CCndnS: A strategy for spreading content and decoupling NDN caches

Mostafa Rezazad Y.C. Tay
National University of Singapore

***Abstract*—Many proposals for information-centric networking (ICN) share the idea of in-network caching. This idea has four issues: (1) Memory capacity can be wasted through caching redundant copies and unpopular content. (2) Memory latency is high because the caches must be large. (3) Traffic filtering can result in high miss rates in the core and load imbalance. (4) Performance coupling among caches makes modeling their behavior intractable.**

CCndnS is a caching strategy for Named Data Networking that segments each file and spreads them among the caches, thus addressing the above issues: (1) It reduces redundant copies and cache pollution by unpopular content. (2) It reduces the number of futile checks on caches, thus reducing the delay from memory accesses. (3) It increases hit rates in the core without reducing hit rates at the edge (thus improving overall hit rates) and balances the load among caches. (4) It decouples the caches, so there is a simple analytical performance model for the network of caches.

The efficacy of CCndnS and the accuracy of the model are validated with simulations using an Abilene-like topology.

I. Introduction

Named Data Networking (NDN[1]) aims to make a paradigm shift in the Internet architecture, from the current location-based addressing scheme to one that is content-based. Specifically, a content file (email message, software binary, mobile TV, etc.) is divided into **Data chunks** that are individually addressable by name.

A **client** retrieves a chunk by sending an **Interest** packet that specifies the name of the Data. The routing scheme forwards the Interest towards the **source**, where the chunk can be found. Along the way, the Interest may find its Data cached at a router; the chunk then follows the reverse path taken by the Interest to reach the client.

A router along the reverse path may decide to cache a copy of the Data. This is why Data can be found in the routers, away from its source. The routers thus become a **network of caches**, so there is **in-network caching**. There are several issues with this idea:

($I1$) A chunk may be cached at multiple routers, and routers may cache unpopular chunks that are rarely hit. Both are a waste of caching capacity.

($I2$) The number of content files in the Internet is huge, and many are large (e.g. video). To hold a significant fraction of these files, the caches will also have to be large, and therefore use cheaper and slower technology (e.g. DRAM instead of SRAM) [20]. The need to access this memory (to check if a chunk is there, or insert a chunk) thus slows down the router. This delay can increase exponentially if traffic is heavy and queues build up.

($I3$) A router that caches some popular Data will **filter** out Interests for this Data, and only forward Interests for unpopular Data to downstream routers. If content popularity has a long tail, then chances are slim that the filtered Interests will find their Data at some intermediate router before the source [9]. Routers in the **core** of the network thus cache unpopular content ($I1$). The resulting imbalance in workload degrades performance (e.g. longer memory queues at the edge).

($I4$) Filtering ($I3$) causes a cache to affect its downstream caches. This coupling makes it difficult to construct a mathematical model for analyzing how cache sizes, Interest rates, etc. affect caching performance.

This paper presents **CCndnS**, a content caching strategy for NDN that is a modification of **CCndn** [24]. CCndnS divides a file into **segments**, each larger than a chunk and smaller than a file. (This subdivision is consistent with segmentation in HTTP streaming.) The chunks in a segment are cached together in the same router, but different segments may be cached at different routers. A file is thus spread across multiple routers. CCndnS addresses the above issues in the following ways:

($I'1$) The caching strategy reduces the number of redundant copies of a chunk. This frees up space in the core for caching more chunks for popular files, thus squeezing unpopular files out of caches in the core (see Fig. 1).

($I'2$) CCndnS allows an Interest to **skip** a cache, i.e. pass through a router without checking if its Data is there. This reduces cache misses and shortens memory queues, thus reducing router latency.

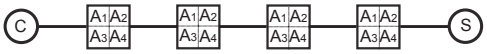
($I'3$) CCndnS raises hit rates for caches in the core, so they are better utilized. This increases the chance that an Interest can find its Data before reaching the source, and balances the workload among caches in the core and at the edge.

($I'4$) Skipping decouples the dependency among caches. This makes it possible to model aggregate behavior of the network with some simple equations. Our model is a key contribution for this paper, as the analysis of the filtering effect is known to be intractable [14].

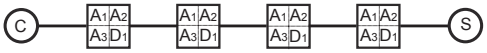
Sec. II begins by describing how CCndn spreads a file. Sec. III then describes how Interests can skip router checks under CCndnS. The model for CCndnS is derived in Sec. IV. Along the way, we present simulation results to examine the caching strategy and validate the model. Sec. V reviews related work before Sec. VI concludes with a summary.

[1] http://www.named-data.net/

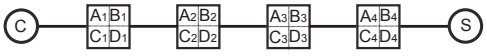
ISBN 978-3-901882-68-5



(a) All caches have the same content with default NDN caching



(b) A chunk of an unpopular file D_1 replaces chunk A_4 of a popular file from all caches.



(c) With CCndnS, we can cache chunks from 4 popular files in the network.

Fig. 1: CCndnS can reduce redundancy and increase the number of distinct chunks in the network between client C and source S. (a) shows redundant caching of 1 popular file, while (c) shows nonredundant caching of 4 popular files.

$N_{\mathcal{F}}^{\text{chunk}}$	number of chunks in a file $\mathcal{F}$
$N_{\text{seg}}^{\text{chunk}}$	number of chunks in a segment
S	number of segments in a file
H	CCndnS bound on hop count for spreading a file
$m_{\mathcal{F}}$	$\lceil S/(H-1) \rceil$ for file $\mathcal{F}$
$P_{\text{router}}^{\text{hit}}$	hit probability at a router
$P_{\text{net}}^{\text{hit}}$	probability that Data is found in a router, not at the source
N_{copies}	number of copies of a chunk among the routers
N_{hops}	number of hops made by Interest before finding Data

TABLE I: Summary of Notation

II. CCNDN: SPREADING CONTENT

We first describe CCndn in Sec. II-A. Sec. II-B then presents results from experiments on CCndn behavior.

A. CCndn: Description

In NDN, clients request content by sending an Interest that explicitly addresses a Data chunk of a content file. The content may be an email, a voice call, etc., but, for convenience, we refer to them as **files**.

Each NDN router has a **content store** (CS) for caching Data chunks. CCndn is a caching policy that specifies which router should cache a chunk and how chunks should be replaced.

By default, NDN caches a chunk at every router that the chunk visits [11]. This redundancy wastes caching capacity. Moreover, hits at edge routers filter out their Interests, so copies in the core get few hits [9].

