VIP: A Framework for Joint Dynamic Forwarding and Caching in Named Data Networks

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

Tracey Ho[†]
California Inst. of Technology
Pasadena, CA, USA
tho@caltech.edu

Ying Cui
MIT
Cambridge, MA, USA
yingcui@mit.edu

Michael Burd
California Inst. of Technology
Pasadena, CA, USA
burdmi@gmail.com

Ran Liu
Northeastern University
Boston, MA, USA
liu.ran1@husky.neu.edu

Derek Leong
Inst. for Infocomm Research
Singapore
dleong@i2r.a-star.edu.sg

ABSTRACT

Emerging information-centric networking architectures seek to optimally utilize both bandwidth and storage for efficient content distribution. This highlights the need for joint design of traffic engineering and caching strategies, in order to optimize network performance in view of both current traffic loads and future traffic demands. We present a systematic framework for joint dynamic interest request forwarding and dynamic cache placement and eviction, within the context of the Named Data Networking (NDN) architecture. The framework employs a virtual control plane which operates on the user demand rate for data objects in the network, and an actual plane which handles Interest Packets and Data Packets. We develop distributed algorithms within the virtual plane to achieve network load balancing through dynamic forwarding and caching, thereby maximizing the user demand rate that the NDN network can satisfy. Numerical experiments within a number of network settings demonstrate the superior performance of the resulting algorithms for the actual plane in terms of low user delay and high rate of cache hits.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—network communications

[*]E. Yeh gratefully acknowledges support from the National Science Foundation Future Internet Architecture grant CNS-1205562 and a Cisco Systems research grant.

[†]T. Ho gratefully acknowledges support from the Air Force Office of Scientific Research grant FA9550-10-1-0166.

General Terms

Theory, Design, Management

Keywords

Named data networking; content centric networking; information centric networking; forwarding; caching; routing

1. INTRODUCTION

Emerging information-centric networking (ICN) architectures are currently changing the landscape of network research. In particular, Named data networking (NDN) [1], or content-centric networking (CCN)[2], is a proposed network architecture for the Internet that replaces the traditional client-server model of communications with one based on the identity of data or content. This abstraction more accurately reflects how the Internet is primarily used today: instead of being concerned about communicating with specific nodes, end users are mainly interested in obtaining the data they want. The NDN architecture offers a number of important advantages in decreasing network congestion and delays, and in enhancing network performance in dynamic, intermittent, and unreliable mobile wireless environments [1].

Content delivery in NDN is accomplished using two types of packets, and specific data structures in nodes. Communication is initiated by the data consumer or requester. To receive data, the requester sends out an *Interest Packet*, which carries the (hierarchically structured) name of the desired data (e.g. `/northeastern/videos/WidgetA.mpg/1`). The Interest Packet is forwarded by looking up the data name in the *Forwarding Information Base (FIB)* at each router the Interest Packet traverses, along routes determined by a name-based routing protocol. The FIB tells the router to which neighbor node(s) to transmit each Interest Packet. Each router maintains a *Pending Interest Table (PIT)*, which records all Interest Packets currently awaiting matching data. Each PIT entry contains the name of the interest and the set of node interfaces from which the Interest Packets for the same name arrived. When multiple interests for the same name are received, only the first is sent toward the data source. When a node receives an interest that it can fulfill with matching data, it creates a *Data Packet* containing the data name, the data content, together with a signature by the producer's key. The Data Packet follows in reverse the path taken by the corresponding Interest Packet, as recorded by the PIT state at each router traversed. When the Data Packet arrives at a router, the router locates the matching PIT entry, transmits the data on all interfaces listed in the PIT entry, and then removes the PIT entry. The router may optionally cache a copy of the received Data Packet in its local *Content Store*, in order to satisfy possible future requests. Consequently, a request for a data object can be fulfilled not only by the content source but also by any node with a copy of that object in its cache [1].

Assuming the prevalence of caches, the usual approaches for forwarding and caching may no longer be effective for ICN architectures such as NDN. Instead, these architectures seek to optimally utilize both bandwidth and storage for efficient content distribution. This highlights the need for joint design of traffic engineering and caching strategies, in order to optimize network performance in view of both current traffic loads and future traffic demands. Unlike many existing works on centralized algorithms for static caching, our goal is to develop distributed, dynamic algorithms that can address caching and forwarding under changing content, user demands, and network conditions.

To address this fundamental problem, we introduce the *VIP framework* for the design of high performing NDN networks. The VIP framework relies on the new metric of *Virtual Interest Packets* (VIPs), which captures the measured demand for the respective data objects in the network. The central idea of the VIP framework is to employ a *virtual* control plane which operates on VIPs, and an *actual* plane which handles Interest Packets and Data Packets. Within the virtual plane, we develop distributed control algorithms operating on VIPs, aimed at yielding desirable performance in terms of network metrics of concern. The flow rates and queue lengths of the VIPs resulting from the control algorithm in the virtual plane are then used to specify the forwarding and caching policies in the actual plane.

The general VIP framework allows for a large class of control and optimization algorithms operating on VIPs in the virtual plane, as well as a large class of mappings which use the VIP flow rates and queue lengths from the virtual plane to specify forwarding and caching in the actual plane. Thus, the VIP framework presents a powerful paradigm for designing efficient NDN-based networks with different properties and trade-offs. In order to illustrate the utility of the VIP framework, we present two particular instantiations of the framework. The first instantiation consists of a distributed forwarding and caching policy in the virtual plane which achieves effective load balancing and adaptively maximizes the throughput of VIPs, thereby maximizing the user demand rate for data objects satisfied by the NDN network. The second instantiation consists of distributed algorithms which achieves not only load balancing but also stable caching configurations. Experimental results show that the latter set of algorithms have superior performance in terms of low user delay and high rate of cache hits, relative to several baseline routing and caching policies.

We begin with a formal description of the network model in Section 2, and discuss the VIP framework in Section 3. We present two instantiations of the VIP framework in Sections 4 and 5. The performance of the proposed forwarding and caching policies is numerically evaluated in comparison with several baseline routing and caching policies using simulations in Section 5.3.

Although there is now a rapidly growing literature in information centric networking, the problem of optimal joint forwarding and caching for content-oriented networks remains open. In [3], a potential-based forwarding scheme with random caching is proposed for ICNs. A simple heuristically defined measure (called potential value) is introduced for each node. A content source or caching node has the lowest potential and the potential value of a node increases with its distance to the content source or caching node. Potential-based forwarding guides Interest Packets from the requester toward the corresponding content source or caching node. As the Data Packet travels on the reverse path, one node on the path is randomly selected as a new caching node. The results in [3] are heuristic in the sense that it remains unknown how to guarantee good performance by choosing proper potential values. In [4], the authors consider one-hop routing and caching in a content distribution network (CDN) setting. Throughput-optimal one-hop routing and caching are proposed to support the maximum number of requests. Given the simple switch topology, however, routing is reduced to cache node selection. Throughput-optimal caching and routing in multi-hop networks remains an open problem. In [5], the authors consider single-path routing and caching to minimize link utilization for a general multi-hop content-oriented network, using primal-dual decomposition within a flow model. Here, it is assumed that the path between any two nodes is predetermined. Thus, routing design reduces to cache node selection [5]. The benefits of selective caching based on the concept of betweenness centrality, relative to ubiquitous caching, are shown in [6]. Cooperative caching within ICNs has been investigated in [7], where an age-based caching scheme is proposed. These proposed cooperative caching schemes have been heuristically designed, and have not been jointly optimized with forwarding strategies. Finally, adaptive multipath forwarding in NDN has been examined in [8], but has not been jointly optimized with caching strategies.

