Published in: IEEE Global Communications Conference (GLOBECOM), 2016.

Hybrid Memory Allocation for Content-Centric Networking

Teng Liu and Alhussein A. Abouzeid
Department of Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA
Email: liut7@rpi.edu, abouzeid@ecse.rpi.edu

***Abstract*—In this work, we study the problem of allocating two types of memory, which have different access speeds, to a set of routers in a single content-centric network. Total network delay is adopted as the performance metric. We first formulate the allocation problem and show that it is NP-hard. Then we prove that this problem is monotone submodular with a matroid constraint. Hence, we are able to derive a guaranteed $(1-\frac{1}{e})$-approximation solution when the network is originally stable. We consider tree-structured networks with variable hierarchies under the widely used *En-route* caching assumption. Simulation results show that the developed algorithm performs well compared to the optimal solution, and only a limited fraction of nodes becomes hybrid regardless of the network size. To the best of our knowledge, this is the first theoretical work on quantitative analysis and algorithm development for hybrid memory allocation for content-centric networking (CCN).**

***Index Terms*—content-centric networking, network delay, hybrid memory, submodular optimization.**

I. Introduction

In recent years, the demand for content has exponentially increased [1]. Content-centric networking (CCN) is a promising architecture that addresses this large-scale content delivery challenge [2, 3, 4]. CCN is characterized by ubiquitous in-network caching in routers, and it handles requests differently from traditional packet-switched networking [5]. When a cache-enabled router receives a request, it first searches its cache. If the requested file exists in its cache, the router serves the request by transmitting the file to the requester. Otherwise, the router forwards the request to another router under some routing protocol policy.

The cost of allocating cache to routers in CCN networks depends on the kind of memory used. DRAM is generally slower but cheaper than SRAM, so in [6] only DRAM is used for cache memory in CCN routers. The total storage is often a limited resource in CCN networks, therefore storage allocation strategies in CCN networks taking this into account is an active area of research [7, 8, 9]. [10] studies whether additional storage is justified and argues that there is only a small performance gap between edge caching and pervasive caching. [11] analyzes the allocation of a content store space based on centrality metrics, and concluds that more cache should be added to the router with more line cards. [12] divides the cache of a router into two parts: coordinated and non-coordinated, and derives the optimal strategy for provisioning both parts in order to lower the average latency and coordination cost.

Combining one type of memory with a faster type of memory enhances the processing speed while lowering the cost [13], and there often exists a budget for the faster but more expensive memory [14]. Study on SRAM/DRAM hybrid cache architectures in standalone computer systems already exists, for example, [15] introduced an SRAM/DRAM hybrid cache architecture in 3D-implemented microprocessors with an access time 25% faster than the conventional stacked DRAM implementation. Without loss of generality, we use the term SRAM to refer to the costly but faster type of memory, and use the term DRAM to refer to the cheaper memory with slower access speed.

In most prior research, the cache in CCN routers is assumed to be composed of *only one type* of memory - DRAM [6]. DRAM has a reasonable cost, but it is less efficient than the more expensive SRAM technology [14]. Although there are works on hybrid storage in CDN networks, where each cache server can be allocated with both HDD and SSD, none of them considers the design and performance of hybrid cache in routers in the context of CCN networks [16].

In this work, we study the problem of allocating two types of memory to a set of routers in a CCN network, thus turning them into hybrid cache-enabled routers. We consider a single CCN network with a logical tree structure that consists of a server - the data source, and multiple routers, each of which can be allocated with both DRAM and SRAM, hence the name *hybrid memory allocation*. For ease of discussion, we refer to the network with only DRAM as the *original network*. When a request arrives at a node, it gets served after a time period which we call *waiting time* (as in queuing theory), or *delay*. Since users in CCN networks usually care more about content downloading speed [17], our goal is to minimize the overall network delay, *subject to a limited SRAM budget*. We formulate this problem as a monotone submodular optimization problem subject to a matroid constraint, and thus a greedy algorithm can achieve $1-\frac{1}{e}$ of optimal [18, 19] if the network is originally stable, assuming that the file popularity does not change. To the best of our knowledge, this is the first theoretical work on quantitative analysis and algorithm development for hybrid memory allocation in CCN networks.

The remainder of this paper is organized as follows. In Section II we present the model and assumptions of the hybrid memory allocation problem. In Section III, we analyze the problem formulation and its properties. In Section IV a heuristic Hybrid Memory Allocation algorithm (HMAC) is designed and its time complexity is derived. In Section V, we compare the HMAC solution to the optimal solution, and study the fraction of hybrid nodes after running HMAC. Finally, Section VII discusses possible future work.

