Leveraging In-network Caching for Efficient Content Delivery in Content-centric Network

Diliang He, Wei K. Chai and George Pavlou

Department of Electronic & Electrical Engineering, University College London, WC1E 7JE, Torrington Place, London, UK

Abstract: The recent proposal on content-centric network advocates the use of in-network caching to enhance content delivery in the Internet. In this paper, we proposed an in-network caching algorithm based on the concept of ego network betweenness centrality and compared its performance with Van Jacobson's networking named content caching technique.

1. Introduction.

Content-centric network (CCN) is a new networking concept focusing on content dispersion rather than end-to-end resource sharing. In CCN, content names are decoupled from their host addresses, effectively separating the role of identifier and locator as opposed to the current IP addresses which are used for both purposes. Naming content directly enables the exploitation of in-network caching since the content can now be accessed in an application-independent manner.

The most influential proposal for content-centric network is [1] where it was proposed that content chunk be cached in each and every router it traverses along the delivery path with each router applying the least recently used (LRU) cache eviction policy. Such caching strategy ensures quick diffusion of content across the network.

In this paper, we argue that such caching strategy is not optimal and study an alternative caching strategy for enhancing the overall content delivery performance. We propose a caching algorithm inspired by the concept of centrality in the social network analysis area [2] where only selected nodes in the content delivery path cache the content chunk with the rationale that some nodes have higher probability of getting a cache hit compared to others and by strategically caching the content at "better" nodes, we can decrease the cache eviction rate and increase the cache hit. We show by simulation that our proposed algorithm indeed perform better than [1] under different scenarios.

2. A Cache Point Selection Strategy

The current host-centric Internet requires all content requests to be resolved to the location of host servers before content can be delivered while in CCN (e.g., [[1], [3]]), content are labelled and identified by their own content names without needing to know the host machine. Thus, in CCN, a content request can be satisfied by any matching content regardless of its location (i.e., a cached content can serve a request). In [1], this feature of CCN is exploited by caching every content chunk that passes a router with the assumption that routers are equipped with (large) cache stores. With this approach, content would disperse into cache store in a fast way and the cache stores would update frequently following the LRU policy. For the rest of the paper, we refer this as *CCN-LRU* as the benchmark for performance comparison.

We propose a new caching algorithm based on the concept of betweenness centrality [2] which measures the number of times a specific node occurs on the shortest paths between all pairs of nodes in a network topology. The idea is that if a node lies in the path of many shortest paths, then it is more likely to get a cache hit.

For a topology G=(V, E) with V vertices and E edges, the betweenness centrality, $C_B(v)$ of node *v* is computed as follows using Eq. (1).

$$\text{Betweenness centrality: } C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}. \quad (1)$$

where σ_{st} is the number of shortest paths from *s* to *t* and $\sigma_{st}(v)$ is the number of shortest paths from *s* to *t* that pass through a node *v*.

Each node in a topology thus has its own betweenness centrality value. When a content client initiates a content delivery, the initiation message (e.g., GET) will record the node with the highest centrality value. When the content is delivered, it will be cached at the recorded node along the delivery path. If more than one node has the same highest centrality value, then the node closest to the content client is chosen. The same LRU cache eviction policy is assumed at each node. Hereafter, this caching strategy is referred to as *Betweenness-LRU*.

We show a preliminary comparison of the two caching strategies in Figure 1 in a 13-node string topology where all content requests originate from one end while all content are hosted at the other. We simulate a total of 50,000 content requests for 100 different content with content requests generated based on Zipf-distribution. The

cache store size of each node is 10% of the total number of content. We measure the performance by the running average of the ratio on hops saved from each request with in-network caching i.e.,

$$\text{Hop gain ratio} = \frac{Number\ of\ hops\ to\ hit\ the\ requested\ content(with\ cache)}{Number\ of\ hops\ to\ the\ content\ server\ (without\ cache)}. \quad (2)$$

Number of hops without cache equals the hops needed for content to be sent from initial server to the client if a matching cache is available. If no matching cache is found along the delivery path, then the ratio equals one (i.e., no gain). Number of hops with cache equals the hops needed for content to be sent from the nearest cache store to the client. Higher gain is signified by smaller ratio. This preliminary result shows that while *CCN-LRU* save approximately 40% (i.e., on average if the distance from content client to server is 10 hops, the request can find a cache hit at 6th hop along the delivery path) of the number of hops for content delivery, our approach can reach a gain of 60%. From our results, we also note that *Betweenness-LRU* not only saved hops, but also saved server hit from content server. On average only 14164 requests reached the server in *Betweenness-LRU*, while 28733 requests reached the content server in *CCN-LRU*.

