Cost-aware Joint Caching and Forwarding in Networks with Heterogeneous Cache Resources

Faruk Volkan Mutlu and Edmund Yeh
Northeastern University, Boston, USA
{fvmutlu, eyeh}@ece.neu.edu

***Abstract*—Caching is vital for high-throughput networks in data-intensive applications. Dynamic random-access memory (DRAM), often used for caching due to its high data transfer rate, faces limitations in capacity and cost, hindering scalability needed to meet growing demand. Evolving flash storage can augment DRAM, but necessitates caching techniques adapted to its characteristics for optimal network performance. This paper models the cache as a set of storage blocks with varying rate parameters and utilization costs. Utilizing a framework that enables joint caching and forwarding, we introduce an optimization technique based on the drift-plus-penalty method. Our approach minimizes the drift-plus-penalty expression in a virtual control plane and offers a throughput-cache utilization cost trade-off. We implement a corresponding practical policy in the data plane. Simulations demonstrate the superior performance of our approach in total delay and cache utilization costs.**

I. Introduction

In-network caching is pivotal for high-throughput data delivery networks. In this context, cache devices are ubiquitously distributed across the network and placed directly on the forwarding path of requests. Capacity and read rate are critical cache device parameters that influence network performance. As bottlenecks in the caching system can lead to compounding network-wide performance issues, in-network caching places strict demands on these parameters. Dynamic random-access memory (DRAM), commonly chosen for its ability to match network link rates, faces obstacles in scaling network performance due to its limited capacity and high upfront cost. In contrast, persistent storage like solid-state drives (SSDs) offer large capacities at low upfront costs, but are much slower than DRAM. However, they can still serve as additional cache tiers alongside DRAM to enable larger capacities. Some system designs already incorporate this vertical scaling of caches, and given the expected performance improvements in upcoming generations of SSDs, wider implementation of routers with large multi-tiered caches is imminent.

However, a transfer rate gap between memory and storage will persist, and operational costs when using devices like SSDs as short-term caches arise due to power consumption and wear. Therefore, the benefits and costs of using these devices must be carefully balanced in order to build sustainable high performance networks. The first step in this balancing act is the cache admission and replacement policy. However, policies in wide adoption today focus on the characteristics of data rather than caches, making them unsuitable for the task.

In this paper, we propose a *cost-aware multi-tiered caching policy* that intelligently utilizes a variety of memory and storage elements with different characteristics and offers a trade-off between performance and utilization costs. While this policy retains the goal of caching a small set of data objects with high demand in the fastest tier of cache, it also aims to collect a larger set of objects with less demand in slower tiers with more capacity. In this way, it can balance the frequency of cache hits served by cache tiers based on their transfer rates. Furthermore, it makes cache replacement decisions carefully to mitigate the number of low benefit replacements, where a data object is replaced by another with similar measured demand. We build this policy on the VIP framework [1], which provides powerful design and analysis tools that fit our goals. Most notably, we use this framework to enable a joint caching and forwarding policy. The benefits of this coupling were shown in the literature [2], and it allows us to take full advantage of the additional tiers distributed across the network. The contributions of this paper can be summarized as follows:

- An object-level caching model that considers cache tiers with diverse cost and performance parameters, and an extension of the VIP framework to this caching model,
- A distributed joint caching and forwarding algorithm built in the virtual control plane of this extended framework which optimally balances queue backlog and cache utilization cost bounds,
- Practical caching and forwarding policies in the data plane driven by the virtual control plane algorithm and an experimental evaluation of these policies through simulations.

II. Related Work

Feasibility of devices like SSDs for in-network caching has been primarily explored in the information-centric networking (ICN) paradigm. In [3], a two-layer cache with a large SSD layer masked behind a fast DRAM layer was designed, which was later used in a follow-up work to build a high-speed router prototype which achieved more than 10Gbps throughput [4]. Another design was proposed in [5], where block device I/O was separated from forwarding paths by sending packets to a dedicated caching process, allowing forwarding to continue operating at line-rate. While these works address the integration of persistent storage into caching systems, they are not backed by policies designed for multi-tiered caches under the constraints of in-network caching.

The leave-copy-everywhere (LCE) strategy [6], paired with traditional replacement policies such as least recently used (LRU) is a known standard in architectures like Named Data

ICC 2024 - IEEE International Conference on Communications | 978-1-7281-9054-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICC51166.2024.10623104

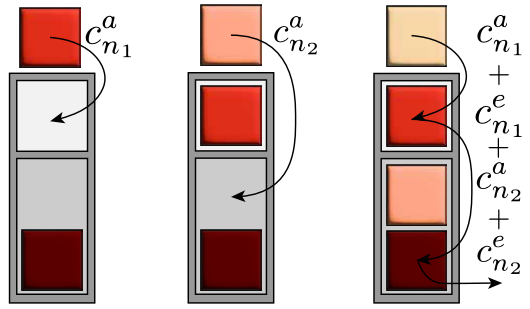


Figure 1: Costs of various cache operations. Admissions into tiers $j = 1$ and $j = 2$ are shown in (a) and (b) respectively. The migration shown in (c) is due to a new object admitted to tier $j = 1$ replacing the existing object there, which is then admitted to tier $j = 2$, in turn causing an eviction in that tier.