A possible improvement is to selectively cache popular content at the edge, and unpopular ones in the core. Doing so with file popularity is impractical, since that requires ranking a large number of files on the fly, as new files are added and popularity shifts. Moreover, the popular files can be large, and so squeeze out other files at the edge. Instead, CCndn spreads a file over multiple routers along the discovered path from client to source, with the head nearer the client and the tail nearer the source. We have three reasons for doing so:

(i) Previous measurements have shown that users tend to abort downloads [29], [34]. so chunks at the head of a file are more popular than those at the tail. CCndn is thus consistent with the well-known observation that popular content eventually saturates the edge in a network of caches [8], [9], [12]. Such saturation happens even with LRU (least recently used) replacement; however, LRU allows an unpopular chunk to displace a popular chunk at the edge.

(ii) Cache space that is close to clients is precious. By caching only the head at the edge, this valuable resource can be shared by more files.

(iii) For content like video, user-perceived latency depends on how soon the application can have enough chunks to get started. Retrieval delay for other chunks can be hidden by the time it takes for the application to consume the head; there is a similar idea in file layout for solid state disks [10].

To spread a file, CCndn divides a file into segments. The chunks for a segment are cached at the same router.

How should a file be segmented? We could use a hop count h to divide a file into $h-1$ segments, and cache segment i at router i along the path from client to source. However, if two clients with different hop counts request the same file, they would thus use different segment sizes and cannot share segments. Instead, CCndn fixes the number of segments S, so each file $\mathcal{F}$ has S segments, with size

$$N_{\text{seg}}^{\text{chunk}} = \lceil N_{\mathcal{F}}^{\text{chunk}}/S \rceil, \tag{1}$$

where $N_{\text{seg}}^{\text{chunk}}$ is the number of chunks per segment and $N_{\mathcal{F}}^{\text{chunk}}$ is the number of chunks per file. (Table I is a summary of our notation.)

CCndn uses another parameter H that does not exceed the smallest hop count between client and source. Suppose an Interest $\mathcal{I}$ from client $\mathcal{C}$ takes a path through routers $\mathcal{R}_1, \ldots, \mathcal{R}_{H-1}, \ldots$ to reach the source, where $\mathcal{R}_i$ is i hops from $\mathcal{C}$. Let $m_{\mathcal{F}} = \lceil S/(H-1) \rceil$. Then $\mathcal{R}_1$ caches segments $1, 2, \ldots, m_{\mathcal{F}}$; $\mathcal{R}_2$ caches segments $m_{\mathcal{F}}+1, \ldots, 2m_{\mathcal{F}}$, etc.

H is a bound on the number of routers for spreading the S segments. If $S \leq H-1$, then only the first S routers cache 1 segment each. Although H can be as large as the hop count from client to source, a smaller H will keep file $\mathcal{F}$'s tail away from the edge at the source, so that premium space can be used by clients there. Sec. II-B3 will examine tuning for S and H.

How does a router know which segments it should cache? There is no addressing scheme for locating a router in NDN. However, since the path taken by a chunk exactly reverses the path taken by its Interest, we can use hop count to identify a router, as follows:

An Interest $\mathcal{I}$ from a client $\mathcal{C}$ searches the CS in every router along its path to the source. $\mathcal{I}$ keeps track of the hop count from $\mathcal{C}$ in its search. $\mathcal{I}$ may hit its Data along the way and thus not reach the source $\mathcal{Z}$. If $\mathcal{I}$ reaches $\mathcal{Z}$, $\mathcal{Z}$ knows the hop count h from $\mathcal{C}$. It then chooses an appropriate H value and calculates $m_{\mathcal{F}}$. $\mathcal{Z}$ then puts h and i in each chunk's header. As the chunk passes through a router on the reverse path to $\mathcal{C}$,

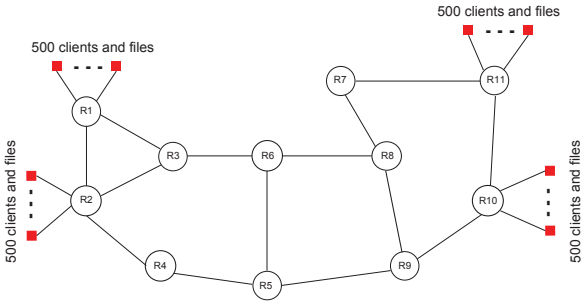


Fig. 2: Abilene topology and its clients set-up.

h is decremented. This value becomes i at router $\mathcal{R}_i$, so $\mathcal{R}_i$ knows it has to cache that chunk; other routers do not cache that chunk as it passes through (this is also true if Data is retrieved from an intermediate router).

When a cache is full, CCndn uses LRU for chunk (not file) replacement. This policy is close to optimal in practice [27], but we will later suggest a modification.

Most of CCndn's processing overhead (e.g. router assignment) is at the source. The routers just need to increment and compare hop count, possibly using separate hardware.

B. CCndn: Experiments

We now present experiments for tuning CCndn and comparing it to other caching policies.

1) Performance metrics: One obvious performance metric for a content caching strategy is the probability $P^{\text{hit}}_{\text{router}}$ of hitting a router cache, i.e. an Interest finds its Data in that CS. We also use the following metrics to measure a strategy's performance over the entire network of caches:

network hit probability $P^{\text{hit}}_{\text{net}}$: the probability that an Interest finds its Data at some intermediate router, instead of the source; a strategy with a higher $P^{\text{hit}}_{\text{net}}$ is more effective in using the caches to relieve load on the source.

redundancy N_{copies}: the average number of copies per Data chunk among the routers; a strategy with a smaller N_{copies} reduces memory wastage and can cache a bigger variety of chunks in the network.

hop count N_{hops}: the average number of hops before an Interest finds its Data; a strategy with a smaller N_{hops} has a smaller latency for retrieving Data, and reduces bandwidth and power consumption.

2) Simulation set-up: Many experiments in the literature on network of caches use a tree topology [12], [15], [19], [22]. Such networks do not test a caching strategy's effectiveness for multipaths and cross traffic. Others use large random graphs [4], [25], but these make it difficult to control the experiments and analyze their results. Instead, we choose a realistic, Abilene-like topology[2], as shown in Fig. 2.

We use a purpose-built simulator and attach clients only at $R1, R2, R10$ and $R11$, so these are edge routers; the other 7 are core routers. Each edge router has 500 clients, NDN aims to support user-generated — instead of server-based — content, so we have each client generate a file that has a geometrically-distributed size of average 500 chunks. The **catalog** of all files thus has about $4 \times 500 \times 500 = 1$ million chunks.

We do not model download abortion (Sec. II-A), as we expect it will improve CCndn performance — abortion shortens a file's tail and frees up cache space near the source.

Unless otherwise stated, routers have the same CS size, which we vary from 1K to 25K chunks. This is 0.1% to 2.5% of the catalog size, comparable to the 5% used by Fayazbakhsh et al. [8]. We also follow these authors in using a request-level simulator that does not model details like router queues and TCP congestion control.