II. Model and Formulation

In this section, we set up a CCN network model and develop a formulation for the hybrid memory allocation optimization problem. We present the assumptions of the problem in Section II-A, show a queuing system model in Section II-B, and describe the request arrival process in Section II-C.

A. Hybrid Memory Model

In a CCN network with hybrid memory, the cache in each router can be composed of both SRAM and DRAM. The sum of the size of these two types of memory in a router equals its total cache capacity. All cache-enabled routers originally have only DRAM cache. If we allocate some units of SRAM to a node, the same units of DRAM are removed from the node. The hybrid cache is managed as one cache, where only one copy of a file can be stored, and SRAM is checked before DRAM. Since our focus is memory allocation, we take as given the content management policy, which we assume can be reflected in the cache hit probability. For example, if LRU replacement policy is used and the requests follow Zipf's distribution, then cache hit probability is a non-decreasing function of cache size [20]. We further assume that SRAM(DRAM)'s hit probability is proportional to its fraction in the cache. More specifically, we have

$$\mathbb{P}_i^s = x_i \mathbb{P}_i^h \tag{1}$$

$$\mathbb{P}_i^d = (1 - x_i)\mathbb{P}_i^h \tag{2}$$

where $x_i \in [0, 1]$ is the fraction of SRAM in node i's cache, $\mathbb{P}_i^h$ is the cache hit probability of node i, $\mathbb{P}_i^s$ is the probability that there is a cache hit and the hit is in SRAM part of the cache, $\mathbb{P}_i^d$ is the probability that there is a cache hit and the hit is in DRAM part of the cache. Without loss of generality, we assume all files have the same size, and that the SRAM budget is a multiple of the file size.

B. Queuing of Requests

We model each node as an M/M/1 queuing system consisting of a server with an exponential service time that represents the router's processing time, and a FIFO queue of requests (Fig. 1). Assume there are N nodes in the network of a single administrative domain. Every node send requests along the shortest path to the server that stores all content, and En-route caching is adopted. The union of those shortest paths forms a tree with the root server at the top level. We call this CCN topology the Shortest Path Tree (SPT) [8]. When a request arrives at a node, it waits until all previous requests in the FIFO queue are served. Once a request is at the head of the queue, the node starts to serve it by checking its SRAM cache before checking its DRAM cache. We denote SRAM and DRAM access times by t_s and t_d respectively. Three mutually exclusive events can happen next: 1) if there is an SRAM cache hit, it takes an access time of t_s for the node to serve the request; 2) if there is a DRAM cache hit, it takes an access time of $t_s + t_d$ to serve the request; 3) if there is a cache miss, the node forwards the request to its parent node, so the serve time depends on the expected waiting time of the M/M/1 queue at the parent node and the downloading bandwidth. A request is usually very short compared to the size of the requested content, so the transmission time of a request is considered negligible.

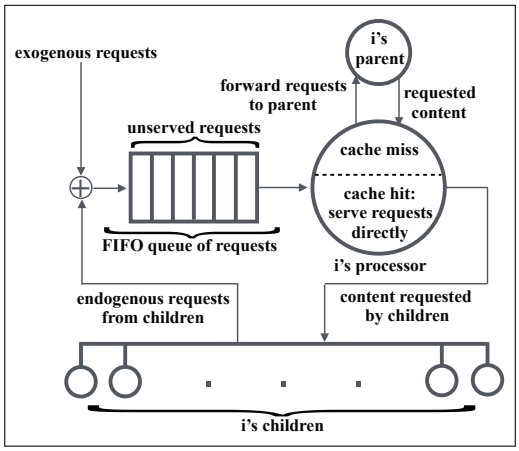

Fig. 1: M/M/1 queuing system consisting of an FIFO request queue and a server (processor of the router) at node i. A request is either forwarded to i's parent on a cache miss, or served by i directly on a cache hit. Request arrival consists of exogenous requests and endogenous requests from children.

Let W_i be the expected waiting time when a request arrives at node i. Define $P(i)$ as the shortest path from node i to the server with the the path length $|P(i)|$. Represent the k-th node along the path $P(i)$ as $P_k(i)$. Denote the content size, downloading bandwidth and cache miss probability by s, B_i and $\mathbb{P}_i^m = (1 - \mathbb{P}_i^h)$ respectively. The expected service time for a request at node i can be expressed as

$$T_i = \mathbb{P}_i^s t_s + \mathbb{P}_i^d (t_s + t_d) + \mathbb{P}_i^m (t_s + t_d + W_{P_2(i)} + \frac{s}{B_i}) \tag{3}$$

$$= T_i^c - \mathbb{P}_i^s t_d + \mathbb{P}_i^m W_{P_2(i)}, \tag{4}$$

where $T_i^c = (t_s + t_d) + \mathbb{P}_i^m \frac{s}{B_i}$. Here we need to distinguish between T_i and W_i. T_i denotes the expected service time when a request is at the head of the request queue, so its inverse is the service rate μ_i. W_i is the expected waiting time, that is the period between the time when a request arrives at node i to the time when the request is served. Therefore, W_i can be considered as the expected delay of the node. As we assume cache hit probability is known and fixed, $\mathbb{P}_i^m$ is also fixed for each node i. Thus T_i^c can be pre-computed as a constant.

