Prefetching Oracles for Pervasive Caching in Information-Centric Networks

Abhiram Ravi[1], Parmesh Ramanathan[2] and Krishna M. Sivalingam[1]
[1]Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India
[2]University of Wisconsin-Madison, Madison, USA
Email: abhiram@cse.iitm.ac.in, parmesh@ece.wisc.edu, skrishnam@cse.iitm.ac.in

ABSTRACT

This paper revisits pervasive content caching at the routers of an Information-Centric Network and demonstrates detailed insights on improving content access latency. This work is motivated by the recent significant growth in Internet video traffic; Fraction of video traffic is predicted to exceed 70% by 2017. The paper extends prefetching models in computer processors to an arbitrary network of caches and proposes a prefetching oracle. The oracle exploits both content popularity *across* users (spatial information) and temporal predictability of *each* user's content access (temporal information) to reduce latency by placing the *right content* at the *right router cache* at the *right time*. An Integer Linear Programming (ILP) formulation of the oracle's challenge is first presented. A heuristic to solve this ILP without significant computational complexity is also presented. Simulation results show that the available bandwidth in the network is intelligently exploited to achieve significant reductions in access latency, and that small levels of temporal lookahead of user requests are sufficient to achieve a large chunk of this gain. Overall, the paper demonstrates that the potential performance benefits of pervasive router level caching are significant and that it must be an integral part of any Information-Centric Network architecture.

1. INTRODUCTION AND RELATED WORK

There is a growing consensus that the *host-centric* architecture of current Internet must evolve to a new *information-centric* architecture to better support emerging applications [1]. For instance, leading projects in the United States National Science Foundation's Future Internet Design (FIND) program (e.g., Named Data Networking [2], eXpressive Internet Architecture [3], MobilityFirst [4]) are based on this premise. In an information-centric architecture, content is treated as a *primitive* entity at the network layer. Applications send out interests for information they desire. Within the network, these interests are matched against published content from different providers. When a match is detected, the information is *diffused* over the network to the applications. The challenge is to diffuse the information in such a way that: (i) the network resources are used effectively, and (ii) application's latency to access the information is small. This paper focuses on a router-level content *prefetching and caching* strategy to address the challenge.

The goal of router-level prefetching and caching is to exploit the available bandwidth in the network to place content items in the right caches *at the right time* so that applications can access them with minimum latency. There is extensive research in literature on content distribution networks that have addressed the problem of effective application-level content caching. They typically assume stochastic knowledge of content popularity among users (e.g., Zipf distribution with known parameters [5]) and then strive to prefetch and place contents near edge networks so that application requests from end hosts do not have to travel far to reach the content they desire. This body of literature typically assumes that the application-level caches are servers with large storage capacities and with ability to simultaneously provide contents for a very large number of users [6, 7]. In contrast, our paper focuses on router-level caching at the edge networks.

Specifically, in this paper, the *source* of the content is assumed to be an application-level cache of a content distribution network. Within the edge network, each router is assumed to have a content store to cache items that are likely to be needed in the near future. Note that, router caches are usually much smaller compared to server storage capacities typically assumed in application-level caches. Of course, one major issue is to predict who will the need a particular content and when. This paper, however, does not focus on this issue. Instead it focuses on identifying the network layer transactions needed to place the content items at the right cache at the right time and for the right duration. It assumes perfect knowledge of applications' immediate future interests. The rationale is to characterize the potential benefits of router-level prefetching and caching and the key factors that determine these benefits so that one can design the right protocols for the information-centric Internet architecture.

Similar to this paper, video streaming research in literature also usually assume full knowledge of future application requests for content. For example, there is extensive work on optimizing the delivery rate of video streaming in a multicast scenario where a potentially large number of users are simultaneously watching the same video. This synchrony in access among all the users plays a key role in the solution [8] and these solutions are not effective in situations where the access to the same content is not synchronized. The oracle in this paper does not require synchrony in content access among users. It can, however, take advantage of the synchrony that may exist between the users.

Specifically, our prefetching oracle unleashes the potential of pervasive router-level caching by exploiting both spatial information (content popularity across users) and temporal information in the interest sequence of each user. The oracle's strategy is first formulated as an Integer Linear Program (ILP). This paper also proposes a heuristic to solve the optimization problem. Although sub-optimal, the heuristic is computationally inexpensive and still shows the tremendous potential router-level prefetching in information-centric architecture. For several different edge network topologies, the improvement in latency relative to conventional content access scheme is over 50%. Also, as expected, the improvement increases with increase in temporal prediction interval. Finally, the distribution of the latency to access different contents shows that the oracle is able to place the right contents at the right place, at the right time.

The use of spatial and temporal localities in prefetching and caching has been well-explored in the computer architecture community [9, 10, 11, 12]. The approach in this paper extends the principles and notions from this context to an *arbitrary* network of caching elements with multiple request generators and content sources. The number of request generators and sources is much larger in our context than the common case in the computer architecture area.

The rest of this paper is organized as follows. The system model is described in Section 2. An Integer Linear Programming formulation of the oracle's strategy is presented in Section 3 and a heuristic to efficiently solve this optimization formulation is proposed in Section 4. The results of an empirical evaluation of the oracle's performance is presented in Section 6. The paper concludes with Section 7.

2. PREFETCHING IN A NETWORK OF CACHES

In this section, we generalize the notions of prefetching to an arbitrary network of caches and set the system model and foundations for the problem that we address in this work.