Networking (NDN) [7] that employ in-network caching. While simple and effective, this approach does not extend well to multi-tiered caches. Though more sophisticated methods exist in the literature, they tend to focus on monolithic caches and prioritize the characteristics of data objects. Furthermore, cost-aware policies also see little attention. One notable work relating to our motivations in this context is [8], where the endurance costs of using flash-based caches are reflected into the caching policy.

The lack of robust multi-tiered caching policies is one of the reasons why SSD-based caches have yet to see wider adoption in modern high-performance ICN prototypes. In [9], where a state-of-the-art forwarder for NDN was introduced, caching in persistent memory or disk storage was stated as a future direction and the necessity for novel caching algorithms accounting for varying device characteristics in a multi-tiered cache was emphasized. Most recently, an experimental study for a NDN-based data delivery system, intended for science experiments with data volumes in the exabyte range, featured nodes with DRAM-only caches of 20 GB capacity [10].

While neither traditional policies nor novel ones from the literature readily propose extensions to multi-tiered caches, the VIP framework described in [1] stands out as a primary candidate that can support such an extension since it models the transfer rates of caches. We will leverage this to extend the framework to multiple tiers and establish the basis for our multi-tiered caching policy.

III. System Model

Let the unit of content in the network be a *data object* (*object* for short) and assume that each object has the same size. Denote the set of objects in the network as $\mathcal{K}$. Let $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ be a directed graph where $\mathcal{N}$ and $\mathcal{L}$ denote the sets of nodes and links in the network respectively. Assume that $(b, a) \in \mathcal{L}$, if $(a, b) \in \mathcal{L}$ and let C_{ab} be the transmission capacity of link $(a, b) \in \mathcal{L}$ in objects/second. For each object $k \in \mathcal{K}$, assume that there is a unique and fixed node $\mathcal{S}(k) \in \mathcal{N}$ which serves as the content source of k. Requests for data objects can enter the network at any node and are transmitted via request packets of negligible size. A request for object k is forwarded through the network until it reaches either $\mathcal{S}(k)$ or a node n that caches k, at which point the data object is produced at n and delivered to the requester, following the path of the request in reverse.

Assume that any node in the network can have one or more cache(s), each referred to as a *cache tier*. Denote the set of cache tiers at node $n \in \mathcal{N}$ as $\mathcal{J}_n$ and let tiers $j \in \mathcal{J}_n$ be indexed as $j = 1, 2, \ldots, |\mathcal{J}_n|$ where $|\mathcal{J}_n|$ denotes the number of tiers at n. Let each tier have capacity L_{n_j} in objects. Assume that an object k can be cached in at most one tier at a node n, and cannot be cached in any of the tiers at node $\mathcal{S}(k)$. Migrations of objects from one cache tier to another occur when a replacement takes place in one tier and the evicted object is moved into another tier. Admitting a new object into tier j at node n incurs the cost $c^a_{n_j}$. This can represent the time or energy spent on this write operation, its endurance impact, or a combination of these and any other potential costs. Evicting an object from tier j incurs the cost $c^e_{n_j}$, representing similar costs for the read operation. Figure 1 illustrates the costs of various operations in this model.

IV. Virtual Plane

As stated in Section I, we build our caching policy on the VIP framework [1]. In this framework, user demand for objects is measured using *virtual interest packets* (VIPs), which are tracked in a *virtual plane*. The virtual plane operates separately from the request and data forwarding processes described in Section III, which take place in what we refer to as the *data plane*. VIPs are used as the common metric for caching and forwarding decisions, which allow algorithms based on this framework to drive requests toward caching nodes while avoiding congestion in the network. Additionally, this separation reduces the complexity of design and analysis for algorithms.

In view of our design goal described in Section I, the VIP framework provides two additional advantages. First, it embeds the read rates of caches into its analysis of network dynamics in the virtual plane. This is critical for our purposes. Second, as this analysis is based on the *Lyapunov drift* technique, it allows us to formulate a performance-cost trade-off using the *drift-plus-penalty* approach [11].

A. VIP Dynamics

We first summarize the dynamics of VIPs in the virtual plane, as adapted from [1] for the multi-tiered cache model. Time in the virtual plane is represented in slots of length 1 indexed as $t = 1, 2, \ldots$ where time slot t refers to the time interval $[t, t + 1)$. Each node $n \in \mathcal{N}$ keeps a separate *VIP counter* $V^k_n(t)$ for each object $k \in \mathcal{K}$, which are all set to 0 at the beginning of time slot $t = 1$, with $V^k_n(t) = 0$ for all $t \geq 1$ if $n = \mathcal{S}(k)$. When a request for object k enters the network at node n in the data plane, the corresponding counter $V^k_n(t)$ is incremented. The number of requests for object k that enter the network at node n during slot t is denoted as $A^k_n(t)$ and the external *arrival rate* of VIPs at node n for object k is $\lambda^k_n \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} A^k_n(\tau)$. At each

