MPC: Popularity-based Caching Strategy for Content Centric Networks

César Bernardini, Thomas Silverston, Festor Olivier

MPC: Popularity-based Caching Strategy for Content Centric Networks

César Bernardini*†, Thomas Silverston*† and Olivier Festor†
*Université de Lorraine, LORIA, UMR 7503, Vandoeuvre-les-Nancy, F-54506, France
†Inria, Villers-les-Nancy, F-54600, France
Email: {cesar.bernardini, thomas.silverston, olivier.festor}@inria.fr

***Abstract*—Content Centric Networking (CCN) has recently emerged as a promising architecture to deliver content at large-scale. It is based on named-data where a packet address names content and not its location. Then, the premise is to cache content on the network nodes along the delivery path. An important feature for CCN is therefore to manage the cache of the nodes.**

In this paper, we present Most-Popular Content (MPC), a new caching strategy adapted to CCN networks. By caching only popular content, we show through extensive simulation experiments that MPC is able to cache less content while, at the same time, it still achieves a higher Cache Hit and outperforms existing default caching strategy in CCN.

I. Introduction

The Internet is currently mostly used for accessing content. Indeed in the 2000s, P2P traffic for file-sharing counted for about 80% of the overall Internet traffic. Nowadays, video streaming services such as Youtube represent the most important part of the Internet traffic. It is expected that the sum of all forms of video (TV, VoD and P2P) will be approximately 86% of global consumer traffic by 2016 [1].

While the Internet was designed for -and still focuses on- host-to-host communication (IP), users are only interested in actual content rather than source location. Hence, new Information-Centric Networking architectures (ICN) such as CCN [2], NetInf [3] and Pursuit/PSIRP [4] have been defined giving high priority to efficient content distribution at large scale. Among all these new architectures, Content Centric Networking (CCN) has attracted considerable attention from the research community [5].

CCN is a network architecture based on named data where a packet address names content and not its location. The notion of host as defined into IP does not exist anymore. In CCN, the content is not retrieved from a dedicated server, as it is the case for the current Internet. The premise is that content delivery can be enhanced by including per-node-caching features as content traverses its distribution path across the network. Content is therefore replicated and located at different points of the network, increasing availability for incoming requests.

An important feature for CCN is to manage the cache of nodes with caching strategies and replacement policies, which decide whether to cache and in case the cache is full, the element to be replaced respectively. Recent works have mostly focused on cache replacement policies [6] [**?**] (MRU, MFU, LRU, FIFO) over several topologies as binary trees [7] or educational ISP [8]; even comparing improvements via other features as the multipath support [9] or investigating theoretical cache properties [10]. Regarding caching strategies, some study shows that enormous cache memory would be necessary in order to achieve high caching performances [11], while other [12] claims that caching less can achieve a similar level of performances by only storing content in a subset of nodes along the delivery path. It is therefore essential to design efficient caching strategies adapted to CCN networks.

In this paper, we present Most Popular Content (MPC), our new caching strategy designed for CCN network. Instead of storing all the content at every node on the path, MPC caches only popular content. MPC caches less than CCN default strategy but still improves in-network caching performance while -at the same time- decreases resource consumption.

The reminder of this paper is organized as follows. Section II describes MPC, our new caching strategy for CCN. Section III presents the simulation environment while Section IV shows the performances of MPC. Finally, Section V concludes the paper and presents future work.

II. Most Popular Caching

In this section, we first provide a brief overview of CCN, and then we present our new caching strategy MPC.

A. CCN Overview

CCN architecture is mostly based on two primitives: *Interest* and *Data*. A consumer requests content by sending an *Interest* message in the network; any node hearing the request and having the data can issue a response with a *Data* message. The content is therefore transmitted to the consumer and every node on the delivery path can cache the data. With no clearly defined caching strategy, the CCN default caching strategy always stores content at all nodes on the delivery path (*Always* strategy). This strategy has shown the best results in comparison with other ones [8]. Caching less by only storing content in a subset of nodes along the delivery path achieves also similar cache performances [12]. Since the first approach could lead to replace popular content by unpopular one, and based on the second *caching less* approach, we argue that caching only popular content will allow to achieve high performances and save resources at the same time.

