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

Faruk Volkan Mutlu
fvmutlu@ece.neu.edu
Northeastern University
Boston, MA, USA

Edmund Yeh
eyeh@ece.neu.edu
Northeastern University
Boston, MA, USA

## ABSTRACT

Caching is crucial for enabling high-throughput networks for data intensive applications. Traditional caching technology relies on DRAM, as it can transfer data at a high rate. However, DRAM capacity is subject to contention by most system components and thus is very limited, implying that DRAM-only caches cannot scale to meet growing demand. Fortunately, persistent memory and flash storage technologies are rapidly evolving and can be utilized alongside DRAM to increase cache capacities. To do so without compromising network performance requires caching techniques adapted to the characteristics of these technologies. In this paper, we model the cache as a collection of storage blocks with different rate parameters and utilization costs. We introduce an optimization technique based on the drift-plus-penalty method and apply it in a framework which enables joint caching and forwarding. We show that it achieves an optimal trade-off between throughput and cache utilization costs in a virtual control plane. We then develop a corresponding practical policy in the data plane. Finally, through simulations in several settings, we demonstrate the superior performance of our proposed approach with respect to total user delay and cache utilization costs.

## 1 INTRODUCTION

One of the core components enabling high-throughput data delivery networks is in-network caching. The main parameters of a cache that impact network performance are its transfer rate and capacity. The traditional cache device is DRAM, which despite its small capacity, can match network link rates. However, as the volume of data grows, the small capacity of DRAM becomes an obstacle in scaling network performance via caching. Fortunately, persistent storage technologies are advancing rapidly and they may soon have transfer rates high enough to justify their use as additional tiers of cache alongside DRAM to enable larger capacities. System designs that consider this *vertical* scaling of caches already exist in the literature, and given the expected performance improvements in upcoming evolutions of the PCIe and NVMe standards, as well as technologies like SPDK [18], wider implementation of routers with large multi-tiered caches is imminent.

Nonetheless, a transfer rate gap between memory and storage will continue to exist for the foreseeable future, even if it narrows over time. Furthermore, while storage technologies today offer large capacities at low upfront costs, utilizing them as short-term caches introduce significant operational costs, as they consume significant amounts of power and their reliability diminishes with use. 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 tend to assume a singular cache capable of operating at the rate of the forwarding path and emphasize the characteristics of data objects rather than caches, making them unsuitable for the task.

In contrast, a *multi-tiered caching policy* maintains the goal of caching a small set of data objects with high demand in the fastest tier of cache, but also aims to collect a larger set of objects with less demand in slower tiers with more capacity. The objects in the slower tiers are requested less often, but there are a larger number of them. Therefore, such a policy aims to balance the frequency of cache hits served by each tier 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 with another one of only slightly more demand. Such replacements are costly, happen more frequently in slower tiers, and bring no significant performance gains in the long term. In this paper, we propose a multi-tiered caching policy that can intelligently utilize a variety of memory and storage elements with different characteristics. The goal of this policy is to find an optimal trade-off between the performance gained via caching and the costs incurred by utilizing caches. The proposed policy is built on the VIP framework [20], which provides powerful design and analysis tools that fit our goals. The contributions of this paper can be summarized as follows:

- An object-level caching model that considers multiple tiers of cache with different 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 growth bound and cache utilization costs, and an exact solution to the optimal cache placement problem posed by this algorithm,
- Practical caching and forwarding policies in the data plane driven by the virtual control plane algorithm and a thorough experimental evaluation of these policies through simulations.

## 2 RELATED WORK

The challenge of optimal data placement in multi-layered storage architectures have long been a focus of research in systems like data centers or cloud storage platforms. In these contexts, vast amounts of data need to be kept in storage for long periods of time. To provide low latency access to popular portions of this data, techniques that use SSDs as caches for HDDs are widely adopted [8]. Novel techniques aimed at modern enterprise storage systems where all storage elements are SSDs (all-flash) have also been developed [19]. Most recently, reinforcement learning was applied to the data placement problem in hybrid storage systems to provide an adaptive and continuously improving solution [13].

The optimal data placement challenge for the in-network caching context may appear similar. However, the crucial difference is that in-network caching places cache devices directly on the forwarding path of requests and data. Therefore, bottlenecks in the caching system lead to compounding network-wide performance issues. As such, in-network caching has much stricter demands from storage systems and lower tolerance for overheads. Still, the feasibility of devices beyond DRAM as caches in a data delivery network context have been explored, primarily for the information-centric networking (ICN) paradigm. While SSDs were initially seen as either too costly or not performant enough for ICN [9], as storage technologies improved, system designs that make use of them for ICN caches appeared. In [10], 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 [5]. Another design was proposed in [14], 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.

However, while such designs address the integration of persistent storage into caching systems, they are not backed by policies designed for multi-tiered caches and the constraints of in-network caching. The lack of such policies is one of the reasons why SSD-based caches have yet to see wider adoption in modern high-performance ICN prototypes. In [11], where a high-throughput software-defined forwarder for Named Data Networking (NDN) was introduced, caching in persistent memory or NVMe 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 [17].

While neither traditional policies nor emerging ones from the literature readily propose extensions to multi-tiered caches, the VIP framework described in [20] stands out as a primary candidate that can support such an extension. Although the algorithm proposed in [20] targets single-tier caches, the framework models the transfer rate of that cache. In Section 4, we describe how we use this fact to extend the framework to multiple tiers.

## 3 SYSTEM MODEL

Let the unit of content in the network be a *data object* (or simply *object* for short) and assume that each object has the same size[1]. 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 be indexed as $j = 1, 2, ...$, i.e. $j \in \mathcal{J}_n = \{1, 2...\}$. 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 $\mathcal{S}(k)$. We refer to this as the *cache exclusivity* assumption. 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, instead of being evicted entirely from the node. To facilitate such migrations, we assume that each node has a *migration buffer* capable of storing one object at a time, where a migrating object is kept temporarily. While this is only required when the destination tier is full, for simplicity we assume that a migrating object is always moved to this buffer first. 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 $j$ and placing it in the migration buffer incurs the cost $c^e_{n_j}$, which includes both the cost of reading from $j$ and that of writing to the buffer.

## 4 CONTROL PLANE

As stated in Section 1, we build our multi-tiered caching policy on the VIP framework [20]. In this framework, user demand for objects is measured using *virtual interest packets* (VIPs), which are tracked in a *virtual control plane* (or simply *virtual plane*). The virtual plane operates separately from the request forwarding and data delivery processes described in Section 3, which take place in what we refer to as the *data plane*. This approach facilitates the development of joint caching and forwarding algorithms in the virtual plane. VIPs are used as the common metric that measures the value of data objects, which allow such algorithms to drive requests toward caching nodes while avoiding congestion in the network. Additionally, this separation reduces the complexity of design and analysis for such algorithms, compared to building them directly in the data plane.

In view of our design goal described in Section 1, the VIP framework provides two additional advantages. First, it embeds the most critical parameter of a cache, the rate at which it can push objects into the network, directly into its analysis of network dynamics in the virtual plane. Secondly, 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 [6].

In this section, we describe the dynamics of VIPs, introduce our virtual plane joint caching and forwarding algorithm and show that this algorithm achieves an optimal trade-off between VIP backlogs and cache utilization costs.

