Neighborhood Search and Admission Control in Cooperative Caching Networks

Walter Wong*, Liang Wang†, Jussi Kangasharju†‡

*School of Electrical and Computer Engineering, University of Campinas, Brazil
†Department of Computer Science, University of Helsinki, Finland
‡Helsinki Institute for Information Technology, University of Helsinki, Finland

***Abstract*—In-network caching of content is a popular technique for eliminating redundant traffic from the network and improve the performance of network applications. In this paper we present a novel cooperative caching strategy to improve performance of in-network caches. Our cooperative scheme is composed of an admission policy for the incoming data and a content exchange protocol between neighbor network caches to improve the search zone. The admission policy enforces that a previously cached data is not unnecessary replicated in other caches, resulting in more space for new data. The content exchange protocol allows for exchange on cached data, increasing the hit rate for incoming requests. The benefits are twofold: first, we reduce the redundant content caching in the network, and second, we improve the hit rate by informing the content cached in the nearby caches. As a proof-of-concept, we have implemented a prototype and evaluated its performance using different large-scale topologies against standard non-cooperative caching algorithms. Our numerical results show that both admission and content exchange policies yield large performance gains over standard algorithms.**

I. Introduction

The hype with user generated content (UGC) such as Youtube videos and IPTV, has put the current Internet infrastructure under high burden. According to a Cisco survey [1], the global IP traffic is expected to grow four times from 2009 to 2014, approaching 64 Exabytes per month in 2014, and by that time, various forms of video (TV, video-on-demand, and P2P) will exceed 91% of the traffic.

Despite these predictions, ISPs lack incentives to upgrade their infrastructure, an issue known as the *middle mile* problem [2]. The middle mile is the infrastructure that interconnects the transit points between different ISPs, and ISPs do not have enough incentives to upgrade this infrastructure because they do not receive any direct revenue from that upgrade. On the flip side, ISPs have incentives to upgrade the *last mile*, which is the infrastructure that connects subscribers to the Internet. Also, content providers have incentives to upgrade the *first mile* to provide better service availability and user experience.

In order to reduce the immediate upgrade pressure in the infrastructure, ISPs have deployed Web caches [3] in their networks to reduce the redundant traffic passing through their networks. Caches provide a simple but effective storage mechanism to reduce the latency and bandwidth usage. Content delivery networks (CDNs) have been deployed in the Internet to improve the user experience by placing content in the client side of the network, i.e., in the same ISP or close to it, reducing the latency and eventual bandwidth bottlenecks between ISPs. There are two main incentives for the usage of caches in the Internet. First, storage prices have decreased substantially faster than bandwidth costs. Second, data consumption is time-correlated in the Internet, i.e., a given piece of data is produced once and consumed many times in the Internet following a Zipf probabilistic distribution [4]. Recent surveys [5], [6] confirm that a large portion of the network traffic is redundant and could easily be cached.

In this paper, we propose a cooperation protocol for network caches to improve the caching capacities. The general idea is to use content routers to provide routing and caching capabilities together in the network. Additionally, these content routers implement a novel admission control in the caches, described as *cached-bit*, and a content look up procedure described as neighbor search. The cached-bit strategy reduces the redundant data in the network through a bit set in the content header and the neighbor search procedure allows for content discovery in the neighbor caches, resulting in footprint reduction. As a proof-of-concept, we implemented the caching network together with these caching strategies and evaluated them experimentally through emulation using Rocketfuel topologies. We compare the bandwidth savings, cache hit and the control overhead against simpler models.

The organization of this paper is as follows. Section II describes background about in-network caching and Bloom filters. Section III proposes the routing mechanism based on exchanged cache digests. Section IV presents the prototype implementation and describes the evaluation in several scenarios. We present and discuss the results in Section V Section VI presents the related work and compares with our approach. Finally, Section VII summarizes the paper.

II. Background

The basic in-network caching mechanism is performed by a *content router* (CR) [7]. A CR is a data forwarder similar to a regular router, but it also has internal memory that can be used to store data in transit. The simplest model works as follows: first, for each data response, any CR on the path between a server and clients caches the data in their memory. Further requests can be served by the local copy in the CR. This model is simple and works just as a network storage mechanism with a simple income queue, presenting some benefits such as

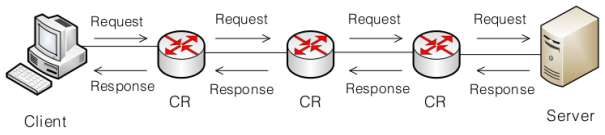


Fig. 1: Basic store-n-forward content router.

reduction in the content retrieval latency and bandwidth usage. Fig. 1 illustrates an example of CR in a network topology.