A client sends Interests for a file at a constant rate. The time between the first Interests of any two files is exponentially distributed. This means that, sometimes, a client may be downloading multiple files concurrently. The clients at $R1$ and $R2$ request files from clients at $R10$ and $R11$, and vice versa, so there are $2 \times 2 \times 2 = 8$ flows. There is cross traffic in all routers because of Interests flow to and fro.

Interests are routed via the shortest path to the source. If there are multiple shortest paths from a router R to the source, R multicasts the first Interest of the file to all of them; R then chooses the one that brings back the Data chunk first, and uses that for the rest of the simulation [35].

As is common practice, we use a Zipf distribution with parameter α to model the popularity of files [3].

In the plots below, each data point is the average of 5 runs but, to avoid clutter, we omit error bars.

3) Tuning S and H: CCndn has two parameters, S (number of segments) and H (for hop count), to be tuned to suit the workload and topology. What should these values be for our experimental set-up? Fig. 3 plots network hit probability $P^{\text{hit}}_{\text{net}}$ and hop distance N_{hops} for $H = 7$.

For Zipf $\alpha = 2.5$, Fig. 3(a) shows that $P^{\text{hit}}_{\text{net}}$ quickly exceeds 0.9 when CS size increases. This is because the file popularity is heavily skewed, so the edge caches are large enough to contain the small number of popular files. In fact, Fig. 3(b) shows that, for $\alpha = 2.5$, N_{hops} is minimum when $S = 1$, i.e. the entire file can be cached in one router at the edge.

A caching network is of marginal interest if the popularity is so skewed that the caches at the edge suffice to achieve a high $P^{\text{hit}}_{\text{net}}$. Besides, it is well-known that file sizes have a long tail distribution. In particular, a recent study of traces from a content delivery network shows that files (text, images, multimedia, binaries, etc.) have sizes that fit a Zipf distribution with $\alpha \approx 1$ (specifically: 0.99, 0.92 and 1.04) [8].

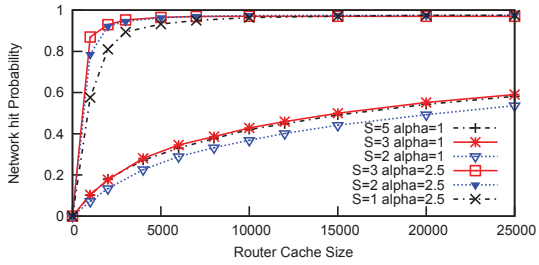
Henceforth, we focus on $\alpha = 1$.

For $\alpha = 1$, Fig. 3(a) shows that $S = 3$ and $S = 5$ have $P^{\text{hit}}_{\text{net}}$ higher than $S = 2$, whereas $S = 2$ and $S = 3$ have smaller N_{hops} than $S = 5$. These suggest that $S = 3$ is the best choice for this topology and workload.

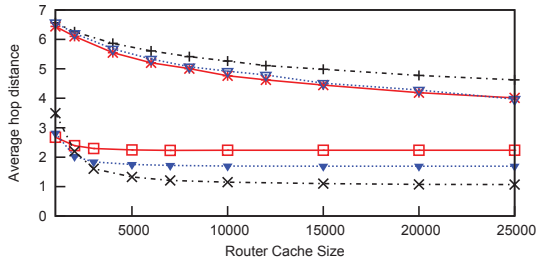
Note that $H = 7$ is about the number of hops between requesters and sources in Fig. 2. With $S = 3$, each file is spread over 3 routers, so the tails avoid contending for cache space with the heads. In fact, $H = 4$ will work just as well.

Our experimental set-up is relatively symmetrical (same number of clients per edge router, same popularity distribution, etc.), so $H = 3$ is intuitively right. In general, where there is

[2] http://nic.onenet.net/technical/category5/core_ topology.htm

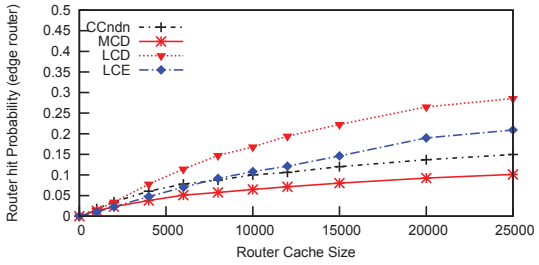


(a) Network hit probability $P_{\text{net}}^{\text{hit}}$

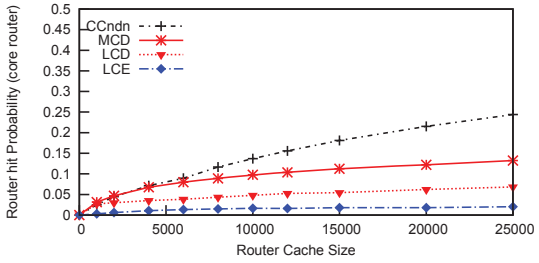


(b) Distance to content N_{hops}

Fig. 3: CCndn performance for $H = 7$, Zipf $\alpha = 1$ and $\alpha = 2.5$. All routers have the same CS size.



(a) CS hit probability for an edge router (R1)



(b) CS hit probability for a core router (R6)

Fig. 4: Comparing the strategies' $P_{\text{router}}^{\text{hit}}$ for edge and core routers ($S = 3$).

more asymmetry, the best value for H will depend on the traffic and topological imbalance.

Since each router caches $m_{\mathcal{F}} = \lceil S/(H-1) \rceil$ segments, why not just set $S = H - 1$, so $m_{\mathcal{F}} = 1$? Recall that different clients may have different hop counts to the source, so they have different best values for H. On the other hand, segment size S must be fixed, so clients with different hop counts can share segments.

S and H should therefore be set independently. We will revisit S and H tuning after introducing CCndnS.

4) Comparison with other strategies: There are several proposals for how a network of caches can cooperate to improve performance [18], [31] that one can use to compare with CCndn. However, cooperative caching requires information exchange that would significantly slow down NDN packet traffic [4]. We therefore consider three simpler strategies (the names follow terminology for hierarchical caches [14]):

LCE (Leave Copy Everywhere): A copy of the Data is cached at every router along the path traced by its Interest; this is the default strategy for NDN [11].

LCD (Leave Copy Down): If an Interest finds its Data after i hops, a copy of the Data is cached at hop $i - 1$.

MCD (Move Copy Down): If an Interest finds its Data after i hops, a copy of the Data is cached at hop $i - 1$, and the chunk is deleted from hop i, unless that is the source.

WAVE [6] is a newer scheme that is similar to LCD and MCD in performance.