### 4.1 VIP Dynamics

Time in the virtual plane is represented in slots of length 1 indexed as $t = 1, 2, ...$ where time slot $t$ refers to the time interval $[t, t + 1)$. Each node $n \in \mathcal{N}$ keeps a separate counter $V^k_n(t)$ for each object $k \in \mathcal{K}$, which are all set to 0 at the beginning of time slot $t = 1$,

[1]While this assumption simplifies our model and analysis, the approach presented in this paper can be generalized to a model with data objects of unequal size. Section 4.3.3 contains further discussion on this issue.

with $V_n^k(t) = 0$ for all $t \geq 1$ if $n = \mathcal{S}(k)$. These are called *VIP counts*. When a request for object $k$ enters the network at node $n$ in the data plane, the corresponding counter $V_n^k(t)$ is incremented. The number of requests for object $k$ that enter the network at node $n$ during slot $t$ is denoted as $A_n^k(t)$ and the external *arrival rate* of VIPs at node $n$ for object $k$ is $\lambda_n^k \triangleq \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} A_n^k(\tau)$. At each 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 $k$. Note that, in the data plane, the communication of VIPs can be handled with a single message of negligible size at each slot. We define the *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, with $s_{n_j}^k(1) = 0$, $\forall n \in \mathcal{N}, k \in \mathcal{K}$. 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. A tier $j$ at node $n$ has *readout 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 the cache tiers in the set are numbered in descending order of their readout rates, i.e. $r_{n_1} \geq r_{n_2} \geq \cdots$.

## 4.2 Joint Caching and Forwarding Algorithm

We now describe our distributed joint caching and forwarding algorithm for the virtual plane in terms of the definitions above.

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:** *At each node $n \in \mathcal{N}$, choose $s_{n_j}^k(t)$ for each data object $k \in \mathcal{K}$ and cache tier $j \in \mathcal{J}_n$ to*

$$\text{maximize} \quad \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}_n} b_{n_j}^k(t) s_{n_j}^k(t) \tag{1}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{K}} s_{n_j}^k(t) \leq L_{n_j},\ j \in \mathcal{J}_n \tag{2}$$

$$\sum_{j\in\mathcal{J}_n} s_{n_j}^k(t) \leq 1,\ k \in \mathcal{K} \tag{3}$$

$$s_{n_j}^k(t) \in \{0,1\},\ k \in \mathcal{K},\ j \in \mathcal{J}_n \tag{4}$$

*where,*

$$b_{n_j}^k(t) \triangleq \begin{cases} r_{n_j}V_n^k(t) - \omega c_{n_j}^a, & \text{if } s_{n_j}^k(t-1) = 0 \\ r_{n_j}V_n^k(t) + \omega c_{n_j}^e, & \text{if } s_{n_j}^k(t-1) = 1 \end{cases} \tag{5}$$

*and $\omega \geq 0$ is the importance weight of the penalty function defined as*

$$p_{n_j}^k(t) = \begin{cases} c_{n_j}^a, & \text{if } s_{n_j}^k(t) - s_{n_j}^k(t-1) = 1 \\ c_{n_j}^e, & \text{if } s_{n_j}^k(t) - s_{n_j}^k(t-1) = -1 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

**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*

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

*where,*

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

**Queue Evolution:** *Update VIP counts according to*

$$\begin{aligned} V_n^k(t+1) \leq \Bigg( &\Big(V_n^k(t) - \sum_{b\in\mathcal{N}} \mu_{nb}^k(t)\Big)^+ + A_n^k(t) \\ &+ \sum_{a\in\mathcal{N}} \mu_{an}^k(t) - \sum_{j\in\mathcal{J}_n} r_{n_j} s_{n_j}^k(t) \Bigg)^+ \end{aligned} \tag{9}$$

*where $(x)^+ \triangleq \max(x,0)$.*

The caching subproblem of Algorithm 1 defined by (1)–(4) ensures an optimal placement of objects across cache tiers with respect to both the value of objects and the costs of admissions and replacements. The importance weight $\omega$ acts as a control parameter that lets us decide how much we value each side of this trade-off. The penalty function (6) gets its name from the drift-plus-penalty approach and unifies the representation of cache utilization costs. Throughout the rest of the paper, we use the term *penalty* in reference to this function, in order to distinguish it from constants $c_{n_j}^a$ and $c_{n_j}^e$. The last term in (9) represents VIPs removed via caching and lets us model the impact of the rate at which caches can satisfy requests, which directly affects network capacity and performance.

Forwarding according to (7) and (8) applies the backpressure algorithm [15] to the VIP queues and forwards VIPs using links that maximize the queue backlog differential $W_{ab}^k$, which balances the VIP backlog around the network in the virtual plane. This, in turn, translates to balancing user demand for objects, thus enabling the implementation of a data plane forwarding policy that avoids congestion building up in parts of the network.

## 4.3 Trade-off Optimality Analysis

We now provide an optimality analysis for our algorithm in the virtual plane, which consists of two steps: (i) defining the stability region of the VIP queue network and (ii) showing the optimal trade-off between queue backlog reduction and total penalty.

*4.3.1 Stability Region.* As stated previously, 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. However, to show this property of the algorithm, a *stability region* of the network with respect to arrival rates, for which there exists some feasible policy that can make all VIP queues rate stable, must first be defined [6]. We formally define this region with the following theorem.

Theorem 1 (Stability Region). *The VIP stability region $\Lambda$ of the network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, is the set $\Lambda$ consisting of all VIP arrival rates*

*$\boldsymbol{\lambda} = (\lambda_n^k)_{k\in\mathcal{K},n\in\mathcal{N}}$ such that the following holds*

$$\lambda_n^k \le \sum_{b\in\mathcal{N}} f_{nb}^k - \sum_{a\in\mathcal{N}} f_{an}^k + \sum_{j\in\mathcal{J}_n} r_{n_j} \sum_{i=1}^{\sigma} \beta_{n,i} \mathbf{1}_{[(k,j)\in\mathcal{B}_{n,i}]}, \quad \forall n\in\mathcal{N},\ k\in\mathcal{K},\ n\neq\mathcal{S}(k) \tag{10}$$

$$0 \le \beta_{n,i} \le 1, \quad i = 1,\cdots,\sigma, \quad n\in\mathcal{N} \tag{11}$$

$$\sum_{i=1}^{\sigma} \beta_{n,i} = 1, \quad \forall n\in\mathcal{N} \tag{12}$$

*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 $t\ge 1$.*

$$F_{ab}^k(t)\ge 0, F_{nn}^k(t) = 0, F_{\mathcal{S}(k)n}^k(t) = 0,\ \forall a,b,n\in\mathcal{N}, k\in\mathcal{K}, \tag{13}$$

$$F_{ab}^k(t) = 0,\ \forall a,b\in\mathcal{N}, k\in\mathcal{K}, (a,b)\notin\mathcal{L}^k \tag{14}$$

$$\sum_{k\in\mathcal{K}} F_{ab}^k(t) \le C_{ba},\ \forall (a,b)\in\mathcal{L} \tag{15}$$

*$\mathcal{B}_{n,i}$ denotes the $i$-th one among $\sigma$ 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 in the form of $(k,j)$, such that if $(k,j)\in\mathcal{B}_{n,i}$, object $k$ is cached in tier $j$.*

Proof. Please see Appendix A. □

### 4.3.2 *Backlog-Penalty Bound Tradeoff.*

We now show that our algorithm can achieve the optimal trade-off between queue backlog growth 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*

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n\in\mathcal{N},k\in\mathcal{K}}\mathbb{E}[V_n^k(t)] \le \frac{NB}{\epsilon} + \frac{\omega}{2\epsilon}\Psi(\boldsymbol{\lambda}) \tag{16}$$