By investigating the structure of (4), we note that if the miss probability of node i's children are low, the decrement of node i's waiting time has a limited contribution to the increment of the service rate of its children. Thus, if a (parent) node has

a lower children miss probability, it tends to push the SRAM allocation away from itself. In addition, SRAM allocation also depends on a constant term which involves the downloading bandwidth, and cache miss probability.

C. Request Arrival Process

Request arrival at any leaf node is assumed to consist of only exogenous requests following a Poisson distribution with the same parameter λ. Request arrival process of a non-leaf node i consists of two parts: (1) exogenous request arrival that follows a Poisson distribution with parameter λ_i^e, and (2) endogenous request arrival from downstream nodes due to the cache misses at children. From Burke's Theorem, the arrival distribution of a stable M/M/1 system is the same as its departure distribution [21]. Therefore, endogenous arrival rate is the miss-probability-weighted sum of the arrival rates of the children. Denote the set of node i's immediate children by $c(i)$, and the set of leaf nodes by L. By Little's Theorem, the arrival rate and the expected waiting time of node i can be respectively expressed as

$$\lambda_i = \begin{cases} \sum_{v \in c(i)} \mathbb{P}_i^m \lambda_v + \lambda_i^e, \ i \notin L \\ \lambda, \ i \in \end{cases} \tag{5}$$

$$W_i = \begin{cases} \frac{1}{\mu_i - \lambda_i}, \text{ if } \mu_i > \lambda_i \\ \infty, \text{otherwise} \end{cases} \tag{6}$$

Note that the service time consists of two parts: the service time if there is a cache hit: $T_i^h = \mathbb{P}_i^s t_s + \mathbb{P}_i^d (t_s + t_d)$, and the service time if there is a cache miss: $T_i^m = \mathbb{P}_i^m (t_s + t_d + W_{P_2(i)} + \frac{s}{B_i})$. The service rate (inverse of service time) due to cache miss (denoted by μ_i^m) indicates the number of requests that can be served per unit time if node i forwards the request to its parent due to cache miss. So T_i^m is lower bounded by $\mathbb{P}^m \frac{s}{B_i}$. As a result, (3) and (4) hold only if $\mu_i^m \leq \frac{B_i}{s\mathbb{P}^m}$. Denote the SRAM budget by b. Let $[A, B]^- = \min(A, B)$. Given the network topology $G = (V, E)$, the problem can be formulated as maximizing the negative of the total network delay:

$$\begin{aligned} \underset{x_i}{\text{maximize}} \ & -\sum_{i \in V} W_i \\ \text{subject to} \ & \sum_{i \in V} c_i x_i \leq b \end{aligned} \tag{7}$$

where:

$$\mu_i = [T_i^h + (\mu_i^m)^{-1}]^{-1} \tag{8}$$

$$T_i^h = x_i \mathbb{P}_i^h t_s + (1 - x_i) \mathbb{P}_i^h (t_s + t_d) \tag{9}$$

$$\mu_i^m = [(T_i^m)^{-1}, \frac{B_i}{s(1 - \mathbb{P}^h)}]^- \tag{10}$$

$$T_i^m = (1 - \mathbb{P}_i^h)(t_s + t_d + W_{P_2(i)} + \frac{s}{B_i}) \tag{11}$$

Eq. (8) through (11) correspond respectively to the service rate, delay due to cache hit, service rate due to cache miss, and upper bound of delay due to cache miss.

From the above formulation (7), we can obtain insightful observations about the influence of arrival rates, number of children, cache miss probability, and children's miss probability, on the hybrid memory allocation. Note that we assume the request pattern does not change. If a node has a high arrival rate, we need to increase the service rate to keep its waiting time low. Note that the arrival consists of exogenous arrival and miss-probability-weighted sum of request arrivals of children. Therefore, high exogenous arrival rate, as well as a large number of children, can potentially cause more SRAM allocation to the corresponding node. The consideration of number of children is similar to the *degree centrality allocation* scheme [11], which concludes that more storage should be added to the routers with more line cards. Leaf nodes have the same exogenous request arrival rate and zero number of children, so the effect of miss probability dominates if their downloading bandwidths are similar. High miss probability at a leaf node means the node needs to forward more requests to its parent. So if we want to lower the delay at the leaf level, more SRAM should be allocated to the high-miss-probability leaves to compensate for the request forwarding time.