Fig. 4 compares the $P_{\text{router}}^{\text{hit}}$ of CCndn, LCE, LCD and MCD for $S = 3$. Fig. 4(a) shows that, for an edge router ($R1$), $P_{\text{router}}^{\text{hit}}$ is higher for LCE and LCD than for CCndn. This is because they cache the most popular files in their entirety at the router, whereas CCndn only caches the heads.

For the core router $R6$, however, Fig. 4(b) shows that CCndn has higher $P_{\text{router}}^{\text{hit}}$ than all three alternatives. In fact, a comparison of Fig. 4(a) and Fig. 4(b) shows that $P_{\text{router}}^{\text{hit}}$ for CCndn at the core $R6$ is *even higher than at the edge* $R1$. This is because $R6$ gets hits from clients at both $R1$ and $R2$ for the same files, whereas $R1$ only gets hits from its own clients.

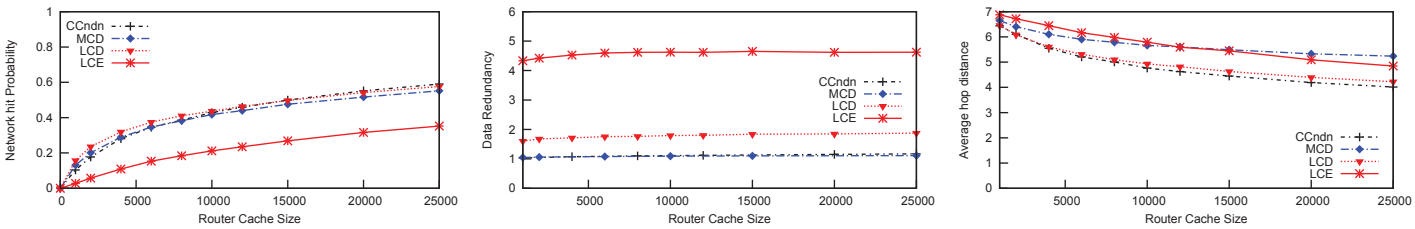
Fig. 5 shows how router hits translate into aggregated network performance: While CCndn, LCD and MCD have similar $P_{\text{net}}^{\text{hit}}$, CCndn has smaller N_{copies} than LCD and smaller N_{hops} than MCD.

Chai et al. noted that hop reduction does not imply higher net hit probability [4]; indeed, Fig. 5 shows that, although CCndn has a significantly smaller average hop distance than LCE and MCD, its $P_{\text{net}}^{\text{hit}}$ value is much higher than that for LCE but similar to that for MCD.

In Fig. 5, LCD has an advantage over CCndn in that the edge caches are occupied by popular content for LCD, but are polluted with unpopular content for CCndn. One simple improvement for CCndn is to adopt the **SLRU** policy previously proposed for disk systems [13]. (SLRU stands for *segmented* LRU, but "segment" in SLRU has a different meaning.)

In our SLRU implementation, a newly arriving chunk c that is to be inserted into the cache does not occupy the head of the LRU list. Instead, if this list has ℓ positions, then c is inserted at position $0.9 \times \ell$, near the tail. Later insertions then push c down this LRU list. If c is not hit, it gets pushed out of the list (i.e. replaced from the cache) sooner than other chunks; if c is hit, then it goes to the top of the list, like vanilla LRU. In this way, unpopular chunks stay in the cache for a shorter time than popular chunks.

Fig. 6 shows that LCD has similar performance if it uses SLRU instead of LRU. For CCndn, however, using SLRU



(a) Network hit probability $P_{\text{net}}^{\text{hit}}$ (b) Data redundancy N_{copies} (c) Average distance to content N_{hops}

Fig. 5: Comparing CCndn (S=3) with alternatives.

makes a significant (up to 60%) difference, resulting in a clear performance improvement over LCD.

III. CCndnS: Decoupling Caches

CCndn uses a segment as the granularity for caching content. It follows that if an Interest finds chunk k of a file in router R_x, then the Interest for chunk $k+1$ should skip other routers and go directly to R_x. This is the key idea in CCndnS.

As before, Sec. III-A below describes CCndnS and Sec. III-B presents experiments for analyzing its behavior.

A. CCndnS: Description

In CCndn, an Interest searches every CS in its path for its Data. With CCndnS, an Interest can use hop count to skip this search for some routers, as follows:

(S1) The first Interest of a file checks every CS along its path to the Data source.

(S2) An Interest keeps track of hop count in its search. If it finds its Data at some *intermediate* router R_x, the Data copies the hop count from the Interest and carries it back to the client. The Interest for the next chunk of the file uses this hop count to skip CS checks, until it reaches R_x. It checks CS at R_x and if the chunk is not there, it checks every CS along the rest of the path to the source.

(S3) Suppose an Interest does not find its Data at intermediate routers and retrieves the chunk from the source $\mathcal{Z}$. $\mathcal{Z}$ puts in the header of this chunk the file size $N_{\mathcal{F}}^{\text{chunk}}$, the segment size $N_{\text{seg}}^{\text{chunk}}$, and the value i for the router R_i that should cache this Data. When the client receives this chunk, it learns hop count i and puts that in the Interest it sends out for the next chunk so, like in (S2), CS checking can be skipped for $R_1, \ldots, R_{i-1}$. If the Interest does not find its Data at R_i, it skips subsequent routers and retrieves from $\mathcal{Z}$.

(S4) There is an exception to (S3): When a client learns the segment size, it can determine if the next interest $\mathcal{I}$ is for the first chunk in a segment; if so, $\mathcal{I}$ returns to (S1) and checks every CS. This is because the next segment of a file may be cached at an *earlier* router, possibly because some other client has retrieved it.

To illustrate (S4), consider the topology in Fig. 7. Assume Client1 downloaded file $\mathcal{F}$, which has 6 segments $seg1, seg2, \ldots$ cached in the following way:
$seg1, seg2$ at $R1$; $seg3, seg4$ at $R2$; $seg5, seg6$ at $R3$; ...
Suppose Client2 also wants $\mathcal{F}$. Since its hop count to the source is different, the caching scheme for Client2 may be
$seg1, seg2, seg3$ at $R2$; $seg4, seg5, seg6$ at $R3$; ...
When Client2 sends the first Interest for $seg4$, it may expect to find it at $R3$ when, in fact, $seg4$ is cached at a closer router $R2$. This is why CCndnS requires the first Interest of a segment to search all CS in its path to the source.

Next, we use Fig. 7 to illustrate a separate issue in (S2) and (S3), namely **multipaths**. Assume a segment $segA$ has 4 chunks. When $R3$ forwards Client1's Interests towards the source, it may send the first two Interests to $R4$, then change its routing decision and send the next two Interests to $R5$. The first two chunks of $segA$ may thus be cached at $R6$, say, and the next two chunks are cached at $R7$.