The simple CR model has limitations, especially in scenarios where they need to work as a single system. First, the usual admission policy is basically to *cache everything that is possible*, meaning that all in-transit data should be cached in the CR. However, CRs have limited storage capacity and they need more fine grained admission control policies to filter the insertion of new entries in the caches. Second, there is no cooperation between CRs, leading them to cache the same piece of content in the network. For example, a line of CRs between a client and a server will cache the same content, reducing the effectiveness of the caching mechanism. Therefore, we focus on two main topics: cache admission policies and cache cooperation strategies.

In [7], we proposed the general idea of a neighbor search mechanism that improves the cache hit ratio in cooperative caching networks. The idea is to allow queries to be diverted to neighbor caches, resulting in higher hit rate. However, this strategy alone does not improve the hit rate much due to the fact that all caches tends to have the same content in a linear path. Therefore, neighbor searching in CRs that have the same content results in poor performance. Another limitation of the old neighbor search strategy is that each CR holds a pointer to the CR that has the content, requiring additional memory to store neighbor information. In this paper we propose a new admission policy that improves the efficiency of storage space use and propose a new neighbor search algorithm combined with Bloom filters to store neighbor information. Bloom filters are space efficient structures that allows for aggregated neighbor information storage, resulting in higher hit ratio. Bloom filters are detailed below.

III. Cooperative Caching Strategy

Our cooperative caching strategy is composed of an admission policy and a content discovery protocol. We first describe the admission policy, followed by the neighbor search scheme with Bloom filters and the combination of both of them.

A. Admission Policy

An admission policy of a CR decides whether a piece of data should be cached or not. For network caching scenarios, we need a simple yet effective way to decide the admission since the CRs work as routers and need fast decisions. According to [4], the cache hit rate grows logarithmically with the cache size. Thus, it is interesting if we could aggregate the cache sizes by removing all the redundant content in these caches.

As a solution, we propose the *cached bit* admission control. The cached bit is a single bit set in the header informing

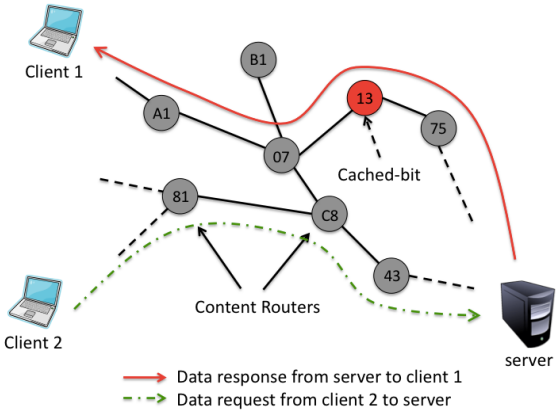


Fig. 2: Basic cache-and-forward with Cached Bit strategy.

whether a given piece of data has already been cached in the network or not. Whenever there is a message carrying some data, the first CR that caches the data also sets the bit in the header, informing further CRs along the path that piece of content has already been cached in the network. Therefore, other CRs know that they do not need to cache that piece of content again. The benefit of this approach is that the overall caching capacity can be improved in the network. Compared to the *cache all* admission policy, the cached bit can reduce the amount of redundant data in the network. The cached bit policy solves the previous limitation of the CR model in [7], where caches along the path had the same content. In this approach, just one CR has a given piece of content. Fig. 2 illustrates the basic CR model with the cached bit strategy.

In this example, the content server answers with data to client one (red line). The CR with $id = 13$ caches the data, thus, neither CR with $id = 07$ nor CR with $id = A1$ will cache the same data again. Thus, the cached bit reduces the amount of replicated data along a network path. Despite the improvement with the cached bit policy, we can see that if there is a second client (client 2) located in another edge of the topology, she will not benefit from the cached bit policy.

B. Neighbor Search with Bloom Filters

The Neighbor Search (NbS) strategy is a cooperative scheme to improve the hit rate in the cases shown in the above example where subsequent requests for the same content do not follow the path of the original request. This increases chances of finding the content and improves hit rate and reduces network traffic. The basic model is illustrated in Fig. 3.