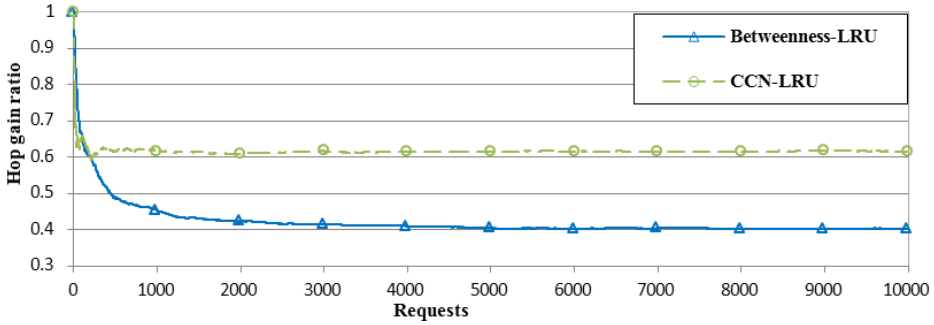


Figure 1: Comparison of CCN-LRU and Betweenness-LRU in string topology

In the real world, it is usually not practical to assume each node capable of computing its betweenness centrality since it requires the knowledge of the shortest paths between all pairs of nodes. Such computation is simply not scalable. We then further develop an approximation of our caching strategy based on the ego network betweenness concept [4].

Ego network consists of a central node together with the nodes (one hop neighbors) they are connected to and all the links among those nodes. The advantage of ego network is the ease for central node to collect data from the neighbors compared to collecting the data of whole network and it is simple to calculate the betweenness centrality of the central node within its ego network. This kind of centrality is called ego network betweenness centrality. Although ego network betweenness only reflects the importance of a node within its ego network, its scalability and ease of implementation makes it a good alternative for betweenness in large networks. The caching algorithm using ego network betweenness centrality and LRU cache eviction policy is referred to *EgoBetweenness-LRU* hereafter.

3. Evaluation and Simulation Study

3.1 Hop gain ratio in 5-tier binary tree topology

We evaluate the performance of the three caching algorithms: *CCN-LRU*, *Betweenness-LRU*, *EgoBetweenness-LRU* based on a 5-tier binary tree topology shown in Figure 2 with two peered tier-1 ASes (i.e., AS 1 and AS 2). The spread factor of the binary tree is two and totally there are 62 autonomous systems (ASes) in the topology.

We simulate 240,000 requests initiated from random clients for 300 different content with content requests based on Zipf-distribution. Unless otherwise specified, the cache store size is 10% of the total content population.

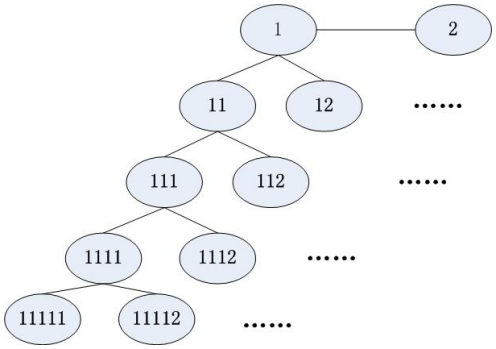


Figure 2:5-tier binary tree topology

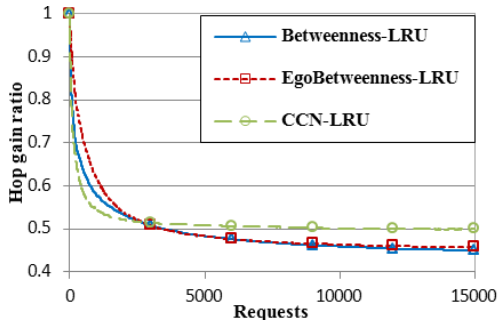


Figure 3: Performance of the three caching strategies in 5-tier binary tree topology