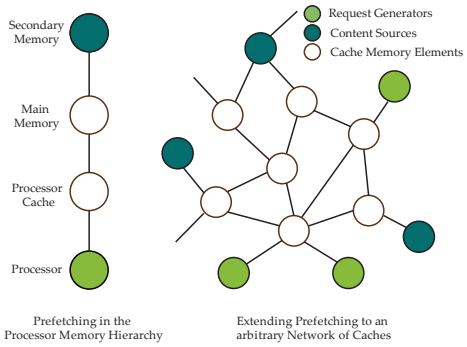


Figure 1: **Network of Caches** Framework.

2.1 Generalizing the Prefetching framework

In the processor setting, prefetching is typically performed from off-chip to on-chip memories, or from the secondary memory to the main memory. Each processor core generates a sequence of requests, and the goal of the system is to reduce the latency of serving these requested items from memory. Requests that *hit* in the on-chip cache are typically served with latencies that are orders of magnitude lesser than those served off-chip. In the single processor setting, we can view the system as a linear graph (see Figure 1), and in the multiprocessor setting as a hierarchy of memory elements. In our setting, the multiprocessor hierarchy generalizes to arbitrary networks of memory (see Figure 1). The nodes of the graph represent memory elements, and edges represent direct connectivity. Each edge is associated with a latency value for unit data transfer. This setup can be naturally viewed from the ICN perspective where we have a network of router *content stores.* Note that there can be multiple clients and content origin servers that can lie *anywhere* in the network. If we assume that the routers of the system can generate requests in addition to forwarding them, i.e., behave like clients in some sense, we have a framework where routers can *prefetch* content for a nearby client.

Evaluation methods for prefetchers also extend directly from the processor context, with *Prefetch coverage*, *prefetch accuracy*, and *timeliness* [9] being the terms in concern. The three traditional questions regarding *what* to prefetch, *when* to prefetch, and *where* to place the prefetched content also apply in this extended context. Our goal in this paper is to set up an *oracle* solution that answers these three questions, as a benchmark for designing distributed prefetching policies.

2.2 Model and Definitions

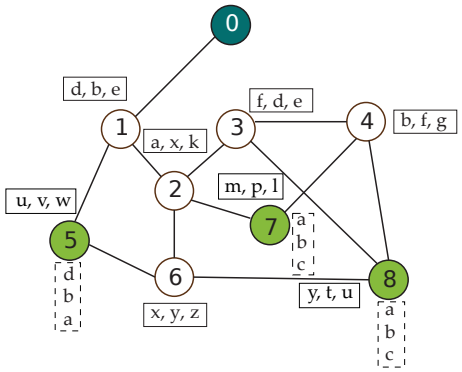


Figure 2: **Benefits of Prefetching** :- Request queues at the clients are shown in dotted boxes, along with the snapshots of the content stores at the routers in solid boxes.

System Model

Extending the processor model used by [12], we make the following simplifying assumptions about the network in consideration. Let $G = (V, E)$ be a *simple unit capacity* network. We assume that :-

- Each content item $\alpha \in \zeta$ be of unit size.
- Each client $c \in C$ in the system is associated with a interest sequence $\mathbf{r}_c = (r_{c,1}, r_{c,2}, ..., r_{c,R})$ of length R, where each request item $r_{c,i} \in \zeta$. The interests are serialized and issued in sequence i.e., if $r_{c,i}$ has been issued, then $\forall j < i$, $r_{c,j}$ must have been served successfully. Since each client has a small buffer, the requested content need not arrive in serial order. However, each client maintains a window of size w and it will not issue a request for a new content that is w steps ahead of any current pending requests.
- Each node $i \in C \cup R$ is associated with a content-store capacity L_i. Each server node $s \in S$ is the origin server for some subset of ζ.

Benefits of Prefetching

Figure 2 illustrates a simple case showing how prefetching can be used to exploit both spatial and predictive information via the availability of spare bandwidth in the system to reduce latency at the clients. Consider the given network where node 0 is the origin server, and nodes 5, 7 and 8 are clients. The request queues and content store snapshots at a certain time are as shown. Since the nearest replica of a for node 8 is at node 2, we need at least two hops to bring it to 8. Observe that node 4 has content b readily available for node 8, and sends it across right away. The choice now remains whether to bring a from node 2 via node 3 or via node 6. Now, by exploiting *spatial information*, i.e., the fact

Notation	Explanation
$G = (V, E)$	A unit capacity network.
$e = (e_s, e_f)$	A unit capacity edge in the network
C, N, $S \subseteq V$	Set of Client, Router, and Server nodes respectively.
L_i	Content-store capacity at node i as number of items.
ζ	(Finite) Space of content items.
$r_{c,i}$	The i^{th} request of client c.
R	Length of the request sequence of each client.
$\tau_{et\alpha\beta}$	Indicator variable for transaction along edge e at time t. delivering α and replacing β.
$d_{i,j}$	Shortest path distance between nodes i and j.
$d_c(\alpha)$	Distance of client c to the nearest replica of content α.
$i_\alpha(c)$	First index (starting from 0) of content α in c's request queue.
$n_c(\alpha)$	Set of nodes containing the nearest replica of content α for client c.
$C_{\alpha,t} \subseteq C$	Set of clients for whom α is in the request queue at time t.
$C_{\alpha,t,k} \subseteq C$	Set of clients for whom α is among the first k items in its request queue at time t.
$y_{i,t,\alpha}$	Indicator variable representing if content store of node i has α at time t.
$\mu_{c,i,t}$	Indicator variable representing if $r_{c,i}$ was served at time t.
w	Maximum number of pending requests at a client

Table 1: Summary of Notation

that node 5 also needs a in the *near* future, we would prefer to bring a via node 6, so that a copy of it can be retained[1] for service to node 5. By exploiting predictive information i.e the fact that node 7 needs c in the near future, we could bring c from the origin server to node 1. Note that these choices exist because of the availability of spare bandwidth along those links. Though it is not always the case that this bandwidth will be available for our exploitation, applying these prefetching techniques will in essence *create* spare bandwidth along several links for future requests. For example, since now c has already been placed at node 1, we have spare bandwidth along the $0 - 1$ link when c is actually needed by node 7. Note that, this example has not worked out the full details for the given snapshot, but has rather motivated the need for executing these *key* decisions to reduce latency.

