Congestion-Aware Caching and Search in Information-Centric Networks*

Mikhail Badov
University of Massachusetts
Amherst
mbadov@cs.umass.edu

Anand Seetharam
California State University
Monterey Bay
aseetharam@csumb.edu

Jim Kurose
University of Massachusetts
Amherst
kurose@cs.umass.edu

Victor Firoiu
BAE Systems
victor.firoiu@baesystems.com

Soumendra Nanda
BAE Systems
soumendra.nanda@baesystems.com

ABSTRACT

The performance of in-network caching in information-centric networks, and of cache networks more generally, is typically characterized by network-centric performance metrics such as hit rate and hop count, with approaches to locating and caching content evaluated and optimized for these metrics. We believe that user-centric performance metrics, in particular the delay from when a content request is made by the user to the time at which the requested content has been completely downloaded, are also important. For such metrics, performance is often determined by link capacity constraints and network congestion. We investigate network cache management and search policies that account for path-level (content-server to content-requestor) congestion and file popularity in order to directly minimize user-centric, content-download delay. Through simulation, we find that our policies yield significantly better download delay performance than existing policies, even though these existing policies provide better performance according to traditional metrics such as cache hit rate and hop count.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design - Distributed networks

General Terms

Performance, Design

*This work was funded in part by the DARPA Fixed Wireless program under contract FA8750-13-C-0169 and by the National Science Foundation under grant CNS-1117764. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation or the U.S. Government.

ICN'14, September 24–26, 2014, Paris, France.

http://dx.doi.org/10.1145/2660129.2660145 .

Keywords

Information-Centric Networking, Caching, Congestion

1. INTRODUCTION

A key component of many information-centric network (ICN) designs is the use of in-network caching at storage-enabled routers lying between content custodians (origin servers) and content requesters [3]. The key advantage of serving content to the requester from an in-network cache (particularly a cache enroute to the custodian) is that content will be returned to the user faster than if the content had been served by the content custodian. When designing and evaluating the effectiveness of cache management and content location schemes for such networks, one of the primary performance metrics has been cache hit probability — the fraction of content requests passing through a cache node that find the content stored in that node, — a performance metric used since the earliest analyses of standalone caches more than 40 years ago. More recently, when analyzing networks of caches, the number of hops between the requester and the in-network cache or custodian returning content, has been used as an additional performance metrics of interest, e.g., [1].

Cache hit rates and hop counts are *network-centric* performance metrics. Since a network exists to provide service to its users, *user-centric* performance metrics are also of great importance. For the case of content retrieval, the content download delay — the time from when a user first issues a request to the time when the content has been completely received by the user — is a natural performance metric of interest. Here, the capacity of the links on the download path between the requester and the in-network cache or custodian returning content, and the number of ongoing content flows using those links will influence the content download delay. Given the differences between traditional network-centric metrics and the user-centric metric of download delay, one might expect (and indeed we will see in this paper) that cache management and content-request routing approaches developed for network-centric metrics do not necessarily perform well when evaluated using download delay as the performance metric of interest, and that new approaches designed with congestion-sensitive download delay in mind can achieve better performance.

In this paper, we propose and evaluate new cache management (content replacement) and content-request routing

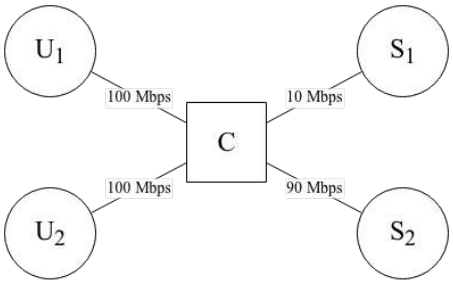


Figure 1: A cache-enabled content network

schemes using a user-centric performance metric of content-download delay. We first consider the case of fixed routing of requests towards a custodian and present a congestion-aware cache management policy that considers the relative costs of downloading various content. The intuition behind our approach is that rather than optimizing hit rates, one should use space in the local cache to avoid using the most congested links to download requested content. We then consider the complementary challenge of content-request routing, investigating an approach in which the requester adopts a congestion-aware search for content, before routing the request to the custodian. Through simulation, we find that our policies yield significantly better download delay performance than existing policies, even though these existing policies provide better performance according to traditional metrics such as cache hit rate and hop count.

The remainder of this paper is structured as follows. Section 2 provides a simple, motivating example that provides intuition and insight into why cache management and content-routing policies that provide a high hit rate or low content-download hop count may perform poorly when download delay is the performance metric of interest. In Section 3, we describe our network model. We present our congestion-aware caching and search policies in Sections 4 and 5, respectively. These policies incorporate not only the popularity of a given piece of content, but also the estimated download delay in the presence of network congestion, as a basis for deciding what to cache, what to evict, and from where to download content. In Section 6, we evaluate our policies through simulation across a variety of topologies (including a grid, a scale-free network, a hybrid MANET, and ISP backbones based on Rocketfuel), finding that our policies yield the lowest average download delay in all studied networks. We discuss these findings in Section 7 and discuss related work in Section 8. Finally, Section 9 concludes the paper and outlines future work.

2. A MOTIVATING EXAMPLE

We begin with a simple example, shown in Figure 1, to provide intuition and insight into why cache management policies that optimize the network-centric goal of maximizing hit rate need not lead to good performance for a user's point of view, where content download delay is the metric of interest. Figure 1 shows two users, U_1 and U_2, and two servers, S_1 and S_2. A cache-enabled router C, with the ability to store exactly one piece of content, connects all nodes. The content universe consists of two equally sized pieces, F_1 and F_2, residing in S_1 and S_2, respectively. We assume that the users request a single piece of content at a time

Cached Content	Content Request Ratios					
	F_1:0.50, F_2:0.50		F_1:0.10, F_2:0.90		F_1:0.01, F_2:0.99	
	Hit Rate	Delay	Hit Rate	Delay	Hit Rate	Delay
F_1	0.5	0.106	0.10	0.110	0.01	0.111
F_2	0.5	0.550	0.90	0.190	0.99	0.109

Table 1: Hit rate and average delay, under a static caching policy, for content of size 10 Mb with various content request ratios

(i.e., each user issues their next content request only after completely downloading the previous content). The links U_1-C and U_2-C both have a capacity of 100 Mbps, while the links C-S_1 and C-S_2 have a capacity 10 Mbps and 90 Mbps, respectively.