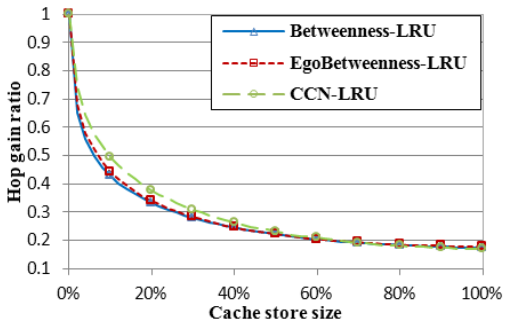


Figure 4: Performance of the three caching strategies at different cache sizes

From Figure 3 we could see the hop gain ratio of *CCN-LRU* drops faster than other two caching strategies and this phenomenon could be explained by *CCN-LRU*'s all cache property. Although with this property clients in *CCN-LRU* could benefit from cached content at early stage, the final result shows *CCN-LRU* saved only 50% of the number of hops while our *Betweenness-LRU* saved 58% and *EgoBetweenness-LRU* saved 57%. This is because *CCN-LRU* caches content in every content store along the delivery path and this causes the content stores to update too frequently. *EgoBetweenness-LRU* has similar performance with *Betweenness-LRU*, but due to its approximation the performance is slightly worse.

3.2 The effect of cache store size

We evaluate the three caching strategies with different cache sizes based on the 5-tier binary tree topology. The cache size ranges from 0% to 100% of the total content population. We generate 240,000 requests while other configurations stay the same as 3.1

Figure 4 shows *CCN-LRU* performs no better than *Betweenness-LRU* and *EgoBetweenness-LRU* at all cache sizes. The performance gap between *CCN-LRU* and our algorithms is bigger when the cache store size is small and the performance of all algorithms converges when the cache size is set to be large enough to cache the whole population of the content. This is only academic since at 100%, every content is eventually cached everywhere.

3.3 The performance with varying content popularity distribution

In this section, we investigate the effect of content popularity distribution on the caching gain.

Throughout our simulation, all our content requests follow Zipf-distribution. The equation of Zipf-distribution is as follow:

$$\sum_{k=1}^{\text{content number}} \frac{C}{k^{\alpha}} = 1. \quad (3)$$

where k represents the k^{th} most popular content and the possibility for a request for the k^{th} content is $\frac{C}{k^{\alpha}}$. The sum of the possibility of all content equals one. We vary the popularity factor, α and from Figure 5, find similar observations as before, i.e., both our algorithms achieve similar gains and better than *CCN-LRU*. .

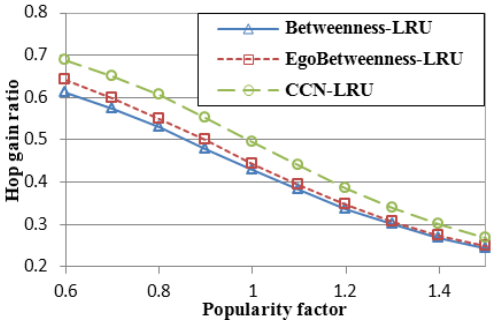


Figure 5: Performance of the three caching strategies with different popularity factor