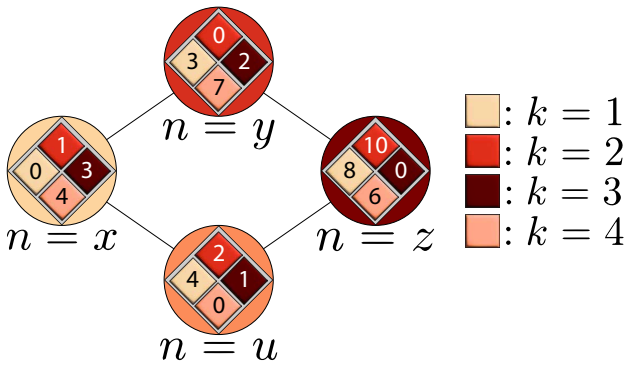


Figure 2: Snapshot of VIP counts of objects at the beginning of slot t for an example network. $\mathcal{S}(1) = x$, $\mathcal{S}(2) = y$, $\mathcal{S}(3) = z$, $\mathcal{S}(4) = u$. $C_{ab} = 2$ for all $(a, b) \in \mathcal{L}$. $|\mathcal{J}_n| = 2$, $L_{n_1} = 1$, $L_{n_2} = 2$, $r_{n_1} = 2$, $r_{n_2} = 1$ for all $n \in \mathcal{N}$. VIP counter values are given by text inside the squares depicting the objects, e.g. $V_z^2(t) = 10$, $V_y^4(t) = 7$.

time slot, nodes communicate their VIP counts and forward VIPs to their neighbors according to decisions made by a virtual plane algorithm. The allocated transmission rate of VIPs for object k over link (a, b) during slot t is denoted as $\mu_{ab}^k(t)$. VIPs for an object k, after being forwarded through the virtual plane, are removed at either $\mathcal{S}(k)$ or nodes that cache k. In each time slot, the communication of VIPs can be handled with a single message of negligible size in the data plane. We define the virtual plane *cache state* for object k at node n in tier j during slot t as $s_{n_j}^k(t) \in \{0, 1\}$, where $s_{n_j}^k(t) = 1$ if the object is cached and $s_{n_j}^k(t) = 0$ otherwise[1], with $s_{n_j}^k(1) = 0$, $\forall n \in \mathcal{N}, k \in \mathcal{K}$.

A tier j at node n has *read rate* r_{n_j}, meaning it can produce r_{n_j} copies of each object it caches per time slot, independent of other objects and tiers. We assume that cache tiers in set $\mathcal{J}_n$ are numbered in descending order of their read rates, i.e. $r_{n_1} \geq r_{n_2} \geq \cdots \geq r_{n_{|\mathcal{J}_n|}}$. Then, the dynamics of each VIP counter can be summarized with the following inequality, where $(x)^+ \triangleq \max(x, 0)$.

$$V_n^k(t+1) \leq \Bigg(\Big(V_n^k(t) - \sum_{b \in \mathcal{N}} \mu_{nb}^k(t) \Big)^+ + A_n^k(t) + \sum_{a \in \mathcal{N}} \mu_{an}^k(t) - \sum_{j \in \mathcal{J}_n} r_{n_j} s_{n_j}^k(t) \Bigg)^+ \tag{1}$$

To demonstrate VIP dynamics concretely, we use the example of Figure 2 and numerically describe how certain VIP counters would evolve from slot t to $t+1$. Assume that $s_{z_1}^2(t) = 1$, $A_z^2(t) = 3$, and an arbitrary virtual plane algorithm allocates $\mu_{zy}^2 = \mu_{zu}^2 = 2$, $\mu_{yz}^2 = \mu_{uz}^2 = 0$. Then, by (1), $V_u^2(t+1) = 7$. Node z has enough VIPs to use the entirety of allocation rates μ_{zy}^2 and μ_{zu}^2. Thus, it forwards 4 of its VIPs for object 2 to its neighbors. It receives 3 VIPs from external request arrivals and drains 2 VIPs via its tier 1 cache since $r_{z_1} = 2$. Now, assume $s_{x_1}^2(t) = s_{x_2}^2 = 0$, $A_x^2(t) = 2$ and the virtual plane algorithm allocates $\mu_{xu}^2 = 0$, $\mu_{xy}^2 = 2$. By (1), $V_x^2(t+1) = 2$. Node x can only forward 1 VIP to y as $V_x^2(t) = 1$, even though the allocated rate is higher. Finally, as node x does not cache object 2 in any of its tiers, the 2 VIPs due to the external arrivals can't be drained via caching.

[1]We assume that in the virtual plane, at each time t, a node can immediately gain access to any data object in the network and cache it locally.

B. Joint Caching and Forwarding Algorithm

Before we state our distributed joint caching and forwarding algorithm for the virtual plane, we first define a *virtual plane cache benefit* function as follows.

$$b_{n_j}^k(t) \triangleq \begin{cases} r_{n_j} V_n^k(t) - \omega c_{n_j}^a, & \text{if } s_{n_j}^k(t-1) = 0 \\ r_{n_j} V_n^k(t) + \omega c_{n_j}^e, & \text{if } s_{n_j}^k(t-1) = 1 \end{cases} \tag{2}$$

where $\omega \geq 0$ is the *importance weight*, an adjustable parameter which determines how we weigh costs against performance. This function incorporates the potential profits and costs associated with object-tier pairings. Objects in higher demand, i.e. those with high VIP counts, are more valuable if kept in faster tiers. This is why read rate is a scaling factor to VIP counts in (2). On the other hand, moving objects between cache tiers or admitting new objects have associated costs, whereas maintaining the cache state of an object does not. This is why the potential benefit of admitting an object is reduced by the weighed admission cost in (2), while that of maintaining an already cached object is increased by the weighed eviction cost, since choosing to maintain this object means we won't be paying the eviction cost while keeping its value[2]. Having defined this function, we now state our algorithm.