III. Formulation Analysis

In this section, we show that the budget constraint in formulation (7) is a matroid in Section III-A, then we prove the monotonicity, as well as the submodularity, of the objective of formulation (7) in Section III-B, and hence the problem is NP-hard [22]. We apply the results from [18, 19] to develop an approximation greedy algorithm in the next section.

A. Matroid Constraint

A tuple $M = (\Omega, I)$ is a matroid if the following three conditions hold: i) I is a nonempty set; ii) If $Y \subset I$, $X \subset Y$, then $X \subset I$; iii) If $X, Y \subset I, |X| < |Y|, \exists y \in Y \setminus X$, such that $X \cup y \subset I$. Let X be a feasible solution to the formulation in (7). Define a ground set $\Omega = \{v_{1,1}, v_{1,2}, \cdots, v_{2,1}, v_{2,2}, \cdots, v_{N,1}, v_{N,2}, \cdots\}$, such that Ω can be partitioned into disjoint independent sets: $\Omega = \{\Omega_1; \Omega_2; \cdots; \Omega_N\}$, where $\Omega_i = \{v_{i,1}, v_{i,2}, \cdots\}$ and $|\Omega_i| = c_i$. A subset I of Ω is: $I = \{v_{i,1}, v_{i,2}, \cdots, v_{j,1}, v_{j,2}, \dots\}$, where the number of $v_{i,m}$ in I equals the number of SRAM units to be allocated to node i. So W_i can be regarded as a set function $W_i : 2^\Omega \to \mathbb{R}$. The budget constraint is a uniform matroid if we define $M = (\Omega, I)$, where $I = \{X \subset \Omega : \ |X| \leq b\}$.

B. Monotone Submodular Objective

We now prove the objective in formulation (7) is monotone submodular by equivalently showing that its negative, i.e. $\sum_{i \in V} W_i$, is non-increasing supermodular. Since the sum of non-increasing supermoular functions is also non-increasing supermodular, we only need to show W_i is non-increasing supermodular for any $i \in V$.

A set function f is monotonically non-increasing if the following holds: given $A \subseteq B \subset \Omega, f(A) \geq f(B)$. In formulation (7), $A \subseteq B \subset \Omega$ means: besides allocating SRAM the same way as set A does, set B allocates more SRAM in the network. The waiting time of node i in the context of B, denoted by $(W_i)_B$, must be no greater than that in the context

of A, denoted by $(W_i)_A$, i.e. $(W_i)_A \geq (W_i)_B$. So W_i is a non-increasing function on set Ω.

A set function is supermodular if the following holds: given $A \subseteq B \subset \Omega$ and $v \in \Omega \setminus B$,

$$f(A \cup \{v\}) - f(A) \leq f(B \cup \{v\}) - f(B) \qquad (12)$$

Inequality (12) is called increasing returns, since adding a new element to a larger set yields more gain than adding this element to a smaller set. For simplicity, we denote $f_i(M \cup \{v\}) - f_i(M)$ as $(\Delta f_i)_M,\ \forall i, v \in V$, where M is a subset of Ω . Now we prove W_i is supermodular via induction set up by the following three claims, from which we have $(W_i)_{A\cup\{v\}} - (W_i)_A \leq (W_i)_{B\cup\{v\}} - (W_i)_B,\ \forall i, v \in V$.

Claim 1: If $v_{j,n} \in \Omega \setminus B, n \in [1, c_j]$, and if node j is not on the path $P(i)$, then increasing returns holds for node i.

Proof: Choosing an element $v_{j,n}$ from the set $\Omega \setminus B$ means we allocate another unit of SRAM at node j. The effect of the change of node j's waiting time only propagates to j's subtree due to the recurrence term $W_{P_2(j)}$ in (7). If node j is not on the path $P(i)$, he added SRAM has no effect on node i's waiting time W_i. So $(W_i)_{A\cup\{v\}} - (W_i)_A = (W_i)_{B\cup\{v\}} - (W_i)_B = 0$.