Suppose Client2 now sends Interests for $segA$, and $R3$ forwards them via $R5$. It is possible that the Interests for the chunks at $R7$ skip that router. This situation happens because caching for $segA$ is disrupted by a change in routing decision at $R3$. We assume such incidents are rare.

As for the chunks of $segA$ cached at $R6$, Client2's Interests may miss them because $R3$ forwards them via $R5$. Such a miss can happen even if there is no skipping.

Should the Interest routing policy be changed to suit CCndnS? This is an issue that is outside the scope of this paper, but under consideration in our current work.

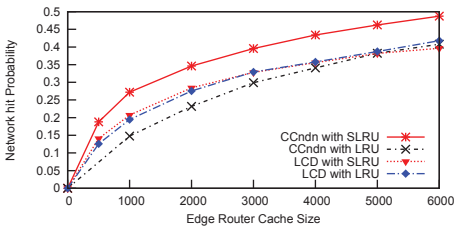
B. CCndnS: Experiments

By targeting a specific router that is likely to cache a desired chunk, CCndnS should reduce the probability of a miss. Indeed, Fig. 8 shows a big reduction in misses for both edge router $R1$ and core router $R6$.

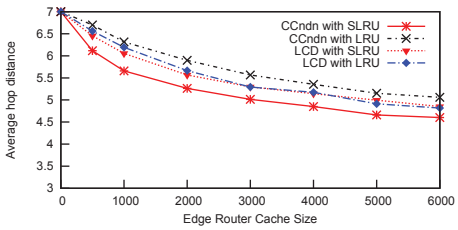
1) Hits: However, Fig. 9 shows that the network hit probability $P_{\text{net}}^{\text{hit}}$ is unaffected. This may seem unintuitive: How is it possible to drastically reduce router miss probability without affecting network hit probability? With a little derivation, this becomes clear:

Prob(check CS)+Prob(don't check CS)=1,
i.e. Prob(check CS)+Prob(skip)=1,
so Prob(check&hit)+Prob(check&miss) +Prob(skip)=1. Using conditional probability,
Prob(hit|check)Prob(check)+Prob(miss|check)Prob(check)
+Prob(skip)=1,

$$\text{so} \quad \text{Prob(hit|check)} = (1 - \text{Prob(skip)})/\text{Prob(check)} - \text{Prob(miss|check)}. \qquad (2)$$



(a) Network hit probability $P_{\text{net}}^{\text{hit}}$



(b) Distance to content N_{hops}

Fig. 6: Comparing CCndn and LCD, using LRU and SLRU ($S = 3$). $R3, R4, R7, R8$ and $R9$ have 50% more cache space, while $R5$ and $R6$ have 100% more, than the edge routers $R1, R2, R10$ and $R11$. LCE and MCD are omitted since LCD has better performance.

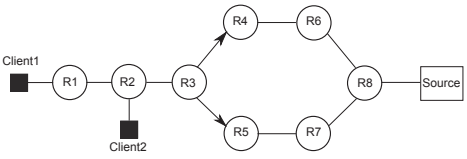


Fig. 7: An example of a multipath routing problem.

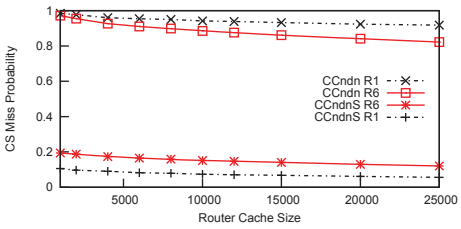


Fig. 8: Skipping drastically reduces miss probabilities at both edge and core routers. ($S = 5, H = 7$)

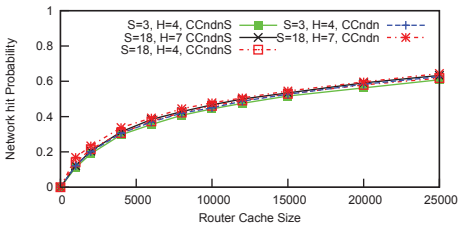


Fig. 9: Skipping does not affect $P_{\text{net}}^{\text{hit}}$.

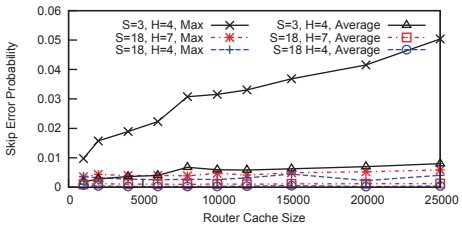


Fig. 10: Tuning S can drastically reduce maximum skip error.

Without skipping, we have Prob(skip)$= 0$ and Prob(check)$= 1$, and Eqn. (2) has the usual form $\text{Prob(hit)} = 1 - \text{Prob(miss)}$, so Prob(hit) is affected by any change in Prob(miss). With skipping, however, the increase in Prob(skip) is balanced by the decrease in Prob(miss|check) in Eqn. (2), so Prob(hit|check) is unaffected.

2) Skip errors: It is possible that an Interest does not check a CS when its Data is in fact there (e.g. brought in by cross traffic), thus committing a **skip error**. Fig. 10 shows that this error is less than 0.01 when averaged over all routers in our experiment.

However, Fig. 10 also shows that, for $S = 3$ and $H = 4$, the maximum (over all routers) is much larger than the average, and increases with CS size.

3) Tuning S and H: Interests from a client skip a CS for all non-first chunks in a segment, so any skip error is amplified by the size of a segment. One can thus decrease skip error by reducing segment size, i.e. increasing the number of segments S. Fig. 10 shows that, when S is increased from 3 to 18 for $H = 4$, the maximum skip error is drastically reduced.

However, there is a price to pay: increasing S spreads a file over more routers, thus increasing hop count N_{hops}; we can see this from Fig. 11, where S is increased from 3 ($H = 4$) to 18 ($H = 7$) .

To reduce maximum skip error by increasing S but without increasing N_{hops}, we can use H to limit the spread of the files. Fig. 11 shows that, when $H = 4$, increasing S from 3 to 18 does not increase N_{hops}; Fig. 9 also shows no change in $P_{\text{net}}^{\text{hit}}$.

4) Eliminating the filter effect: One fundamental issue with the default caching strategy for NDN is the filter effect, where only Interests that miss their Data at the edge enter the core. This can lead to traffic imbalance among the routers. Such an imbalance can seriously degrade performance, since traffic load has a nonlinear effect on queueing delay.

We see this imbalance in Fig. 12 where, under LCE, there is a wide range in the number of Interests received by different routers in the same simulation run. In contrast, except for $R2$, the routers receive a similar number of Interests under CCndnS. $R2$ is different because it is both an edge router and a core router, there are more flows running through it, it is assigned more chunks to cache, and so it receives more Interest traffic.

