4, it avoids the extreme blowups that Uniform experiences at tight caps and steadily improves with cap.

J. Conclusion

These plots collectively support three conclusions.

a) (1) Uniform throttling is a weak baseline under site-level caps: Uniform DVFS creates a global service-rate reduction that pushes the network into the high-utilization regime where P99 grows explosively. It performs acceptably only after the cap crosses a knee.

b) (2) Priority-aware, localized control improves tail latency under tight budgets: HW Reactive demonstrates that even a simple, congestion-aware strategy can significantly reduce control P99 under the same cap by spending power where queues form, rather than penalizing the whole fabric.

c) (3) The control signal matters: tail-latency targeting beats occupancy targeting: Queue PID illustrates that stabilizing occupancy does not necessarily stabilize P99. The Perf Target policy's superior results suggest that an explicit performance objective (control P99 or an SLO margin) is the right abstraction for WSC-style NoC DVFS under hard caps.

d) Summary: Under site-level-derived NoC power caps, a policy that explicitly targets control-class tail latency achieves the best robustness across caps and loads, while uniform throttling is brittle and occupancy-only PID can misallocate headroom and regress P99 in lightly loaded regimes.

FUTURE WORK

The results we currently have are promising, but they also lead to some next steps, both to strengthen the evidence and to push to more robust or realistic designs.

K. Richer Control Strategies

a) SLO-aware multi-objective control: Right now, the policies effectively optimize one primary objective (control P99) under a hard cap. In practice, WSCs operators care about multiple competing goals: control tail latency, batch throughput, fairness, and stability of actuation (to avoid thrashing). A useful extension is a *multi-objective controller* that explicitly trades off:

$$\min \ \alpha \cdot \text{P99}_{\text{control}} + \beta \cdot \text{slowdown}_{\text{batch}} + \gamma \cdot \|\Delta f\|$$

subject to the power cap and discrete DVFS states. This would make the idea of not killing batch jobs completely measurable and tunable, rather than qualitative.

b) Model-predictive control (MPC) for bursty traffic: A recurring theme we saw is that burstiness dominates tails. Reactive thresholding and PID can lag bursts, and occupancy is often a weak proxy for tail behavior. MPC could be a good fit here: it can use a short horizon forecast (e.g., recent injection history and queue trends) to choose frequency actions that are stable and anticipatory. Even a lightweight MPC variant—receding-horizon search over a small discrete action set—could capture most of the benefit without being computationally heavy.

c) Learning-based policies with safety constraints: There is room to explore reinforcement learning (RL) approaches as well, but only if the policy is constrained by safety: never violate the power cap, and never exceed a control SLO by more than some bound.

L. Better Priority Modeling and Scheduling

a) Beyond two classes: This paper uses a clean control vs. batch split. Real services may want to have multiple priority tiers (e.g., interactive, latency-sensitive background, best-effort batch). Extending the framework to 3–4 classes would stress-test whether the gains persist when prioritization is more nuanced than strict binary separation.

b) Dynamic urgency boosting: Static class labeling is a reasonable starting point, but tail events could also come from *misclassified* or *temporally urgent* traffic (e.g., an RPC that becomes urgent because it is late). A future iteration could incorporate urgency boosting based on age, queueing delay, or remaining slack-to-deadline, and then drive DVFS using urgency-weighted signals rather than raw occupancy.

c) Joint DVFS and arbitration: DVFS is only one knob we chose to focus on. Arbitration policy (strict priority, aging, deficit round robin) and VC allocation can possibly dramatically change head-of-line blocking and cross-class interference. An extension is to treat arbitration parameters as part of the control action, enabling the policy to trade "power headroom" against "scheduling headroom" depending on congestion regime.

M. Expanded Experimental Coverage

a) Workloads closer to production mixtures.: The current experiments demonstrate clear trends, but an important next step would be to evaluate on trace mixes that better reflect co-location: multiple services, multiple batch jobs, and diurnal shifts in load.

We would want to run:

- More diverse injection patterns (microbursts, phase changes).
- Mixed topologies and routing modes (deterministic vs. adaptive).
- Varying domain granularities (per-router, per-quadrant, per-subnetwork).