Each CR has a *neighbor table*, where each entry contains a content ID and the interface where the data came from. Hence, upon receiving a content request, the CR looks up in the neighbor table for the entry, and if positive, it forwards to the interface where the data was last seen. We use *Content Bloom Filters* (CBF) to store content identifiers from neighboring CRs. Each CR broadcasts to its immediate neighbors a Bloom filter informing its currently cached content. Receiving neighbors add this CBF in their neighbor tables, allowing for opportunistic content routing towards CRs that have the content. The benefit of the augmented neighbor search model

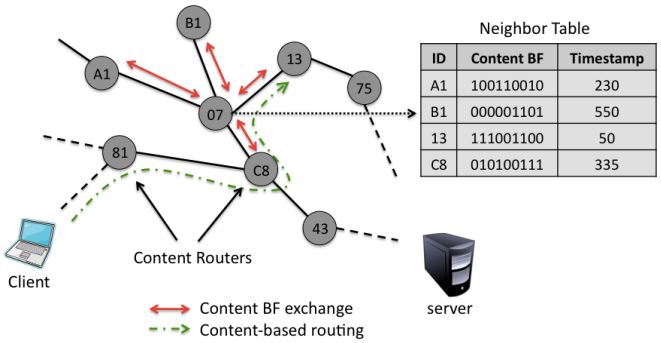


Fig. 3: Content BF distribution in a network.

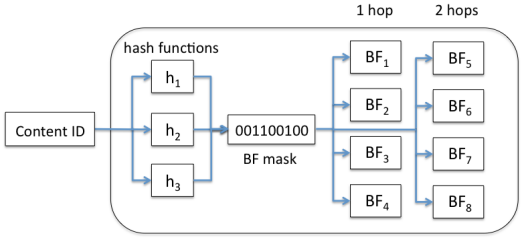


Fig. 4: CR neighbor look up table.

with Bloom filters is to allow re-directions towards CRs that may have the content with much high probability. Effectively this aggregates the storage of all neighbor CRs, and a request does not query a single CR, but a set of CRs in the network.

Fig. 4 illustrates how CBFs can be used to take the routing decisions. First, the content ID is hashed using the internal hash functions (three in this example). Then it will generate a BF mask that is going to be used against the neighbor CBFs stored in memory. In this example, the CRs stores the neighbor CBFs and neighbors-of-neighbors CBFs, indicated by the distance in the number of hops.[1]

CBFs have some drawbacks, *false misses* and *false hits*. False miss is when a piece of content is present in a CR, but the CBF does not reflect it. A false hit is when a CR does not contain the data anymore, but the CBF still has it.

C. Neighbor Search with Cached Bit

Combining the cached bit admission policy with the neighbor search cooperation strategy aims to improve the cache hit rate further. Cached bit reduces redundant data in the network. As the hit rate is logarithmically proportional to the cache size, the reduction of the replicated data increases the storage capacity and the hit rate as well. Neighbor Search increases the search space by aggregating a set of CRs in one larger storage. As consequence, a single query may implicitly go through several CRs for a cache hit. CRs work both cooperatively to reduce the redundant data and also increase the query space.

D. Refreshal Procedure

The CBF generation and refreshal is triggered by two events. First, at regular intervals, CRs generate CBFs and advertise them, for example, every second. The main benefit of this approach is that the overhead generated by the CBF exchange is known *a priori* and can be used in a network management system. CRs can also be configured to invalidate CBFs after that interval, thus, reducing problems with failovers. The drawback of this approach is that CRs do not analyze the data in-transit, for example, in-transit content refreshal. In this scenario, if the data changes too fast, the source CR is not able to update the CBF quickly to reflect its current state, increasing the number of false misses. Conversely, if changes are slow, there will be a number of unnecessary updates with small changes in the CBF.

Second, a CR sends an updated CBF when a certain number of events have happened such that, e.g., a given fraction of content in the CR is not represented in the CBF. The benefit of this approach is that it adapts to the real network conditions, for example, if suddenly the network changes due to popular content, it can quickly update its neighbors. However, this means that the amount of network traffic is unpredictable, since a flash crowd will trigger a lot of CBF updates. Fig. 3 illustrates an example of CBF broadcast to the immediate neighbors. The CRs populate their routing tables with the CBFs which are used in the routing decision towards the most likely location where the content is located.

E. Miscellaneous

The bootstrapping procedure is straightforward: CRs periodically broadcast CBF to its immediate neighbors, informing which content it has. Receiving CRs add an entry in the neighbor list and use it for further forwarding decisions. Failovers can also be detected in the same way. Each CBF has a validity associated and when it expires, the CR invalidates that entry and does not use that entry for the routing purposes.