Algorithm 1 (Caching and Forwarding in the Virtual Plane)**.** *At the beginning of each time slot t, at each node n, observe VIP counts $(V_n^k(t))_{k \in \mathcal{K}, n \in \mathcal{N}}$, then perform forwarding and caching in the virtual plane as follows.*

Caching: *Let i be an integer in the set of integers $\mathcal{I}_n \triangleq \{1, 2, ..., \sum_{j \in \mathcal{J}_n} L_{n_j}\}$, where each i represents a space in cache that can hold one data object. Let $\mathcal{I}_{n_j} \triangleq \{1 + \sum_{\ell=1}^{j-1} L_{n_\ell}, ..., \sum_{\ell=1}^{j} L_{n_\ell}\}$ be the set of integers that represent cache spaces in tier j. At each node $n \in \mathcal{N}$, choose $s_{n_i}^k(t)$ for each $k \in \mathcal{K}$ and $i \in I_n$ to*

$$\textit{maximize} \quad \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}_n} b_{n_i}^k(t) s_{n_i}^k(t) \tag{3}$$

$$\textit{subject to} \quad \sum_{k \in \mathcal{K}} s_{n_i}^k(t) \leq 1, \ i \in \mathcal{I}_n \tag{4}$$

$$\sum_{i \in \mathcal{I}_n} s_{n_i}^k(t) \leq 1, \ k \in \mathcal{K} \tag{5}$$

$$s_{n_i}^k(t) \in \{0, 1\}, \ k \in \mathcal{K}, \ i \in \mathcal{I}_n \tag{6}$$

where $b_{n_i}^k(t) = b_{n_j}^k(t)$ if $i \in \mathcal{I}_{n_j}$. Then, set $s_{n_j}^k(t) = \sum_{i \in I_{n_j}} s_{n_i}^k(t)$.

Forwarding: *Let $\mathcal{L}^k$ be the set of links which are allowed to transmit VIPs of object k, determined by a routing policy. For each data object $k \in \mathcal{K}$ and each link $(a, b) \in \mathcal{L}^k$, choose*

$$\mu_{ab}^k(t) = \begin{cases} C_{ba}, & \textit{if } k = k_{ab}^*(t) \textit{ and } W_{ab}^k(t) > 0 \\ 0, & \textit{otherwise} \end{cases} \tag{7}$$

[2]Please refer to [12] for a formal derivation of this function.

where,

$$W_{ab}^k(t) \triangleq V_a^k(t) - V_b^k(t),$$
$$k_{ab}^*(t) \triangleq \underset{\{k:(a,b)\in\mathcal{L}^k\}}{\arg\max}\ W_{ab}^k(t), \tag{8}$$

Queue Evolution: *Update VIP counts according to* (1).

The intuition behind the caching problem (3)–(6) is better explained with the added context of the following proposition.

Proposition 1. *A solution to the following problem gives an equivalent solution to the problem defined by* (3)–(6)*.*

$$\textit{maximize} \quad \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}_n} b_{n_j}^k(t)s_{n_j}^k(t) \tag{9}$$

$$\textit{subject to} \quad \sum_{k\in\mathcal{K}} s_{n_j}^k(t) \le L_{n_j},\ j\in\mathcal{J}_n \tag{10}$$

$$\sum_{j\in\mathcal{J}_n} s_{n_j}^k(t) \le 1,\ k\in\mathcal{K} \tag{11}$$

$$s_{n_j}^k(t) \in \{0,1\},\ k\in\mathcal{K},\ j\in\mathcal{J}_n \tag{12}$$

Proof. Since $|\mathcal{I}_{n_j}| = L_{n_j}$, $s_{n_j}^k(t) = 1$ can hold for at most L_{n_j} objects. Therefore, (4) is equivalent to (10). Since $s_{n_i}^k = 1$ can hold for at most one $i \in \mathcal{I}_n$, (5) is equivalent to (11). □

The problem posed by (9)–(12) maps directly to the task of fitting data objects in multiple cache tiers according to the potential benefits of object-tier pairings. Presented in this way, the problem appears in the complex form of a generalized assignment problem (GAP), which is NP-hard. However, by representing a cache tier j at node n as a collection of L_{n_j} cache spaces that can each hold one object, we move from many-to-one mappings between objects and tiers, to one-to-one mappings between objects and cache spaces, revealing that our problem in fact has the much simpler form presented in (3)–(6) [3]. While the problem defined by (3)–(6) is close in form to the canonical assignment problem, because we are not *required* to assign all cache spaces to objects, it exhibits the notable difference that constraints (4) and (5) are inequalities. Nevertheless, this problem can still be solved exactly in polynomial time. While a number of methods exist, we use the algorithm presented in [13], which is a variant of the Jonker-Volgenant algorithm [14]. This algorithm already addresses the fact that (6) is an inequality, as it is targeted at the general case where the number of rows (i.e. cache spaces) are smaller than the number of columns (objects). Addressing the fact that (5) is an inequality is trivial, as we can solve the problem treating it as an equality first, then set $s_{n_i}^k(t) = 0$ for all i such that $b_{n_i}^k(t) < 0$, as this will yield a strictly larger objective by (3). While the worst case complexity of this algorithm is on the order of $O(N^3)$, its median execution time in implementation can be fast. We refer the reader to [13] for an in-depth analysis of both the complexity and execution time of the algorithm in various implementations.

A major benefit of adopting the VIP framework is that we do not need to alter the forwarding approach of [1], which applies the backpressure algorithm [15] to VIP queues to balance the VIP backlog around the network in the virtual plane. This approach extends naturally to pair with our caching approach, as the it avoids forwarding interests in a way that would overwhelm the slower tiers across the network.