2. NETWORK MODEL

Consider a connected multi-hop (wireline) network modeled by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ and $\mathcal{L}$ denote the sets of N nodes and L directed links, respectively. Assume that $(b, a) \in \mathcal{L}$ whenever $(a, b) \in \mathcal{L}$. Let $C_{ab} > 0$ be the transmission capacity (in bits/second) of link $(a, b) \in \mathcal{L}$. Let L_n be the cache size (in bits) at node $n \in \mathcal{N}$ (L_n can be zero).

Assume that content in the network are identified as *data objects*, with the object identifiers determined by an appropriate level within the hierarchical naming structure. These identifiers may arise naturally from the application, and are determined in part by the amount of control state that the network is able to maintain. Each data object (e.g. `/northeastern/videos/WidgetA.mpg`) consists of a sequence of *data chunks* (e.g. `/northeastern/videos/WidgetA.mpg/1`). We assume that any data object is demarcated by a *starting chunk* and an *ending chunk*. Content delivery in NDN operates at the level of data chunks. That is, each Interest Packet requests a particular data chunk, and a matching Data Packet consists of the requested data chunk, the data chunk name, and a signature. A request for a data object consists of a sequence of Interest Packets which request all the data chunks of the object, where the sequence starts with the Interest Packet requesting the starting chunk, and ends with the Interest Packet requesting the ending chunk.[1] In the VIP framework which we introduce below, distributed control algorithms are developed in a virtual control plane operating at the data object level, while forwarding of Interest Packets and caching of Data Packets in the actual plane operate at the data chunk level.

We will operate our forwarding and caching algorithms over a set $\mathcal{K}$ of K data objects in the network. As mentioned above, $\mathcal{K}$ may be determined by the amount of control state that the network is able to maintain. Since the data object popularity distribution evolves at a relatively slow time scale compared to the caching and forwarding, one approach is to let $\mathcal{K}$ include the set of the most popular data objects in the network, which is typically responsible for most of the network congestion.[2] For simplicity, we assume that all data objects have the same size z (in bits). The results in the paper can be extended to the more general case where object sizes differ. We consider the scenario where $L_n < Kz$ for all $n \in \mathcal{N}$. Thus, no node can cache all data objects.

For each data object $k \in \mathcal{K}$, assume that there is a unique node $src(k) \in \mathcal{N}$ which serves as the content source for the object. Interest Packets for chunks of a given data object can enter the network at any node, and exit the network upon being satisfied by a matching Data Packet at the content source for the object, or at the nodes which decide to cache the object. For convenience, we assume that the content sources are fixed, while the caching points may vary in time.

Assume that routing (topology discovery and data reachability) has already been accomplished in the network, so that the FIBs have been populated for the various data objects. Upon the arrival of an Interest Packet at an NDN node, the following sequence of events happen. First, the node checks its Content Store (CS) to see if the requested data object chunk is locally cached. If it is, then the Interest Packet is satisfied locally, and a Data Packet containing a copy of the data object chunk is sent on the reverse path. If not, the node checks its PIT to see if an Interest Packet requesting the same data object chunk has already been forwarded. If so, the new Interest Packet (interest, for short) is suppressed while the incoming interface associated with the new interest is added to the PIT. Otherwise, the node checks the FIB to see to what node(s) the interest can be forwarded, and chooses a subset of those nodes for forwarding the interest. Next, we focus on Data Packets. Upon receiving a Data Packet, a node needs to determine whether to make a copy of the Data Packet and cache the copy or not. Clearly, policies for the forwarding of Interest Packets and the caching of Data Packets are of central importance in the NDN architecture. Thus far, the design of the strategy layer for NDN remains largely unspecified. Moreover, in the current CCN implementation, a Data Packet is cached at every node on the reverse path. This, however, may not be possible or desirable when cache space is limited.

We shall focus on the problem of finding dynamic forwarding and caching policies which exhibit superior performance in terms of metrics such as the total number of data object requests satisfied (i.e., all corresponding Data Packets are received by the requesting node), the delay in satisfying Interest Packets, and cache hit rates. We propose a VIP framework to solve this problem, as described in the next section.

[1]The data chunks in between the starting and ending chunks can be requested in any order.

[2]The less popular data objects not in $\mathcal{K}$ may be distributed using simple techniques such as shortest-path forwarding with little or no caching.

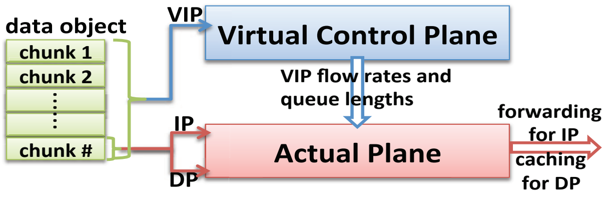

Figure 1: VIP framework. IP (DP) stands for Interest Packet (Data Packet).

3. VIRTUAL INTEREST PACKETS AND THE VIP FRAMEWORK

The VIP framework for joint dynamic forwarding and caching relies on the essential new metric of *virtual interest packets* (VIPs), which are generated as follows. As illustrated in Figure 1, for each request for data object $k \in \mathcal{K}$ entering the network, a corresponding VIP for object $k \in \mathcal{K}$ is generated.[3] The VIPs capture the *measured demand* for the respective data objects in the network, and represent content popularity which is empirically measured, rather than being given a priori. Specifically, the VIP count for a data object in a given part of the network represents the *local* level of interest in the data object, as determined by network topology and user demand.

The VIP framework employs a *virtual* control plane which operates on VIPs *at the data object level*, and an *actual* plane which handles Interest Packets and Data Packets *at the data chunk level*. This design has two motivations. First, this approach reduces the implementation complexity of the VIP algorithm in the virtual plane considerably (as compared with operating on data chunks in the virtual plane). Second, as shown in Section 4.2 below, this approach leads to a desirable implementation which forwards all the Interest Packets for the same ongoing request for a data object on the same path, and which caches the entire data object (consisting of all data chunks) at a caching node (as opposed to caching different chunks of the same data object at different nodes). At the same time, the approach also allows Interest Packets for non-overlapping requests for the same data object to be forwarded on different paths, thus making multi-path forwarding of object requests possible.[4]

Within the virtual plane, we develop distributed control algorithms operating on VIPs, aimed at yielding desirable performance in terms of network metrics of concern. The

[3]More generally, VIPs can be generated at a rate proportional to that of the corresponding data object requests, which can in some cases improve the convergence speed of the proposed algorithms.