Extending the query area represented by the CBFs would be possible when allowing CRs to forward CBFs from other CRs. This may result in routing loops so we would need to add identifiers in CBFs to indicate which is the original source CR so that CRs receiving duplicate CBFs would be able to drop them.

IV. Implementation & Evaluation

We have implemented a CR prototype in Python. Our evaluations are performed on self-developed experiment platform, the core of which is software-implemented router. Software-implemented router simulates a realistic router. On our experiment platform, each router and the links between them can be configured individually in terms of link bandwidth, delay, loss rate and so on. User modules like queuing policy and caching strategy can be easily plugged into router.

Fig. 5 shows the packet header used in the neighbor search mechanism. The header has six fields and it was designed to be aligned to 32 bits. The *type* field has 4 bits that defines the type of the message. The *TTL* has a 8-bit field defining the time-to-live of the message. The *loop-BF* is a 116-bit field containing a BF of all IDs of CRs that the message has been through, in order to prevent loops. The *CR_ID* is a 128-bit

[1]The number of hops can be tuned depending on the amount of memory available in the CRs.

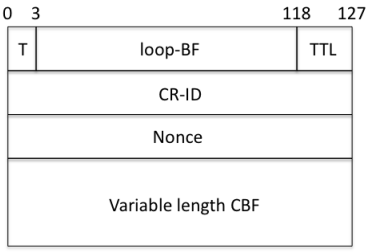


Fig. 5: Neighbor search header.

field containing the CR_ID of the source CR. The *nonce* is a 128-bit field containing a number that is used as message identifier between CRs, i.e., a receiving CR can distinguish the order of the received messages in order to update the neighbor table. The *CBF* is the content BF containing a set of content IDs aggregated in the BF, to be used as neighbor information.

A. Testbed Set-up

All the experiments are performed on our department cluster consisting of 240 Dell PowerEdge M610 nodes. Each node is equipped with 2 quad-core CPUs, 32GB memory, and connected to a 10-Gbit network. All the nodes run Ubuntu SMP with 2.6.32 kernel. The experiment platform will allocate necessary physical resources according to the simulated ISP network size. If ISP's network size is larger than the actual cluster size, routers will be multiplexed onto one node.

B. Methodology

We use the following metrics to evaluate our proposal:

- **Hit Rate:** What fraction of requests was served by CRs. Because all objects are of the same size, the hit rate is also the byte hit rate.
- **Average Number of Hops:** Average number of hops that a request need to go in the network before finding a cache or a server that holds a copy of the requested data.
- **Footprint Reduction:** Network footprint is the product of the amount of data and the network distance from which the data was retrieved. It measures the amount of internal traffic reduction, i.e., a smaller footprint (larger reduction) means less traffic within the ISP's network.

In the experiments, we used two POP-level topologies from real ISP networks [8], the Sprint and AT&T networks. For the deployment, in each network, we selected the top 20 routers with highest degrees as servers, and the rest as clients. The clients keep requesting data from the servers in the experiment, and the requests are uniformly distributed to different servers.

The request pattern follows Zipf distribution with $\alpha = 0.9$, which is very close to real-life distribution shown in [6]. We also tested two traffic patterns, one is the constant traffic rate, while the other follows a gravity model used in [9]. In gravity model, the fraction of the traffic from each client is determined by the city population. However, no significant difference in results has been found in the evaluation.

Our request trace requests chunks of content, which are assumed to be independent of each other (the popularity of chunks follows the above Zipf distribution). We assigned each CR with storage capacity of a certain number of chunks and assumed that every CR in the network had the same amount of storage. We report our results as a function of the per-CR storage, which at the largest sizes (1024 chunks) was around 1% of the total amount of content.