Given this setup, we now begin to establish some basic definitions to ease our understanding of the system.

Definition 1. (Transaction) A *transaction* $\tau_{e,t,\alpha,\beta}$ is an indicator variable that represents the transfer of a single content item α across an edge $e = (e_s, e_f)$ in the network.
It is associated with four indexing parameters :- The directed edge e along which the transfer happens, the time slot t of the transfer, the content α being trans-

[1]If a was needed by 5 only in the *far* future, it may not be advantageous to do so.

ferred, and the content β being replaced at e_f.

With just this notion, in the next section, we go ahead to formalize the arguments presented in the sample case by presenting a centralized optimization formulation to the problem.

3. THE ORACLE PROBLEM FORMULATION

In this section, we formulate the centralized prefetching problem that the system is trying to solve, as an *Integer Linear Program*. We assume a complete-information scenario i.e., *oracle* access to the request queues of all the clients and to the snapshots of the system. The problem now reduces to finding the best *transaction scheduling* scheme. Note that this is a completely deterministic scenario, and there are no stochastics involved. The intention is to set a benchmark for the best achievable latencies, given exact spatial information and temporal predictive information. The goal is to minimize the time required to serve a set of R requests at each client, subject to the constraints regarding the feasibility of transaction sets and servicing conditions of client requests. Assuming that the maximum latency for a single request in the system is upper bounded by some U, we work over a finite time horizon $T = R \times U$, which is the maximum possible time that the system can take to successfully serve the requests of all the clients. Let Γ be the time taken to successfully serve every client's requests. Please refer to Table I for the summary of notation used. We now have,

Minimize Γ

Subject to

1. $\tau_{e,t,\alpha,\beta} \leq y_{e_s,t,\alpha}$ --→ $\forall t\ \forall e\ \forall \alpha\ \forall \beta$
   Source node of the transaction contains α at time t.

2. $y_{e_f,t+1,\alpha} \geq \tau_{e,t,\alpha,\beta}$ --→ $\forall t = 0$ to $T-1\ \forall e\ \forall \alpha\ \forall \beta$
   Destination node of the transaction contains α at time $t+1$.

3. $y_{e_f,t+1,\beta} \leq (1 - \tau_{e,t,\alpha,\beta})$ --→ $\forall t = 0$ to $T-1\ \forall e\ \forall \alpha\ \forall \beta$
   Destination node of the transaction doesn't contain β at time t

4. $y_{e_f,t+1,z} \leq y_{v,t,z} + \sum_\alpha \sum_e \tau_{e,t,z,\alpha}\ \forall t = 0$ to $T-1\ \forall e_f \in V\ \forall z$
   If a cache doesn't have an element at time t, the element cannot exist in the cache at time $t+1$ unless there is a transaction that brings it in.

5. $y_{e_f,t+1,z} \geq y_{v,t,z} - \sum_\alpha \sum_e \tau_{e,t,\alpha,z}\ \forall t = 0$ to $T-1\ \forall e_f \in V\ \forall z$
   If a cache has an element at time t, the element cannot leave the cache at time $t+1$ unless there is a transaction that replaces it.

6. $\sum_\alpha y_{e_s t \alpha} \leq L_{e_s}$ --→ $\forall t\ \forall e_s \in V \setminus S$
   Constraint on content store capacity.

7. $\sum_\alpha \sum_\beta \tau_{e,t,\alpha,\beta} \leq 1$ --→ $\forall t\ \forall e$
   At most one transaction on a given edge e (unit capacity) and time t.

8. $\mu_{c,i,t} \leq y_{c,t,r_{c,i}}$ --→ $\forall c\ \forall i\ \forall t$
   A request $r_{c,i}$ can be served at time t only if the item is in c's content store.

9. $\sum_{i=1}^{R} \mu_{c,i,t} \leq w$ --→ $\forall c\ \forall t$
   At most w content items can be consumed by a client at a given time t, since there are at most w requests pending.

10. $\sum_{t=1}^{T} \mu_{c,i,t} = 1$ --→ $\forall c\ \forall i$
    A request $r_{c,i}$ is served exactly once.

11. $\sum_{t=1}^{k-1} \mu_{c,i-1,t} \geq \mu_{c,i,k}$ --→ $\forall c\ \forall i = 2$ to R
    If request $r_{c,i}$ is served, then $\forall j < i$, $r_{c,j}$ must have been served.

12. $t \times \mu_{c,i,t} \leq \Gamma$ --→ $\forall c\ \forall i\ \forall t = 1$ to T
    Every request is served before Γ time.

The solution to the above optimization forms a time sequence of transaction sets. Executing these sets at the corresponding time steps would result in the client requests being served within the optimal objective time Γ that is being solved for.

Remark: If the value for T is a known upper bound of the optimal value of the objective in the above formulation, then the problem is always feasible. Intuitively, this is enforced by letting the time horizon $T = R \times U$, since in the worst case, each request of each client is served in at most U time units. Another approach is to first use the heuristic described in Section 4 to find an upper bound for the optimal objective value and set it equal to T.

To test the computational feasibility of the proposed optimization, we feed the optimization problem to the Gurobi Integer Program Solver [13]. The solver took about one day to solve the optimization problem on a 32-core machine with 64 GB of RAM, for a topology of 10 nodes and 15 content items, for a request length R of 10 requests. The solver takes an indefinite amount of time for the same setup with 20 content items (The time horizon T that is set is obtained from the heuristic described in Section 4). This clearly indicates that although the above optimization problem has constraints and variables polynomial in the input parameters, it is computationally too expensive to directly solve the problem using an off-the-shelf solver for a reasonably sized topology.