In this paper, we will assume the existence of some mechanism (e.g., TCP-like) that fairly shares link-bandwidth among multiple download flows traversing a common bottleneck link, although this assumption is not needed for this simple example. A flow's bottleneck link determines that flow's overall download rate along the path from content sender to content requester. Temporarily ignoring the cache, the end-to-end throughput between the users and the two servers, S_1 and S_2, would be 10 Mbps and 90 Mbps, respectively, due to the links C-S_1 and C-S_2 acting as bottlenecks. However, if a requester finds its content cached at C, that content can be downloaded at a rate of 100 Mbps.

Let us first consider the case that the request rates for F_1 and F_2 are equal. In this case, if F_1 is cached at C, the average download delay is 0.106 secs - F_1 requests are downloaded at 100 Mbps to U_1 with a delay of .1 secs from C, and F_2 requests are downloaded to U_2 from S_2 at a bottleneck rate of 90 Mbs, with a delay of .112 sec. If F_2 is cached at C, the average download delay increases to 0.555 secs, since F_1 requests are downloaded to U_1 from S_1 at a bottleneck rate of only 10 Mbps (requiring 1 sec to download), while F_2 requests are downloaded from C with a delay of .1 sec. *This example suggests that caching on the downstream end of a low-capacity or congested link can make most effective use of cache space.*

Suppose next that the request rates for F_1 and F_2 are 0.1 and 0.9 respectively (the middle column in Table 1). In this case, if F_1 is cached at C, the average download delay is 0.110 but the hit rate is only 0.1. Now suppose that F_2 (which receives 90% of the requests) is cached at C. The hit rate here increases to 0.9 but the delay *increases* to .19 secs. This increase in delay results from the fact that F_1 (which is only receiving 10% of the requests) must now be downloaded from S_1 over the slow 10 Mbps path. In this example, the policy of caching F_1 at C has the *best* performance from a user-centric point of view (minimizing download delay) but the *worst* performance from a network-centric point of view (maximizing hit rate). Similarly, the policy of caching F_2 at C has the *best* performance from a network-centric point of view but the *worst* performance from a user-centric point of view. *This example suggests that a caching policy that provides the best network-centric performance may provide very poor user-centric performance, and that caching policies designed for network centric performance metrics such as hit rate may not be well-suited for scenarios when user-centric performance metrics are of primary interest.*

Last, suppose the asymmetry in the F_1 and F_2 request rates increases further to 0.01 and 0.99, respectively. In this case, caching F_2 (which is receiving 99% of the requests) at C provides both a higher hit rate *and* a lower average delay, making this policy the winner from both a user-centric and a network-centric point of view. *This example suggests that content popularity, although not always the deciding factor, can play a role in determining the preferred cache management policy.*

We these insights, we can now consider cache management policies whose goals are to decrease the user-centric measure of average download delay.

3. NETWORK MODEL

We consider a typical ICN model with named content, but the cache management and content-request routing policies, insights, and results apply more generally to networks of caches. Our content universe consists of a finite set of distinct, but equally sized, content. Each piece of content has a node responsible for its permanent storage, referred to as the custodian. We assume there is some baseline mechanism for forwarding content requests (e.g., along a shortest path) from content requester to content custodian. Requests arrive exogenously at every node, and nodes route every exogenous and endogenous content request to the respective custodian, e.g. via shortest-path routing. Content delivery follows the request path.

Nodes are cache-enabled; the size of a cache is given in terms of the number of pieces of content that can be stored within. If a content request, en-route to the custodian, reaches a node that contains the content in its cache, then the intermediate node directly services the request from its own cache. This is a technique commonly used in ICN literature [4, 10].

Our goal is to design cache management and content-request routing policies that minimize content download delay. To this end, we factor congestion (resulting from simultaneous downloads using a link) into caching and routing decisions. We adopt a fluid model to capture the effects of congestion and heterogeneous link capacity on download throughput. Specifically, we assume there is some mechanism (e.g., TCP-like) that fairly shares link-bandwidth among multiple download flows that are bottlenecked at a link. If there are N flow crossing a link, each flow is guaranteed to receive a fair share of at least $1/N$ of the link bandwidth. The throughput of a content-download flow is limited by the most congested link on its path. Note that a flow may not use the fair share of bandwidth allocated to it across a link due to a bottleneck elsewhere along the path. In such a scenario, the capacity freed up by the congested flow may be fairly divided among other flows sharing that link.

4. CONGESTION-AWARE CACHING

In the section, we present a novel congestion-aware cache management policy that determines whether a piece of content passing through the router should be cached, and if so, what piece of content currently in the cache must be evicted to make room for this to-be-cached content. The effectiveness of our policy lies in the metrics used in its design — our policy considers the link congestion experienced along the path from content sender to content requester during content retrieval, combined with content popularity, in the caching and eviction process. Our intuition is that caches should preferentially retain content that has been forwarded over congested links, and evict content forwarded over uncongested links. We use a utility function to approximate the value of caching a given content item. We first describe the construction of our utility function, followed by the design of our management policy.

We design a utility function that operates at each node and estimates the download delay saved by caching a piece of content (that is currently being forwarded through that node) at that node. The download delay savings is defined as the difference between the time it takes for a requester to download content from this cache, and the time it would take the requester to download from the transmitting source (either the upstream cache or the custodian). We will use a fluid model to estimate download delay, with the flow's bottleneck link along the path determining flow throughput, and flow throughput, in turn, determining download time, as in our simple example in Section 2.