V. Experimental Results

We used the three metrics explained above: *hit rate, average number of hops* and *footprint reduction*. For the experiments, we use 4 different strategies as defined in Section III:

- **ALL**: no admission policy, CR caches everything
- **CachedBit**: one CR caches the content along a path
- **NbSA**: Neighbor search with ALL policy
- **NbSC**: Neighbor search with CachedBit admission policy

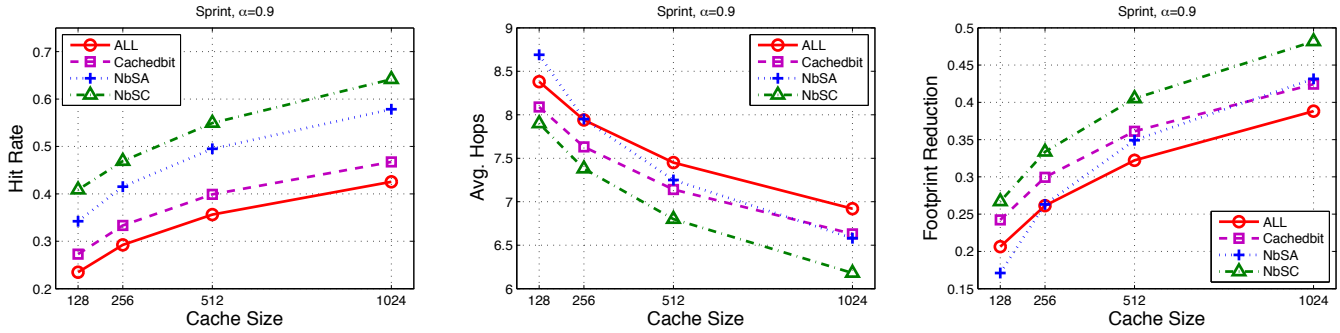
Figures 6 and 7 show the results for Sprint and AT&T networks, respectively. The graphs show hit rate, average hops, and footprint reduction. X-axis is the per-CR storage capacity.

Concerning hit rate, ALL strategy has the worst performance because all CRs along the same path store the same content. Therefore, if there is a miss in one CR, then, it is likely that all CRs in the same path will result in cache miss as well. CachedBit stores at most one copy per path, so it is better able to use the storage and a miss at one CR might be a hit in the next CR. NbSA strategy allows for content lookup within a set of CRs, resulting in a combination of a CR with all its neighbors, thus, increasing the search domain. As a result, it has a better performance than ALL and CachedBit. The best performance comes from the combination of CachedBit and Neighbor search since CachedBit reduces redundant data. Compared to ALL, NbSC can increase the hit ratio up to 50% at a small cost.

Concerning average hops, the same ranking between the strategies holds. One point of importance is the behavior of ALL and NbSA for small cache sizes. When encountering a miss, NbSA searches around for the content, but because of the small cache sizes, the other CRs are unlikely to have the content, hence the searching actually costs a lot of traffic. Therefore, *neighbor search strategies might not perform well for small caches*. However, it performs better as the cache size grows. Again, the combination of CachedBit and Neighbor Search results in the smallest number of hops.

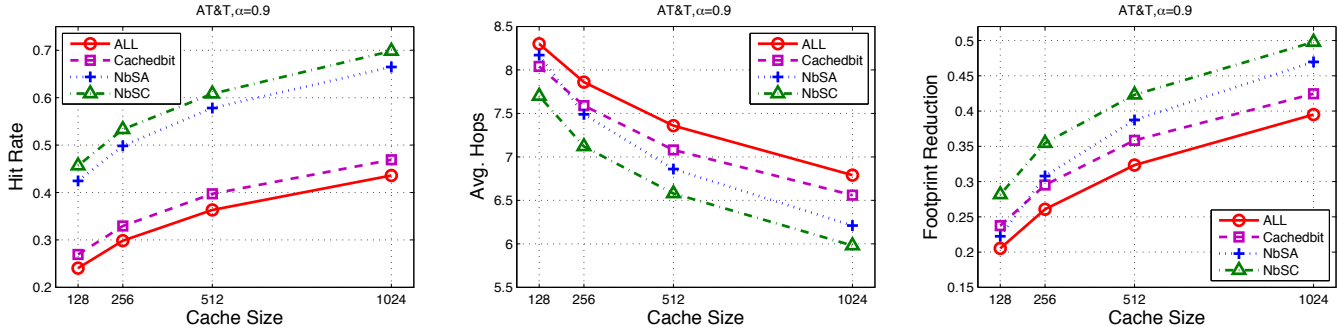
As for footprint reduction, we see the same behavior and ranking as with hit rate and average hops, including NbSA's weakness with small cache sizes. Again, the performance of NbSC is the best of them.