Also note that if requests keep coming in at every time step, the optimization needs to be re-solved after every time step to maintain optimality.

4. TOWARDS HEURISTIC ALGORITHMS

Since optimally solving the *oracle problem* formulation is practically infeasible, we propose heuristic algorithms for transaction scheduling that nevertheless show drastic improvements in performance. We define a few more terms before formally describing our heuristic solver.

Definition 2. (Progressive Transaction) $\tau_{e,t,\alpha,\beta}$ is said to be a *progressive* (resp. *k-progressive*) transaction if $\exists c \in C_{\alpha,t}$(resp. $C_{\alpha,t,k}$) such that $d_{c,e_f} < d_c(\alpha)$, i.e., upon this transaction, $n_c(\alpha) = \{e_f\}$.

Essentially, for some client that is interested in obtaining α, e_f must become the only node containing the nearest replica of α. A *regressive* transaction has $d_{c,e_f} \geq d_c(\alpha)\ \forall c \in C_{\alpha,t}$.

Definition 3. (Dominating Transaction) We say that $\tau_{e,t,\alpha,\beta}$ dominates $\tau_{e',t,\alpha,\beta'}$ if (i) $\forall c \in C_{\alpha,t}$ we have $d_{c,e_f} \leq d_{c,e'_f}$ and (ii) $\exists c \in C_{\alpha,t}$ such that $d_{c,e_f} < d_{c,e'_f}$. For weak domination, (ii) need not hold, and for *strict* domination $d_{c,e_f} < d_{c,e'_f}\ \forall c \in C_{\alpha,t}$. A *dominating* transaction is a transaction that is not dominated by any other transaction.

Definition 4. (Conflicting Transactions) We say that $\tau_{e,t,\alpha,\beta}$ conflicts with $\tau_{e',t,\alpha',\beta'}$ if *any* of the following holds :-

- e and e' are along the same edge.
- $e_f = e'_f$ and $\beta = \beta'$:- Replacing the same item.
- $e_f = e'_s$ and $\beta = \alpha'$:- Replacing an item that the other transaction is transferring.

Essentially, conflicting transactions cannot be performed together, since they violate the constraints imposed on the system.

Definition 5. (Gravity) The gravity $g(\tau)$is a value associated with every transaction[2]. The notion of gravity is used while making cache replacement decisions i.e., $\tau = 1 \Rightarrow g(\tau) > 0$, and for resolving scheduling conflicts which we shall see later. Intuitively, gravity represents the attractive force generated by the system to replace content β with content α at e_f. It can take several functional forms.

Definition 6. (Feasibility) A set of transactions spanning a finite time interval $t = [0, T]$ is said to be *feasible* if and only if it satisfies constraints (1) to (5) in the Oracle Problem Formulation. It can be seen that two *conflicting* transactions cannot be part of the same feasible set.

[2]We drop the indices on τ for convenience

5. PROPOSED HEURISTIC FRAMEWORK

In this section, we present heuristic algorithms to efficiently find solutions to the centralized optimization problem. Instead of solving for sets of transactions across time units simultaneously, we take a *greedy* approach and make transaction scheduling decisions for the immediately next time step, given a current snapshot of the system. Results show that this greedy approach itself provides significant performance gain.

The basic idea behind the heuristic is, for every time step, to greedily choose certain transactions. The overall template of the heuristic is as follows. Given a snapshot of the system, we (i) prune all irrelevant (*regressive*) transactions (ii) order the relevant (*progressive*) transactions in decreasing order of importance (*gravity*) and (iii) iteratively choose *dominating* transactions with highest gravity that do not *conflict* with previously chosen transactions, until no other transaction can be chosen. This idea is reflected in the generic heuristic template shown in Algorithm 1. We now look at specific instances of the template.

Algorithm 1 Heuristic Template

```
1: procedure GreedySolver
2:     while there is a pending request do
3:         pTrans ← Get all k-progressive transactions
4:         sortedPTrans ← Sort pTrans in decreasing
                          order of gravity
5:         chosenTrans ← Empty List
6:         for t in sortedPTrans do
7:             if some t' in chosenTrans conflicts with
   t
                  or dominates t then
8:                 continue
9:             chosenTrans.add (t)
10:        Execute-all (chosenTrans)
11:        Serve & update request-queues of clients
```

Discounted K-Lookahead

We define the gravity of a transaction as follows

$$g(\tau) = h_{e_f,t}(\alpha) - h_{e_f,t}(\beta)$$

$$h_{n,t}(\alpha) = \textstyle\sum_{c \in C_{\alpha,t,k}} f_{n,t,c}(\alpha)$$

$$f_{n,t,c}(\alpha) = \begin{cases} \gamma^{i_\alpha(c)} \cdot \frac{1}{(i_\alpha(c)+1)(d_{c,n}+1)} & d_c(\alpha) > d_{c,n} \\ 0 & \text{Otherwise} \end{cases}, 0 \leq \gamma \leq 1$$

The *pull* $h_{e_f,t}(\alpha)$ represents the priority level of content α at node n, and the *push* $h_{e_f,t}(\beta)$ represents same for content β. Each of them is a sum over the individual pulls $f_{n,t,c}(\alpha)$'s (or pushes) from every client interested in the respective content items. The *gravity* value of

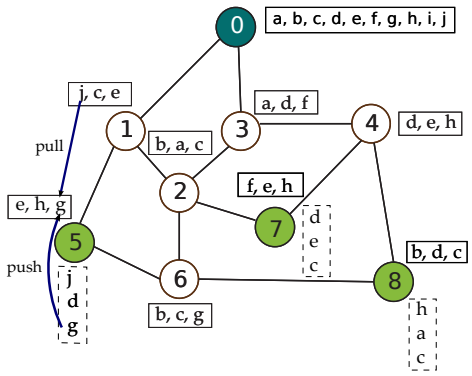


Figure 3: **Gravity** :- The push and pull exhibited by nodes in the system. Content stores are shown in solid boxes, and the request queues at the clients

a transaction represents the effective force with which content α is brought to replace β at node n. The heuristic solver only considers transactions with positive attraction i.e., *pull* > *push* and hence, *gravity* > 0.