More importantly, filtering reduces the effectiveness of caches in the core. We ran an experiment where the core $R5$ and $R6$ each has a 20K cache, and all other routers have a 1K cache. Fig. 13 shows that, when the edge router $R1$ has its CS size increased to 30K, $P_{\text{router}}^{\text{hit}}$ for $R5$ steadily decreases for LCE. In contrast, for CCndnS, $P_{\text{router}}^{\text{hit}}$ for $R5$ is unaffected by the change in CS size at the edge.

CCndnS thus decouples the cache performance of edge and core routers.

IV. CCndnS: Analytical Model

The input to a cache in a network depends on the miss rates in upstream caches. This makes it very difficult to construct a mathematical model to analyze their performance. However, the skipping in CCndnS considerably weakens this dependency, as illustrated in Sec. III-B4. This decoupling makes it possible to model cache behavior with some simple equations.

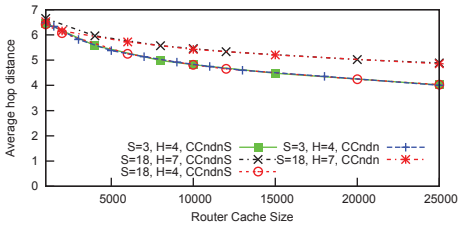


Fig. 11: Average distance to content N_{hops}.

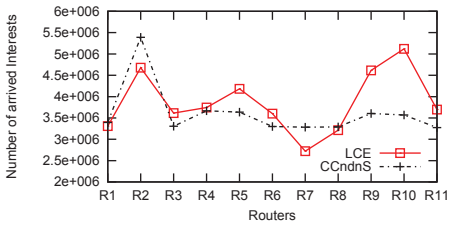


Fig. 12: CCndnS balances workload among edge and core routers, except for $R2$ (it is both core and edge).

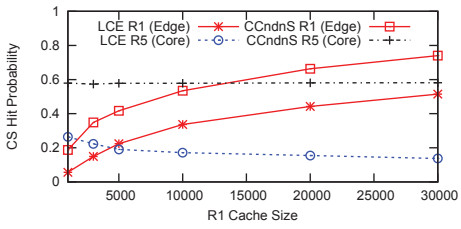


Fig. 13: Under CCndnS, changing CS size of edge router $R1$ does not affect $P^{\text{hit}}_{\text{router}}$ in the core.

We first derive equations for $P^{\text{hit}}_{\text{router}}$, then use it to compute the aggregate measures $P^{\text{hit}}_{\text{net}}$ and N_{hops}. In each case, we validate the model with the simulator.

A. Router Hit Probability $P^{\text{hit}}_{\text{router}}$

Consider router r. Let M_r = size of content store CS at router r, λ_r = arrival rate of Interests at router r and T_r = average time a Data chunk stays in router r.
By Little's Law [28],

$$M_r = \lambda_r T_r. \tag{3}$$

Under CCndnS, only a subset of the catalog is eligible for caching at router r; let C_r be the size of this subset.

Suppose the part of file k to be cached at router r is a fraction f_k of C_r. Let p_k be the probability that an arriving Interest requests file k. Let M_{rk} be the size of router r's CS that is occupied by chunks from file k. Then the probability of a cache hit at router r is

$$P^{\text{hit}}_{rk} = \frac{M_{rk}}{C_r f_k} \tag{4}$$

The Interest arrival rate for file k is $p_k\lambda_r$, so Little's Law gives, in steady state,

$$M_{rk} = \begin{cases} (p_k\lambda_r)T_r & \text{if } p_k\lambda_r T_r < C_r f_k \\ C_r f_k & \text{if } p_k\lambda_r T_r \geq C_r f_k \end{cases}$$

where the second case applies if M_r is so large that router r can accommodate all chunks of file k that is to be cached there. By Eqn. (3), we get

$$M_{rk} = \begin{cases} M_r p_k & \text{if } M_r p_k < C_r f_k \\ C_r f_k & \text{if } M_r p_k \geq C_r f_k \end{cases} \tag{5}$$

The probability of a hit at router r is, by Eqn. (4),

$$P^{\text{hit}}_r = \sum_k P^{\text{hit}}_{rk} p_k = \sum_k \frac{M_{rk}}{C_r f_k} p_k. \tag{6}$$

To validate Eqn. (6), we use the Abilene topology. Clients at $R1$ and $R10$ request each other's files, and similarly for $R2$ and $R11$, so cross traffic is heaviest at $R2$. The 500 files at each edge router have popularity distributed as Zipf with $\alpha = 1$, so $f_k = 1/500$, and $p_k = p_0/k$, where $p_0 = 1/\sum_{i=1}^{500} \frac{1}{i}$. Fig. 14 shows that the equation gives an excellent fit for the simulated $P^{\text{hit}}_{\text{router}}$ at both edge and core routers.

B. Network Hit Probability $P^{\text{hit}}_{\text{net}}$

To calculate an aggregated metric for the network, we need to determine Prob(check), i.e. the probability that an Interest checks a CS (see Sec. III-B1).

Consider a file $\mathcal{F}$ with $N^{\text{chunk}}_{\mathcal{F}}$ chunks, and a router r. How many of the $N^{\text{chunk}}_{\mathcal{F}}$ Interests will check the CS at router r?

If $S \leq H-1$, then router r caches at most 1 segment, which has $N^{\text{chunk}}_{\mathcal{F}}/S$ chunks; if $S > H-1$, then the router caches, on average, $N^{\text{chunk}}_{\mathcal{F}}/(H-1)$ chunks. Router r thus gets approximately $I_1 = N^{\text{chunk}}_{\mathcal{F}}/\min(S, H-1)$ Interests for the chunks that it caches.

In addition, CCndnS requires the Interest for the first chunk of a segment to probe every router; at worst, router r gets $I_2 = S$ Interests for this probing. We therefore estimate Prob(check) as $(I_1 + I_2)/N^{\text{chunk}}_{\mathcal{F}}$, i.e.

$$\text{Prob(check)} \approx \frac{1}{\min(S, H-1)} + \frac{S}{N^{\text{chunk}}_{\mathcal{F}}}. \tag{7}$$

This approximation does not model how Prob(check) depends on r (e.g. Interest for the first chunk of a segment checks every CS in its path only until it gets a hit); we assume the effect is minor.

Now suppose the route taken by Interests passes through routers $1, 2, \ldots, k$ before reaching the source. The network hit probability is then

$$P^{\text{hit}}_{\text{net}} = \sum_{r=1}^{k} \text{Prob(check)} P^{\text{hit}}_r. \tag{8}$$

Fig. 15 shows that this equation fits two very different sets of $P^{\text{hit}}_{\text{net}}$ measurements in our experiment.