Content popularity will also play a role. The intuition here is that caching content that is costly (i.e., has a high download delay from where it is currently being served), but is also unpopular, will not provide significant benefit to the system as a whole. Similarly, caching a popular piece of content on a relatively uncongested path will also not provide significant benefit. Consequently, it is best to cache content that is both popular and costly, relative to the location of the source, the location of the cache, and the level of congestion experienced along the path between them.

Estimating Local Congestion: Each node keeps a count of the number of flows (active downloads) currently passing over each of its interfaces. The bandwidth B_l of link l available to a download flow is estimated by dividing the link capacity, C_l, by the number of flows passing through it, F_l:

$$B_l = \frac{C_l}{F_l}$$

Note that this is an underestimate of the bandwidth actually available to the flow on this link, since (as described above), any of the flows passing over a link may use less bandwidth than is allocated, due to bottlenecks elsewhere along the flow's path.

Estimating Popularity: The popularity of a piece of content f, denoted as P_f, is defined as the number of requests for the content divided by the number of total interest requests for all content. These requests may be counted during a moving window, updated according to an exponentially weighted moving average over the windows, or simply counted since the last time such counts were zeroed; we will assume the latter in our evaluation below, assuming that counts are zeroed when a router starts up. At a given cache, denote the number of times a piece of content has been requested at that cache by N_f. The set F is the set of all content that has been requested at the cache. We define P_f as:

$$P_f = \frac{N_f}{\sum_{f'}^{F} N_{f'}}$$

Considering End-to-End Throughput: A content download flow may be bottlenecked either in the source-to-cache (upstream) path segment or the cache-to-requester (downstream) path segment. Downstream congestion can

limit the gain of caching content at the cache, especially if the effective cache-to-requester throughput is significantly lower than the source-to-cache throughput. Certainly, caching the content would still yield some gains, because other flows would not have to share the upstream link capacity. However, if the downstream links are the more congested, the upstream links are (by definition) less congested and therefore flow using the upstream links are less likely to benefit significantly from caching a piece of content at the cache (and thereby obviating the need for future requests for that content to use those upstream links). It would appear as though it is not beneficial to cache content if the requester has a path with low available bandwidth to the specified cache — it appears to neither decrease the download time for the next request for the content, nor significantly impact other flows sharing the upstream links. For this reason, considering downstream throughput is important in making caching decisions.

Putting it All Together: Let us define L_U as the set of links connecting the *upstream* source (the custodian, or another upstream cache) to the cache, and L_D as the set of links connecting the cache to the *downstream* requester. For every link l in either L_U or L_D, let B_l denote the available bandwidth for that link, in the presence of the congestion resulting from simultaneous downloads. Define the set B_{LU} as the set of available capacities of the links in L_U, and the set B_{LD} as the set of available capacities of the links in L_D. Using the fluid model, the cache-to-requester throughput is then $min(B_{LD})$. For the same reason, $min(B_{LD} \cup B_{LU})$ is the throughput of the end-to-end path. The estimated delay saved by caching the content for future requests is then $S/min(B_{LD} \cup B_{LU}) - S/min(B_{LD})$, where S is the content unit size.

Recalling the insights gained through our simple example in Section 2, we also want to take content popularity (as seen at a given cache) into account when defining the overall utility of caching file f at that given cache. The utility of caching f, given the set of bandwidth available at each link B_L at node N is thus the delay savings weighted by content popularity:

$$U_{f,N} = (\frac{S}{min(B_{LD} \cup B_{LU})} - \frac{S}{min(B_{LD})}) * P_f \quad (1)$$

We also consider a policy that takes only the upstream congestion level into account. This can be seen as absolute instead of relative savings. Such a policy could be relevant in the case of a hop-by-hop [14] congestion control mechanism (where there is no notion of end-to-end throughput). The utility value of the alternate policy is defined as:

$$U_{f,N} = \frac{S}{min(B_{LU})} * P_f \quad (2)$$

The various pieces of information needed to compute the utility value in Equations 1 and 2 can be obtained by piggybacking additional information in content-request and content-download packets already present in many ICN architectures. Specifically, an interest request for content that is being forwarded upstream towards the custodian contains the running minimum of B_l for all links that it has traversed and is (potentially) updated after passing through each link. Similarly, a running minimum of B_l is passed downstream with the content. In this manner, each cache obtains the values of B_{LU} and B_{LD} with each request. The other values needed in Equations 1 and 2 are all locally-available pieces of information.

Eviction and Management: Given our utility function, cache management and eviction are simple. When new content enters a router, its utility is computed. If the computed utility value is lower than the lowest existing utility in the cache, the content is forwarded, without caching. Otherwise, the content with the lowest utility is evicted and the new content is cached. When deciding to cache a piece of content, a cache resets the running downstream B_l value, so that the utility computed at downstream caches will be computed with respect this cache, rather than the original source. This prevents all caches downstream of a congested link from caching the content.

5. CONGESTION-AWARE SEARCH

Typically, ICN proposals have advised using the strategy layer to route to the nearest replica [4]. However, the nearest replica is not guaranteed to be the one to which the requester has the highest throughput path or equivalently the smallest download delay. In this section, we thus describe a simple scoped-flooding protocol that locates requested cached content with minimum download delay. If cached content is not found with the scope of the flood, the content-requesting node routes a standard interest request for content towards the custodian, as described earlier. In our evaluation in the following section, we will compare content-aware search with a simple nearest-cache policy.

Our search policy operates as follows. Content requester R begins by flooding an interest request to all surrounding neighbors. These neighbors, in turn, flood the packet, so on, until a boundary, in terms of number of hops, or link weight, is reached. The weight of link l might correspond to the inverse of B_l, in which case the scoped flood would stop when a link of sufficiently low available bandwidth is encountered. A link may be assigned a high initial weight, preventing any queries from being sent over it. In this sense, the search is scoped, because it does not flood the entire network.