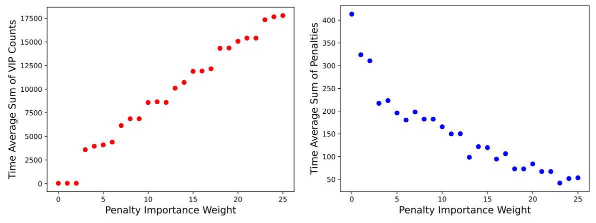
$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[p(t)] \le \frac{2NB}{\omega} + \Psi(\boldsymbol{\lambda}) \tag{17}$$

*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 policy that makes all queues rate stable [6] and,*

$$B \triangleq \frac{1}{2N}\sum_{n\in\mathcal{N}}\Bigg(\big(\textstyle\sum_{b\in\mathcal{N}} C_{nb}\big)^2 + 2\big(\sum_{b\in\mathcal{N}} C_{nb}\big)|\mathcal{K}|r_{n_1} \tag{18}$$

$$+\big(\textstyle\sum_{k\in\mathcal{K}} A_{n,max}^k + \sum_{a\in\mathcal{N}} C_{an} + |\mathcal{K}|r_{n_1}\big)^2\Bigg) \tag{19}$$

Proof. Please see Appendix B. □



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

**Figure 1: Long-term average of total VIP queue backlog and total penalty in the virtual plane (Abilene topology of Figure 2, $|\mathcal{N}| = 1000$, $\sum_{k\in\mathcal{K}}\lambda_n^k = 10$).**

Theorem 2 demonstrates the adjustable trade-off between queue backlogs and penalties: we can set $\omega$ arbitrarily large to push the time average penalty close to the minimum $\Psi(\boldsymbol{\lambda})$ at the cost of significantly increasing the average queue backlog and vice-versa. Figure 1 visualizes this dynamic in the virtual plane.

### 4.3.3 *Optimal Solution to the Caching Subproblem.*

We have shown through Theorem 2 that Algorithm 1 optimizes the backlog-penalty trade-off. However, for Theorem 2 to hold we must solve the caching subproblem defined by (1)–(4) exactly. While this problem appears in the form of a generalized assignment problem (GAP), which is known to be NP-hard, we can show that it is in fact a rectangular assignment problem (RAP), by using a simple transformation of variables, which we present in the following lemma.

Lemma 1. *Let $i$ be an integer in the set of integers $\mathcal{I}_n = \{1,2,...,\sum_{j\in\mathcal{J}_n} L_{n_j}\}$. Let $b_{n_j}^k(t) = b_{n_i}^k(t)$ and $s_{n_j}^k(t) = s_{n_i}^k(t)$ if $i\in\mathcal{I}_{n_j} = \{1+\sum_{\ell=1}^{j-1} L_{n_\ell},...,\sum_{\ell=1}^{j} L_{n_\ell}\}$. Then, a solution to the following problem gives an equivalent solution to the problem defined by (1)–(4).*

$$\text{maximize} \quad \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}_n} b_{n_i}^k(t)s_{n_i}^k(t) \tag{20}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{K}} s_{n_i}^k(t)\le 1,\ i\in\mathcal{I}_n \tag{21}$$

$$\sum_{i\in\mathcal{I}_n} s_{n_i}^k(t)\le 1,\ k\in\mathcal{K} \tag{22}$$

$$s_{n_i}^k(t)\in\{0,1\},\ k\in\mathcal{K},\ i\in\mathcal{I}_n \tag{23}$$

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, (21) is equivalent to (2). Furthermore, since $s_{n_i}^k = 1$ can hold for at most one $i\in\mathcal{I}_n$, (22) is equivalent to (3). □

The problem defined by (20)–(23) has the form of a RAP and can be solved exactly in polynomial time using a modified Jonker-Volgenant algorithm [2, 16]. Since (21) is an inequality, in cases where $b_{n_i}^k(t) < 0$ for some $i\in\mathcal{I}_n$, we can obtain the optimal solution by treating (21) as an equality, then setting $s_{n_i}^k(t) = 0$ for all $i$ such that $b_{n_i}^k(t) < 0$.

It is important to point out that the equivalence of (1)–(4) to (20)–(23) stems from the assumption in our model that all objects have the same size. However, the problem formulated by (1)–(4) can be generalized to a model where objects have different sizes, in which case it would indeed be a GAP. Polynomial time methods that can approximate a solution to the GAP exist [3, 4, 12]. While the results of Theorem 2 would not hold for an approximate solution, it may be possible to show that they would hold for arrival rates in the interior of a subset of $\Lambda$, the bounds of which depend on the factor of approximation.

## 5 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 due to a gap between the assumptions of the virtual plane and the constraints of the data plane. Because we assumed that nodes have immediate access to any data object they decide to cache in the virtual plane, the virtual plane cache state variables $s^k_{n_j}(t)$ may not reflect the actual placement of objects in the data plane at any given time. Furthermore, since VIPs for an object begin to be drained via caching immediately after the cache state for that object is set to 1, Algorithm 1 leads to frequent shifts in cache distribution, as was also mentioned in [20]. While the drift-plus-penalty approach allows us to mitigate this oscillatory behavior, it faces an issue in balancing the constant cost parameters that contribute to penalties against the large swings in VIP counts that can happen within a time slot. Having highlighted this gap, 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 [20]. Let $v^k_{ab}(t) \leq \mu^k_{ab}(t)$ be the actual number of VIPs for object $k$ transmitted over link $(a, b)$ during slot $t$ and let $T$ be the size of a sliding window. The cache score for object $k$ at node $n$ at time $t$ is defined as the average number of VIPs for object $k$ received by node $n$ over a sliding window of $T$ time slots prior to time slot $t$, expressed as follows.

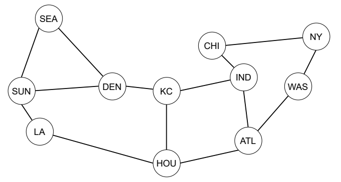
$$CS^k_n(t) = \frac{1}{T} \sum_{\tau=t-T+1}^{t} \sum_{(a,n)\in\mathcal{L}^k} v^k_{an}(\tau) \tag{24}$$

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

$$CB^k_{n_j}(t) = \begin{cases} r_{n_j}(CS^k_n(t) - CS^{k'}_n(t)) - \omega(c^a_{n_j} + c^e_{n_j}), \text{if } j \text{ is full} \\ r_{n_j} CS^k_n(t) - \omega c^a_{n_j}, \text{ otherwise} \end{cases} \tag{25}$$

where $k'_{n_j} = \arg\min_{\{k\in\mathcal{K}_{n_j}\}} CS^k_n(t)$ with $\mathcal{K}_{n_j}$ denoting the set of currently cached objects across all tiers at node $n$ at a given time.

When a data object $k \notin \mathcal{K}_{n_j}$ 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. $j^* = \arg\max_{\{j\in\mathcal{J}_n\}} CB^k_{n_j}(t)$. If $CB^k_{n_{j^*}}(t) > 0$, the object is admitted to tier $j^*$. If tier $j^*$ is full, object $k$ replaces object $k'_{n_j}$ defined above. If $j^*$ has free capacity, no replacements occur. 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 there is benefit to migrating it to a different cache tier.