[4]In principle, the VIP algorithm in the virtual plane can be applied at the chunk level (corresponding to the case where there is only one chunk in each data object). In this case, the virtual and actual planes operate at the same granularity. On the other hand, the complexity of implementing the algorithm in the virtual plane would be much larger.

flow rates and queue lengths of the VIPs resulting from the control algorithm in the virtual plane are then used to specify the forwarding and caching policies in the actual plane (see Figure 1). A key insight here is that control algorithms operating in the virtual plane can take advantage of local information on network demand (as represented by the VIP counts), which is unavailable in the actual plane due to interest collapsing and suppression.

In order to illustrate the utility of the VIP framework, we present two particular instantiations of the framework in Sections 4 and 5. For both instantiations, the following hold. First, the VIP count is used as the common metric for enabling both the distributed forwarding and distributed caching algorithms in the virtual and actual control planes. Second, the forwarding strategy in the virtual plane achieves load balancing through the application of the backpressure algorithm [9] to the VIP queue state. Finally, one caching algorithm determines the caching locations and cache replacement policy for both the virtual and actual planes. The two instantiations differ in the manner in which they use the VIP count to determine caching actions.

3.1 VIP Dynamics

We now specify the dynamics of the VIPs within the virtual plane. Consider time slots of length 1 (without loss of generality) indexed by $t = 1, 2, \ldots$. Specifically, time slot t refers to the time interval $[t, t+1)$. Within the virtual plane, each node $n \in \mathcal{N}$ maintains a separate VIP queue for each data object $k \in \mathcal{K}$. Note that no data is contained in these VIPs. Thus, the VIP queue size for each node n and data object k at the beginning of slot t (i.e., at time t) is represented by a *counter* $V_n^k(t)$.[5] Initially, all VIP counters are set to 0, i.e., $V_n^k(1) = 0$. As VIPs are created along with data object requests, the counters for the corresponding data object are incremented accordingly at the entry nodes. After being forwarded through the network (in the virtual plane), the VIPs for object k are removed at the content source $src(k)$, and at nodes that have cached object k. That is, the content source and the caching nodes are the *sinks* for the VIPs. Physically, the VIP count can be interpreted as a *potential*. For any data object, there is a downward "gradient" from entry points of the data object requests to the content source and caching nodes.

An exogenous request for data object k is considered to have arrived at node n if the Interest Packet requesting the starting chunk of data object k has arrived at node n. Let $A_n^k(t)$ be the number of exogenous data object request arrivals at node n for object k during slot t (i.e., over the time interval $[t, t+1)$).[6] For every arriving request for data object k at node n, a corresponding VIP for object k is generated at n ($V_n^k(t)$ incremented by 1).[7] The long-term exogenous VIP arrival rate at node n for object k is $\lambda_n^k \triangleq \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} A_n^k(\tau)$.

[5]We assume that VIPs can be quantified as a real number. This is reasonable when the VIP counts are large.

[6]We think of a node n as a point of aggregation which combines many network users. While a single user may request a given data object only once, an aggregation point is likely to submit many requests for a given data object over time.

[7]For the general case where object sizes differ, $V_n^k(t)$ is incremented by the object size z_k for every arriving request for object k.

Let $\mu_{ab}^k(t) \geq 0$ be the allocated transmission rate of VIPs for data object k over link (a, b) during time slot t. Note that at each time t and for each object k, a single message between node a and node b can summarize all the VIP transmissions during that time slot.

In the virtual plane, we assume that at each time t, each node $n \in \mathcal{N}$ can gain access to any data object $k \in \mathcal{K}$ for which there is interest at n, and potentially cache the object locally. Let $s_n^k(t) \in \{0, 1\}$ represent the caching state for object k at node n during slot t, where $s_n^k(t) = 1$ if object k is cached at node n during slot t, and $s_n^k(t) = 0$ otherwise. Now note that even if $s_n^k(t) = 1$, the content store at node n can satisfy only a limited number of VIPs during one time slot. This is because there is a maximum rate r_n (in objects per slot) at which node n can produce copies of cached object k.[8]

The time evolution of the VIP count at node n for object k is as follows:

$$V_n^k(t+1) \leq \left(\left(V_n^k(t) - \sum_{b\in\mathcal{N}} \mu_{nb}^k(t) \right)^+ + A_n^k(t) + \sum_{a\in\mathcal{N}} \mu_{an}^k(t) - r_n s_n^k(t) \right)^+ \tag{1}$$

where $(x)^+ \triangleq \max(x, 0)$. Furthermore, $V_n^k(t) = 0$ for all $t \geq 1$ if $n = src(k)$.

From (1), it can be seen that the VIPs for data object k at node n at the beginning of slot t are transmitted during slot t at the rate $\sum_{b\in\mathcal{N}} \mu_{nb}^k(t)$. The remaining VIPs $(V_n^k(t) - \sum_{b\in\mathcal{N}} \mu_{nb}^k(t))^+$, as well as the exogenous and endogenous VIP arrivals during slot t, are reduced by r_n at the end of slot t if object k is cached at node n in slot t ($s_n^k(t) = 1$). The VIPs still remaining are then transmitted during the next slot $t+1$. Note that (1) is an inequality because the actual number of VIPs for object k arriving to node n during slot t may be less than $\sum_{a\in\mathcal{N}} \mu_{an}^k(t)$ if the neighboring nodes have little or no VIPs of object k to transmit.

4. THROUGHPUT OPTIMAL VIP CONTROL

In this section, we describe an instantiation of the VIP framework in which the VIP count is used as a common metric for enabling both the distributed forwarding and distributed caching algorithms in the virtual and actual control planes. The forwarding strategy within the virtual plane is given by the application of the backpressure algorithm [9] to the VIP queue state. Note that while the backpressure algorithm has been used for routing in conventional source-destination-based networks, its use for forwarding in ICNs appears for the first time in this paper. Furthermore, backpressure forwarding is being used in the virtual plane rather than in the actual plane, where interest collapsing and suppression make the application of the algorithm impractical.

The caching strategy is given by the solution of a max-weight problem involving the VIP queue length. The VIP flow rates and queue lengths are then used to specify forwarding and caching strategies in the actual plane, which handles Interest Packets and Data Packets. We show that

[8]The maximum rate r_n may reflect the I/O rate of the storage disk. Since it is assumed that all data objects have the same length, it is also assumed that the maximum rate r_n is the same for all data objects.

the joint distributed forwarding and caching strategy adaptively maximizes the throughput of VIPs, thereby maximizing the user demand rate for data objects satisfied by the network.

We now describe the joint forwarding and caching algorithm for VIPs in the virtual control plane.