Claim 2: If $v_{j,n} \in \Omega \setminus B, n \in [1, c_j]$, the increasing returns holds for node j. (See prove in appendices)

Claim 3: If $v_{j,n} \in \Omega \setminus B, n \in [1, c_j]$, and if a node k has a parent node l that satisfies $(\Delta W_l)_A \leq (\Delta W_l)_B$ for any node $k \neq j$, then $(\Delta W_k)_A \leq (\Delta W_k)_B$. (See prove in appendices)

IV. HMAC Algorithm Design and Analysis

In this section, we present the HMAC algorithm and analyze its computational complexity. We have proved in Section III that (7) is maximizing a monotone submodular function subject to a matroid constraint, which is known to be NP-hard and has a greedy algorithm solution that achieves $1 - \frac{1}{e}$ of optimal by incrementally adding the element that brings the most gain compared to current iteration [18, 19]. The size of DRAM is implied by the size of SRAM during any iteration since the proportion of DRAM in node i is $(1 - x_i)$. We say a network is unstable if there is any request queue with an arrival rate larger than service rate. Since request patterns are assumed to be fixed over time, the stability of the network is known given the request arrival processes and network topology. The greedy algorithm HMAC is described in Algorithm 1.

Now we provide further explanation for the HMAC pseudo-code in Algorithm 1. Whenever a node is allocated with SRAM to its full cache capacity, it is outside of our consideration in the later iterations. But its expected waiting time should be updated if any of its upstream nodes is allocated with more SRAM. To determine where to allocate the next unit of SRAM, we try allocating it to at most N nodes and compare the network delay decrement of those attempts. If the original network is stable, the algorithm will skip line 4 and the solution achieves $1 - \frac{1}{e}$ of optimal delay reduction. In the rare case where the network is originally unstable, we can adopt a greedy approach in line 4 of Algorithm 1 to stabilize an unstable node i: the next unit of SRAM is allocated to its upstream node that leads to the most decrement of $\frac{1}{\mu_i - \lambda_i}$.

Algorithm 1 HMAC in CCN Networks

Input: CCN network $G = (V, E)$, SRAM budget;
Output: Allocation scheme: $X = \{x_i : i \in V\}$;
1: **Initialize:** $x_i = 0 \forall i \in V$, $W^{old} = \sum_{i \in V} W_i$;
2: **while** there remains SRAM unallocated **do**
3: **for** each node i **do**
4: Try stabilize node i if $W_i = \infty$
5: Add one unit of SRAM to node i;
6: Compute new network delay W^{new};
7: Compute delay decrement: $D_i = W^{old} - W^{new}$;
8: Remove one unit of SRAM from node i;
9: **end for**
10: Find maximum D_i, add one unit of SRAM to i;
11: Update x_i, $W^{old} = W^{new}$, budget -= 1;
12: **end while**
13: **return** X;

We keep adding more SRAM units in this manner until the request queue of node i becomes stable. However, the solution is not guarantee to be $1 - \frac{1}{e}$ of optimal if the original network is unstable, since only the local gain, instead of global gain, is considered in line 4 of Algorithm 1 for stabilization. If the original network is unstable and the network delay is still infinity when HMAC terminates, then there is no solution. We assume this case to be less common in practice. It entails an efficient network stabilizing scheme, and thus is beyond our discussion here and is left to our future work.

HMAC is an efficient polynomial algorithm. It terminates after b iterations, where b is the SRAM budget. The network delay decrement resulting from allocating one unit of SRAM to a node is computed by updating the waiting time of the subtree rooted at the node. There are at most N nodes in a subtree, so there are N additions to update the waiting time of a subtree in the worst case. Therefore, the *for loop* takes $O(N^2)$. The largest delay decrement can be computed in linear time, and is dominated by the quadratic time complexity incurred by the *for loop*. Assuming all algebraic computations take constant time, the HMAC algorithm runs in $O(bN^2)$.

V. Simulation Results

In this section, we evaluate HMAC by comparing its performance against the optimal solutions. Then we compute the fraction of hybrid nodes after running HMAC on networks of different sizes. A set of CCN networks is set up by the following two steps:

Step 1: A tree is randomly generated with configurable number of nodes. We choose random trees for the control flexibility of network properties, which enables more general evaluations. Each node in the tree is associated with two properties of interest: number of children and cache hit probability, both of which are random with configurable ranges.

Step 2: Streams of exogenous requests are fed into the network. Every leaf node is fed with Poisson request arrival with the same parameter λ; every non-leaf node is fed with exogenous Poisson request arrival with different uniformly

distributed parameters λ_i. Note that although we adopt a Zipf request pattern and LRU replacement policy, only the arrival rates are relevant to the evaluation. The request pattern and caching policy would affect cache hit probability which is assumed to be known a-priori, as discussed in Section II-A.