Figure 3 gives an intuitive picture of the push-pull mechanism that is proposed. Node 5 is interested in obtaining content item j. Consider the transaction that brings in content item j from node 1 and replaces content item g in the content store of node 5. This transaction gets a pull from node 5 since j is immediately needed, but also gets a push from the same node since the replaced item g is needed in the near future. The overall gravity of this transaction is the difference between the pull and the push. A transaction that brings in j from node 1 and replaces e at node 5 has a higher gravity since there is no push (e is not needed in the near future).

We choose an appropriate functional form for gravity that captures the properties we desire. The design of the *pull* function reflects the following properties.

1. If both α and β appear in the same indices at request queues of clients interested in them, then the client nearer to the node in concern exhibits a larger pull.

2. If a client interested in α, and another in β, are equidistant from node n, then a larger pull value is exhibited on the content that appears at the earliest index in the request queues.

3. A node that already has a replica of α nearer to it than the node's distance to n shows *no* pull to α at n.

4. γ represents a decay of priority for content items that appear in higher indices. A γ close to 0 gives very low priority to content appearing in higher indices.

5. k represents the *Lookahead* value of the heuristic, i.e., number of future requests in the client request queues that are considered.

Note that the given function is just one possible definition for gravity, and there are several alternate choices that one could choose to implement. Simulation results show that our heuristic shows drastic latency gains using this gravity function, and saturates at a lookahead value H, which is the maximum hop distance of any client to the source. We term this the *horizon lookahead* (LA-H).

An Example

The following example demonstrates the functioning of the heuristic. Consider the snapshot of the system as shown in Figure 3. We now observe the decisions made by the heuristic at this snapshot. We assume that all content stores have a capacity of three content items, and we look ahead two requests into the future. We choose $\gamma = 0.25$ for this example.

- *Pruning* :- Node 5 needs j. Since node 1 is the only nearest node that contains j, the transaction from node 1 to node 5 *dominates* the transaction transferring j from 0 to 1. Also, the latter is *regressive*. Hence it (the latter) is discarded from consideration. Similarly, the transaction from node 5 to node 6 transferring h is discarded since it is not a progressive transaction (node 4 already has h ready for node 8). Several transactions are eliminated in this fashion. This represents the pruning stage of the algorithm.

- *Conflict Resolution* :- Consider the transactions from node 1 to node 5 transferring j. We will now calculate the gravities of these transactions for the various possible items that j can replace (e, h, and g). Intuitively, we observe that since node 5 needs g in the near future, it should be less likely that g is replaced. This intuition is captured as follows. The pull for j at 5 is 1. The push for g is $(0.25)^2.1$. The push at e because of node 7 (Only node 7 has e in its request queue) is 0 because node 4 already has the nearest replica of e for node 7. Similarly, the push from node 8 (Only node 8 has h in its request queue) for h's replacement is 0 because node 4 already has a nearer replica. Hence, the gravity of the transaction that replaces g (i.e., 0.9375) is lesser than the gravity of the transaction replacing e or h (i.e., 1). The tie between e and h is resolved randomly by the heuristic.

The above was just an example case to illustrate the behaviour of the heuristic. We now proceed to the evaluation of our heuristic solver.

6. EVALUATION

In this section, we evaluate the performance of our heuristic-based oracle on a variety of parameters. Overall, we observe a significant improvement ($\sim$ 50%) in access latencies for a wide range of topologies. In the multicast scenario, we also observe that the solution achieves a significant percentage ($\sim$ 80%) of the minimum of the maximum flow rates from the clients to the servers. We first describe the simulation environment, and then present several plots, studying the sensitivity of our algorithms to various input and algorithm parameters.

6.1 Simulation Environment

We use a custom simulator and evaluate our heuristic on sub networks of the University of Wisconsin-Madison topology [14], each having 20-25 nodes, and 30-35 edges. We also assume that the servers are at most 6-7 hops away from the client, which is a reasonable assumption at the edge of a content distribution network. We assume that all links have equal bandwidth (*unit* bandwidth, for convenience). Each router is allocated a content store budget of some k% of the total number of content items in the system, for varying values of k. There are several clients, and one or several *source servers* each one housing a subset of the universe of content items. We assume that the clients have buffers of size equal to or greater than their respective maximum flow values to the servers.

The Interest Oracle

The Interest Oracle represents complete-information access to the future interests that will be generated by every client (i.e., $|C| \times R$ Interest matrix $[r_{c,i}]$). We consider three different request patterns at the clients.
1. Zipf :- First, we look at the scenario at one end of the spectrum where each client follows an *Independent Reference Model*, with a *Zipf* distribution over content items (Zipf parameter of 0.8).
2. Stream :- Second, we look at the scenario at the other end of the spectrum where every client requests the exact same sequence of (distinct) content items, representing the multicast streaming scenario.
3. Stream-zipf (Hybrid) :- Third, we consider a hybrid of both request patterns. We assume that there are N movies in the system, each consisting of a sequence of F distinct frames(We assume that each frame corresponds to one content item), where F is uniformly distributed from 1 to M. Each client first chooses a movie from a Zipf distribution over the N movies, and then issues interests sequentially for the frames corresponding to that movie. Once all the frames for the chosen movie are served, the client samples another movie from the zipf distribution and repeats the process. We only work with an $2N/M$ ratio of 1, the pattern that lies in the center of the request spectrum. $N \gg M$ represents *Zipf* at the limit, and $N \ll M$ represents *Stream* at the limit.