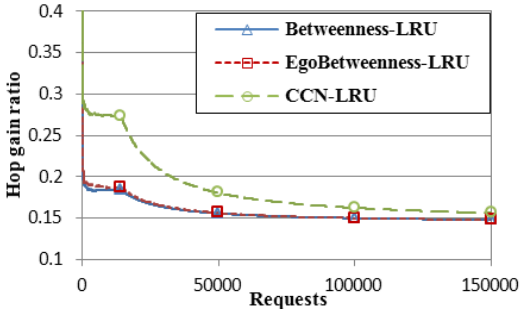


Figure 6: Performance of the three strategies under flash crowd scenario

3.4 Performance under *flash crowd* scenario.

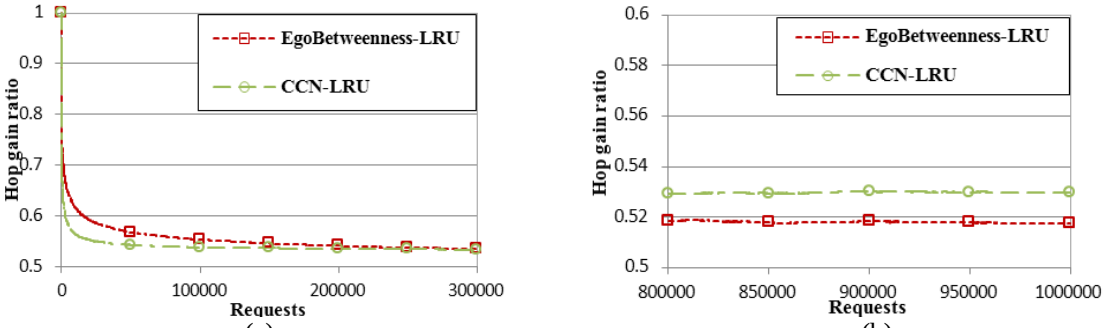
Popular content stand to gain the most in CCN [5]. In this section, we artificially induce a *flash crowd* scenario by generating all requests for one specific content only (the 5$^{\text{th}}$ most popular content as an arbitrary example).

We repeat the simulation in 3.2 until the hop gain ratio become steady before initiating the *flash crowd*. We only track the performance of the 5th most popular content in this part.

In Figure 6 *Betweeneess-LRU* and *EgoBetweenness-LRU* saves 82% of the hops while *CCN-LRU* only saves 73% before *flash crowd* scenario. Comparing the overall gain in 3.1, popular content benefit more from caching. Also we notice the performance gap between our approaches and *CCN-LRU* is 9% for the 5th most popular content while the overall performance gap is 8% in 3.1. Thus our approaches provide even more gain for popular content than *CCN-LRU*.

3.5 Performance of *CCN-LRU* and *EgoBetweenness-LRU* in a real network topology

Then we use the topology of real network to see how *EgoBetweenness-LRU* and *CCN-LRU* perform (*Betweenness-LRU* is not included here for its non-scalability reason). We extracted a 3-tier branch topology from the CAIDA dataset [6] with AT&T as the root of the branch and a total of 6804 ASes in the topology.



(a) (b)

Figure 7: Performance of two strategies in real network topology

From Figure 7(a), we see initially *CCN-LRU*'s hop gain ratio drops more rapidly but the cache gain of *EgoBetweenness-LRU* exceeds *CCN-LRU* after 300000 requests and continues to drop. Unfortunately due to the size of the network, the difference of the two strategies shown in Figure 7(b) is quite small (only 2%). However, we note that the hop gain ratio of *EgoBetweenness-LRU* is still dropping while the hop gain ratio of *CCN-LRU* is already steady.

4. Conclusions

We proposed a caching strategy based on the betweenness centrality concept (i.e., *Betweenness-LRU*) and an approximation of it (*EgoBetweenness-LRU*) for scalable and distributed realization and compare their performance against Van Jacobson's proposal [1] (*CCN-LRU*) on different topologies (string, binary tree and real domain-level topology), cache store sizes, and popularity factors. Although *CCN-LRU* has the highest content dispersion speed, its caching gain is worse than our two approaches under all scenarios. Our simulation results of *Betweenness-LRU* suggest that it has the best hop gain ratio among them but its non-scalability and complexity restricts the implementation. Our results show that *EgoBetweenness-LRU* approximates closely the *Betweenness-LRU* and thus present itself as the more practical candidate for real networks deployment.

References

[1] V. Jacobson, D. K. Smetters, James D. Thornton, Michael Plass, Nick Briggs, Rebecca L. Braynard, "Networking Named Content," *ACM CoNEXT 09*, 2009, pp.1-12.

[2] L. R. Izquierdo, Robert A. Hanneman, "Introduction to the Formal Analysis of Social Networks Using Mathematica", University of California, Riverside.

[3] W. K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. G. de Blas, F. J. Salguero, L. Liang, S. Spirou, A. Beben and E. Hadjioannou, "CURLING: Content-Ubiquitous Resolution and Delivery Infrastructure for Next Generation Services" , *IEEE Communications Magazine, Special Issue on Future Media Internet*, March, 2011

[4] M.Everett, S. P. Borgatti, "Ego network betweenness", *Social Networks, 27(2005)* pp. 31-38.

[5] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, G. Pavlou, "Modelling and Evaluation of CCN-Caching Trees", *Proc. IFIP Networking 2011, Valencia, Spain,* pp. 9-13, May 2011

[6] CAIDA Dataset for AS relationships; http://www.caida.org/research/topology/#Datasets.