C. Average Hop Count N_{hops}

Similarly, the average hop count is

$$N_{\text{hops}} = \sum_{r=1}^{k} r\,\text{Prob(check)} P^{\text{hit}}_r + (k+1)(1 - P^{\text{hit}}_{\text{net}}). \tag{9}$$

Fig. 16 shows that this equation also gives a good N_{hops} fit for the two sets of measurements in our experiment.

Our equations can thus accurately model both local ($P^{\text{hit}}_{\text{router}}$) and global ($P^{\text{hit}}_{\text{net}}$ and N_{hops}) performance. Our current work is on using these equations to partition the caches, and thus enforce Service Level Objectives for specific traffic classes.

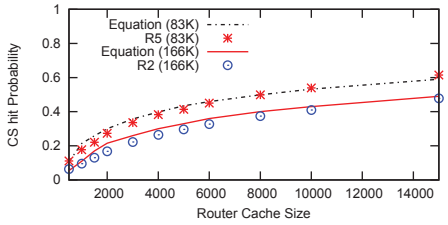


Fig. 14: Validating Eqn. (6) for $P_{\text{router}}^{\text{hit}}$ at edge router R_2 ($C_r = 166$K) and core router R_5 ($C_r = 83$K) for Abilene.

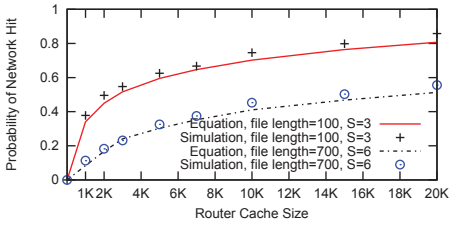


Fig. 15: Validating Eqn. (8) for $P_{\text{net}}^{\text{hit}}$ ($H = 7$).

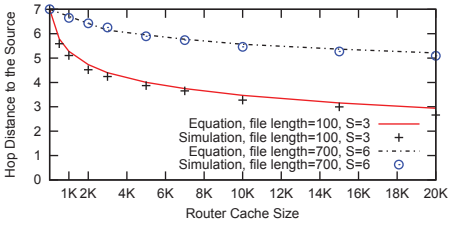


Fig. 16: Validating Eqn. (9) for N_{hops} ($H = 7$).

V. Related Work

NDN grew out of Jacobson et al.'s proposal on Content-Centric Networking [11], and is an example of information-centric networking (ICN). Recent work [1] has confirmed the cacheablity of current Internet traffic. However, caching performance depends on content popularity, caching policy, etc. [26]. In fact, there is some skepticism over whether ICN in general, and NDN in particular, are fundamentally feasible.

For Ghodsi et al., ICN's in-network caching is futile because most hits will occur only at the edge, where the popular files are cached [9]. Perino and Varvello's calculations also show that memory technology (e.g. size and speed) cannot match line speed at reasonable cost for NDN [20].

In this paper, we make no effort to examine whether NDN is feasible. Rather, the question we pose is: How can we improve in-network caching for NDN?

In-network caching resembles cooperative web caching. Several papers have proposed protocols for such cooperation [18], [23], but Wolman et al.'s analysis shows that they have marginal benefits for large populations [31].

However, that analysis uses traffic data from some 16 years ago, when the Web was quite different from today. In the case of on-demand TV, for example, Li and Simon recently show that their cooperative caching strategy can significantly reduce cross-domain traffic [17].

Without cooperation, caches at the edge of the network will get most of the hits and act as a filter. This effect is most obvious for hierarchical systems; it is known for web caching [5], and recently observed for CCN [12], [22]: Caches at the edge are dominated by popular files, while those in the core are occupied by unpopular content that get few hits. This is largely why Fayazbakhsh et al. believe that most of the benefit of in-network caching for ICN can be obtained with TCP/IP by putting all the caches at the edge [8].

On the other hand, Tyson et al.'s simulations with BitTorrent traces show that, if cache sizes are sufficiently large, then there is significant caching in Tier-2 Autonomous Systems using CCN instead of TCP/IP [30].

In the studies above, the files are cached in their entirety. CCndnS, however, segments the files and spreads them over multiple routers, thus removing the filtering effect. (Our use of equal-sized segments is consistent with segmentation in HTTP streaming.) Fig. 4 shows that this increases hits in the core, possibly even exceeding that at the edge if there is cross traffic.

One reason NDN's default en-route caching strategy leaves the core caches cold lies in the lack of cache diversity, i.e. the core contains copies of the content at the edge. One way to reduce this redundancy is for the caches to run some coordination protocol [32], [7]. The coordination may require the measurement of content popularity [16], [33]. Such schemes increase traffic overheads, add complexity to the routers, and may not adapt fast enough to changes in popularity.

Alternatively, this redundancy can be reduced by meta algorithms like LCD and MCD, as considered by Laoutaris et al. for hierarchical web caches [15]. WAVE [6] is a chunk placement policy that is similar to MCD; it also has a chunk marking window that is similar to a segment. These algorithms have the common problem of caching only popular content at the edge, leaving the caches in the core ineffective.

Another possibility is for a router to probabilistically decide whether to cache a chunk that is passing through, i.e. randomized redundancy reduction, like in ProbCache [21]. Arianfar et al. see probabilistic caching as a load-sharing strategy, having pointed out the difficulty for memory latency to match line speed [2]. Chai et al. also note that this need to work at line speed rules out any collaborative caching algorithms that require significant information exchange among the routers [4].

CCndnS does not require such control traffic. The Interests and Data chunks at most carry hop count, file size and segment size in their headers. Nonetheless, it spreads content so the load for checking CS is spread out (Fig. 12). This reduces queueing delays at the CS and significantly postpones router saturation [24]. The redundancy reduction also ensures that caching capacity in the core is not wasted. CCndnS thus partly addresses the line speed and edge/core issues raised by previous authors. Note that CCndnS cannot completely eliminate redundancy, since it is a caching strategy, whereas redundancy is partly determined by the routing algorithm.

The filtering effect that makes en-route caching ineffective also makes the analytical modeling of a caching network "extremely difficult" [25] and "far beyond the borders of tractability" [14]. This is why many models consider only hierarchical topologies [12], [22].

In contrast, by caching at the granularity of a segment, CCndnS facilitates skipping, diminishes the filtering effect, and decouples the behavior among caches (Fig. 13). We thus obtain a simple model (Sec. IV) that can be used to predict and analyze individual and aggregated router performance.