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

***Forwarding**: For each data object $k \in \mathcal{K}$ and each link $(a,b) \in \mathcal{L}^k$, choose*

$$\mu_{ab}^k(t) = \begin{cases} C_{ba}/z, & W_{ab}^*(t) > 0 \text{ and } k = k_{ab}^*(t) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where

$$W_{ab}^k(t) \triangleq V_a^k(t) - V_b^k(t), \tag{3}$$
$$k_{ab}^*(t) \triangleq \arg \max_{\{k:(a,b)\in\mathcal{L}^k\}} W_{ab}^k(t),$$
$$W_{ab}^*(t) \triangleq \left(W_{ab}^{k_{ab}^*(t)}(t)\right)^+.$$

Here, $\mathcal{L}^k$ is the set of links which are allowed to transmit the VIPs of object k, $W_{ab}^k(t)$ is the backpressure weight of object k on link (a,b) at time t, and $k_{ab}^*(t)$ is the data object which maximizes the backpressure weight on link (a,b) at time t.

***Caching**: At each node $n \in \mathcal{N}$, choose $\{s_n^k(t)\}$ to*

$$\text{maximize} \sum_{k\in\mathcal{K}} V_n^k(t) s_n^k \quad \text{subject to} \sum_{k\in\mathcal{K}} s_n^k \leq L_n/z \tag{4}$$

Based on the forwarding and caching in (2) and (4), the VIP count is updated according to (1).

At each time t and for each link (a,b), backpressure-based forwarding algorithm allocates the entire normalized "reverse" link capacity C_{ba}/z to transmit the VIPs for the data object $k_{ab}^*(t)$ which maximizes the VIP queue difference $W_{ab}^k(t)$ in (3). Backpressure forwarding maximally balances out the VIP counts, and therefore the demand for data objects in the network, thereby minimizing the probability of demand building up in any one part of the network and causing congestion.

The caching strategy is given by the optimal solution to the max-weight knapsack problem in (4), which can be solved optimally in a greedy manner as follows. For each $n \in \mathcal{N}$, let $(k_1, k_2, \ldots, k_K)$ be a permutation of $(1, 2, \ldots, K)$ such that $V_n^{k_1}(t) \geq V_n^{k_2}(t) \geq \cdots \geq V_n^{k_K}(t)$. Let $i_n = \lfloor L_n/z \rfloor$. Then for each $n \in \mathcal{N}$, choose

$$s_n^k(t) = \begin{cases} 1, & k \in \{k_1, \cdots, k_{i_n}\} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Thus, the objects with the highest VIP counts (the highest local popularity) are cached.

It is important to note that both the backpressure-based forwarding algorithm and the max-weight caching algorithm are *distributed*. To implement the forwarding algorithm, each node must exchange its VIP queue state with only its neighbors. The implementation of the caching algorithm is local once the updated VIP queue state has been obtained.

To characterize the implementation complexity of Algorithm 1, we note that both the computational and communication complexity of the back pressure forwarding algorithm per time slot is $O(N^2K)$, where the bound can be improved to $O(NDK)$ if D is the maximum node degree in the network. Assuming fixed cache sizes, the computational complexity of the caching algorithm per time slot can be found to be $O(NK)$.

In the following section, we show that the forwarding and caching strategy described in Algorithm 1 is *throughput optimal* within the virtual plane, in the sense of maximizing the throughput of VIPs in the network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ with appropriate transmission rate constraints.

4.1 Maximizing VIP Throughput

We now show that Algorithm 1 adaptively maximizes the throughput of VIPs in the network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ with appropriate transmission rate constraints. In the following, we assume that (i) the VIP arrival processes $\{A_n^k(t); t = 1, 2, \ldots\}$ are mutually independent with respect to n and k; (ii) for all $n \in \mathcal{N}$ and $k \in \mathcal{K}$, $\{A_n^k(t); t = 1, 2, \ldots\}$ are i.i.d. with respect to t; and (iii) for all n and k, $A_n^k(t) \leq A_{n,\max}^k$ for all t.

To determine the constraints on the VIP transmission rates $\mu_{ab}^k(t)$, we note that Data Packets for the requested data object must travel on the reverse path taken by the Interest Packets. Thus, in determining the transmission of the VIPs, we take into account the link capacities on the reverse path as follows:

$$\sum_{k\in\mathcal{K}} \mu_{ab}^k(t) \leq C_{ba}/z, \text{ for all } (a,b) \in \mathcal{L} \tag{6}$$
$$\mu_{ab}^k(t) = 0, \text{ for all } (a,b) \notin \mathcal{L}^k \tag{7}$$

where C_{ba} is the capacity of "reverse" link (b,a).

4.1.1 VIP Stability Region

To present the throughput optimality argument, we first define the VIP stability region. The VIP queue at node n is *stable* if

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} 1_{[V_n^k(\tau)>\xi]} d\tau \to 0 \text{ as } \xi \to \infty,$$

where $1_{\{\cdot\}}$ is the indicator function. The *VIP network stability region* Λ is the closure of the set of all VIP arrival rates $(\lambda_n^k)_{k\in\mathcal{K},n\in\mathcal{N}}$ for which there exists some feasible joint forwarding and caching policy which can guarantee that all VIP queues are stable. By feasible, we mean that at each time t, the policy specifies a forwarding rate vector $(\mu_{ab}^k(t))_{k\in\mathcal{K},(a,b)\in\mathcal{L}}$ satisfying (6)-(7), and a caching vector $(s_n^k(t))_{k\in\mathcal{K},n\in\mathcal{N}}$ satisfying the cache size limits $(L_n)_{n\in\mathcal{N}}$.

Theorem 2 in the Appendix precisely characterizes the VIP stability region in the virtual plane. To our knowledge, Theorem 2 is the first instance where the effect of caching has been fully incorporated into the stability region of a multi-hop network.

4.1.2 Throughput Optimality

By definition, if the VIP arrival rates $\boldsymbol{\lambda} = (\lambda_n^k)_{k\in\mathcal{K},n\in\mathcal{N}} \in \text{int}(\Lambda)$, then all VIP queues can be stabilized. In general, however, this may require knowing the value of $\boldsymbol{\lambda}$. In reality, $\boldsymbol{\lambda}$ can be learned only over time, and may be time-varying. Moreover, stabilizing the network given an arbitrary VIP arrival rate in the interior of Λ may require (time sharing among) multiple forwarding and caching policies.

We now show that the joint forwarding and caching policy in Algorithm 1 adaptively stabilizes all VIP queues in the network for any $\boldsymbol{\lambda} \in \text{int}(\Lambda)$, without knowing $\boldsymbol{\lambda}$. Thus, the policy is *throughput optimal*, in the sense of adaptively maximizing the VIP throughput, and therefore the user demand rate satisfied by the network.

THEOREM 1 (THROUGHPUT OPTIMALITY). *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*

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} \sum_{n\in\mathcal{N},k\in\mathcal{K}} \mathbb{E}[V_n^k(\tau)] \leq \frac{NB}{\epsilon} \tag{8}$$

where $B \triangleq \frac{1}{2N}\sum_{n\in\mathcal{N}}\big((\mu_{n,\max}^{out})^2+(A_{n,\max}+\mu_{n,\max}^{in}+r_{n,\max})^2+2\mu_{n,\max}^{out}r_{n,\max}\big)$, $\epsilon \triangleq \min_{n\in\mathcal{N},k\in\mathcal{K}}\epsilon_n^k$, $\mu_{n,\max}^{in} \triangleq \sum_{a\in\mathcal{N}} C_{an}/z$, $\mu_{n,\max}^{out} \triangleq \sum_{b\in\mathcal{N}} C_{nb}/z$, $A_{n,\max} \triangleq \sum_{k\in\mathcal{K}} A_{n,\max}^k$, and $r_{n,\max} = Kr_n$.