To evaluate the performance of HMAC, we set up a network of 30 nodes and assign each node a random cache size and cache hit probability. Originally, all nodes only have DRAM memory. We set t_s, t_d, and the root server's service time to $0.015\mu s$, $0.055\mu s$, and $0.5\mu s$ rescompectively [14, 23]. The performance is measured as the percentage of decrement in delay relative to the original network delay. We use MATLAB to obtain the optimal solution, which can only be accomplished for small instances of the problem, since it is NP-hard. The lower bound is computed as $1 - \frac{1}{e}$ of the optimal value.

Simulation results are shown in Fig. 2, where the data points are obtained as follows. We vary the exogenous request arrival rates from 5 to 100 with the step size of 5. We run HMAC for each of the arrival rates. Then we take the average of those 20 results. The data points of the optimal solution are obtained in a similar manner. We note that when the SRAM budget is relatively small (less than 270 in this case, or 10% of the total capacity), the HMAC solution almost overlaps with the optimal solution. As the budget increases, HMAC solution diverges from the optimal, but is always notably larger than the lower bound. After the SRAM budget surpasses the total cache capacity (2700 in this case), all nodes are allocated with SRAM to their full capacities, so there is no difference between the HMAC solution and optimal solution.

Next, we vary the network size and SRAM budget, and compute the percentage of nodes that are allocated with SRAM after running HMAC, as shown in Fig. 3. For each data point, we take the average of the results of 200 simulations. In each simulation, the number of children, cache size, and exogenous request arrival rate, are uniformly distributed in $[0, 4]$, $[10, 40]$, and $[50, 100]$ respectively, and hence the simulation covers a wide variety of topologies. We set SRAM budget to be 10%, 20%, and 50% of the total cache capacity. According to Fig. 3, only part of the nodes becomes hybrid after running HMAC on various sizes of networks. For example, if the SRAM budget is 10% of the total capacity, no more than 35% of the nodes becomes either hybrid or SRAM-only, regardless of the network size. This significantly decreases the potential cost of making all nodes hybrid.

VI. Conclusion and Future Work

We have derived a formulation of the hybrid memory allocation problem and developed an efficient polynomial time algorithm HMAC that can achieve $1 - \frac{1}{e}$ of optimal delay reduction for originally stable networks. Theoretical analysis and simulation provide insights on hybrid memory allocation for future CCN deployment. Our approach works much better than the lower bound and avoids the potentially high cost of making all nodes hybrid.

A number of open questions remain. First, our analysis is based on the assumption that the request pattern is invariant.

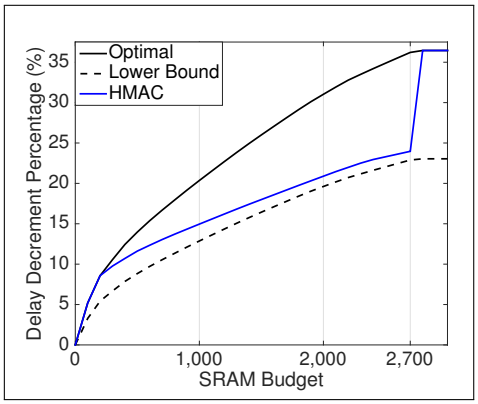

Fig. 2: Delay decrement percentage vs. SRAM budget

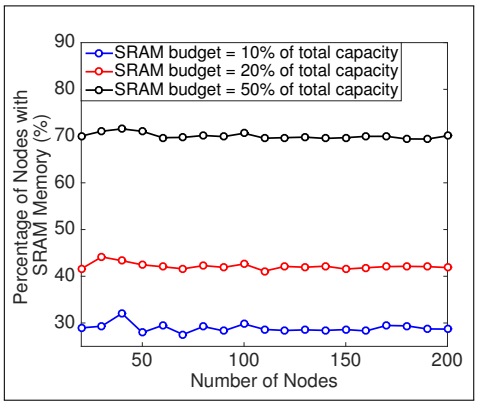

Fig. 3: Percentage of hybrid nodes vs. number of nodes in the network

However in practice, variation in request arrival rates and in file popularity may challenge the performance of HMAC and is worth further investigation. Second, the formulation should be modified to allow for more complicated topologies in future CCN deployment, where there may exist multiple autonomous units, different data sources, etc. Last but not least, the performance of HMAC on an unstable network are unknown - an efficient network stabilizing scheme can be used jointly with HMAC. Both of the stabilizing scheme and the performance comparison with other widely known allocation schemes warrant further study.

Acknowledgement

This material is based upon work supported by the U.S. National Science Foundation under grant numbers 1422153 and 1456887, and a Tekes FiDiPro Fellow award with University of Oulu, Oulu, Finland.