Any node, N, containing the requested file sends an interest reply message that is forwarded back to the requester along the reverse path, accumulating the minimum value of B_l of all links on the path back to R, in a manner similar to that discussed in the previous section. R then receives all replies, and selects the source to which it has the best connection (i.e., the maximum of the minimum throughputs). In this fashion, a user may retrieve content from a node that is further away in terms of hop count, but to which a better connection exists. Formally, if B_L is the set of effective capacities of the links lying on the path between R and N, then $min(B_L)$ is the effective throughput between R and N. The node N_B to which the requester has the best throughput, and thus the one selected to serve the content, is defined as:

$$N_B = max_N(min_l(B_L))$$

Caching Policy Interactions: Interesting interactions arise from combining a congestion-aware caching policy with a congestion-aware search policy. Most notably, the policies complement each other in two ways:

1. Cached content with high utility values would be discoverable even if the cache was not on the shortest path from the requester to the custodian; with search, con-

tent that ordinarily could only be downloaded slowly over congested network links, from a distant custodian, could now be quickly obtained.

2. If content is found via search and is delivered over an uncongested path, then the congestion-aware policy would likely not cache the content anywhere along the path, due to the low computed utility value. This would leave space for other, higher utility content to be cached, that could not be found nearby via search.

Together, these two properties provide a form of implicit coordination among nearby caches, preventing redundant caching not just along a path, but also across a group of nearby nodes.

6. PERFORMANCE EVALUATION

To evaluate the different caching and routing policies, we built a discrete-event simulator. An event is any action that has the potential of changing the throughput of existing flows. For example, a flow entering due to an exogenous interest-request arrival or exiting the network when requested content has been downloaded can impact the fair-share throughput of other flows, and introduce or remove bottlenecks. Additionally, changes in the routing (which will occur in the hybrid MANET scenarios we consider) can also change flow throughput. Before every new event, all existing flows are drained based on the time passed since the last event and the previously-calculated flow rate. Then, the rate of every flow is recomputed in the simulator based on the new network state. The key notion here is that flow rate is discretized, and does not change in between events. Our flow rate estimation is described in Algorithm 1.

We evaluate our caching and routing policies on simple network topologies such as grid and scale-free as well as real network topologies (Rocketfuel). We also perform simulations on a hybrid network consisting of a MANET and a cellular infrastructure to demonstrate the broad applicability of our cache management and request-routing policies across a variety of scenarios.

We assume that exogenous requests arrive at every node in the network according to a Poisson process with rate λ. Unless otherwise stated, we assume that content popularity follows a Zipfian distribution with rate $\alpha = 0.8$ and cache size $C = 500$. We consider a content universe of size $F = 10000$. Custodians are placed randomly throughout the network for each trial. Content unit size varies from 0.5 Mb to 3.0 Mb. We assume that shortest path routes are computed using Dijkstra's shortest-path (lowest-weight) algorithm. Error bars in our results correspond to 90% confidence intervals.

We evaluate the following set of cache management policies:

- **TERC+LRU**: Transparent en-route caching (TERC) is a common caching mechanism used in the ICN literature [4]. This policy caches everywhere along the path, and uses LRU as the cache eviction algorithm.
- **BTW+LRU**: The policy described in [1], using betweenness centrality to cache at the most central node along the download path, and LRU as the cache eviction algorithm.
- **CAC-E2E**: Our congestion-aware caching policy, considering the end-to-end path congestion.
- **CAC-UP**: Our congestion-aware caching policy, considering only upstream throughput.

We will also evaluate our cache management policies both with, and without, the following search policies.

- **SEARCH-CNG**: Congestion-based search, as described in Section 4.
- **SEARCH-HOP**: This policy finds the closest cache containing the content with respect to hop count. The search is also scoped (as described in Section 4)

Algorithm 1 Download Flow Rate Calculation

1: Determine the set of active flows, and mark each as unfinalized.
2: Determine the set of all links being utilized by those flows.
3: Create a list of effective link capacities, initialized to the set of maximum link capacities.
4: Find the bottleneck link - the utilized link with the lowest value of effective link capacity divided by number of unfinalized flows at that link (i.e., the link with the lowest fair share throughput for remaining flows.)
5: Find all unfinalized flows traversing the bottleneck link, and set the throughput of each of these flows to be the effective capacity of the bottleneck link.
6: Mark those flows as finalized.
7: For every link being traversed by any of those flows, reduce its effective capacity by that of the bottleneck link; this effectively allows all finalized flows to receive their fair share bandwidth.
8: Repeat Steps 4-8 until all flows have been finalized. At this point, the download throughputs of all flows have been determined.

6.1 Grid Topology

We begin by examining our caching policy in the context of a 10x10 grid topology. Two kinds of links that exist in the network: high capacity at 10 Mbps, denoted by solid lines, and low capacity at 2 Mbps, denoted by dotted lines in Figure 2. The grid is essentially split into two poorly-connected halves. Figure 2 shows the fraction of cached content across the entire topology whose custodian resides in the bottom-left cluster C, encompassed by the grey box.