**(a) Abilene Network**

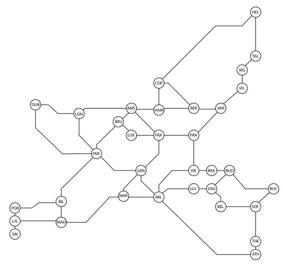

**(b) GEANT Network**

**Figure 2: Network Topologies**

*Forwarding.* Implementing a forwarding policy in the data plane requires a similar, albeit simpler, adjustment to that of caching. Although VIPs in the virtual plane are generated along with requests that arrive in the data plane, they are forwarded in batches periodically at each time slot, whereas each request in the data plane is forwarded as soon as it arrives. Therefore, we need a forwarding policy in the data plane that follows the flow patterns of VIPs in the virtual plane [20]. A request for object $k \notin K_{n_j}$ that arrives at node $n$ is forwarded to node

$$b^k_n(t) = \underset{\{b:(n,b)\in\mathcal{L}^k\}}{\arg\max} \frac{1}{T} \sum_{t'=t-T+1}^{t} v^k_{nb}(t) \tag{26}$$

Here, the sliding window is used once again, this time 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 the policies we describe above are expressed only in object level terms. In an actual implementation, data objects can consist of several *chunks*. Our policies would employ the following principles at the chunk level. If a data object is admitted to a cache tier, all its chunks must be admitted to the same tier. If a data object is evicted from a cache tier, all its chunks must be evicted from that tier. The forwarding decision described in (26) 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.

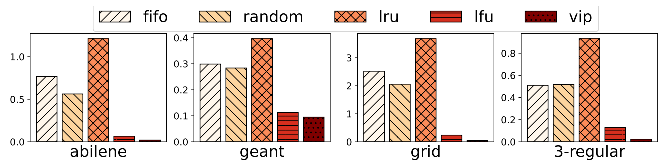
## 6 EXPERIMENTAL RESULTS

We now describe our experiment setting and results obtained for two test scenarios via simulations[2] over this setting.

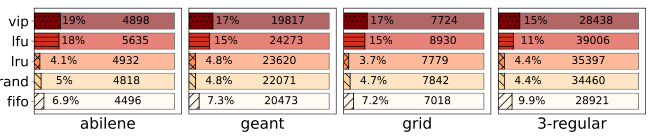
### 6.1 Experiment Setting

We use the following topologies in our simulation experiments: the Abilene and Geant GN4-3N [1] network graphs shown in Figure 2, a two dimensional 4×4 grid, and a 3-regular graph sampled uniformly. For all topologies, we assume that all nodes can be content sources, have caches and generate requests. For each data 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 enough copies of $k$ at any given time to satisfy all requests it receives for it. We apply a simple routing scheme on all topologies

[2]The code for our simulations is available at https://github.com/fvmutlu/multi-tier.



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



(b) Cache hits, percentage of hits in the first tier on the left, total number of hits on the right

**Figure 3: Total delay and cache hits obtained by various policies across different topologies ($\lambda = 10$, $L_{n_2} = 100$, $\omega = 0$).**

that allows each node $n$ to forward requests for object $k$ to any neighboring node that is its next hop on a shortest path (in number of hops) between $n$ and $\mathcal{S}(k)$. In all topologies, each link has a capacity to transmit 10 data objects per second. 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. For all experiments, 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: a fast but small tier with high utilization costs and a slower but larger tier with lower costs. The fast tier can store 5 objects and produce 20 objects per second. It has an admission cost of 4 and an eviction cost of 2. The slow tier can produce 10 objects per second, has an admission cost of 2 and an eviction cost of 1. Its capacity is left as a variable parameter for our experiments. In addition to these parameters, we also define a write rate for each tier equal to its read rate. This parameter is not modeled in our approach, as our design aims to extract many cache hits for each cache admission and avoid frequent low-benefit replacements, making the write rate a negligible factor. We include this parameter in the simulations to incorporate the impact it could have in a real deployment. Note that the choices of numerical values stated here are not meant to replicate real network and cache device specifications, but rather to establish a simple object level setting that can capture the use case for our model and policies.

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 (24) and (26) for the data plane policies. To establish baselines, we use four cache replacement policies: First In First Out (FIFO), Least Recently Used (LRU), Uniform Random (RAND) and Least Frequently Used (LFU). The first three replacement policies are adapted to the multi-tier model in "naive" ways, i.e. they do not account for performance and cost parameters of caches, and are implicitly paired with the Leave Copy Everywhere (LCE) admission policy, i.e. they admit all new (not already cached) data object arrivals:

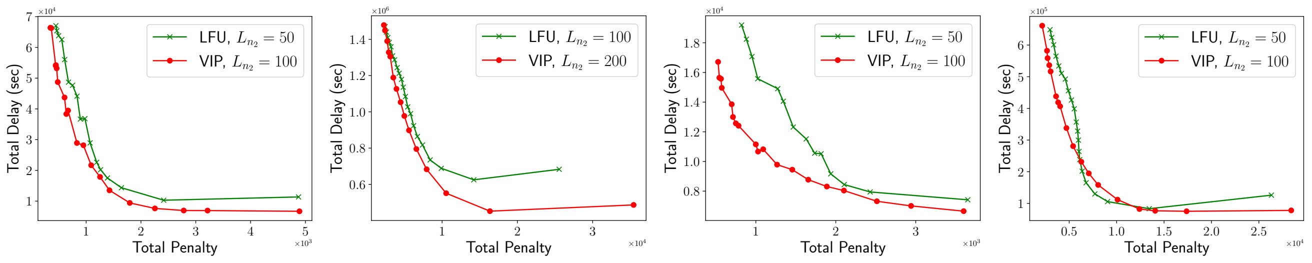
- LRU: An arriving data object is admitted to the first cache tier and the LRU object in the tier is evicted if the tier is full. When an object is evicted from the first tier, it can be migrated to the second tier if there is available capacity there, or the LRU object in the second tier was requested less recently. An object evicted from the second tier cannot be migrated to the first tier and is evicted entirely from the node.
- FIFO: Both caches are operated as a single FIFO queue. When the object at the front of the first tier is evicted, it is migrated to the second tier. If the second tier is also full this causes the object at the front of the second tier to be evicted entirely from the node.
- RAND: A tier is selected to admit the arriving data object uniformly at random. If the selected tier is full, a random object from that tier is evicted entirely from the node.

We adapt LFU using the method in the practical caching policy we developed in Section 5. To do so, in place of the cache score (24), we use the frequency measurement of LFU when computing the cache benefit (25). We use an "ideal" LFU implementation which keeps a counter that tracks the number of accesses for all objects in the network. This multi-tier and cost-aware adaptation of LFU operates similarly to our caching policy in the data plane. As a result, when observing the performance differences between the two policies, we will be able to more directly evaluate the impact of having the virtual plane optimal algorithm drive decisions in the data plane.

We pair all four of the baseline caching policies described above with a Least Response Time (LRT) forwarding scheme, in which 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 (26) yields more than one valid neighbor node.

## 6.2 Results

We start with a simple scenario where penalties are disregarded by setting $\omega = 0$ and the capacity of the second tier is fixed at 100 (denoted as $L_{n_2} = 100$). We present the results of this fixed-parameter scenario in Figure 3. Figure 3a shows the total delay (the sum of delays experienced by all requests generated throughout the simulation run) achieved by the policies we test, as a fraction of the delay that would occur without any caching. It can be immediately seen that naive policies are performing poorly, even increasing the delay beyond that would be experienced without caching in certain cases. When we observe Figure 3b, which shows both the total number of cache hits and the distribution of these hits between the two tiers, we get a clearer picture. For these policies, the issue arises not from a lack of cache hits, but instead from an inability to balance the amount of requests served from each of the tiers. Furthermore, they perform many low-benefit replacements which exacerbates their poor performance. We can see this effect by observing the random strategy, which performs at most one replacement per data object arrival: while it has similar cache hit metrics with LRU, it achieves less delay in all topologies. Essentially, such naive policies trade queueing delays at network links with queueing delays at



**(a) Abilene** **(b) Geant** **(c) Grid** **(d) 3-Regular**

**Figure 4: Delay vs. penalty (tier 2 capacities for each policy listed in the legends)**

cache device controllers. In contrast, we can see that the adapted policies achieve much smaller delay with comparable amounts of total cache hits. In the grid topology, which exhibits the most drastic difference, the naive LRU policy increases total delay by 267% (in comparison to no caching), while our VIP-based policy reduces delay by 95%, which is a significant improvement even in comparison to the adapted LFU policy, which reduces delay by 76%.