The motivation behind introducing the hybrid request pattern is to capture the intermediate region of the spectrum that lies between the assumptions of an independent reference model and that of a multicast streaming scenario. We assume oracle access to the interest matrices that represent each of these three patterns and analyze the performance of our heuristic. The metrics in concern are the *access latencies, network utilization* and the *rates* of content delivery. Except for the case where we analyze performance on the Stream pattern, we initialize the system with content distributed randomly across the routers in proportion to the content popularities before beginning our measurements.

6.2 Key Results

6.2.1 Latency Reduction

Figure 4 plots the average latency gains for our *Discounted Horizon-Lookahead* heuristic (latency in terms of hop count) as compared to latency to the sources for five different topologies. For these results, the client requests are based on *Stream-zipf*. The averages are taken over varying cache sizes ranging between 2–6% of the total number of content items in the request streams. For each topology, the left bar corresponds to the case with *no* prefetching while the right bar corresponds to the oracle with the *Discounted Horizon-Lookahead* heuristic. For each topology, note that, the gains due to prefetching in the oracle are significant. The reduction in latency for the left bar (i.e., no prefetching) is only due to exploitation of content popularity among clients (i.e., spatial information). However, the reduction in latency for the right bar (i.e., oracle) comes due to both spatial and temporal information (i.e., looking ahead in time). For instance, for Topology 1, the average latency is about 80% of the source latency when there is no prefetching, whereas it is about 40% of the source latency for the oracle. This is 50% improvement due to prefetching.

6.2.2 Multicast Rate

Figure 5 plots the delivery rates for different levels of lookahead when the client access pattern corresponds to the *Stream* scenario. Recall that, *Stream* scenario corresponds to a multicast where all clients request the exact same set of distinct content items over time. To evaluate performance in this scenario, we consider the *rate* of content delivery as opposed to latency, and observe that the heuristic achieves 81.5% of the minimum of the maximum flow over all the clients. Note that it may not be possible to achieve maximum flow without network coding [15, 16]. Since the oracle does not

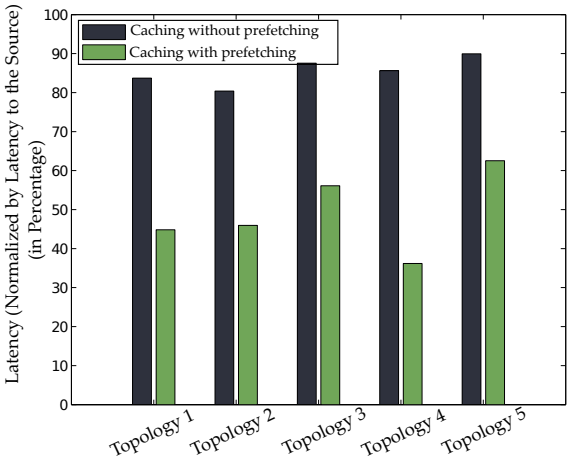


Figure 4: **Latency Reduction** :- Average query latency over clients across different topologies, normalized by the latency to the source, for *Stream-zipf*.

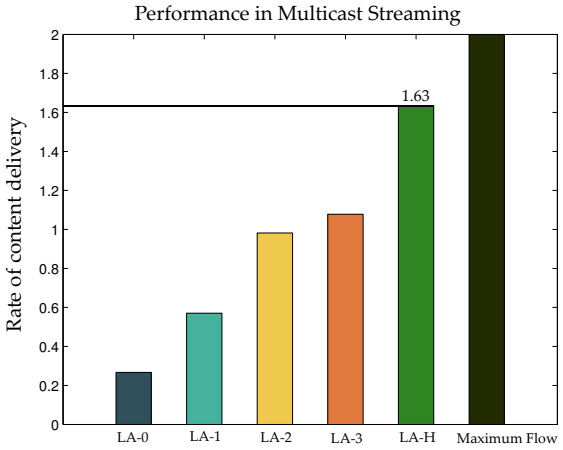


Figure 5: **Performance in Multicast Streaming** (*Stream* request pattern) :- Rate of content delivery for different lookaheads.

use network coding, we only observe that the heuristic is capable of achieving near-maximum flow rates when challenged with a multicast request pattern.

The figure also clearly shows the benefits of lookahead. The leftmost bar in the figure corresponds to the case of zero lookahead (i.e., no prefetching). The next three bars show the results for a lookahead of one, two, and three time steps respectively. The LA-H bar corresponds to the case where the lookahead is equal to the number of hops to the source. There are no advantages to looking ahead beyond the number of hops to the source. Hence, LA-H corresponds to the best performance from the oracle with respect to the levels of lookahead.