PROOF. Please refer to [10]. □

The forwarding and caching policy in Algorithm 1 achieves throughput optimality in the virtual plane by exploiting both the bandwidth and storage resources of the network to maximally balance out the VIP load (or the demand for data objects in the network), thereby preventing the buildup of congestion. Equivalently, Algorithm 1 is throughput optimal in the actual plane when Interest Packets are not collapsed or suppressed. Note that Theorem 1 can be seen as the multi-hop generalization of the throughput optimal result in [4].

4.2 Forwarding and Caching in the Actual Plane

We now focus on the development of forwarding and caching policies for the actual plane, based on the throughput optimal policies of Algorithm 1 for the virtual plane. Forwarding and caching in the actual plane take advantage of the exploration in the virtual plane to forward Interest Packets on profitable routes and cache Data Packets at profitable node locations.

4.2.1 Forwarding of Interest Packets

The forwarding of Interest Packets in the actual plane follows the pattern established by the VIPs under Algorithm 1 in the virtual plane. For a given window size T, let

$$\bar{\nu}_{ab}^k(t) = \frac{1}{T}\sum_{t'=t-T+1}^{t} \nu_{ab}^k(t') \tag{9}$$

be the average number of VIPs for object k *transmitted* over link (a,b) over a sliding window of size T under Algorithm 1 prior to time slot t.[9]

Forwarding: At any node $n \in \mathcal{N}$, Interest Packets for all data objects share one queue and are served on a First-Come-First-Serve basis. Suppose that the head-of-the-queue Interest Packet at node n at time t is an interest for the *starting chunk* of data object k. If (i) node n has not yet received a request for data object k, or if the last type-k data chunk in the last Data Packet received at node n prior to t is the *ending chunk* of object k, *and* if (ii) there is no PIT entry at node n for any chunk of data object k, then forward the Interest Packet to node

$$b_n^k(t) \in \arg\max_{\{b:(n,b)\in\mathcal{L}^k\}} \bar{\nu}_{nb}^k(t). \tag{10}$$

That is, the Interest Packet is forwarded on the link with the maximum average object-k VIP flow rate over a sliding window of size T prior to t, under Algorithm 1. This latter link is a "profitable" link for forwarding the Interest Packet at time slot t, from the standpoint of reducing delays and congestion. If either condition (i) or (ii) does not hold, then forward the Interest Packet on the link used by node n to forward the most recent Interest Packet for a chunk of object k.[10]

If the head-of-the-queue Interest Packet at node n at time t is an interest for a chunk of data object k which is not the starting chunk, then forward the Interest Packet on the link used by node n to forward the most recent Interest Packet for a chunk of object k.

The above forwarding algorithm ensures that a new request for data object k (which does not overlap with any ongoing request for object k) at time t is forwarded on the link with the maximum average object-k VIP flow rate over a sliding window of size T prior to t. At the same time, the algorithm ensures that an ongoing request for data object k keeps the same outgoing link from node n. This ensures that in the actual plane, all the Interest Packets for an ongoing request for data object k are forwarded on the same path toward a caching point or content source for data object k. As a direct result, the Data Packets for all chunks for the same ongoing request for data object k take the same reverse path through the network.

Note that the Interest Packets for non-overlapping requests for data object k can still be forwarded on different paths, since the quantity $b_n^k(t)$ can vary with t. Thus, the forwarding of data object requests is inherently *multi-path* in nature.

It can be seen that the computational complexity (per time slot) of both the averaging operation in (9) and the link selection operation in (10) is $O(N^2K)$. Thus, the complexity of forwarding (per time slot) in the actual plane is $O(N^2K)$.

4.2.2 Caching of Data Packets

As mentioned in Section 3, within the instantiations of the VIP framework we consider, the caching algorithm in the actual plane coincides with the caching algorithm in the virtual plane. Thus, in the current context, the caching algorithm for the actual plane is the same as that described in (5). Thus, at each time slot t, the data objects with the highest VIP counts (highest local popularity) are cached locally.[11]

[9]Note that the number $\nu_{ab}^k(t)$ of VIPs for object k transmitted over link (a,b) during time slot t may not be the same as the allocated transmission rate $\mu_{ab}^k(t)$. $\nu_{ab}^k(t)$ may be less than $\mu_{ab}^k(t)$ if there are few VIPs waiting to be transmitted.

[10]The router nodes need not know the names of the starting chunk and ending chunk beforehand. These names can be learned as the routers forward Interest Packets and receive Data Packets for the popular data objects. Before the names of the starting and ending chunks are learned, Interest Packets for the data object can be forwarded using a simple technique such as the shortest path algorithm.

[11]For practical implementation in the actual plane, we cannot assume that at each time, each node can gain access to the data object with the highest local popularity for caching. Instead, one can use a scheme similar to that discussed in Section 5.2, based on comparing the VIP count of the data object corresponding to a Data Packet received at a given

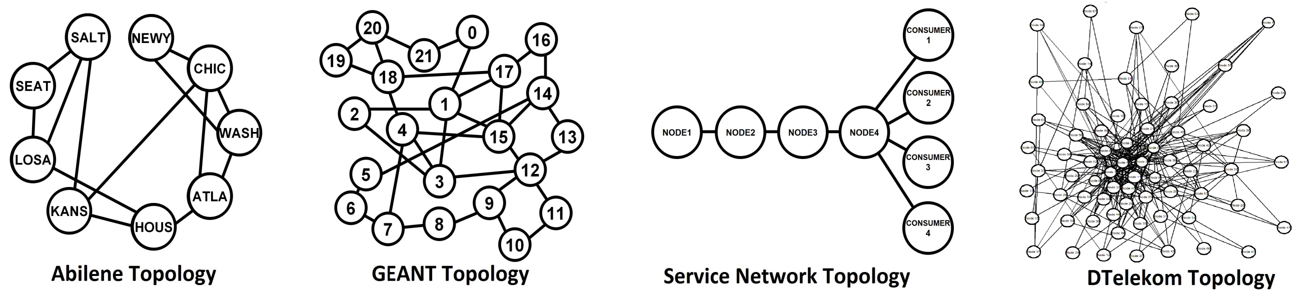

Figure 2: Network Topologies

In attempting to implement the caching algorithm in (5), however, we encounter a problem. Since the VIP count of a data object is decremented by r_n immediately after the caching of the object at node n, the strategy in (5) exhibits oscillatory caching behavior, whereby data objects which are cached are shortly after removed from the cache again due to the VIP counts of other data objects now being larger. Thus, even though Algorithm 1 is throughput optimal in the virtual plane, its mapping to the actual plane leads to policies which are difficult to implement in practice.