We now exclude the naive approaches and instead focus on the performance of our policy and the adapted LFU which, as we projected earlier and as demonstrated in Figure 3, has been performing closely to our policy. We explore the range of trade-offs for both policies by controlling $\omega$. We illustrate our results in Figure 4 by plotting total delay against total penalty (the sum of penalties incurred by all caching decisions throughout the simulation run) to directly demonstrate the performance-cost trade-off. Note that different tier 2 capacities are used for the two policies, as we've experimented with different values of this parameter and picked the value yielding the best curve for each policy.

We can observe that our VIP-based policy outperforms LFU in all topologies except for a small region in the 3-regular topology. While the adapted LFU policy behaves similarly to the VIP-based 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 take advantage of a larger tier 2 cache to obtain a better trade-off.

## 7 CONCLUSION AND FUTURE WORK

Leveraging recent advances in memory and storage technologies 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 an existing framework that provides powerful design tools for this challenge. We showed that our approach yields an optimal algorithm for the performance-cost trade-off in the virtual plane of this framework. We demonstrated through simulation experiments that the data plane policy driven by our algorithm in the virtual plane outperforms both traditional policies and an ideal LFU implementation that is adapted using the same principles as our data plane policy.

We conclude our work by underlining two potential future extensions. Firstly, given the data delivery mechanics of our data plane model, our policy can run into issues where an object that has been previously cached in a slower tier sees significant increase in demand, but cannot be moved to a faster tier since requests for it are not forwarded upstream. To work around this, we could devise *proactive migration* schemes for the data plane, where objects can be moved between cache tiers independently of request arrival and data delivery processes. An example of a practical approach is to periodically poll the placement of objects across all tiers and carry out the most beneficial migrations that can be completed in a reasonable time frame. Secondly, the assumption in the virtual plane that the readout rate of a cache tier is independent for each object does not translate well to the data plane since cache devices have a shared bandwidth. A way to amend this gap is to express the readout rates as dynamic parameters which are tied together by a total rate constraint. While the optimal selection of these dynamic parameters is an added problem dimension, it can be approached in the virtual plane using the backpressure-based tools provided by the framework.

## REFERENCES

[1] GEANT Association. 2022. GN4-3N. Retrieved March 10, 2023 from https://network.geant.org/gn4-3n/

[2] David F Crouse. 2016. On implementing 2D rectangular assignment algorithms. *IEEE Trans. Aerospace Electron. Systems* 52, 4 (2016), 1679–1696.

[3] Uriel Feige and Jan Vondrák. 2006. Approximation algorithms for allocation problems: Improving the factor of 1-1/e. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 667–676.

[4] Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. 2006. Tight approximation algorithms for maximum general assignment problems. In *SODA*, Vol. 6. Citeseer, 611–620.

[5] Rodrigo B. Mansilha, Lorenzo Saino, Marinho P. Barcellos, Massimo Gallo, Emilio Leonardi, Diego Perino, and Dario Rossi. 2015. Hierarchical Content Stores in High-Speed ICN Routers: Emulation and Prototype Implementation. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking* (San Francisco, California, USA) *(ACM-ICN '15)*. Association for Computing Machinery, New York, NY, USA, 59–68. https://doi.org/10.1145/2810156.2810159

[6] Michael J Neely. 2010. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks* 3, 1 (2010), 1–211.

[7] M. J. Neely, E. Modiano, and C. E. Rohrs. 2005. Dynamic power allocation and routing for time-varying wireless networks. *IEEE Journal on Selected Areas in Communications* 23, 1 (2005), 89–103.

[8] Junpeng Niu, Jun Xu, and Lihua Xie. 2018. Hybrid storage systems: a survey of architectures and algorithms. *IEEE Access* 6 (2018), 13385–13406.

[9] Diego Perino and Matteo Varvello. 2011. A reality check for content centric networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. 44–49.

[10] G. Rossini, D. Rossi, M. Garetto, and E. Leonardi. 2014. Multi-Terabyte and multi-Gbps information centric routers. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 181–189. https://doi.org/10.1109/INFOCOM.2014.6847938

[11] Junxiao Shi, Davide Pesavento, and Lotfi Benmohamed. 2020. NDN-DPDK: NDN forwarding at 100 Gbps on commodity hardware. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*. 30–40.

[12] David B Shmoys and Éva Tardos. 1993. An approximation algorithm for the generalized assignment problem. *Mathematical programming* 62, 1-3 (1993), 461–474.

[13] Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gómez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu. 2022. Sibyl: Adaptive and extensible data placement in hybrid storage systems using online reinforcement learning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 320–336.

[14] Won So, Taejoong Chung, Haowei Yuan, David Oran, and Mark Stapp. 2014. 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* (Los Angeles, California, USA) *(ANCS '14)*. Association for Computing Machinery, New York, NY, USA, 241–242. https://doi.org/10.1145/2658260.2661767

[15] Leandros Tassiulas and Anthony Ephremides. 1990. 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, 2130–2132.

[16] A Volgenant. 1996. Linear and semi-assignment problems: a core oriented approach. *Computers & Operations Research* 23, 10 (1996), 917–932.

[17] Yuanhao Wu, Faruk Volkan Mutlu, Yuezhou Liu, Edmund Yeh, Ran Liu, Catalin Iordache, Justas Balcas, Harvey Newman, Raimondas Sirvinskas, Michael Lo, Sichen Song, Jason Cong, Lixia Zhang, Sankalpa Timilsina, Susmit Shannigrahi, Chengyu Fan, Davide Pesavento, Junxiao Shi, and Lotfi Benmohamed. 2022. N-DISE: NDN-Based Data Distribution for Large-Scale Data-Intensive Science *(ICN '22)*. Association for Computing Machinery, New York, NY, USA, 103–113. https://doi.org/10.1145/3517212.3558087

[18] Ziye Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E Paul. 2017. SPDK: A development kit to build high performance storage applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 154–161.

[19] Zhengyu Yang, Morteza Hoseinzadeh, Allen Andrews, Clay Mayers, David Thomas Evans, Rory Thomas Bolt, Janki Bhimani, Ningfang Mi, and Steven Swanson. 2017. AutoTiering: automatic data placement manager in multi-tier all-flash datacenter. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[20] Edmund Yeh, Tracey Ho, Ying Cui, Michael Burd, Ran Liu, and Derek Leong. 2014. Vip: A framework for joint dynamic forwarding and caching in named data networks. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*. 117–126.