Figure 3 shows the average download delay, average hop count, and average hit rate vs. arrival rate for content across all nodes. The higher the arrival rate, the more congested links become; the more flows in the system, the more they must complete for bandwidth. This is especially true in the case of links with a low initial capacity. Figure 2 shows TERC+LRU caches the files in C relatively uniformly throughout the network, with slightly higher concentrations in the cluster itself. This is due to the cache everywhere approach. Our congestion-aware caching scheme demonstrates an interesting property. The nodes on the top half of the grid with low-capacity links all tend to cache a high percentage of content originating from C. On the contrary, nodes on the bottom half of the grid, connected to the weak links cache little relatively from C. This is a direct consequence of our caching scheme; it is congestion-aware. Our policy assigns a high utility value to content traversing highly congested, low-capacity links. Likewise, it assigns a relatively low value

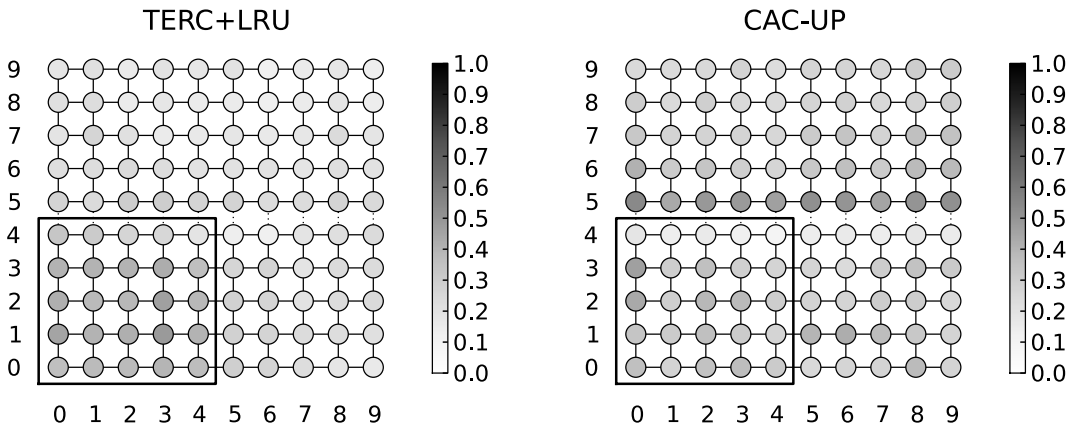
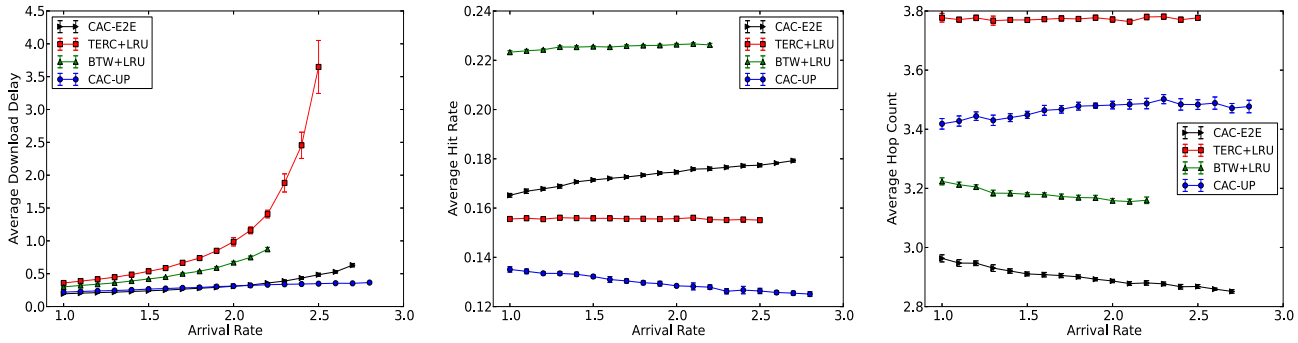


Figure 2: A heatmap of percentage of content cached per node belonging to the custodians located within the grey box. The dotted links denote the poor connectivity between the top and bottom half of the grid.



(a) Average Download Delay (b) Average Hit Rate (c) Average Hop Count

Figure 3: Grid network performance with increasing λ

to content that is obtained over uncongested, high-capacity links, making such content easy to evict. Together, these two properties work to not only cache, but to also retain, content that takes a long time to download from the custodian.

Figure 3 presents the average download delay, hit rate, and hop count for varying λ. As λ increases, the links in the network become more congested. As seen in Figure 3(a), the delay for TERC+LRU increases at the fastest rate. CAC-E2E and CAC-UP perform well even for higher arrival rates. It is also very instructive to note results in Figures 3(b) and 3(c). Note BTW+LRU has the *highest* average hit rate, but not the lowest delay — as in the case of our simple example in Section 2, *a higher hit rate does not necessarily translate to better download delay performance.* Although topology-aware, BTW+LRU does not consider network state, and thus makes no effort to cache content with respect to congested links. CAC-E2E and CAC-UP cache with respect to available link bandwidth, resulting in fewer flows traversing the low-capacity links and a consequently lower average delay.

6.2 Scale-free Network

We consider a scale-free network, such as the one considered in [1]. Here, there are 100 nodes, each link has a capacity of 10 Mbps, and the content unit size is 1 Mb. Even in the case of homogeneous link capacity, effective throughput can vary greatly across links. A more central link will naturally have more flows traversing it. Within the Internet, central links are often provisioned with higher capacity to account for this. However, in a wireless or mesh network, all links may have identical capacity regardless of position. In this case, more central links become more congested, and are best avoided. Therefore, a congestion-aware caching policy can still provide gains even in the case of homogeneous link capacity. Figure 4 presents delay and hit rate as λ increases. As in the grid case, BTW+LRU provide the highest hit rate, but CAC-E2E and CAC-UP provide the lowest content-download delay.