6.3 Sensitivity & Analysis

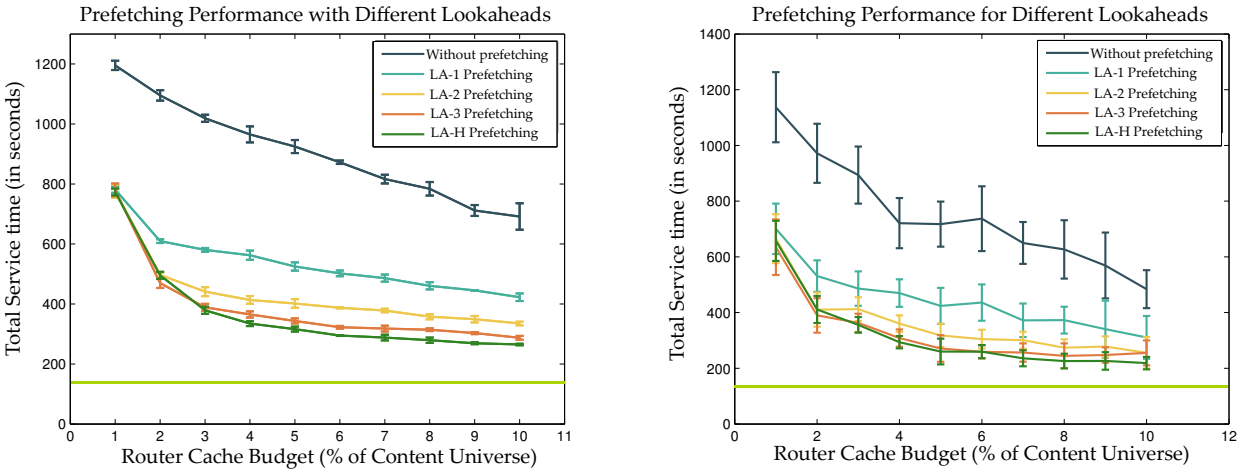
In the next set of results, we evaluate the sensitivity of oracle's performance to different input and algorithm parameters.

6.3.1 Cache Budget and Lookahead

Figure 6 shows the variation in average total service time for each client to complete 2500 accesses as a function of cache size, for the *Zipf* and *Stream-zipf* patterns. Smaller total service times are better than larger total service times. There are five curves in the figure. The topmost curve corresponds to the case with only caching and no prefetching (i.e., lookahead is zero). The next three curves from top to bottom correspond to lookaheads of one, two, and three, respectively. The lowest curve corresponds to lookahead equal to the number of hops to the server (H = 6). The averaging is over many different access request sequences for length 2500 for each client. As expected, the average total service time decreases when the cache size increases. We also observe that a lookahead of just 2 or 3 time steps provides a significant reduction in total service time, and beyond that the each additional lookahead provides smaller and smaller reduction in total service times. This indicates that it is *enough* to look ahead a small time into the future to make significant performance improvements. The reference line represents the ideal latency if the cache at the edges was infinite, which is eventually reached by all the curves at 100% cache budget. For a more realistic scenario, where the cache budget is $\sim$ 5% of the content space, we see a clear improvement in performance for higher lookaheads in both *zipf* and *stream-zipf* request patterns. Error bars representing the standard deviation for 20 runs are also shown for each data point. Variance of data points for *Stream-zipf* is higher because of the varying levels of spatial and temporal properties in the interest matrices generated for different runs.

6.3.2 Network Utilization

One of the key features of prefetching is that it exploits the *available* spare bandwidth to carry out useful transactions. Figure 7 plots the average percentage of active links for different lookahead values. Here again, there are five curves in the figure corresponding to zero, one, two, three, and horizon lookaheads (starting from bottom to top). We see that higher lookaheads utilize the available bandwidth in the system to bring content closer to the clients and improve latency. Note that the baseline exploits $\sim$ 25% of the available bandwidth of the system at any given time. Higher lookaheads exploit more available bandwidth. Note that, even for LA-H prefetching, there is still a significant gap of around $15 - 20\%$ to full utilization. This does not imply that our heuristic is sub-optimal. We observe from the link



(a) **Zipf Request Pattern** (b) **Stream-Zipf Request Pattern**

Figure 6: **Prefetching Performance for Different Lookaheads** on a single topology, with error bars. H=6

utilization data that the unused links were irrelevant in the context of prefetching (*Prefetch-dead* links :- Links that do not help in bringing content closer to any client). In fact, it is possible that the most optimal scheduling scheme would utilize lesser bandwidth and still provide better latencies! We observe that with increase in cache size, the network utilization initially increases. This is because very small caches do not have space for prefetched data i.e prefetching cannot take greater advantage of the network due of storage constraints at the routers. Note that even in this scenario, although the effect of higher lookahead is little, prefetching can still take significant advantage of the network (at least 40% utilization, as opposed to the 25% baseline) to improve latencies. We also observe that the utilization gradually decreases, for higher cache sizes. This decrease represents the cache-bandwidth trade-off i.e., the system needs lower bandwidth resources for larger cache sizes to deliver at the same latency.

6.3.3 *Prefetch Quality*

Figure 8 plots the cumulative distribution function of latency for the Stream-zipf request pattern. Higher fraction for the same latency indicates that smaller latencies are much more prevalent. The left curve in this figure corresponds to caching with prefetching with horizon lookahead while the right curve corresponds to the case of no prefetching (i.e., lookahead of zero). We see that with prefetching enabled, a large number of requests ($> 80\%$) are served within one hop. This graph is reflective of the *prefetch timeliness* metric, which indicates that when the content was needed, it was at most two hops away with a 90% chance. We would also like to point out that the other commonly used evaluation metrics of cache performance, namely, Prefetch Coverage and Prefetch Accuracy are irrelevant here since we have assumed oracle access to the client requests.