## A PROOF OF THEOREM 1

Define the following for $n \in \mathcal{N}$, $j \in \mathcal{J}_n$ and $i = 1, \cdots, \sigma$.

$$\mathcal{T}_{n,j,i} = \left\{ \tau \in \{1, \cdots, \tilde{t}\} : \begin{array}{ll} s^k_{n_j}(\tau) = 1 & \forall (k,j) \in \mathcal{B}_{n,i} \\ s^k_{n_j}(\tau) = 0 & \forall (k,j) \notin \mathcal{B}_{n,i} \end{array} \right\}$$

$$\mathcal{T}_{n,i} = \left\{ \tau \in \{1, \cdots, \tilde{t}\} : \begin{array}{ll} \sum_{j \in \mathcal{J}_n} s^k_{n_j}(\tau) = 1 & \forall k : \exists (k,j) \in \mathcal{B}_{n,i} \\ \sum_{j \in \mathcal{J}_n} s^k_{n_j}(\tau) = 0 & \forall k : \forall (k,j) \notin \mathcal{B}_{n,i} \end{array} \right\}$$

Note that, due to the cache exclusivity assumption, $\sum_{j \in \mathcal{J}_n} |\mathcal{T}_{n,j,i}| = |\mathcal{T}_{n,i}|$. Then, defining $\beta_{n,i} = |\mathcal{T}_{n,i}|/\tilde{t}$, we can prove (11) and (12).

Based on the evolution dynamics of VIP queues, we have that

$$V^k_n(t) \geq \sum_{\tau=1}^{t} A^k_n(\tau) + \sum_{\tau=1}^{t} \sum_{a \in \mathcal{N}} F^k_{an}(\tau) - \sum_{\tau=1}^{t} \sum_{b \in \mathcal{N}} F^k_{nb}(\tau) - \sum_{j \in \mathcal{J}_n} r_{n_j} s^k_{n_j}(\tau) \tag{27}$$

Network stability, as defined in Lemma 1 of [7], implies that there exists a finite $M$ such that $V^k_n(t) \leq M$, $\forall n \in \mathcal{N}, k \in \mathcal{K}$ holds infinitely often. Given an arbitrarily small value $\epsilon > 0$, there exists a slot $\tilde{t}$ such that

$$V^k_n(\tilde{t}) \leq M, \quad \frac{M}{\tilde{t}} \leq \epsilon, \quad \left| \frac{\sum_{\tau=1}^{\tilde{t}} A^k_n(\tau)}{\tilde{t}} - \lambda^k_n \right| \leq \epsilon \tag{28}$$

Then, by (28) and (27) we have

$$\begin{aligned} \lambda^k_n - \epsilon \;\leq\; \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} A^k_n(\tau) \;\leq\; & \frac{1}{\tilde{t}} V^k_n(\tilde{t}) + \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_{b \in \mathcal{N}} F^k_{nb}(\tau) \\ & - \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_{a \in \mathcal{N}} F^k_{an}(\tau) + \sum_{j \in \mathcal{J}_n} r_{n_j} \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} s^k_{n_j}(\tau) \end{aligned} \tag{29}$$

Since $\sum_{\tau=1}^{\tilde{t}} s^k_{n_j}(\tau) = \sum_{i=1}^{\sigma} |\mathcal{T}_{n,j,i}| \mathbf{1}_{[(k,j) \in \mathcal{B}_{n,i}]}$, by (29), we have

$$\lambda^k_n \;\leq\; \sum_{b \in \mathcal{N}} f^k_{nb} - \sum_{a \in \mathcal{N}} f^k_{an} + \sum_{j \in \mathcal{J}_n} r_{n_j} \sum_{i=1}^{\sigma} \beta_{n,i} \mathbf{1}_{[(k,j) \in \mathcal{B}_{n,i}]} + 2\epsilon \tag{30}$$

Thus, by letting $\epsilon \longrightarrow 0$, we can prove (10), which defines the boundaries of the VIP network stability region $\Lambda$.

Now, we will further show that $\boldsymbol{\lambda} \in int(\Lambda)$ is sufficient for stability, i.e. there exists a feasible policy that can stabilize all queues given $\boldsymbol{\lambda} \in int(\Lambda)$. A feasible policy is one that, at each time slot $t$, yields a VIP forwarding allocation vector $(\mu^k_{ab}(t))_{k \in \mathcal{K}, (a,b) \in \mathcal{L}}$ and cache state vector $(s^k_{n_j}(t))_{k \in \mathcal{K}, n \in \mathcal{N}, j \in \mathcal{J}_n}$, which satisfy the following.

$$\sum_{k \in \mathcal{K}} \mu^k_{ab}(t) \leq C_{ab}, \text{ for all } (a,b) \in \mathcal{L} \tag{31}$$

$$\mu^k_{ab}(t) = 0, \text{ for all } (a,b) \notin \mathcal{L}^k \tag{32}$$

$$\sum_{k \in \mathcal{K}} s^k_{n_j}(t) \leq L_{n_j}, \; j \in \mathcal{J}_n \tag{33}$$

$$\sum_{j \in \mathcal{J}_n} s^k_{n_j}(t) \leq 1, \; k \in \mathcal{K} \tag{34}$$

$\boldsymbol{\lambda} \in int(\Lambda)$ implies that there exists $\boldsymbol{\epsilon} = (\epsilon^k_n)_{n \in \mathcal{N}, k \in \mathcal{K}}$, where $\epsilon^k_n > 0$, such that $\boldsymbol{\lambda} + \boldsymbol{\epsilon} \in \Lambda$. Let $(f^k_{ab})_{(a,b) \in \mathcal{L}, k \in \mathcal{K}}$ and $\beta_{n,i}$ now denote the flow and cache placement set frequency variables associated with $\boldsymbol{\lambda} + \boldsymbol{\epsilon}$. Then, the following holds.

$$\lambda^k_n + \epsilon^k_n \leq \sum_{b \in \mathcal{N}} f^k_{nb} - \sum_{a \in \mathcal{N}} f^k_{an} + \sum_{j \in \mathcal{J}_n} r_{n_j} \sum_{i=1}^{\sigma} \beta_{n,i} \mathbf{1}_{[(k,j) \in \mathcal{B}_{n,i}]}, \quad \forall n \in \mathcal{N}, k \in \mathcal{K}, n \neq \mathcal{S}(k) \tag{35}$$

We now construct a random forwarding policy. For every link $(a,b) \in \mathcal{L}$ such that $\sum_{k \in \mathcal{K}} f^k_{ab} > 0$, this policy transmits the VIPs of the single object $\tilde{k}_{ab}$ which is chosen randomly to be $k$ with probability $f^k_{ab} / \sum_{k \in \mathcal{K}} f^k_{ab}$. Then, the number of VIPs that can be transmitted in slot $t$ is as follows:

$$\tilde{\mu}^k_{ab}(t) = \begin{cases} \sum_{k \in \mathcal{K}} f^k_{ab}, & \text{if } k = \tilde{k}_{ab} \\ 0, & \text{otherwise} \end{cases} \tag{36}$$

For every link $(a,b) \in \mathcal{L}$ such that $\sum_{k \in \mathcal{K}} f^k_{ab} = 0$, the policy sets $\tilde{\mu}^k_{ab}(t) = 0$ for all $k \in \mathcal{K}$. Thus, we have

$$\mathbb{E}[\tilde{\mu}^k_{ab}(t)] = f^k_{ab} \tag{37}$$

Next, we construct a randomized caching policy. For every node $n$, this policy caches the distribution $\tilde{\mathcal{B}}_n$, where $\tilde{\mathcal{B}}_n$ is chosen randomly to be $\mathcal{B}_{n,i}$ with probability $\beta_{n,i}$. Then, the caching state of $k$ for tier $j$ during slot $t$ is as follows:

$$\tilde{s}_{n_j}^k(t) = \begin{cases} 1, & \text{if } (k,j) \in \tilde{\mathcal{B}}_n \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