6.3 Rocketfuel Topologies

Borrowing from the methodology of [2], we use topologies generated by Rocketfuel [6] (Sprint and Tiscali) to measure performance on a realistic, Internet-scale backbone network. The Rocketfuel topologies provide latencies and link weights inferred from measurement. We use the inverse of the link weights to estimate the relative link capacities, as network operators often use the inverse of link capacity to set link weight. Links in the two networks range from 10 Mbps to 100 Mbps. Here we set the content size to 3 Mb. In Figures 5 and 6, we vary λ. We can see that for realistic backbone topologies, the trend holds; BTW+LRU and TERC+LRU

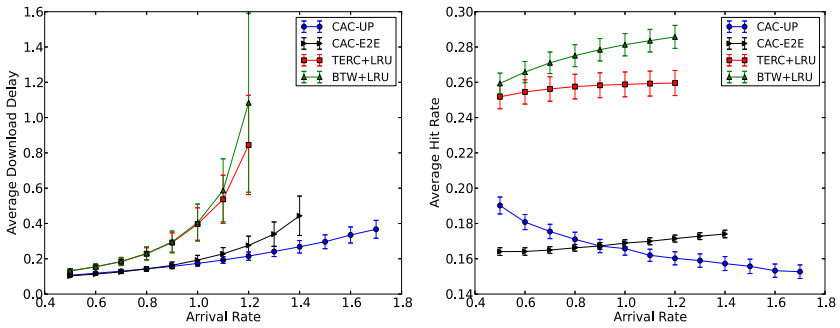


Figure 4: Scale-free download delay and hit rate with increasing λ

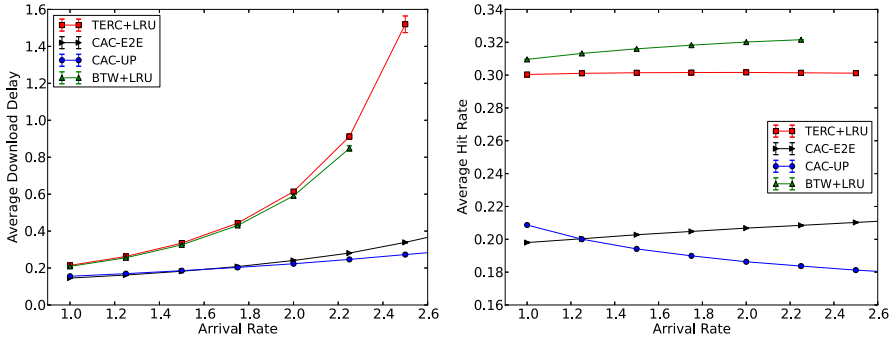


Figure 5: Sprint (US) topology download delay and hit rate with increasing λ

have high hit rates, but the average delay for both CAC policies is significantly lower. In Figures 7 and 8 we vary C and α, and set $\lambda = 1.0$. Across a variety of cache sizes, ranging from 0.1% to 1.0% of the content universe size, we see that the results remain the same; congestion-aware schemes yield lower delay. As α gets larger, we see the policies converge; as popularity becomes more skewed, it becomes more important to user-centric performance. We observed this in Section 2. We suspect that at higher arrival rates, as congestion plays a larger role, the policies again diverge.

Figures 9 and 10 show the performance of our proposed search policies (both scoped to two hops) as λ increases. Note that for both TERC+LRU and CAC-UP, using SEARCH-HOP results in a lower hop count. However, using SEARCH-CNG yields a lower average delay for both policies. Note that CAC-UP, using either search policy, outperforms TERC+LRU. We attribute this to the coordination described in Section 5. TERC+LRU, augmented with a search policy, would not necessarily have the same coordinative properties; although search would allow requesters to find content nearby, the caching would not be coordinated. Content that could be downloaded quickly would be cached everywhere along the path, creating unnecessary redundancy.

6.4 Hybrid Network

We simulate a Hybrid network, that is, a MANET where all nodes are connected to a base station. Nodes are grouped into clusters, and each cluster is either stationary or mobile over the course of the entire scenario. We use a random-waypoint mobility model to simulate cluster movement. Therefore, clusters merge and separate over time. In the worst case, a cluster may become completely disconnected from all others, and must rely solely on the base station for connectivity. In this scenario, there are 100 nodes total, grouped into 10 clusters, 5 of which are mobile. Here, the content size is 0.5 Mb. The capacity of MANET links is 10 Mbps, while the capacity of the wireless base station links is 2 Mbps. For this reason, the base station, while allowing for connectivity where there would otherwise be none, is not a desirable resource; the base station weight is set to be 5x higher than that of the MANET links. Due to the scoped aspect of the search policies, neither sends search queries over the base station (although it is still used if the content is not found via search, and shortest-path routing so dictates).

Again, CAC-E2E and CAC-UP perform better than TERC+LRU and BTW+LRU, as our congestion-aware policies tend to cache content received over lower capacity links. Naturally, there is higher in variation in delay due to node mobility. As was true for the Rocketfuel evaluation, both congestion-aware policies perform better than both TERC policies when search is added. Here, the search policies perform fairly similarly; this is because inter-cluster congestion is low, and there is not a large difference between selecting the closest node, or the one to which the throughput is greatest.

7. DISCUSSION

Our initial exploration has demonstrated the benefits of designing caching and search policies based on congestion, but a number of open research questions remain.

In the design of our cache management policy we have considered the available capacity of the bottleneck link on an end-to-end path to decide whether content should be cached.