Therefore, we designed MPC, *Most Popular Content*, a new caching strategy for CCN, where nodes cache only popular content.

B. Most Popular Caching Strategy

In MPC, every node counts locally the number of requests for each content name, and stores the pair *(Content Name; Popularity Count)* into a *Popularity Table*. Once a content name reaches locally a *Popularity Threshold*, the content name is tagged as being popular and if the node holds the content it suggests its neighbor nodes to cache it through a new *Suggestion* primitive. These suggestion messages may be accepted or not, according to local policies such as resource availability.

As the popularity of a content can decrease with time after the suggestion process, the Popularity Count is reinitialized according to a *Reset Value* in order to prevent flooding the same content to neighbors.

MPC Strategy influences directly in CCN node requirements. In addition to CCN cache space required, MPC needs an extra space to store the *Popularity Table*. For instance, keeping one million entries in the table means $1GB$ of RAM memory using $1023B$ per content name and $1B$ for the Popularity Count (we used fixed length for the name to simplify calculation). We expect to share dedicated caching memory with MPC tables in order to perform fair comparisons.

C. MPC Example Scenario

We present in this section an example to clarify the MPC workflow. In this example, nodes are requesting content located around the network and we explain how the MPC strategy works in comparison with the default CCN strategy. The example is depicted in Figure 1, and Figure 1(a) describes the scenario and the sequence of content requests. Fig. 1(b) and 1(c) show the final states after applying MPC and CCN caching strategies respectively. In our scenario, three nodes A, B and C will request a popular content d_1 initially stored at node D. Node A will also request an unpopular content e_1 initially stored at node E

Using MPC strategy on Fig. 1(b), when node A sends an *Interest* message for content e_1, e_1's popularity is increased in the *Popularity Tables* of every node along the path [A, C, D, E], i.e., e_1's popularity is set to 1 at nodes A, C, D and E. Similarly, when B sends an *Interest* message for content d_1, the popularity of d_1 is set to 1 along the path [B, C, D]. Since content d_1 is a popular one, A and C send in turn an *Interest* Message for d_1. The *Interest* message from A will increase d_1's popularity to 2 at C and D (through the path [C, D]). Finally, C requests d_1, increasing d_1's popularity to 3 at node C and D. At this moment, D is the only node to hold the content d_1. Assuming that d_1's popularity has reached the *Popularity Threshold*, D spreads *Suggestion* messages to his neighbors C and E. C and E will therefore cache the content and will be able to reply to upcoming requests and transmit the content d_1. For instance, if there are interested nodes close to A, their requests will be redirected towards C and not D, reducing the number of hops to get the content. After d_1 is spread to its neighbors, D will reset d_1's popularity at the *Reset Value* in order to prevent continuously flooding the same content to neighbors.

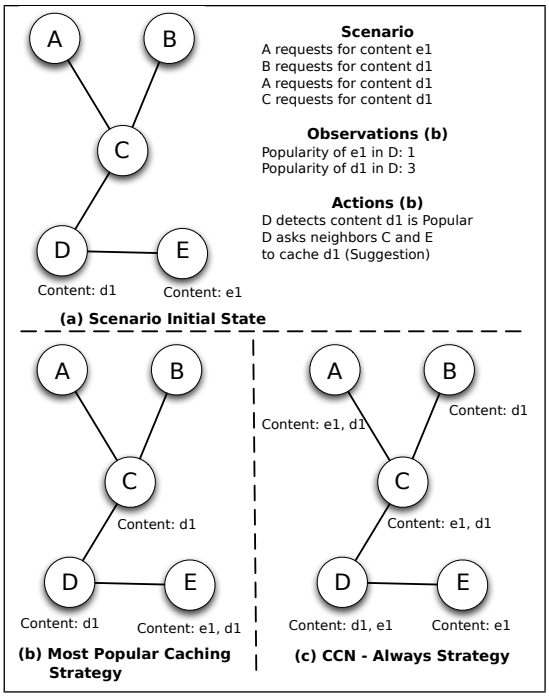


Fig. 1: MPC Workflow Example

. (a) Initial state of the network and the scenario; (b) MPC final state; (c) Final state of CCN with the Always strategy.