In the next set of experiments, we evaluate the performance of neighbor search versus how many hops away we look for cached content. Results are in Figures 8 and 9 for Sprint and AT&T, respectively. As expected, querying a larger set of neighbors yields a slightly higher hit rate, and at the cost of increased network traffic. However, hit rate improves much faster by increasing the per-CR storage than extending the search radius, thus there is little benefit from searching for content from far away. The ranking between NbSA and NbSC remains the same with NbSC having the advantage, even with



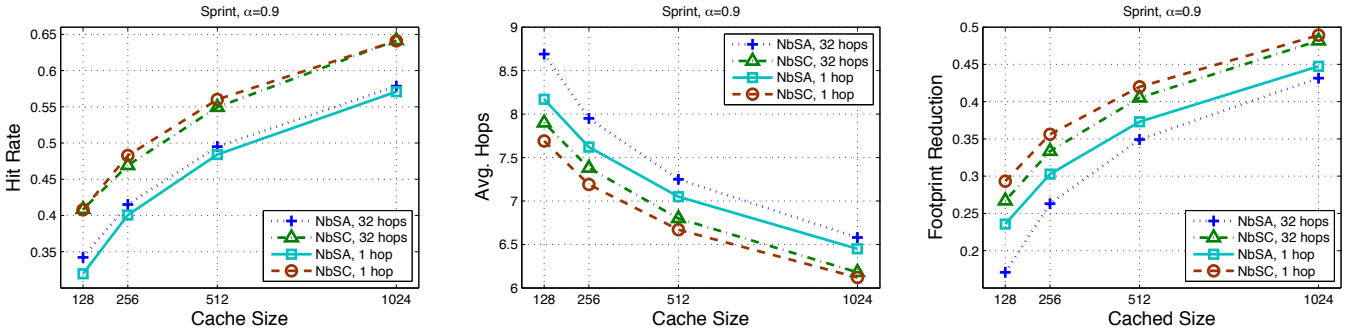
(a) Hit rate. (b) Average number of hops. (c) Footprint reduction.

Fig. 6: Comparison between ALL, CachedBit, NbSA and NbSC in the Sprint Network.



(a) Hit Rate (b) Average Number of Hops (c) Footprint Reduction

Fig. 7: Comparison between ALL, CachedBit, NbSA and NbSC in the AT&T Network.



(a) Hit rate. (b) Average number of hops. (c) Footprint reduction.

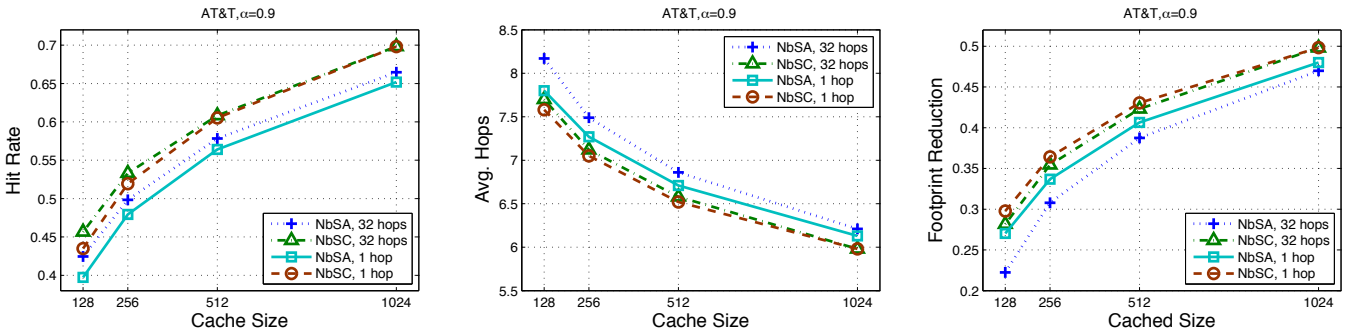
Fig. 8: Search zone effects for 1 hop and 32 hop search radii and both admission policies, Sprint network

the larger search radius. This underscores the importance of the admission policy.

In the next experiment, we analyze the impact of the Bloom filters and the false positive rate. All the previous experiments have been performed with a 1% false positive rate. False positives happen when a CR has a content Bloom filter that is unsynchronized with the source CR. Therefore, the CR forwards the request to a peer CR that actually does not have that content anymore, resulting in cache miss.

We experimented with 0.1%, 1%, and 5% false positive rates for both neighbor search mechanisms, with 1 hop search radius. We found out that the difference in performance between 0.1% and 1% false positive rates was negligible, i.e., it is not necessary to use very large bloom filters in an attempt to minimize the number of false positives. Going to a 5% false positive rate had a negative effect on performance, in particular on average hops and footprint reduction, but less so with hit rate. For example, for the AT&T network with CR cache size of 512 chunks, the footprint reduction was 40% and 24% for false positive rates of 0.1% and 5%, respectively.