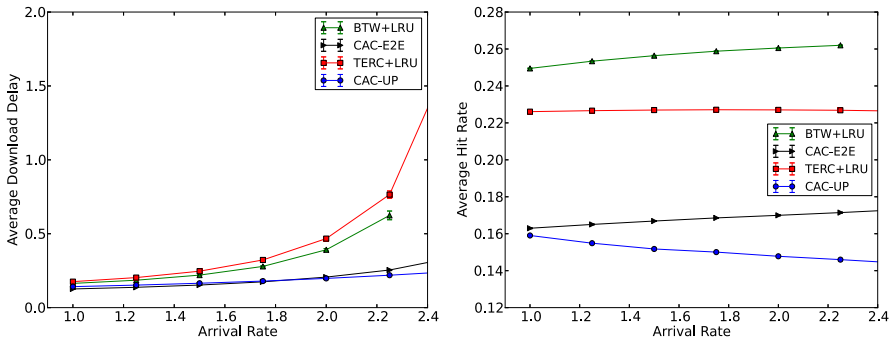


Figure 6: Tiscali (EU) topology download delay and hit rate with increasing λ

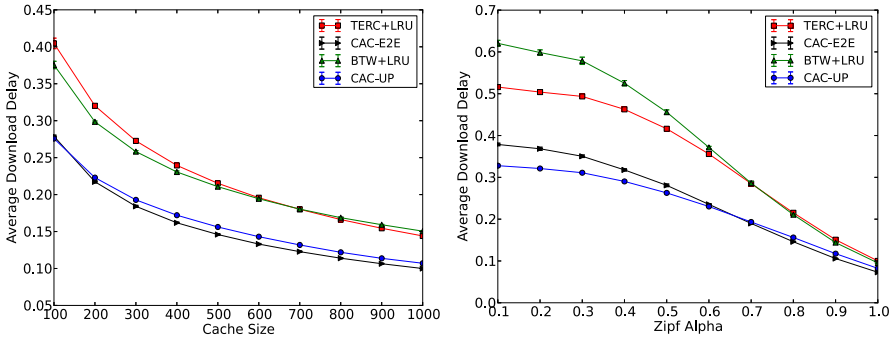


Figure 7: Sprint (US) topology download delay with increasing C and α.

We also experimented by only considering congestion on the upstream path from a node and observed that it provides very similar performance to the scheme that considers end-to-end path congestion. Though this observation needs to be validated through additional experimentation, we conjecture that this is due to cross streams; what is downstream for one node may be upstream for another, and in that manner, the congestion of links in all parts of the network is still factored into utility values of the upstream-only policy.

In our current model, the utility of a cached content does not vary over time. Therefore, a previously hard to obtain content may never be evicted from a cache, regardless of how easy to obtain it becomes in the future. This situation is more likely to arise in a mobile scenario where groups of nodes merge and split over time. Therefore, as part of our future work we plan to incorporate a notion of aging content periodically in the design of our cache policy. We have observed that a naive approach of decreasing the utility of cached content by a constant factor periodically does not perform well in our simulations.

One lesson we have learnt from this work is that congestion and caching are closely coupled; one directly influences the other. Though our long term objective is to design jointly optimal caching and routing policies for ICN, as a first attempt we aim to adopt a fixed point approximation approach. The approximation can start with an initial caching policy, determine the congestion, and then utilize the congestion to update the caching policy. The iterative process converges once the average congestion falls below a threshold.

8. RELATED WORK

Previous works have proposed caching schemes for ICN, leveraging various network properties to increase performance. In [1], the authors use betweenness centrality to make caching decisions, placing content at the point where they expect it to receive the most cache hits. In [9], the authors estimate the caching capacity of a path to fair-share cache space appropriately. The authors of [7] describe a policy which uses a coordinated age parameter to evict content. While not a caching scheme, [11] examines the impact of heterogenous cache sizes in ICN. In [8], the authors model bandwidth and cache sharing, and then use their model to estimate download delay.

Routing is another challenge of ICN. In [10], the authors propose a best-effort content location scheme that uses previous requests to estimate where content may reside. In [12], the authors develop a coordinated scheme to directly route users to replicated content.

Web proxy caching is similar in nature to ICN, and two works have examined non-traditional metrics as a basis for caching. In [13], the authors demonstrate that considering delay, and not just hit rate, is important to caching policies. In [15], the authors create a policy that estimates delay, and bases caching decisions on their estimation.

Several works propose caching policies that exploit topological and popularity factors in their caching decision. While [7] and [2] evaluate ICNs with respect to delay and congestion, to the best of our knowledge, no prior works have directly examined the relationship between hit rate, hop count, and end-user delay. The authors of [13] and [15] propose factoring delay into caching decisions. In [5],

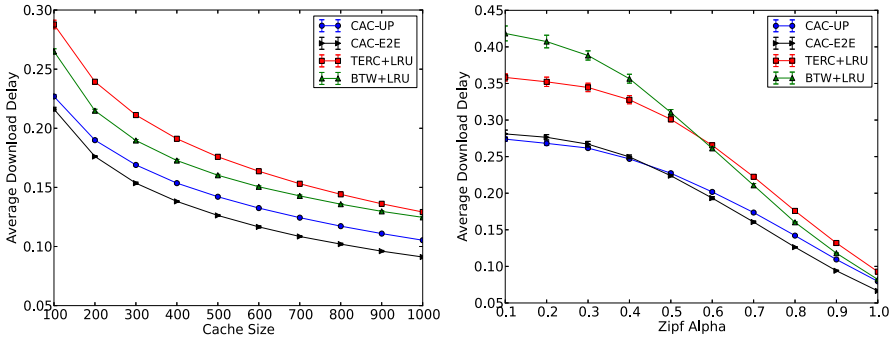


Figure 8: Tiscali (EU) topology download delay with increasing C and α.

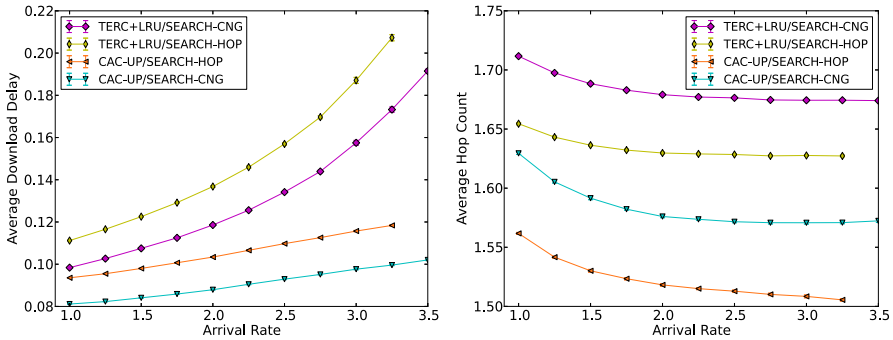


Figure 9: Sprint (US) topology download delay and hop count with increasing λ (using search)

the authors propose a caching algorithm factoring in overall popularity, and not just temporal locality, in caching decisions. However, no policies have been proposed to cache and route based on end-to-end congestion measurements.