In the next section, we demonstrate another instantiation of the VIP framework yielding a forwarding and caching policy for the actual plane, which has more stable caching behavior.

5. STABLE CACHING VIP ALGORITHM

In this section, we describe a practical VIP algorithm, called Algorithm 2, that looks for a stable solution in which the cache contents do not cycle in steady-state. Although Algorithm 2 is not theoretically optimal in the virtual plane, we show that it leads to significant performance gains in simulation experiments.

5.1 Forwarding of Interest Packets

The forwarding algorithm in the virtual plane for Algorithm 2 coincides with the backpressure-based forwarding scheme described in (2)-(3) for Algorithm 1. The forwarding of Interest Packets in the actual plane for Algorithm 2 coincides with the forwarding scheme described in (10). That is, all the Interest Packets for a particular request for a given data object are forwarded on the link with the maximum average VIP flow rate over a sliding window of size T prior to the arrival time of the Interest Packet for the first chunk of the data object.

5.2 Caching of Data Packets

The caching decisions are based on the VIP flow in the virtual plane. Suppose that at time slot t, node n receives the Data Packet containing the first chunk of data object k_{new} which is not currently cached at node n. If there is sufficient unused space in the cache of node n to accommodate the Data Packets of all chunks of object k_{new}, then node n proceeds to cache the Data Packet containing the first chunk of data object k_{new} as well as the Data Packets containing all subsequent chunks for data object k_{new} (which, by the forwarding algorithm in Section 4.2.1, all take the same reverse path through node n). That is, the entire data object k is cached at node n. Otherwise, the node compares the *cache scores* for k_{new} and the currently cached objects, as follows. For a given window size T, let the cache score for object k at node n at time t be

$$CS_n^k(t) = \frac{1}{T} \sum_{t'=t-T+1}^{t} \sum_{(a,n)\in\mathcal{L}^k} \nu_{an}^k(t') = \sum_{(a,n)\in\mathcal{L}^k} \bar{\nu}_{an}^k(t), \tag{11}$$

i.e., the average number of VIPs for object k *received* by node n over a sliding window of size T prior to time slot t. Let $\mathcal{K}_{n,old}$ be the set of objects that are currently cached at node n. Assuming that all data objects are of equal size, let $k_{min} \in \mathcal{K}_{n,old}$ be a current cached object with the smallest cache score. If k_{new} has a lower cache score than k_{min}, then object k_{min} (consisting of all chunks) is evicted and replaced with object k_{new}. Otherwise, the cache is unchanged. If objects have different sizes, the optimal set of objects is chosen to maximize the total cache score under the cache space constraint. This is a knapsack problem for which low complexity heuristics exist.

At each time t, the VIP count at node n for object k is decreased by $r_n s_n^k(t)$ due to the caching at node n. This has the effect of attracting the flow of VIPs for each object $k \in \mathcal{K}_{n,new}$, where $\mathcal{K}_{n,new}$ denotes the new set of cached objects, to node n.

The Data Packets for data objects evicted from the cache are potentially cached more efficiently elsewhere (where the demand for the evicted data object is relatively bigger). This is realized as follows: before the data object is evicted, VIPs and Interest Packets flow toward the caching point as it is a sink for the object. After eviction, the VIP count would begin building up since the VIPs would not exit at the caching point. As the VIPs build further, the backpressure load-balancing forwarding policy would divert them away from the current caching point to other parts of the network.

We now find the caching complexity for Algorithm 2. Note that the complexity of calculating the cache scores (per time slot) in (11) is $O(N^2K)$. Due to link capacity constraints, the number of new data objects which arrive at a given node in a time slot is upper bounded by a constant. Thus, for fixed cache sizes, the total computational complexity for the cache replacement operation (per time slot) is $O(N)$. In sum, the caching complexity for Algorithm 2 per time slot is $O(N^2K)$.

node to the VIP counts of the data objects currently cached at the node.

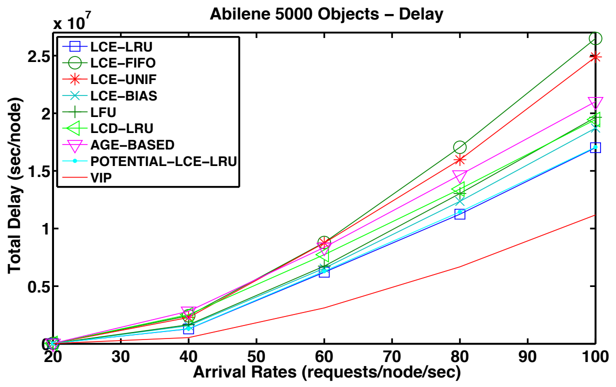

Figure 3: Abilene Network: Delay

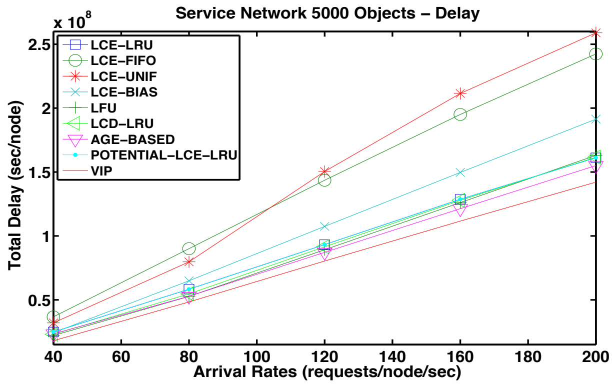

Figure 5: Service Network: Delay

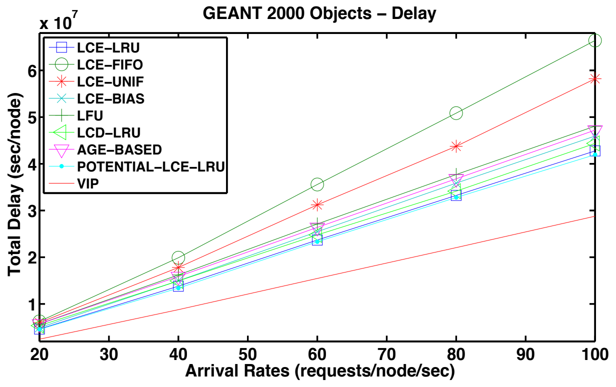

Figure 4: GEANT Network: Delay

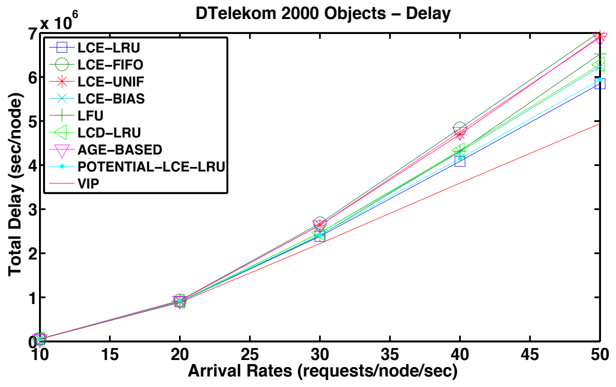

Figure 6: DTelekom Network: Delay

5.3 Experimental Evaluation

This section presents the experimental evaluation of the Stable Caching VIP Algorithm (Algorithm 2).[12] Experimental scenarios are carried on four network topologies: the Abilene Topology (9 nodes), the GEANT topology (22 nodes), the Service Network topology (8 nodes), and the DTelekom Topology (68 nodes) in Figure 2.