Figure 1(c) describes the CCN final state after using the default *Always* caching strategy. When the first request is sent by A, content e_1 will be cached at every nodes along the path [A, C, D, E]. Then, according to our scenario, content d_1 will be cached along the path [B, C, D] after the first request from B. The request from A will be directed only through C and d_1 will be cached into A. Finally, node C stores already the content since it has been cached from previous requests.

To summarize this scenario, CCN counts three replicas of e_1 at nodes A, C and D and three of d_1 at nodes A, B and C (Fig. 1(c)). MPC counts only two replicas of d_1 at nodes C and E, and e_1 has never been replicated (Fig. 1(b)). Clearly, by caching only popular content, our proposed strategy MPC is saving resources such as memory, bandwidth and the number of caching operations compared with the CCN Always strategy. The duplication of d_1 at node E has been opportunistic since E never requests for it (Fig. 1(b)). However, node E may receive upcoming requests for d_1 since it is a popular content.

III. Simulation Setup

In order to evaluate our new strategy MPC, we use ccnSim [8], a chunk-level CCN simulator, developed in C++ over the Omnet++ framework. The simulator was first designed to evaluate the CCN performance with different strategies and scenarios. To promote cross comparison between MPC and CCN, we detail the simulation environment, including ccnSim inherent parameters and network topologies.

Parameter	Value	Parameter	Value
Run/Simulation	10	Request Rate	50 req/s
Warmup Phase	True	Chunk Size	10 kb
Multipath	Disabled	Caching Replacement	LRU
Routing Strategy	Closest	Caching Strategy	Always

TABLE I: ccnSim parameters setting

	Segment	N	δ	σ_γ	$\Delta[ms]$	D
Abilene	Core	11	2.54	0.19	11.13	8
Tiger2	Metro	22	3.60	0.17	0.11	5
Geant	Aggr	22	3.40	0.41	2.59	4
DTelekom	Core	68	10.38	1.28	17.21	3
Level3	Core	46	11.65	0.86	8.88	4
Tree	Ref.	15	2.54	0.21	1.00	3

TABLE II: Properties of the network topologies [8]

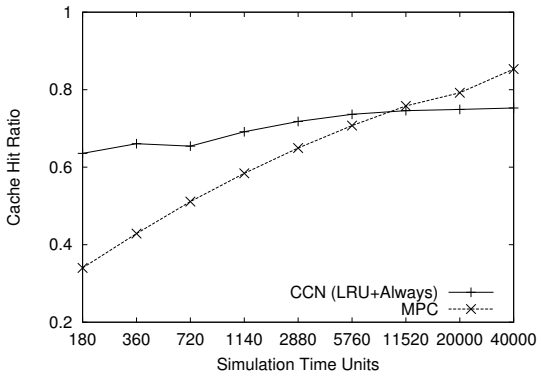


Fig. 2: Evaluation of the simulation time

The ccnSim configurable parameters are depicted into Table I. For all our simulations, CCN nodes use the *Always* caching strategy along with the *LRU* caching replacement policy. We choose these policies because they have shown the best performance for CCN [8]. Other parameters are presented in the table such as routing policy, chunk size or requests rate.

In ccnSim, the popularity of files have been modeled following a MZipf distribution function [11] [8]. In our experiments, we choose the same distribution function to model the file requests. For a Youtube-like catalog of 10^8 files, it means that 99% of the requests are concentrated approximately into the 6,000 most popular files.

As several network topologies are included into ccnSim, we use these topologies which are then described in Table II.

All along the experimentation section, we use two scenarios defined in Table III. The first scenario S_{small} is the scenario commonly used for CCN evaluation [7] and it consists of a catalog of 10^4 files with 10^2 chunks in average per file; cache size is fixed at 10^3 chunks per node. The second one $S_{youtube}$ is a larger-scale scenario with a Youtube-like catalog containing 10^8 files with 10^3 chunks per file: approximately a catalog of $1PB$. Ratio of caches over total chunks has been set to $\frac{C}{|F|F} = 10^{-5}$; in this scenario, cache size is then set to 10^6 chunks ($10\ GB$).