9. CONCLUSION

In this paper, we have investigated network cache management and content-request routing policies that account for path-level (content-server to content-requestor) congestion and file popularity in order to directly minimize user-centric, content-download delay. Through simulation of several congestion aware congestion-aware cache management and content-request routing policies, and for a number of different network topologies, we showed that these content-aware policies yield significantly better download delay performance than existing policies, even though these existing policies provide better performance according to traditional metrics such as cache hit rate and hop count. Our finding that existing policies that provide superior performance according to network-centric performance metrics do *not* do so for user-centric metrics, makes a compelling case for considering user-centric performance metrics in designing and evaluating future ICN and other network protocols. Our future work includes investigating a more explicit coupling between local search and custodian-targeted routing of content requests, and an evaluation of our approaches in the presence of temporally-correlated content requests.

10. REFERENCES

[1] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks. IFIP'12, pages 27–40, Berlin, Heidelberg, 2012. Springer-Verlag.

[2] 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 '13, pages 147–158, New York, NY, USA, 2013. ACM.

[3] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: Seeing the forest for the trees. HotNets-X, pages 1:1–1:6, New York, NY, USA, 2011. ACM.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[5] S. Jin and A. Bestavros. Popularity-aware greedy dual-size web proxy caching algorithms. ICDCS '00, pages 254–, Washington, DC, USA, 2000. IEEE Computer Society.

[6] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. IMW '02, pages 231–236, New York, NY, USA, 2002. ACM.

[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] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and storage sharing performance in

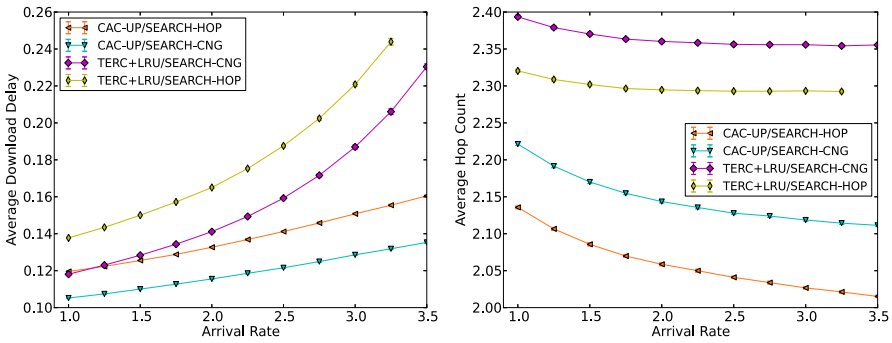


Figure 10: Tiscali (EU) topology download delay and hop count with increasing λ (using search)

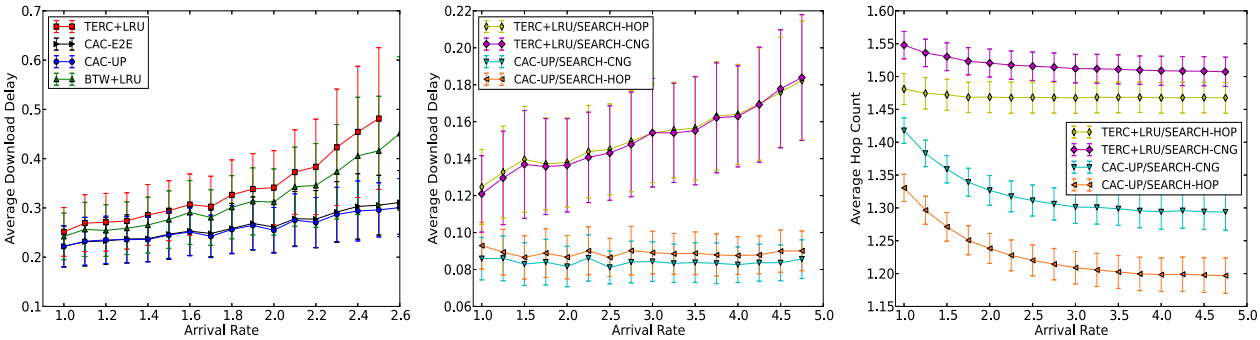


Figure 11: Hybrid MANET download delay (with and without search) and hop count with increasing λ

information centric networking. ICN '11, pages 26–31, New York, NY, USA, 2011. ACM.

[9] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. ICN '12, pages 55–60, New York, NY, USA, 2012. ACM.

[10] E. J. Rosensweig and J. Kurose. Breadcrumbs: Efficient, best-effort content location in cache networks. In *INFOCOM*, pages 2631–2635. IEEE, 2009.

[11] D. Rossi and G. Rossini. On sizing ccn content stores by exploiting topological information. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 280–285, March 2012.

[12] L. Saino, I. Psaras, and G. Pavlou. Hash-routing schemes for information centric networking. ICN '13, pages 27–32, New York, NY, USA, 2013. ACM.

[13] P. Scheuermann, J. Shim, and R. Vingralek. A case for delay-conscious caching of web documents. *Comput. Netw. ISDN Syst.*, 29(8-13):997–1005, Sept. 1997.

[14] Y. Wang, N. Rozhnova, A. Narayanan, D. Oran, and I. Rhee. An improved hop-by-hop interest shaper for congestion control in named data networking. *Computer Communication Review*, 43(4):55–60, 2013.

[15] R. P. Wooster and M. Abrams. Proxy caching that estimates page load delays. *Comput. Netw. ISDN Syst.*, 29(8-13):977–986, Sept. 1997.