In the Service Network and Abilene topologies, all link capacities are chosen to be 500 Mb/s. In the GEANT and DTelekom topologies, all link capacities are chosen to be 200 Mb/s. The Interest Packet size is 125 B; the Data Packet size is 50 KB; the data object size is 5 MB. At each node requesting data, object requests arrive according to a Poisson process with an overall rate λ (in requests/node/sec). Each arriving request requests data object k (independently) with probability p_k, where $\{p_k\}$ follows a (normalized) Zipf distribution with parameter 0.75. In the GEANT and DTelekom topologies, a total of 2000 data objects are considered, while in the other topologies (Service Network and Abilene), 5000 data objects are considered. The buffers which hold the Interest and Data Packets at each node are assumed to have infinite size. We do not consider PIT expiration timers and interest retransmissions.

In the Abilene, GEANT, and DTelekom topologies, object requests can be generated by any node, and the content source for each data object is independently and uniformly distributed among all nodes. The cache sizes at all nodes are identical, and are chosen to be 5 GB (1000 data objects) in the Abilene topology and and 2 GB (400 data objects) in the GEANT and DTelekom topologies. In the Service Network topology, NODE 1 is the content source for all objects and requests can be generated only by the CONSUMER nodes. The cache sizes at NODE 2, NODE 3, NODE 4 and the CONSUMER nodes are 5 GB.

In the virtual plane, the slot length is 200 $msec$ in the GEANT and DTelekom topologies and 80 $msec$ in the other topologies. Forwarding uses the backpressure algorithm with a cost bias to help direct VIPs toward content sources.[13] The cost bias is calculated as the number of hops on the shortest path to the content source, and is added to the VIP queue differential. In the actual plane, the time step for forwarding and caching decisions is 5 μsec in the GEANT and DTelekom topologies and 2 μsec in the other topologies, i.e., the transmission time of one Interest Packet. The window size T is 5000 slots. Each simulation generates requests for 100 sec and terminates when all Interest Packets are fulfilled. Each curve in Figures 3-10 is obtained by averaging over 10 simulation runs.

Simulation experiments were carried out to compare the Stable Caching VIP Algorithm against a number of popular caching algorithms used in conjunction with shortest path forwarding and a potential-based forwarding algorithm. In shortest path forwarding, at any given node, an In-

[12]Our simulations are carried out on a computer with dual Intel E5 2650 CPU's (2.60GHz) and 128 GB RAM space.

[13]It can be shown that the cost-biased version is also throughput optimal in the virtual plane, as in Theorem 2.

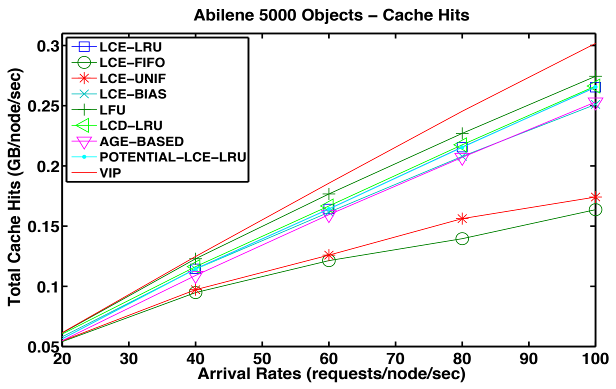

Figure 7: Abilene Network: Cache Hits

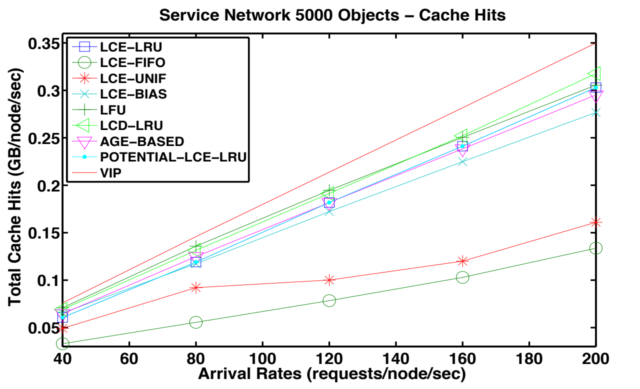

Figure 9: Service Network: Cache Hits

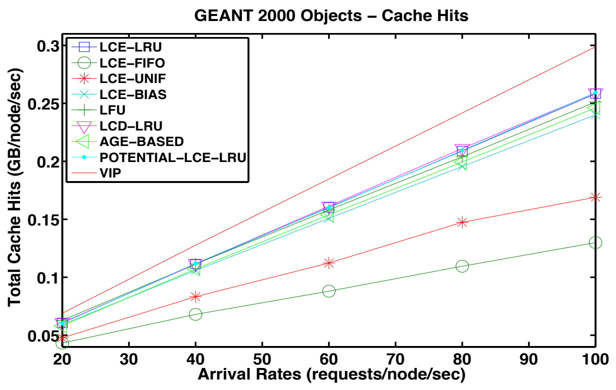

Figure 8: GEANT Network: Cache Hits

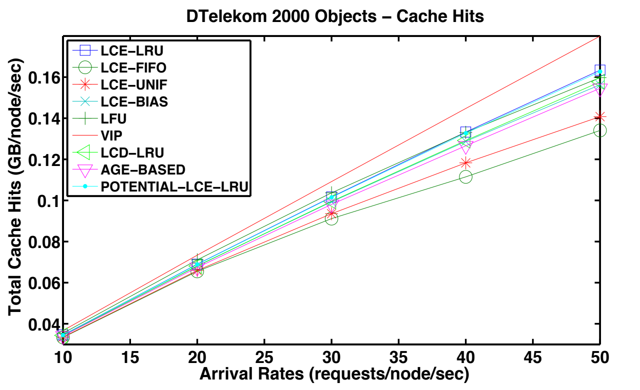

Figure 10: DTelekom Network: Cache Hits

terest Packet for data object k is forwarded on the shortest path to the content source for object k.[14] The Data Packet corresponding to the Interest Packet may be retrieved from a caching node along the shortest path. In potential-based forwarding, a potential value for each data object at each node is set as in [3]. At each time and for each node, an Interest Packet for object k is forwarded to the neighbor with the lowest current potential value for object k. Each caching algorithm consists of two parts: caching decision and caching replacement. Caching decision decides whether or not to cache a new data object when the first chunk of this object arrives and there is no remaining cache space. If a node decides to cache the new data object, then caching replacement decides which currently cached data object should be evicted to make room for the new data object. We considered the following caching decision policies: (i) Leave Copies Everywhere (LCE), which decides to cache all new data objects, and (ii) Leave a Copy Down (LCD)[11], where upon a cache hit for data object k at node n, object k is cached at the node which is one hop closer to the requesting node (while object k remains cached at node n). We considered the following caching replacement policies: (i) Least Recently Used (LRU), which replaces the least recently requested data object, (ii) First In First Out (FIFO), which replaces the data object which arrived first to the cache; (iii) UNIF, which chooses a currently cached data object for replacement, uniformly at random, and (iv) BIAS, which chooses two currently cached data objects uniformly at random, and then replaces the less frequently requested one. In addition, we considered Least Frequently Used (LFU) and age-based caching [7]. In LFU, the nodes record how often each data object has been requested and choose to cache the new data object if it is more frequently requested than the least frequently requested cached data object (which is replaced). In age-based caching [7], each cached object k at node n is assigned an age which depends on p_k, the (Zipf) popularity of object k, and the shortest-path distance between n and $src(k)$. The cache replacement policy replaces the cached object for which the age has been exhausted the longest.