Then, for each experiment, we randomly set one catalog and 8 requester nodes on the topologies. We performed 10 runs per simulation and provide the average value.

IV. Performance Evaluation of MPC

In this section we present the results of our simulations. We evaluate the performance of MPC according to the following metrics:

Parameter	Description	S_{small}	$S_{Youtube}$
F	avg. chunks per file	10^2	10^3
$\|F\|$	number of files	10^4	10^8
C	number of chunks stored per cache	10^3	10^6
α	MZipf exponent parameter	1.5	1.5
q	MZipf plateau parameter	0	0

TABLE III: Simulation experiments scenarios

- *Cache Hit Ratio*: the probability to obtain a cache hit all along the path from a requester to a cache node;
- *Stretch*: the number of CCN hops that the data chunk has travelled in the network with respect to the server storing original copy;
- *Ratio of Cached Elements*: proportion of cached elements with regards to the total number of Interests;
- *Diversity*[8]: the ratio of different chunks stored in the caches.

As MPC introduces new parameters such as *Popularity Table, Popularity Threshold* and *Reset Value*, the first part of this section will entail the tuning of these parameters to infer the appropriate criteria. The simulation time will also be studied. Secondly, we will present the comparison between MPC strategy and CCN with *Always* strategy.

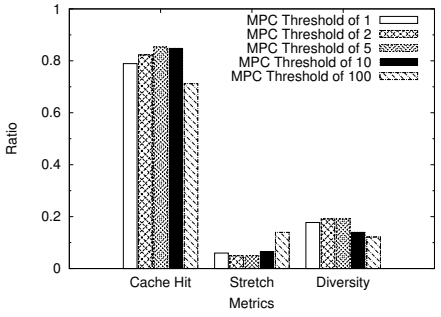
A. MPC Parameters Tuning

The selection of MPC's parameters has been done according to an extensive simulation process over the $S_{youtube}$ scenario and the Tree topology.

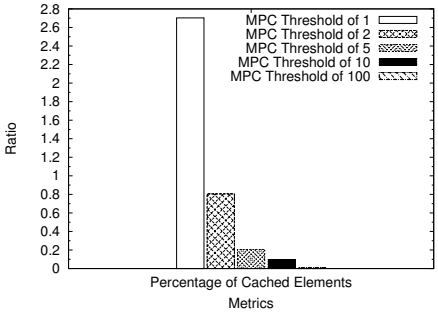
The simulation time is an important parameter for simulating accurately the different strategies. The ccnSim simulator warms up placing chunks in the caches according to a statistical simulation of the strategy. As *Popularity Tables* are not warmed up, MPC needs also time to start caching popular content. We therefore evaluate the time MPC needs to outperform CCN. We varied simulation time up to 40,000 Simulation Time Units (STU). CCN and MPC *Cache Hit* results are presented on Figure 2.

Clearly, for short time simulations and until 11,520 STU, CCN reached higher Cache Hit Ratio than MPC. Then the CCN's Cache Hit ratio remains stable while MPC's one is increasing and outperforms CCN. For instance, at 40,000 STU, Cache Hit Ratio is 85% for MPC and only 75% for CCN. In the rest of this paper, we focus on long time simulations and set simulation time to 40,000 Simulation Time Units.

Regarding the *Popularity Threshold* parameter, Fig. 3a and Fig. 3b show results with different threshold values. Clearly, *Popularity Threshold* set to 5 shows the highest Cache Hit, Diversity and lower Stretch than other



(a)



(b)

Fig. 3: Popularity Threshold Parameter

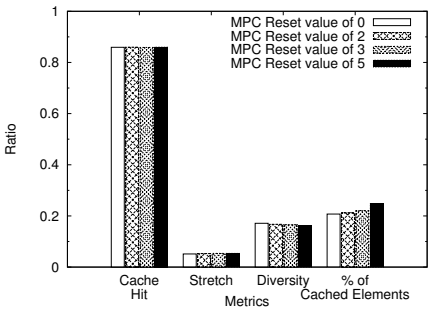


Fig. 4: Reset Value Parameter

tested values (Fig. 3a). At the same time, the ratio of cached elements is kept very low at only 20% (Fig. 3b). A lower value for this threshold (e.g. 1) may cache up to 2.7 times more than with the CCN default strategy.