This would clarify which conclusions are topology-specific versus fundamental to DVFS as service-rate control.

b) Batch impact and end-to-end metrics.: This paper focuses on control P99 because it aligns with the research question. A next step is to quantify the batch-side cost: throughput, slowdown, and tail behavior for batch as well. Ideally we would want to degrade batch gracefully and predictably.

N. More Realistic DVFS and Power Modeling

a) Transition dynamics and overheads.: One area that could affect conclusions is DVFS transition modeling. Real DVFS has latency, intermediate steps, and sometimes constraints on how frequently transitions may occur. More realistic modeling would include:

- Non-zero transition time with settling behavior.
- Discrete voltage/frequency points rather than continuous scaling.
- Switching overheads (energy and/or temporary performance disruption).

This would make the "stability vs. responsiveness" story in the controller design more grounded.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Professor Sagar Karandikar for his guidance and support throughout this project. His expertise in the course CS 294-252: Architectures and Systems for Warehouse-Scale Computers was instrumental in shaping the direction of this research. We also want to thank my classmates for their valuable suggestions and feedback during our in-class discussions, which helped refine our approach to priority-aware power management. Finally, we are grateful for the resources provided by the University of California, Berkeley, which made this work possible.

REFERENCES

[1] Y. Li, C. R. Lefurgy, K. Rajamani, M. S. Allen-Ware, G. J. Silva, D. D. Heimsoth, S. Ghose, and O. Mutlu, "A Scalable Priority-Aware Approach to Managing Data Center Server Power," in *Proc. IEEE Int'l Symp. High-Performance Computer Architecture (HPCA)*, 2019.

[2] S. Li, X. Wang, X. Zhang, V. Kontorinis, S. Kodakara, D. Lo, and P. Ranganathan, "Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale," in *Proc. USENIX Symp. Operating Systems Design and Implementation (OSDI)*, 2020.

[3] A. Bhattacharya, J. M. Culler, R. Kansal, S. Govindan, and S. Sankar, "The Need for Speed and Stability in Data Center Power Capping," in *Proc. Int'l Green Computing Conf. (IGCC)*, 2012.
[4] H. Kasture, D. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast Analytical Power Management for Latency-Critical Systems," in *Proc. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, 2015.
[5] V. Adhinarayanan, I. Paul, J. L. Greathouse, W. Huang, A. Pattnaik, and W.-c. Feng, "Measuring and Modeling On-Chip Interconnect Power on Real Hardware," in *Proc. IEEE Int'l Symp. Workload Characterization (IISWC)*, 2016.
[6] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-core Simulations," in *Proc. Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.
[7] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS)*, 2013.
[8] T. Simunic, S. P. Boyd, and P. Glynn, "Managing Power Consumption in Networks on Chips," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, pp. 96–107, 2004.
[9] R. Hesse and N. E. Jerger, "Improving DVFS in NoCs with Coherence Prediction," in *Proc. IEEE Int'l Symp. Networks-on-Chip (NOCS)*, 2015.
[10] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. T. Kandemir, and C. R. Das, "PEPON: Performance-Aware Hierarchical Power Budgeting for NoC based Multicores," in *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques (PACT)*, 2012.
[11] P. Bogdan and R. Marculescu, "Towards a Science of Cyber-Physical Systems Design: A Time and Energy Perspective," *ACM Trans. Design Automation of Electronic Systems*, vol. 17, no. 3, 2012.
[12] AMD, "NoC and QoS Requirements (UG994)," documentation, 2025.
[13] A. K. Mishra, R. Das, S. Eachempati, R. R. Iyer, V. Narayanan, and C. R. Das, "A Case for Dynamic Frequency Tuning in On-Chip Networks," in *Proc. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, 2009.
[14] M. R. Casu and P. Giaccone, "Rate-based vs Delay-based Control for DVFS in NoC," in *Proc. Design, Automation & Test in Europe (DATE)*, 2015.
[15] M. R. Casu and P. Giaccone, "Power-performance assessment of different DVFS control policies in NoCs," *Journal of Parallel and Distributed Computing*, 2017.
[16] P. Juang, K. Skadron, M. Martonosi, and D. Clark, "Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors," in *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED)*, 2005.