Thus, we have

$$\begin{aligned} \mathbb{E}\Big[\sum_{j\in\mathcal{J}_n} r_{n_j}\tilde{s}_{n_j}^k(t)\Big] &= \sum_{j\in\mathcal{J}_n} r_{n_j}\mathbb{E}[\tilde{s}_{n_j}^k(t)] \\ &= \sum_{j\in\mathcal{J}_n} r_{n_j}\sum_{i=1}^{\sigma}\beta_{n,i}\mathbf{1}_{[(k,j)\in\mathcal{B}_{n,i}]} \end{aligned} \tag{39}$$

Then, by (35), (37) and (39), we have

$$\begin{aligned} \lambda_n^k + \epsilon_n^k &\le \mathbb{E}\Big[\Big(\sum_{b\in\mathcal{N}}\tilde{\mu}_{nb}^k(t) - \sum_{a\in\mathcal{N}}\tilde{\mu}_{an}^k(t) + \sum_{j\in\mathcal{J}_n} r_{n_j}\tilde{s}_{n_j}^k(t)\Big)\Big] \\ &= \sum_{b\in\mathcal{N}} f_{nb}^k - \sum_{a\in\mathcal{N}} f_{an}^k + \sum_{j\in\mathcal{J}_n} r_{n_j}\sum_{i=1}^{\sigma}\beta_{n,i}\mathbf{1}_{[(k,j)\in\mathcal{B}_{n,i}]} \end{aligned} \tag{40}$$

Equation (40) shows that the service rate is greater than the arrival rate for a randomized feasible policy given $\boldsymbol{\lambda} \in int(\Lambda)$, which proves that this condition is sufficient for stability. □

## B PROOF OF THEOREM 2

Define the Lyapunov function as

$$\mathcal{L}(\mathbf{V}(t)) = \sum_{n\in\mathcal{N},k\in\mathcal{K}} (V_n^k(t))^2 \tag{41}$$

Then the Lyapunov drift at slot $t$ is given by

$$\Delta(\mathbf{V}(t)) = \mathbb{E}[\mathcal{L}(\mathbf{V}(t+1)) - \mathcal{L}(\mathbf{V}(t))|\mathbf{V}(t)] \tag{42}$$

Taking the square of both sides of (9), we have the following

$$\begin{aligned} (V_n^k(t+1))^2 &\le (V_n^k(t))^2 + \Big(\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\Big)^2 - 2V_n^k(t)\sum_{b\in\mathcal{N}}\mu_{nb}^k(t) \\ &+ 2V_n^k(t)\Big(A_n^k(t) + \sum_{a\in\mathcal{N}}\mu_{an}^k(t)\Big) \\ &- 2\Big(V_n^k(t) - \sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\Big)^+\Big(\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\Big) \\ &+ \Big(A_n^k(t) + \sum_{a\in\mathcal{N}}\mu_{an}^k(t) - \sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\Big)^2 \end{aligned}$$

The terms can be collected as follows

$$\begin{aligned} &(V_n^k(t+1))^2 - (V_n^k(t))^2 \\ &\le \Big(\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\Big)^2 + 2\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t) \\ &+ \Big(A_n^k(t) + \sum_{a\in\mathcal{N}}\mu_{an}^k(t) + \sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\Big)^2 + 2V_n^k(t)A_n^k(t) \\ &- 2V_n^k(t)\Big(\sum_{b\in\mathcal{N}}\mu_{nb}^k(t) - \sum_{a\in\mathcal{N}}\mu_{an}^k(t)\Big) - 2V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t) \end{aligned}$$

We can then take the sum over $n \in \mathcal{N}, k \in \mathcal{K}$ to obtain

$$\begin{aligned} &\mathcal{L}(\mathbf{V}(t+1) - \mathcal{L}(\mathbf{V}(t)) \\ &\le \sum_{n\in\mathcal{N},k\in\mathcal{K}}\Big(\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\Big)^2 + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}}\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t) \\ &+ 2\sum_{n\in\mathcal{N},k\in\mathcal{K}}\Big(A_n^k(t) + \sum_{a\in\mathcal{N}}\mu_{an}^k(t) + \sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\Big)^2 \\ &+ 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)A_n^k(t) - 2\sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}}\mu_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big) \\ &- 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t) \end{aligned}$$

We now define the following constants: $\mu_{n,max}^{out} \triangleq \sum_{b\in\mathcal{N}} C_{nb}$, $\mu_{n,max}^{in} \triangleq \sum_{a\in\mathcal{N}} C_{an}$, $A_{n,max} \triangleq \sum_{k\in\mathcal{K}} A_{n,max}^k$, $r_{n,max} \triangleq Kr_{n_1}$. Based on these definitions, the following hold.

$$\sum_{k\in\mathcal{K}}\Big(\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\Big)^2 \le (\mu_{n,max}^{out})^2, \tag{43}$$

$$\begin{aligned} &\sum_{k\in\mathcal{K}}\Big(A_n^k(t) + \sum_{a\in\mathcal{N}}\mu_{an}^k(t) + \sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\Big)^2 \\ &\le (A_{n,max} + \mu_{n,max}^{in} + r_{n,max})^2, \end{aligned} \tag{44}$$

$$\sum_{k\in\mathcal{K}}\sum_{b\in\mathcal{N}}\mu_{nb}^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t) \le \mu_{n,max}^{out}r_{n,max} \tag{45}$$

Then, by (43)-(45) and the definition of $B$ we have

$$\begin{aligned} \mathcal{L}(\mathbf{V}(t+1) - \mathcal{L}(\mathbf{V}(t)) &\le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)A_n^k(t) \\ &- 2\sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}}\mu_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big) \\ &- 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t) \end{aligned} \tag{46}$$

Taking conditional expectations on both sides of (46), we have

$$\begin{aligned} \Delta(\mathbf{V}(t)) &\le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\lambda_n^k(t) \\ &- 2\mathbb{E}\Big[\sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}}\mu_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big)\Big|\mathbf{V}(t)\Big] \\ &- 2\mathbb{E}\Big[\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\Big|\mathbf{V}(t)\Big] \end{aligned} \tag{47}$$

Now, let $\boldsymbol{s}(t) = (s_{n_j}^k(t))_{k\in\mathcal{K},n\in\mathcal{N},j\in\mathcal{J}_n}$ and $\hat{p}(\boldsymbol{s}(t)) = p(t)$, then add the penalty function to both sides of (47) to get the drift-plus-penalty inequality.

$$
\begin{aligned}
&\Delta(\mathbf{V}(t)) + \omega\mathbb{E}[p(t)|\mathbf{V}(t)] \le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\lambda_n^k(t)\\
&- 2\mathbb{E}\left[\sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}} \mu_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big)\middle|\mathbf{V}(t)\right]\\
&- 2\mathbb{E}\left[\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\middle|\mathbf{V}(t)\right]\\
&+ \omega\mathbb{E}[\hat{p}(\boldsymbol{s}(t))|\mathbf{V}(t)]
\end{aligned}
\tag{48}
$$

Now, let $\tilde{\mu}_{ab}^k(t)$ and $\tilde{s}_{n_j}^k(t)$ refer to all feasible VIP forwarding allocation rates and cache state variables respectively. Then, given $\mu_{ab}^k(t)$ and $s_{n_j}^k(t)$ obtained via Algorithm 1, the following holds.