As explained before, when a content name reaches the *Popularity Threshold* and is spread through its neighbors, we need to reset this value in case of future requests for this content. As shown in Figure 4, even though the cache hit is at the same rate for all the tested values, a *Reset Value* set to 0 is more appropriate because it exhibits slightly higher diversity, less stretch and replications of the same content in the cache.

Since MPC uses a table to keep track of the content name's popularity, we need to choose an appropriate table size, without wasting the memory of cache nodes. Figure 5 shows that *Popularity Table* with 2,5 millions of entries has the highest Cache Hit, Diversity and keep the Stretch at a low value. By assuming that a content name is 1KB, it means 2.5GB of memory. Then, the cache-size space is shared with MPC requirements (*Popularity Table*), which stands for 25% of caching resources (2.5 GB in case of the $S_{Youtube}$ scenario of the total 10 GB for caches).

To sum up this evaluation part, we tune our MPC parameters for the upcoming experiments: simulation time is set to 40,000, *Popularity Threshold, Reset Value* and *Popularity Table size* are set to 5, 0, and 2.5 GB respectively.

B. MPC vs. CCN Always Strategy

In order to compare fairly MPC and CCN/Always, we first performed the same experiment with the S_{small} scenario as in [8]. It consists in varying the ratio of the cache over the total number of chunks with different popularity distribution function (varying the MZipf exponent from 0.5 to 2.5) and observes the Cache Hit ratio. The results are shown on Fig. 6a and Fig. 6b for CCN and MPC respectively. The similarity in the charts shows how likewise they behave for every configuration. However, the Table IV depicts the *Ratio of Cached Elements* for MPC. In CCN, using the *Always* policy, content is cached by all nodes it passes by: the ratio for CCN would always be 1. In the MPC case, the ratio never exceeds 38%, which means MPC is caching far fewer elements than CCN.

	Ratio of Cache			
MZipf exponent	1/10	1/100	1/1000	1/10000
0.5	ϵ	ϵ	ϵ	ϵ
1	ϵ	ϵ	ϵ	ϵ
1.5	0.22	0.09	ϵ	ϵ
2	0.33	0.31	0.04	ϵ
2.5	0.37	0.38	0.15	ϵ

TABLE IV: Ratio of cached elements for MPC. This ratio is 1 for the CCN/Always policy. (ϵ is for value close to 0.)

We will now compare MPC with CCN by using the $S_{Youtube}$ scenario with all the topologies. On Fig. 7a, clearly MPC Cache Hit ratio is higher than CCN and above 85%. Even when CCN reaches its highest results with Level3 or DTelecom topologies, MPC still outperforms CCN. The *Ratio of Cached Elements* is presented in the Figure 7b. For CCN with the *Always* policy, content is always cached and it reaches a ratio of 100%. The MPC strategy caches much less content than CCN for the Tree, Abilene, Geant and Tiger topologies (approximately 20%), performing less caching operations and saving memory. MPC caches more content with DTelecom and Level3 topologies (80% and 60% respectively) but is still at lower rate than CCN. The increase of cached elements for these two topologies is due to the high connection degree of nodes (10 or 11 neighbors in table II). In such highly connected topology, MPC is caching *off-the-path* because it sends *Suggestion* messages to store content to more neighbors.

Due to the space limitation, we do not present the Figure for the Stretch metric. MPC and CCN exhibit similar results: the Stretch of CCN is about 10% for all the topologies and the MPC's one is slightly lower at 8%. By caching only popular content, MPC strategy is still able to cache content close to requesters.