The BF overhead is $m = -\frac{n \ln p}{(\ln 2)^2}$, for a given false positive rate p and number of entries n. Assuming a cache size C and



(a) AT&T Hit Rate (b) AT&T Average Number of Hops (c) AT&T Footprint Reduction

Fig. 9: Search zone effects for 1 hop and 32 hop search radii and both admission policies, AT&T network

an average file size F, the number of bits needed by a BF is equal to $m = -\frac{C \ln p}{F(\ln 2)^2}$. In terms of memory consumption q, namely the percent of cache needed to store the BF, is $q = \frac{km}{C} = -\frac{k \ln p}{F(\ln 2)^2}$, assuming each router has k neighbors on average. The formula shows that for larger file sizes, we will have smaller BF and vice-versa. Also, additional neighbors require more memory to store the neighbors' BFs.

Figure 10 expands on the results on effects of bloom filter false positive rate and search radius on hit rate and footprint reduction for NbSC in the Sprint network. Results for AT&T network and NbSA are similar and are not shown. The results show that the hit rate remains practically the same across different false positive rates but the footprint reduction drops as the conditions get worse. This is expected since a cache miss due to an unsynchronized Bloom filter (large false positive rate) will result in additional hops and it does not influence the content cached in the network. As consequence of the increase in the number of hops, the footprint reduction decreases.

Fig. 10b shows the comparison of hit ratio, footprint reduction as a function of the search radius. As the results show, the number of searched zones have a small effect when the search zone increases from one to two, but larger search zones result in a drop in footprint reduction, since we search content wider, but often without finding it.

A. Summary of Results

We summarize our main findings as follows:

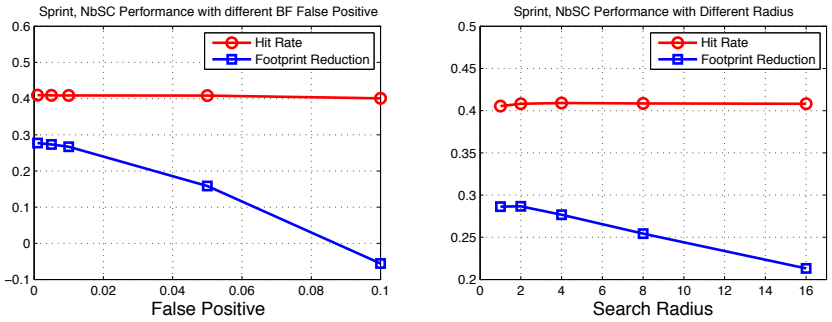
- Admission policy in CRs is very important in ensuring good performance. This is evidenced by the better performance of the strategies using the Cached Bit admission policy vs. the admit-all policy.
- Searching content in neighbor CRs is beneficial, but even a single hop search radius is typically sufficient for getting good gains and increasing search radius yields diminishing returns.
- Bloom filter size (and the associated false positive rate) is not critical and a false positive rate of 1% appears sufficient.

VI. Related Work

In [10], [11], the authors present an analytical model for data transfer, bandwidth and caching performance in information-centric networks. The proposed model uses a traffic generator with variable request rate for different parts of the same content, different caching capacities along the same path towards a piece of content and the LRU replacement policy within the network caches. Compared to our model, we use a simpler traffic generator that requests the content chunks based on a Zipf distribution, and our focus is on evaluating the admission policy and the caching strategy, leaving the analysis as future work. As our results show, admission policy is critical for good caching performance.

Internet Cache Protocol (ICP) [12] was proposed to increase the cooperation among Web-caches. Whenever a Web-cache receives a client query, it holds the query and multicasts an ICP message to all peer Web-caches to query for the requested content. Upon receiving the answer from some Web-cache, it stores locally a copy of the requested data and forwards to the data to the client. Although ICP could increase the hit ratio, the protocol increased the bandwidth usage due to the multicast query to all Web-cache peers. SummaryCache [13] was proposed to overcome ICP's limitation by reducing the number of queries that a Web-cache needed to send. Each Web-cache broadcasts a Bloom filter with its current content to all peers and upon receiving a query, it checks which Bloom filter has the content and sends just to those Web-caches. Compared to our model, CRs can not hold the request while it looks for content since it is in the network level. Hence, the ICP model can not be applied to our model, which is much more constrained in terms of storage and latency.