[3]Note that the problem can be presented in this way only due to our assumption that all objects have the same size (reflected in constraint (10)).

C. Trade-off Analysis

We now provide a two-step analysis of our algorithm in the virtual plane: (i) characterizing the stability region of the VIP queue network and (ii) showing the trade-off between minimizing queue backlog and total penalty bounds.

1) Stability Region: Our analysis of VIP queues in the virtual plane is based on the Lyapunov drift technique, which allows us to show that our algorithm can stabilize all VIP queues without a-priori knowledge of arrival rates. To show this property of the algorithm, we first characterize the *stability region* [16], [11].

Theorem 1 (Stability Region)**.** *The VIP stability region Λ of the network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, is the set Λ consisting of all VIP arrival rates $\boldsymbol{\lambda} = (\lambda_n^k)_{k\in\mathcal{K},n\in\mathcal{N}}$ such that the following holds*

$$\lambda_n^k \le \sum_{b\in\mathcal{N}} f_{nb}^k - \sum_{a\in\mathcal{N}} f_{an}^k + \sum_{j\in\mathcal{J}_n} r_{n_j}\sum_{i=1}^{\sigma}\beta_{n,i}\mathbf{1}_{[(k,j)\in\mathcal{B}_{n,i}]},$$
$$\forall n\in\mathcal{N},\ k\in\mathcal{K},\ n\ne\mathcal{S}(k) \tag{13}$$
$$0\le\beta_{n,i}\le 1, \quad i=1,\cdots,\sigma, \quad n\in\mathcal{N}$$
$$\sum_{i=1}^{\sigma}\beta_{n,i} = 1, \quad \forall n\in\mathcal{N}$$

where $f_{ab}^k = \sum_{\tau=1}^t F_{ab}^k(\tau)/t$ denotes the time average VIP flow for object k over link (a,b) and $F_{ab}^k(t)$ is the number of VIPs of k transmitted over (a,b) during slot t and satisfying the following for all $t \ge 1$.

$$F_{nn}^k(t) = 0, F_{\mathcal{S}(k)n}^k(t) = 0,\ \forall n\in\mathcal{N}, k\in\mathcal{K},$$
$$F_{ab}^k(t) \ge 0, \sum_{k\in\mathcal{K}} F_{ab}^k(t) \le C_{ba},\ \forall (a,b)\in\mathcal{L} \tag{14}$$
$$F_{ab}^k(t) = 0,\ \forall\notin\mathcal{L}^k, k\in\mathcal{K}$$

Here, $\mathcal{B}_{n,i}$ denotes the i-th among σ total number of possible cache placement sets at node n, and $\beta_{n,i}$ represents the fraction of time that objects are placed at n according to $\mathcal{B}_{n,i}$. The elements of set $\mathcal{B}_{n,i}$ are pairs (k,j), where $(k,j)\in\mathcal{B}_{n,i}$, if object k is cached in tier j.

Proof. Please see Appendix A in [12]. □

2) Backlog-Penalty Bound Trade-off: We now show that, given any positive weight $\omega > 0$, our algorithm can minimize the upper bounds for both the queue backlog and total penalty, using the drift-plus-penalty approach. We assume that request arrival processes are mutually independent for all nodes and objects, and are i.i.d. with respect to time slots. We also assume that there is a finite value $A_{n,max}^k$ such that $A_n^k(t) \le A_{n,max}^k$ for all n, k and t.

Theorem 2 (Throughput and Penalty Bounds)**.** *Given the VIP arrival rate vector $\boldsymbol{\lambda} = (\lambda_n^k)_{k\in\mathcal{K},n\in\mathcal{N}} \in int(\Lambda)$, if there exists*

$\boldsymbol{\epsilon} = (\epsilon_n^k)_{n\in\mathcal{N},k\in\mathcal{K}} \succ \mathbf{0}$ such that $\boldsymbol{\lambda}+\boldsymbol{\epsilon}\in\Lambda$, then the network of VIP queues under Algorithm 1 satisfies

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n\in\mathcal{N},k\in\mathcal{K}}\mathbb{E}[V_n^k(t)] \le \frac{NB}{\epsilon}+\frac{\omega}{2\epsilon}\Psi(\boldsymbol{\lambda}) \quad (15)$$

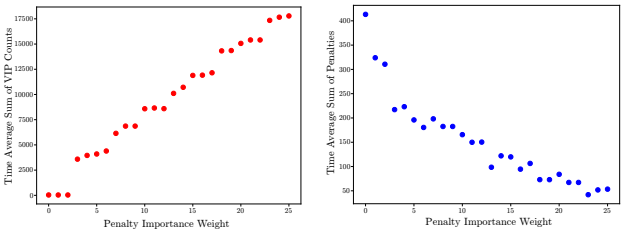
$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[p(t)] \le \frac{2NB}{\omega}+\Psi(\boldsymbol{\lambda}) \quad (16)$$

where $\epsilon \triangleq \min_{n\in\mathcal{N},k\in\mathcal{K}}\epsilon_n^k$, $p(t)=\sum_{k\in\mathcal{K},n\in\mathcal{N},j\in\mathcal{J}_n} p_{n_j}^k(t)$ is the total penalty accumulated across the network at slot t, $\Psi(\boldsymbol{\lambda})$ is the minimum long term average total penalty achievable by any feasible and stabilizing policy [11] and,

$$B \triangleq \frac{1}{2N}\sum_{n\in\mathcal{N}}\Big(\big(\textstyle\sum_{b\in\mathcal{N}}C_{nb}\big)^2+2\big(\sum_{b\in\mathcal{N}}C_{nb}\big)|\mathcal{K}|r_{n_1} + \big(\textstyle\sum_{k\in\mathcal{K}}A_{n,max}^k+\sum_{a\in\mathcal{N}}C_{an}+|\mathcal{K}|r_{n_1}\big)^2\Big) \quad (17)$$

Proof. Please see Appendix B in [12]. □

Theorem 2 demonstrates an achievable trade-off between performance and costs: we can set ω large to prioritize minimizing the bound on penalties in exchange for significantly increasing the bound on queue backlog, and vice-versa. Figure 3 visualizes this dynamic in the virtual plane.