We considered LCE-LRU, LCE-FIFO, LCE-UNIF, and LCE-BIAS combined with shortest path forwarding. We also considered (under shortest path forwarding) LCD combined with LRU, as well as LCE-LRU combined with potential-based forwarding.

The delay for an Interest Packet request is the difference between the fulfillment time (i.e., time of arrival of the requested Data Packet) and the creation time of the Interest Packet request. A cache hit for a data chunk is recorded when an Interest Packet reaches a node which is not a content source but which has the data chunk in its cache. When a cache hit occurs, the corresponding metric is incremented by the size of the chunk in cache.

Figures 3-6 show the delay performance of the algorithms. It is clear that the Stable Caching VIP Algorithm significantly outperforms all other algorithms tested. For instance, for the Abilene topology at $\lambda = 100$ requests/node/sec, the

[14]We assume that all chunks of a data object are cached together.

total delay for the VIP algorithm is only 55% of the delay for the closest competitor (LCE-LRU), and only about 36% of the delay for the worst performing algorithm (LCE-FIFO). Figures 7-10 show the cache hit performance for the algorithms. Again, the Stable Caching VIP Algorithm has significantly higher total cache hits than other algorithms. For the Service topology at $\lambda = 200$ requests/node/sec, the total number of cache hits for Algorithm 2 is about 10% higher than that for the closest competitor (LCD-LRU) and is more than two times the number of cache hits for the worst performing algorithm (LCE-FIFO).

In sum, the Stable Caching VIP Algorithm significantly outperforms all competing algorithms tested, in terms of user delay and rate of cache hits.

6. CONCLUSION

The joint design of traffic engineering and caching strategies is central to information-centric architectures such as NDN, which seek to optimally utilize both bandwidth and storage for efficient content distribution. In this work, we have introduced the VIP framework for the design of high performing NDN networks. In the virtual plane of the VIP framework, distributed control algorithms operating on virtual interest packets (VIPs) are developed to maximize user demand rate satisfied by the network. The flow rates and queue lengths of the VIPs are then used to specify the forwarding and caching algorithms in the actual plane, where Interest Packets and Data Packets are processed. Experimental results show that the latter set of algorithms have superior performance in terms of user delay and cache hit rates, relative to baseline routing and caching policies.

7. REFERENCES

[1] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, kc claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh. Named data networking (ndn) project. October 2010.

[2] 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*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[3] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga. Catt: Potential based routing with content caching for icn. In *Proceedings of SIGCOMM 2012 ICN*, pages 49–54, Helsinki, Finland, August 2012.

[4] M. Amble, P. Parag, S. Shakkottai, and L. Ying. Content-aware caching and traffic management in content distribution networks. In *Proceedings of IEEE INFOCOM 2011*, pages 2858–2866, Shanghai, China, April 2011.

[5] H. Xie, G. Shi, and P. Wang. Tecc: Towards collaborative in-network caching guided by traffic engineering. In *Proceedings of IEEE INFOCOM 2012:Mini-Conference*, pages 2546–2550, Orlando, Florida, USA, March 2012.

[6] W. Chai, D. He, L. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks. In *Proceedings of the 11th International IFIP TC 6 Conference on Networking - Volume Part I*, IFIP'12, pages 27–40, Berlin, Heidelberg, 2012. Springer-Verlag.

[7] Z. Ming, M. Xu, and D. Wang. Age-based cooperative caching in information-centric networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 268–273, March 2012.

[8] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. Adaptive forwarding in named data networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):62–67, June 2012.

[9] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks. 37(12):1936–1949, December 1992.

[10] E. M. 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. http://www.ece.neu.edu/~eyeh/papers/vipicn.pdf. Technical report, 2014.

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

APPENDIX

THEOREM 2 (VIP STABILITY REGION). *The VIP stability region of the network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ with link capacity constraints* (6)-(7), *and with VIP queue evolution* (1), *is the set Λ consisting of all VIP arrival rates $(\lambda_n^k)_{k\in\mathcal{K},n\in\mathcal{N}}$ such that there exist flow variables $(f_{ab}^k)_{k\in\mathcal{K},(a,b)\in\mathcal{L}}$ and storage variables $(\beta_{n,i,l})_{n\in\mathcal{N};i=1,\cdots,\binom{K}{l};\ l=0,\cdots,i_n\triangleq\lfloor L_n/z\rfloor}$ satisfying*

$$f_{ab}^k \geq 0,\ f_{nn}^k = 0,\ f_{src(k)n}^k = 0, \quad \forall a,b,n \in \mathcal{N},\ k \in \mathcal{K} \tag{12}$$

$$f_{ab}^k = 0, \quad \forall a,b \in \mathcal{N},\ k \in \mathcal{K},\ (a,b) \notin \mathcal{L}^k \tag{13}$$

$$0 \leq \beta_{n,i,l} \leq 1,\ i = 1,\cdots,\binom{K}{l},\ l = 0,\cdots,i_n,\ n \in \mathcal{N} \tag{14}$$

$$\lambda_n^k \leq \sum_{b\in\mathcal{N}} f_{nb}^k - \sum_{a\in\mathcal{N}} f_{an}^k + r_n \sum_{l=0}^{i_n} \sum_{i=1}^{\binom{K}{l}} \beta_{n,i,l} \mathbf{1}[k \in \mathcal{B}_{n,i,l}], \quad \forall n \in \mathcal{N},\ k \in \mathcal{K}, n \neq src(k) \tag{15}$$

$$\sum_{k\in\mathcal{K}} f_{ab}^k \leq C_{ba}/z, \quad \forall (a,b) \in \mathcal{L} \tag{16}$$

$$\sum_{l=0}^{i_n} \sum_{i=1}^{\binom{K}{l}} \beta_{n,i,l} = 1, \quad \forall n \in \mathcal{N} \tag{17}$$

Here, $\mathcal{B}_{n,i,l}$ denotes the caching set consisting of the i-th combination of l data objects out of K data objects at node n, where $i = 1,\cdots,\binom{K}{l}$, $l = 0,\cdots,i_n \triangleq \lfloor L_n/z \rfloor$.

PROOF. Please refer to [10]. □