References

[1] B. Tan and L. Massoulié, "Optimal content placement for peer-to-peer video-on-demand systems," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 566–579, Apr. 2013.

[2] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, Aug. 2007.

[3] J. Kurose, "Information-centric networking: The evolution from circuits to packets to content," *Comput. Networks*, vol. 66, pp. 112 – 120, 2014.

[4] X. Zhang, Y. Li, J. Li, K. Zhao, and T. Zhang, "Proximate control stream assisted video transcoding for heterogeneous content delivery network," in *ICIP*, 2014, pp. 2552–2555.
[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *5th Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009, pp. 1–12.
[6] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *ACM Re-Architecting the Internet Workshop (ReARCH)*, 2010, pp. 5:1–5:6.
[7] N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis, "On the optimization of storage capacity allocation for content distribution," *Comput. Networks*, vol. 47, no. 3, pp. 409–428, 2005.
[8] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie, "Optimal cache allocation for content-centric networking," in *IEEE Int. Conf. Network Protocols*, 2013, pp. 1–10.
[9] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at "the edge" of information-centric networks," in *1st Int. Conf. on Information-centric Networking (ICN)*, 2014, pp. 47–56.
[10] 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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 147–158, Aug. 2013.
[11] D. Rossi and G. Rossini, "On sizing CCN content stores by exploiting topological information," in *IEEE INFOCOM Workshops*, 2012, pp. 280–285.
[12] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," *IEEE 33rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, vol. 0, pp. 62–72, 2013.
[13] S. Lee, J. Jung, and C.-M. Kyung, "Hybrid cache architecture replacing SRAM cache with future memory technology," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2012, pp. 2481–2484.
[14] G. de Brito, P. Velloso, and I. Moraes, *Information Centric Networks: A New Paradigm for the Internet*, ser. FOCUS Series. Wiley, 2013.
[15] K. Inoue, S. Hashiguchi, S. Ueno, N. Fukumoto, and K. Murakami, "3D implemented SRAM/DRAM hybrid cache architecture for high-performance and low power consumption," in *Midwest Symp. on Circuits and Syst.*, 2011, pp. 1–4.
[16] T. Kim and E.-J. Kim, "Hybrid storage-based caching strategy for content delivery network services," *Multimedia Tools Appl.*, vol. 74, no. 5, pp. 1697–1709, 2015.
[17] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda, "Congestion-aware caching and search in information-centric networks," in *1st Int. Conf. on Information-centric Networking (ICN)*, 2014, pp. 37–46.
[18] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions. II," *Math. Programming Stud.*, no. 8, pp. 73–87, 1978.
[19] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a monotone submodular function subject to a matroid constraint," *SIAM J. on Computing*, vol. 40, no. 6, pp. 1740–1766, 2011.
[20] L. R. Beaumont, "Calculating web cache hit ratios," available at: http://www.content-networking.com/papers/web-caching-zipf.pdf, [Accessed Dec. 6, 2015].
[21] P. J. Burke, "The output of a queuing system," *Oper. Res.*, vol. 4, no. 6, pp. 699–704, Dec. 1956.
[22] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, Jul. 1998.
[23] "An overview of cache," available at: http://www.intel.com/design/intarch/papers/cache6.htm, [Accessed Dec. 6, 2015].

APPENDIX A
OUTLINE OF PROOF OF CLAIM 2

i) Assume the download link of j is not saturated. Define $\mathbb{C}_j = [T_j^c + \mathbb{P}_j^m W_{P_2(j)}]t_d^{-1}$. Adding one unit of SRAM to node j does not affect the waiting time of its parent, i.e. $W_{P_2(j)} = W_{P_2(j)}^{'}$, so $(\mathbb{C}_j)_A = (\mathbb{C}_j^{'})_A$. Then the gain can be written as:

$$(\Delta W_j)_A = \frac{(-t_d\lambda_j^2)^{-1}[(\mathbb{P}_j^{'s})_A - (\mathbb{P}_j^s)_A]}{[\frac{1}{t_d\lambda_j} - (\mathbb{C}_j)_A + (\mathbb{P}_j^{'s})_A][\frac{1}{t_d\lambda_j} - (\mathbb{C}_j)_A + (\mathbb{P}_j^s)_A]}$$