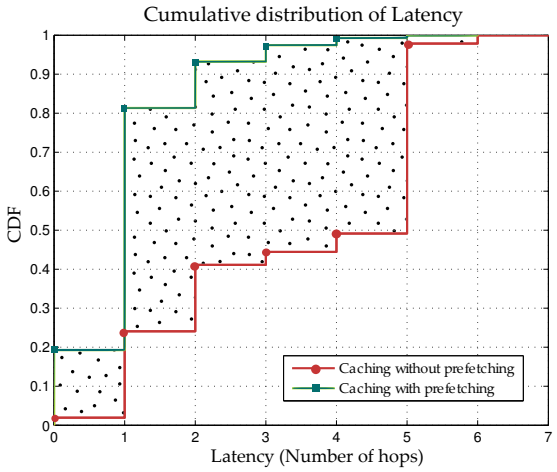
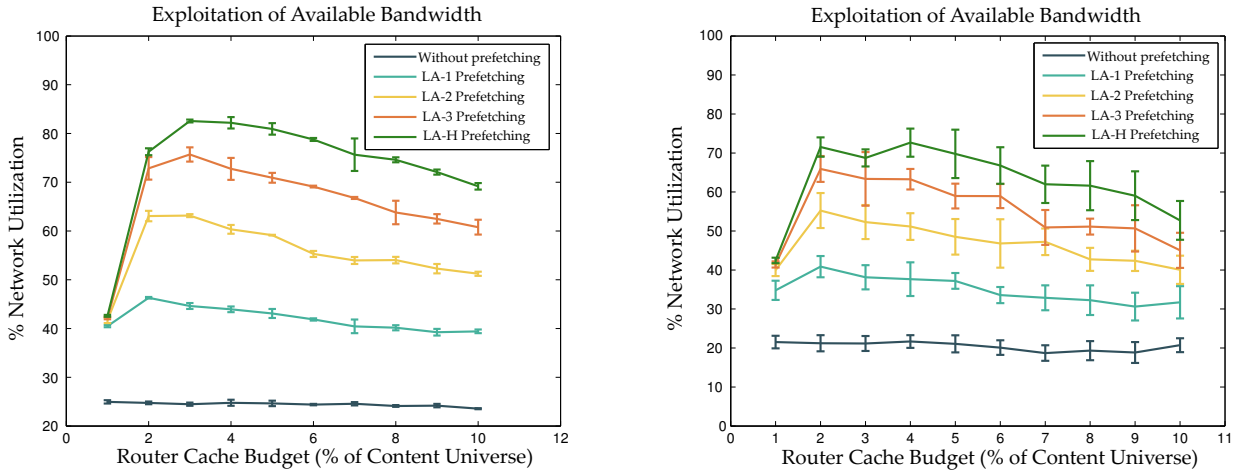


Figure 8: **Prefetch Quality**: CDF of Latency for router cache budget $k = 5\%$.

7. SUMMARY AND FUTURE WORK

Inspired by the prefetching model in processors and extending them to an arbitrary network of caches, we have presented a formulation to the prefetching oracle as a problem of transaction scheduling in the form of an Integer Linear Program. We have also proposed a feasible heuristic solver for the same, and shown that the heuristic solution can decrease latencies by more than 50% in typical university topologies. We have demonstrated that small sized router caches, that lie in the heart of an Information-centric Network, can be used to leverage both spatial and temporal information about client requests and drastically improve access latency.



(a) **Zipf Request Pattern**

(b) **Stream-Zipf Request Pattern**

Figure 7: **Network Utilization** (Exploitation of Available Bandwidth) **for Different Lookaheads** :- Average percentage of active links in the system.

In addition, we have also shown that small lookaheads into the future requests of the clients contribute to a significant portion of the gain. Overall, we have established a framework to study prefetching in the ICN context, and have paved the way for a thorough study of incomplete information & distributed prefetching algorithms by setting up a benchmark to compare against.

Note that, the oracle had two distinct advantages: (i) correct knowledge of future requests from all clients, and (ii) centralized decision-making with prefetching over the entire network. Future work must focus on relaxing both these requirements. That is, each router must execute prefetching in a distributed (but possibly coordinated) fashion. Furthermore, each router may have to predict future requests based on the request patterns it observes, and is hence prone to errors in prediction. We are addressing these issues in an ongoing work.

References

R. Jain, "Internet 3.0: Ten problems with current internet architecture and solutions for the next generation," in *Proceedings of the 2006 IEEE Conference on Military Communications*, MILCOM'06, IEEE Press, 2006.

L. Zhang et. al, "Named Data Networkng (NDN) Project," tech. rep., University of California, Los Angeles, Oct 2010.

D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "XIA: Efficient supoprt for evolvable internetworking," in *Proceedings of NSDI*, Apr. 2012.

D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "MobilityFirst: A Robust and Trustworthy Mobility-Centric Architecture for the Future Internet," *SIGMOBILE Mobile Computing and Communication Review (M2CR)*, 2012.

S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," SIGCOMM, ACM, 2013.

D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in *Proceedings of the 6th International Conference*, CoNEXT '10, ACM, 2010.

B. Tan and L. Massoulié, "Optimal content placement for peer-to-peer video-on-demand systems," *IEEE/ACM Transactions on Networking*, Apr. 2013.

K. Kim, S. Choi, S. Kim, and B.-h. Roh, "A push-enabling scheme for live streaming system in content-centric networking," in *Proceedings of the 2013 Workshop on Student Workhop*, CoNEXT Student Workhop '13, ACM, 2013.

D. Joseph and D. Grunwald, "Prefetching using markov predictors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ISCA '97, ACM, 1997.

S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Computing Surveys*, June 2000.

W. Jin, R. Barve, and K. Trivedi, "A simple characterization of provably efficient prefetching algorithms," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 571–580, 2002.

S. Albers, N. Garg, and S. Leonardi, "Minimizing stall time in single and parallel disk systems," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, ACM, 1998.

I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015.

"University of wisconsin at madison network statistics," April 2015.

R. Ahlswede, C. Ning, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, 2000.

N. Sundaram and P. Ramanathan, "A distributed bandwidth partitioning scheme for concurrent network-coded multicast sessions," in *Proceedings of the IEEE Conference on Global Telecommunications*, IEEE Press, 2009.