Content Centric Network (CCNx) [14] is a content-oriented communication model driven by *interests*. Content requests are identified by URLs and they are routed directly based on URLs towards the server. Caches on the path are able to identify these requests and answer with the requested data if they have cached, otherwise, they forwards the requests and mark their interest in the content. Despite the similar approach in the caching model, our proposal implements more efficient caching strategies with cached-bit and neighbor search to



(a) NbSC Performance vs. BF false positive rates in Sprint network.

(b) NbSC Performance vs. Radius.

Fig. 10: Performance comparison of NbSC vs. false positive rates and radius, 128 chunks of per-CR storage

increase the caching capacity and efficiency in the network. Our CR model is very close to CCN, but we do not explicitly consider how the requests are sent and interpreted by the CRs.

Content-centric Caching (CONIC) [15] is a clean-slate approach for content retrieval in caches based on Conic Routers (CR). Clients requesting data send requests to the servers and CRs on the path intercept the request and query other clients cache for the content. CONIC works for HTTP data and has look up latency associated for each request.

Cache-and-Forward (CNF) [16] is an in-network caching architecture where routers have a large amount of memory for storage. These routers perform content-aware caching, routing and forward *packet* requests based on location-independent identifiers. In contrast, we provide extra features to improve the cache hit in the network and also content search capabilities. Also, we are not restricted to HTTP as CNF is.

Cachecast [17] proposes a redundant traffic elimination technique in the multicast communication. They place small caches on links that inspects all frames and replaces with a shim header that represents the redundant data and the shim header is translated to the actual data in the destination. Our work is different since instead of removing redundant traffic, we assume that clients can use caches as alternative sources.

These approaches are architectural in nature, and do not investigate the effectiveness of the distributed caching from caching effectiveness point of view, which is our focus.

VII. Conclusion

In this paper we have presented an admission policy and a neighbor search mechanism for in-network caching architectures. Our admission policy attempts to ensure only a single cached copy of a piece of content, via informing other content routers via a bit in the header. The neighbor search strategy implements a cache cooperation protocol to exchange information about cached data between peer caches, allowing for content search in neighbor caches. Our key findings in the paper are that an admission policy is needed for good performance and that neighbor search is beneficial in finding content in nearby CRs, thus avoiding the need to retrieve the content from the origin. We have implemented and evaluated the admission policy and neighbor search protocol and the results show an increase of 33% on the hit ratio and decrease of 23% and 20% on the average number of hops and footprint reduction.

References

[1] [online] Cisco Virtual Network, "http://www.cisco.com/web/cy/ solutions/sp/sp_strategy," 2009.

[2] T. Leighton, "Improving performance on the internet," *Commun. ACM*, vol. 52, no. 2, pp. 44–51, 2009.

[3] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of web caching architectures: Hierarchical and distributed caching," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 404–418, 2001.

[4] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *IEEE INFOCOM*, pp. 126–134, 1999.

[5] B. Ager, F. Schneider, J. Kim, and A. Feldmann, "Revisiting cacheability in times of user generated content," in *IEEE Global Internet Symposium*, Mar. 2010.

[6] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *ACM SIGMETRICS*, 2009.

[7] W. Wong, M. Giraldi, M. Magalhaes, and J. Kangasharju, "Content routers: Fetching data on network path," *IEEE ICC*, June 2011.

[8] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 2 – 16, feb. 2004.

[9] A. Anand, V. Sekar, and A. Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 87–98, 2009.

[10] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proceedings of ITC*, 2011.

[11] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *ACM SIGCOMM ICN Workshop*, 2011.

[12] D. Wessels and K. Claffy, "RFC 2186: Internet Cache Protocol (ICP), version 2," Sept. 1997.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 281–293, June 2000.

[14] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM CoNext*, 2009.

[15] Y. Zhu, M. Chen, and N. Akihiro, "Conic: Content-oriented network with indexed caching," *Global Internet Symposium*, 2010.

[16] Paul, S. Yates, R. Raychaudhuri, D. Kurose, J., "The cache-and-forward network architecture for efficient mobile content delivery services in the future internet," *Innovations in NGN: Future Network and Services*, pp. 367–374, May 2008.

[17] P. Srebrny, T. Plagemann, V. Goebel, and A. Mauthe, "Cachecast: Eliminating redundant link traffic for single source multiple destination transfers," *ICDCS*, 2010.