We can write $(\Delta W_j)_B$ in a similar form. Since $\mathbb{P}_j^s$ is strictly increasing w.r.t x_j and $A \subset B$, we have:

$$\begin{cases} (\mathbb{P}_j^s)_A < (\mathbb{P}_j^{'s})_A,\ (\mathbb{P}_j^s)_B < (\mathbb{P}_j^{'s})_B \\ (\mathbb{P}_j^s)_A \leq (\mathbb{P}_j^s)_B,\ (\mathbb{P}_j^{'s})_A \leq (\mathbb{P}_j^{'s})_B \\ (\mathbb{P}_j^{'s})_A - (\mathbb{P}_j^s)_A = (\mathbb{P}_j^{'s})_B - (\mathbb{P}_j^s)_B = \frac{1}{c_i}\mathbb{P}_i^h \\ (\mathbb{C}_j)_A \geq (\mathbb{C}_j)_B \\ W_i = ((\mathbb{C}_i t_d - \mathbb{P}_i^s t_d)^{-1} - \lambda_i)^{-1} > 0 \end{cases} \tag{13}$$

From the above inequalities, we get: $(\Delta W_j)_A \leq (\Delta W_j)_B$.

ii) If the download link of j is saturated, we have:

$$(W_j)_A = ((\mathbb{P}_j^s t_s + \mathbb{P}_i^h(t_s + t_d) + B_j^{-1})^{-1} - \lambda_j)^{-1} \tag{14}$$

Define $\mathbb{C}_j = [(1 - \mathbb{P}_i^m)(t_s + t_d) + B_j^{-1}]t_d^{-1}$. $(\Delta W_j)_A$ and $(\Delta W_j)_B$ can be re-written accordingly. By similar argument, the increasing returns still holds. ■

APPENDIX B
OUTLINE OF PROOF OF CLAIM 3

i) If k's download link is not saturated, we have:

$$(\Delta W_k)_A = \frac{(t_d\lambda_j^2)^{-1}[(\mathbb{C}_k^{'})_A - (\mathbb{C}_k)_A + (\mathbb{P}_k^s)_A - (\mathbb{P}_k^{'s})_A]}{[(\mathbb{C}_k^{'})_A - (\mathbb{P}_k^{'s})_A - \frac{1}{t_d\lambda_k}][(\mathbb{C}_k)_A - (\mathbb{P}_k^s)_A - \frac{1}{t_d\lambda_k}]}$$

Since $k \neq j$, $\mathbb{P}_k^s = \mathbb{P}_k^{'s}$. So the gain becomes:

$$(\Delta W_k)_A = \frac{(t_d\lambda_j^2)^{-1}[(\mathbb{C}_k^{'})_A - (\mathbb{C}_k)_A]}{[\frac{1}{t_d\lambda_k} - (\mathbb{C}_k^{'})_A + (\mathbb{P}_k^s)_A][\frac{1}{t_d\lambda_k} - (\mathbb{C}_k)_A + (\mathbb{P}_k^s)_A]}$$

We can write $(\Delta W_j)_B$ in a similar form. From *Claim 2*, we have:

$$(\Delta W_{P_2(k)})_A \leq (\Delta W_{P_2(k)})_B \tag{15}$$

$(\mathbb{C}_k^{'})_A - (\mathbb{C}_k)_A$ and $(\mathbb{C}_k^{'})_B - (\mathbb{C}_k)_B$ can be re-written as:

$$\begin{cases} (\mathbb{C}_k^{'})_A - (\mathbb{C}_k)_A = \dfrac{\mathbb{P}_k^m}{t_d}(\Delta W_{P_2(k)})_A \\ (\mathbb{C}_k^{'})_B - (\mathbb{C}_k)_B = \dfrac{\mathbb{P}_k^m}{t_d}(\Delta W_{P_2(k)})_B \end{cases} \tag{16}$$

(15), (16) and the condition $A \subset B$ yield:

$$\begin{cases} (\mathbb{C}_k^{'})_A - (\mathbb{C}_k)_A \leq (\mathbb{C}_k^{'})_B - (\mathbb{C}_k)_B \\ (\mathbb{C}_k)_A \geq (\mathbb{C}_k^{'})_A,\ (\mathbb{C}_k)_B \geq (\mathbb{C}_k^{'})_B \\ (\mathbb{C}_k)_A \geq (\mathbb{C}_k)_B,\ (\mathbb{C}_k^{'})_A \geq (\mathbb{C}_k^{'})_B \\ (\mathbb{P}_k^s)_A \leq (\mathbb{P}_k^s)_B \end{cases} \tag{17}$$

From (13) and (17), we get $(\Delta W_k)_A \leq (\Delta W_k)_B$.

ii) If k's download link is saturated, the decrement of waiting time of any $v \in P(k)$ does not affect W_k, i.e. $(\Delta W_k)_A = (\Delta W_k)_B$, so the increasing returns holds. ■