(a) Sum of VIP counts. (b) Sum of penalties.

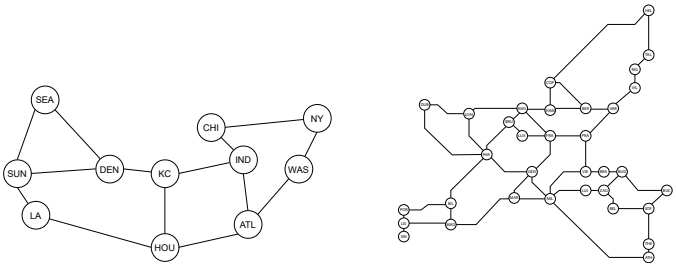
Figure 3: Long-term average of total VIP queue backlog and total penalty in the virtual plane as ω increases (Abilene topology of Figure 4, $|\mathcal{N}|=1000$, $\sum_{k\in\mathcal{K}}\lambda_n^k=10$).

V. Data Plane

In this section, we develop practical policies that operate in the data plane and make forwarding and caching decisions by using Algorithm 1 as a guideline.

Caching. Implementing a caching policy in the data plane based on Algorithm 1 requires certain adjustments. First, unlike in the virtual plane, nodes do not have immediate access to data objects in the data plane. Therefore, the policy must make decisions based on objects available to a given node. Furthermore, in the virtual plane, VIPs of an object are drained via caching immediately after the cache state for that object is set to 1, leading Algorithm 1 to make frequent changes in cache distribution, as was also mentioned in [1]. This oscillatory behavior is not practical in the data plane. Having observed these, we present a practical caching policy in the data plane that is based on the VIP flows established by Algorithm 1.

We adopt the *cache score* metric devised for the stable caching algorithm presented in [1]. The cache score for object



(a) Abilene Network (b) GEANT Network

Figure 4: Network topologies used in experiments.

k at node n at time t is defined as the average number of VIPs for k received by n over a sliding window of T time slots prior to t, expressed as follows.

$$CS_n^k(t) = \frac{1}{T}\sum_{\tau=t-T+1}^{t}\sum_{(a,n)\in\mathcal{L}^k}F_{an}^k(\tau) \quad (18)$$

We then define the data plane *cache benefit* metric, which balances the cache score against utilization costs, as follows.

$$CB_{n_j}^k(t) = \begin{cases} r_{n_j}CS_n^k(t)-\omega c_{n_j}^a, \text{if } j \text{ is empty} \\ r_{n_j}(CS_n^k(t)-CS_n^{k'_{n_j}}(t)) \\ \qquad -\omega(c_{n_j}^a+c_{n_j}^e), \text{otherwise} \end{cases} \quad (19)$$

where $k'_{n_j}=\arg\min_{\{k\in\mathcal{K}_{n_j}(t)\}}CS_n^k(t)$ with $\mathcal{K}_{n_j}(t)$ denoting the set of cached objects in tier j at node n at a given time.

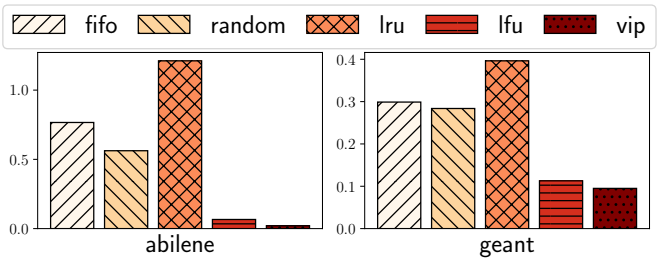
When a data object $k\notin\cup_{j\in\mathcal{J}_n}\mathcal{K}_{n_j}(t)$ arrives at node n during the time interval $[t,t)$, the caching policy first determines the cache tier which offers the highest cache benefit, i.e. $j^*=\arg\max_{\{j\in\mathcal{J}_n\}}CB_{n_j}^k(t)$. If $CB_{n_{j^*}}^k(t)>0$, the object is admitted to tier j^*. If tier j^* is full, object k replaces object k'_{n_j}. We repeat this process as long as there are replacements, treating any replaced object as if it were a new data object arrival to see if it is beneficial to migrate it to a different tier.