$$
\begin{aligned}
&\Delta(\mathbf{V}(t)) + \omega\mathbb{E}[p(t)|\mathbf{V}(t)] \le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\lambda_n^k(t)\\
&- 2\mathbb{E}\left[\sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}} \mu_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big)\middle|\mathbf{V}(t)\right]\\
&- 2\mathbb{E}\left[\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}s_{n_j}^k(t)\middle|\mathbf{V}(t)\right]\\
&+ \omega\mathbb{E}[\hat{p}(\boldsymbol{s}(t))|\mathbf{V}(t)]\\
&\le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\lambda_n^k(t)\\
&- 2\mathbb{E}\left[\sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}} \tilde{\mu}_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big)\middle|\mathbf{V}(t)\right]\\
&- 2\mathbb{E}\left[\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}\tilde{s}_{n_j}^k(t)\middle|\mathbf{V}(t)\right]\\
&+ \omega\mathbb{E}[\hat{p}(\tilde{\mathbf{s}}(t))|\mathbf{V}(t)]
\end{aligned}
\tag{49}
$$

This is because Algorithm 1 maximizes the following two quantities over all feasible $\tilde{s}_{n_j}^k(t)$ and $\tilde{\mu}_{ab}^k(t)$, by (1)–(4) and by (7)–(8) respectively.

$$
\begin{aligned}
&\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\bigg(\sum_{b\in\mathcal{N}}\tilde{\mu}_{nb}^k(t) - \sum_{a\in\mathcal{N}}\tilde{\mu}_{an}^k(t)\bigg)\\
&= \sum_{(a,b)\in\mathcal{L},k\in\mathcal{K}} \tilde{\mu}_{ab}^k(t)\Big(V_a^k(t) - V_b^k(t)\Big),
\end{aligned}
$$

$$
\bigg(\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\sum_{j\in\mathcal{J}_n} r_{n_j}\tilde{s}_{n_j}^k(t)\bigg) - \omega\hat{p}_{n_j}^k(\tilde{s}_{n_j}^k(t))
$$

Then, using (40), we have

$$
\begin{aligned}
&\Delta(\mathbf{V}(t)) + \omega\mathbb{E}[p(t)|\mathbf{V}(t)]\\
&\le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\lambda_n^k(t) - 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\\
&\times \mathbf{E}\left[\bigg(\sum_{b\in\mathcal{N}}\tilde{\mu}_{nb}^k(t) - \sum_{a\in\mathcal{N}}\tilde{\mu}_{an}^k(t) + \sum_{j\in\mathcal{J}_n} r_{n_j}\tilde{s}_{n_j}^k(t)\bigg)\middle|\mathbf{V}(t)\right]\\
&+ \omega\mathbb{E}[\hat{p}(\tilde{\mathbf{s}}(t))|\mathbf{V}(t)]\\
&\le 2NB + 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)\lambda_n^k(t)\\
&- 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t)(\lambda_n^k + \epsilon_n^k) + \omega\mathbb{E}[\hat{p}(\tilde{\mathbf{s}}(t))|\mathbf{V}(t)]\\
&\le 2NB - 2\sum_{n\in\mathcal{N},k\in\mathcal{K}} \epsilon_n^k V_n^k(t) + \omega\mathbb{E}[\hat{p}(\tilde{\mathbf{s}}(t))|\mathbf{V}(t)]\\
&\le 2NB - 2\epsilon\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t) + \omega\mathbb{E}[\hat{p}(\tilde{\mathbf{s}}(t))|\mathbf{V}(t)]
\end{aligned}
$$

Now, assume that, for a given arrival rate vector $\boldsymbol{\lambda}$, at time $t$, the policy that achieves $\Psi(\boldsymbol{\lambda})$ has the cache state variable vector $\boldsymbol{s}^*(t)$ and $\mathbb{E}[\hat{p}(\boldsymbol{s}^*(t))] = p^*(t)$. Therefore, we have

$$
\begin{aligned}
&\Delta(\mathbf{V}(t)) + \omega\mathbb{E}[p(t)|\mathbf{V}(t)]\\
&\le 2NB - 2\epsilon\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t) + \omega\mathbb{E}[\hat{p}(\boldsymbol{s}^*(t))|\mathbf{V}(t)]\\
&= 2NB - 2\epsilon\sum_{n\in\mathcal{N},k\in\mathcal{K}} V_n^k(t) + \omega p^*(t)
\end{aligned}
\tag{50}
$$

Using (42) and the law of iterated expectations we have

$$
\begin{aligned}
&\mathbb{E}[\Delta(\mathbf{V}(t))] + \omega\mathbb{E}[\mathbb{E}[p(t)|\mathbf{V}(t)]]\\
&= \mathbb{E}[\mathbb{E}[\mathcal{L}(\mathbf{V}(t+1)) - \mathcal{L}(\mathbf{V}(t))|\mathbf{V}(t)]] + \omega\mathbb{E}[\mathbb{E}[p(t)|\mathbf{V}(t)]]\\
&= \mathbb{E}[\mathcal{L}(\mathbf{V}(t+1))] - \mathbb{E}[\mathcal{L}(\mathbf{V}(t))] + \omega\mathbb{E}[p(t)]\\
&\le 2NB - 2\epsilon\sum_{n\in\mathcal{N},k\in\mathcal{K}} \mathbb{E}[V_n^k(t)] + \omega\Psi(\boldsymbol{\lambda})
\end{aligned}
\tag{51}
$$

Summing both sides over $t \in \{1, 2, ..., T\}$ for some positive integer T, we obtain the following.

$$
\begin{aligned}
&\mathbb{E}[\mathcal{L}(\mathbf{V}(T+1))] - \mathbb{E}[\mathcal{L}(\mathbf{V}(1))] + \omega\sum_{t=1}^{T}\mathbb{E}[p(t)]\\
&\le 2NBT - 2\epsilon\sum_{t=1}^{T}\sum_{n\in\mathcal{N},k\in\mathcal{K}}\mathbb{E}[V_n^k(t)] + \omega T\Psi(\boldsymbol{\lambda})
\end{aligned}
\tag{52}
$$

Note that the values of $V_n^k(t)$, $\mathcal{L}(\mathbf{V}(t))$ and $p(t)$ for any given $k$, $n$, and $t$ are all non-negative. Recall also that $\epsilon > 0$ and $\omega \ge 0$. Therefore, we can arrange the above into two inequalities:

$$
\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[p(t)] \le \frac{2NB}{\omega} + \frac{\mathbb{E}[\mathcal{L}(\mathbf{V}(1))]}{\omega T} + \Psi(\boldsymbol{\lambda})\\
&\frac{1}{T}\sum_{t=1}^{T}\sum_{n\in\mathcal{N},k\in\mathcal{K}}\mathbb{E}[V_n^k(t)] \le \frac{NB}{\epsilon} + \frac{\mathbb{E}[\mathcal{L}(\mathbf{V}(1))]}{2\epsilon T} + \frac{\omega}{2\epsilon}\Psi(\boldsymbol{\lambda})
\end{aligned}
\tag{53}
$$

If we take the limit as $T \to \infty$ for both inequalities, we reach the two final bounds.

$$
\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[p(t)] \le \frac{2NB}{\omega} + \Psi(\boldsymbol{\lambda}) \tag{54}
$$

$$
\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\sum_{n\in\mathcal{N},k\in\mathcal{K}}\mathbb{E}[V_n^k(t)] \le \frac{NB}{\epsilon} + \frac{\omega}{2\epsilon}\Psi(\boldsymbol{\lambda}) \tag{55}
$$

□