VI. Conclusion

This paper makes three contributions towards improving in-network caching for NDN:

(1) We show how, by spreading a file over multiple routers, CCndn's caching strategy overcomes the edge/core problem. Instead of caching unpopular content in the core that get few hits, spreading popular files raises hit rates in the core (Fig. 4(b)). Cache pollution by unpopular content can be further reduced by using SLRU instead of LRU, thus reducing its performance impact (Fig. 6). Redundant chunk caching in the core is reduced (Fig. 5(b)) because CCndn assigns chunks to routers.

(2) Since chunks are assigned to routers, an Interest can skip some CS checks. We show how this skipping idea helps CCndnS reduce router miss rates (Fig. 8) without hurting network hit rates (Fig. 9), and helps to even out Interest traffic among the routers (Fig. 12). Reducing misses and balancing traffic helps to reduce memory latency so the routers can sustain line rates.

(3) We present a simple analytical model for hits (Sec. IV) that is accurate for both edge and core routers (Fig. 14). It can also be used to calculate aggregate performance for the network (Fig. 15 and Fig. 16). The key to overcoming the modeling intractability lies in the skipping used by CCndnS, which decouples the caches (Fig. 13).

As with any networking protocol, CCndnS has parameters S and H that require tuning (Fig. 3, Fig. 10 and Fig. 11) to suit the topology and traffic.

The model can be used for resource allocation, e.g. memory or bandwidth assignment to traffic classes that have different service level objectives. This requires techniques for estimating model parameters (C_r, f_k, p_k) and adjusting them dynamically.

Space constraint prevents us from elaborating on tuning, allocation and estimation, as well as Interest routing (Sec. III-A). We will address them in future work.

References

[1] B. Ager, F. Schneider, J. Kim, and A. Feldmann. Revisiting cacheability in times of user generated content. In *Proc. INFOCOM Workshops*, pages 1–6, 2010.

[2] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *Proc. ReArch*, pages 5:1–5:6, 2010.

[3] L. Breslau, P. Cao, et al. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. INFOCOM*, pages 126–134, 1999.

[4] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks. In *Proc. IFIP Networking*, May 2012.

[5] H. Che, Y. Tung, and Z. Wang. Hierarchical Web caching systems: modeling, design and experimental results. *JSAC*, 20(7):1305–1314.

[6] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack. WAVE: Popularity-based and collaborative in-network caching for content-oriented networks. In *INFOCOM Workshops*, pages 316–321, 2012.

[7] L. Dong, D. Zhang, Y. Zhang, and D. Raychaudhuri. Optimal caching with content broadcast in cache-and-forward networks. In *Proc. ICC*, pages 1–5, June 2011.

[8] S. K. Fayazbakhsh, Y. Lin, et al. Less pain, most of the gain: Incrementally deployable ICN. In *SIGCOMM*, pages 147–158, 2013.

[9] A. Ghodsi, S. Shenker, T. Koponen, et al. Information-centric networking: seeing the forest for the trees. In *HotNets-X*, pages 1:1–1:6, 2011.

[10] D. Huang, X. Zhang, W. Shi, et al. LiU: Hiding disk access latency for HPC applications with a new SSD-enabled data layout. In *Proc. MASCOTS*, pages 111–120, 2013.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, et al. Networking named content. In *Proc. CoNEXT*, pages 1–12, 2009.

[12] Z. Jia, P. Zhang, J. Huang, et al. Modeling hierarchical caches in content-centric networks. In *Proc. ICCCN*, pages 1–7, 2013.

[13] R. Karedla, J. Love, and B. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, March 1994.

[14] N. Laoutaris, H. Che, and I. Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Perform. Eval.*, 63(7):609–634, 2006.

[15] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *Proc. IPCC*, pages 445–452, 2004.

[16] J. Li, H. Wu, B. Liu, et al. Popularity-driven coordinated caching in named data networking. In *Proc. ANCS*, pages 15–26, 2012.

[17] Z. Li and G. Simon. Time-shifted TV in content centric networks: The case for cooperative in-network caching. In *ICC*, pages 1–6, 2011.

[18] S. Michel, K. Nguyen, A. Rosenstein, et al. Adaptive web caching: Towards a new global caching architecture. *Comput. Netw. ISDN Syst.*, 30(22-23):2169–2177, Nov. 1998.

[19] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and storage sharing performance in information centric networking. In *Proc. ICN*, pages 26–31, 2011.

[20] D. Perino and M. Varvello. A reality check for content centric networking. In *Proc. ICN*, pages 44–49, 2011.

[21] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *Proc. ICN*, pages 55–60, 2012.

[22] I. Psaras, R. G. Clegg, R. Landa, et al. Modelling and evaluation of CCN-caching trees. In *Proc. IFIP Networking*, pages 78–91, 2011.

[23] M. Rabinovich, J. S. Chase, and S. Gadde. Not all hits are created equal: Cooperative proxy caching over a wide-area network. *Computer Networks*, 30(22-23):2253–2259, 1998.

[24] M. Rezazad and Y. C. Tay. ndn||mem: An architecture to alleviate the memory bottleneck for named data networking. In *Proc. CoNEXT Student Workshop*, pages 1–3, 2013.

[25] E. J. Rosensweig, J. Kurose, and D. Towsley. Approximate models for general cache networks. In *Proc. INFOCOM*, pages 1100–1108, 2010.

[26] D. Rossi and G. Rossini. Caching performance of content centric networks under multi-path routing (and more). Technical Report Telecom ParisTech, 2011.

[27] A. Sharma, A. Venkataramani, and R. K. Sitaraman. Distributing content simplifies ISP traffic engineering. In *Proc. SIGMETRICS*, pages 229–242, 2013.

[28] Y. C. Tay. *Analytical Performance Modeling for Computer Systems*. Morgan & Claypool, 2013.

[29] D. N. Tran, W. T. Ooi, and Y. C. Tay. SAX: A tool for studying congestion-induced surfer behavior. In *Proc. PAM*, 2006.

[30] G. Tyson, S. Kaune, S. Miles, et al. A trace-driven analysis of caching in content-centric networks. In *Proc. ICCCN*, pages 1–7, 2012.

[31] A. Wolman, M. Voelker, N. Sharma, et al. On the scale and performance of cooperative web proxy caching. In *Proc. SOSP*, pages 16–31, 1999.

[32] W. Wong, L. Wang, and J. Kangasharju. Neighborhood search and admission control in cooperative caching networks. In *Proc. GLOBECOM*, pages 2852–2858, Dec 2012.

[33] H. Wu, J. Li, Y. Wang, and B. Liu. EMC: The effective multi-path caching scheme for named data networking. In *Proc. ICCCN*, pages 1–7, July 2013.

[34] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proc. EuroSys*, pages 333–344, 2006.

[35] L. Zhang, A. Afanasyev, J. Burke, et al. Named Data Networking. *SIGCOMM Computer Communication Review*, 44(3):66–73, Jul 2014.