Forwarding. We adopt the forwarding strategy of [1], which we describe briefly. A request for object $k\notin K_{n_j}(t)$ that arrives at n is forwarded to node

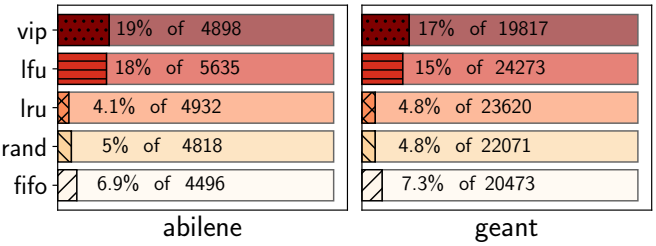
$$b_n^k(t) = \underset{\{b:(n,b)\in\mathcal{L}^k\}}{\arg\max}\frac{1}{T}\sum_{t'=t-T+1}^{t}F_{nb}^k(t) \quad (20)$$

The sliding window is used here as well, to represent the time-average behavior of the periodic VIP forwarding process in the virtual plane. This policy forwards requests on the most profitable links, as established by the flow patterns of VIPs in the virtual plane.

Note that these policies, like our model, are expressed in object level terms. In real data plane implementations, objects consist of several *chunks*. Our policies would employ the following principles at the chunk level. If a data object is admitted to or evicted from a cache tier, all its chunks must be admitted to or evicted from the same tier. The forwarding decision described in (20) is made when a request for the first chunk of object k arrives at node n. Requests for subsequent chunks of k are forwarded to the same node.



(a) Total delay shown as fraction of total delay without any caching.



(b) First tier hits shown as percentage of total hits.

Figure 5: Total delay (a) and cache hit distribution (b) ($\omega = 0$).

VI. Experimental Results

We use the Abilene [17] and GEANT [18] network topologies shown in Figure 4 in our simulation[4] experiments. In both topologies, each link has a capacity of 10 objects per second. All nodes can be content sources, have caches and generate requests. For each object k, a node is selected as the content source $\mathcal{S}(k)$ uniformly at random and independent of all other objects. We assume that node $\mathcal{S}(k)$ can produce copies of k instantaneously. We apply a simple routing scheme that allows each node n to forward requests for object k to any neighboring node that is its next hop on any shortest path (in number of hops) between n and $\mathcal{S}(k)$. For all settings, we consider a total of 1000 data objects. At each requester, requests arrive according to a Poisson process with a rate of $\lambda = 10$. The object requested by each arrival is determined independently of previous arrivals according to a Zipf distribution with parameter 0.75. Requests are generated for the first 100 seconds of the simulation, and a run completes once all requests are fulfilled. We use the same two cache tiers at each node. The faster tier (tier 1) can store 5 objects and produce 20 objects per second. It has an admission cost of 4 and an eviction cost of 2. The slower tier (tier 2) can store 100 objects and produce 10 objects per second, has an admission cost of 2 and an eviction cost of 1.

We run our virtual plane algorithm with a time slot length of 1 second, and use a window size T of 100 slots to compute (18) and (20) for the data plane policies. To establish baselines, we use four cache replacement policies: First In First Out (FIFO), Uniform Random (RAND), Least Recently Used (LRU) and Least Frequently Used (LFU). FIFO, RAND and LRU policies are paired with the LCE admission policy, i.e. they admit all new (not already cached) data object arrivals. For LRU, admissions happen in the first tier, evicting the LRU object if the tier is full; the evicted object can then be migrated to the second tier if there is available capacity there, or the LRU object in the second tier was requested less recently. For FIFO, the two tiers are operated as a single FIFO queue; admissions happen in the first tier and when the object at the front of the first tier is evicted, it is migrated to the second tier. For RAND, one of the tiers is selected uniformly at random to admit the object; if the selected tier is full, an object in the tier is selected uniformly at random to be evicted. We adapt LFU in a less naive manner, by using the method in the practical caching policy we developed in Section V. To do so, in place of the cache score (18), we use the frequency measurement of LFU when computing the cache benefit (19).

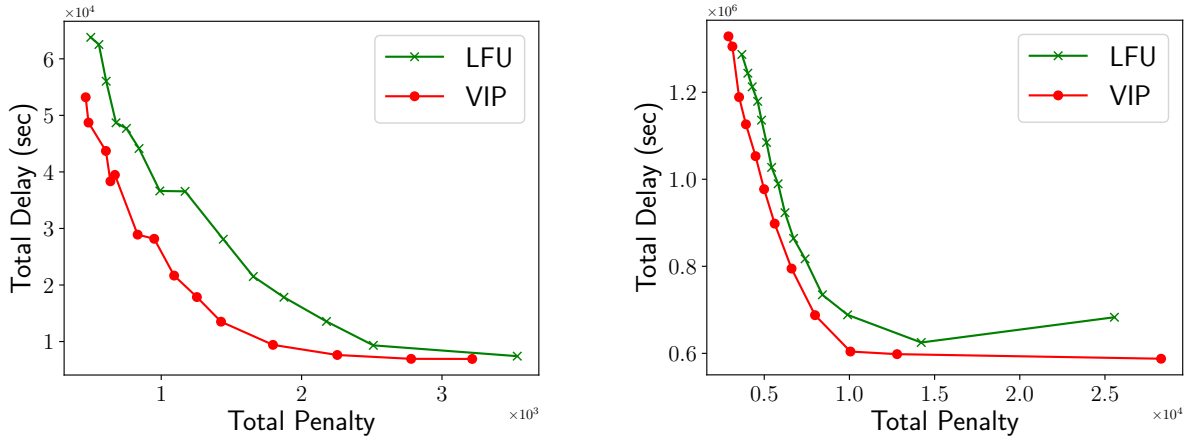
We pair all four of the baseline caching policies described above with a Least Response Time (LRT) forwarding scheme, where nodes keep track of the round trip delay of the last request sent over each outgoing link and choose the link with the smaller delay when more than one link is available to forward a request. For our data plane policy, we use this round trip delay measurement to break ties when (20) yields more than one valid neighbor node.

We first experiment with a simple scenario where penalties are disregarded by setting $\omega = 0$. Results of this scenario can be seen in Figure 5. Figure 5a shows the total delay experienced by all requests generated throughout the simulation run, as a fraction of the delay that would occur without any caching. It can be immediately seen that naive policies perform poorly, even increasing total delay in certain cases. For these policies, the issue is their inability to balance the amount of requests served from cache tiers in accordance with their read rates. Figure 5b, which shows both the total number of cache hits and the distribution of these hits between the two tiers, demonstrates this. Essentially, these policies trade queuing delays at network links with read delays of cache devices. In contrast, we can see that the adapted policies achieve much smaller delay with comparable amounts of total cache hits. In the Abilene topology, which exhibits a more drastic case, the naive LRU policy actually *increases* total delay by 21% compared to the case with no caching at all, while our VIP-based policy reduces delay by 98%.

We now exclude the naive approaches and focus on the performance-cost trade-off by observing our policy in comparison with the adapted LFU. We explore the range of trade-offs for both policies by controlling ω. We illustrate our results in Figure 6 by plotting total delay against the total penalty incurred by all caching decisions throughout the simulation run to directly demonstrate the performance-cost trade-off.

We can observe that our VIP-based policy outperforms LFU in both topologies, with respect to delay achieved for comparable penalty regimes or vice-versa. While the adapted LFU policy uses a similar metric to make caching decisions

[4]Simulation code available at https://github.com/fvmutlu/multi-tier.



(a) Abilene

(b) Geant

Figure 6: Delay vs. penalty curves for our VIP-based policy and the adapted cost-aware LFU policy.

in the data plane, this performance difference between the two arises from the joint caching and forwarding optimization approach in the virtual plane that drives the VIP-based policy, which can manage the larger but slower tier more effectively.

VII. Conclusion

Leveraging advances in storage technology to scale cache capacities is becoming a necessity for high-performance data delivery networks. In this paper, we presented an object-level multi-tiered caching policy that can address the challenges in realizing this transition. We built our policy using the VIP framework which proved to be a suitable tool for this challenge. We showed that our approach yields a stabilizing algorithm which provides minimal upper bounds on both the queue backlogs and cost accumulation across the network, and establishes the trade-off between these two bounds. We demonstrated through simulation experiments that the data plane policy driven by our algorithm in the virtual plane outperforms adapted traditional policies.

References

[1] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "Vip: A framework for joint dynamic forwarding and caching in named data networks," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, 2014, pp. 117–126.

[2] G. Rossini and D. Rossi, "Coupling caching and forwarding: Benefits, analysis, and implementation," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, 2014, pp. 127–136.

[3] G. Rossini, D. Rossi, M. Garetto, and E. Leonardi, "Multi-terabyte and multi-gbps information centric routers," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 181–189.

[4] R. B. Mansilha, L. Saino, M. P. Barcellos, M. Gallo, E. Leonardi, D. Perino, and D. Rossi, "Hierarchical content stores in high-speed icn routers: Emulation and prototype implementation," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, ser. ACM-ICN '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 59–68.

[5] W. So, T. Chung, H. Yuan, D. Oran, and M. Stapp, "Toward terabyte-scale caching with ssd in a named data networking router," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 241–242.

[6] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," in *IEEE International Conference on Performance, Computing, and Communications, 2004*. IEEE, 2004, pp. 445–452.

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12.

[8] S. Shukla and A. A. Abouzeid, "On designing optimal memory damage aware caching policies for content-centric networks," in *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE, 2016, pp. 1–8.

[9] J. Shi, D. Pesavento, and L. Benmohamed, "Ndn-dpdk: Ndn forwarding at 100 gbps on commodity hardware," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 30–40.

[10] Y. Wu, F. V. Mutlu, Y. Liu, E. Yeh, R. Liu, C. Iordache, J. Balcas, H. Newman, R. Sirvinskas, M. Lo, S. Song, J. Cong, L. Zhang, S. Timilsina, S. Shannigrahi, C. Fan, D. Pesavento, J. Shi, and L. Benmohamed, "N-dise: Ndn-based data distribution for large-scale data-intensive science," ser. ICN '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 103–113.

[11] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[12] F. V. Mutlu and E. Yeh, "Cost-aware joint caching and forwarding in networks with heterogeneous cache resources," 2023. [Online]. Available: https://arxiv.org/abs/2310.07243

[13] D. F. Crouse, "On implementing 2d rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016.

[14] A. Volgenant, "Linear and semi-assignment problems: a core oriented approach," *Computers & Operations Research*, vol. 23, no. 10, pp. 917–932, 1996.

[15] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," in *29th IEEE Conference on Decision and Control*. IEEE, 1990, pp. 2130–2132.

[16] M. J. Neely, E. Modiano, and C. E. Rohrs, "Dynamic power allocation and routing for time-varying wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 1, pp. 89–103, 2005.

[17] Internet 2, "Abilene Network," archived from original in 2012. [Online]. Available: https://web.archive.org/web/20120324103518/http://www.internet2.edu/pubs/200502-IS-AN.pdf

[18] GEANT Association, "GÉANT Network - GN4-3N." [Online]. Available: https://network.geant.org/gn4-3n/