The Diversity metric is presented on Fig. 7c. The CCN Diversity ranges from 28% to 35% for all the topologies and the MPC Diversity is much lower from 3% to 18%. Regarding Diversity, it was expected that MPC is less efficient than CCN since MPC has been designed in order to cache only popular content, limiting the diversity of the chunks in the cache of nodes. However, neither CCN nor MPC achieves high Diversity and further work is needed to improve Diversity in

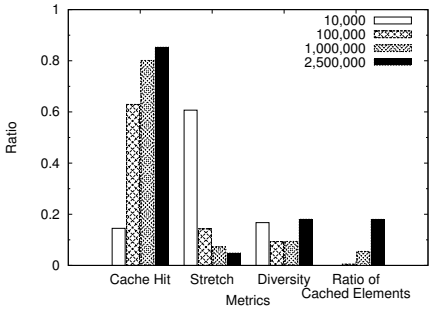
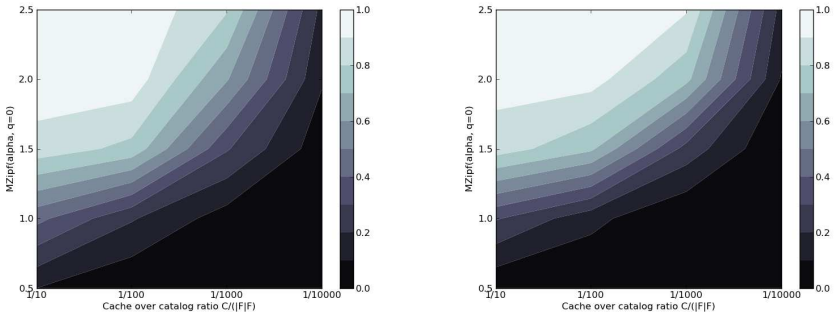
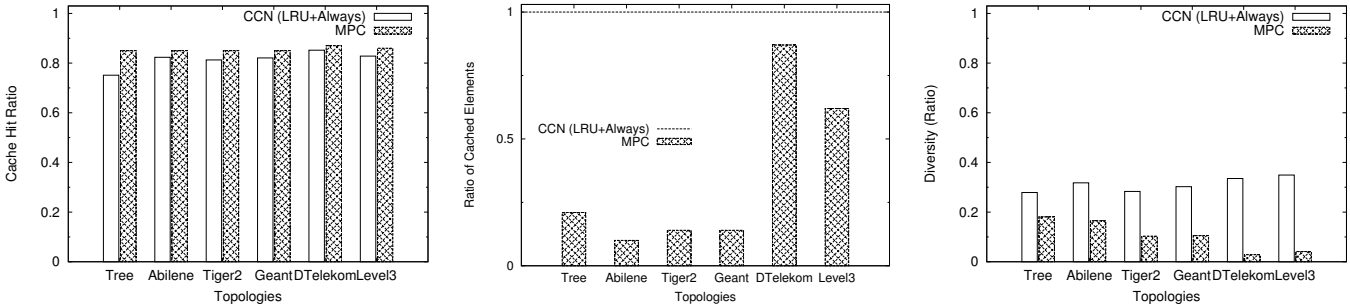


Fig. 5: Popularity Table Size Parameter



(a) CCN - LRU/Always (b) MPC

Fig. 6: Contour plot of Cache Hit



(a) Cache Hit Ratio (b) Ratio of Cached Elements (c) Diversity

Fig. 7: MPC vs. CCN (LRU/Always) over different topologies

CCN networks.

Finally, our simulation experiments show that MPC strategy outperforms CCN Always strategy since it achieves a higher Cache Hit Ratio. Moreover, MPC caches less content, saves resources such as memory and reduces the number of cache operations.

V. Conclusion

In this paper, we presented MPC, a new caching strategy for CCN networks. MPC strategy caches only popular content and reduces the cache load at each node. Our simulation experiments showed that MPC outperforms the CCN/Always default strategy. MPC achieves a higher Cache Hit Ratio and still reduces drastically the number of replicas of elements. By caching less data and improving the Cache Hit Ratio, MPC improves network resources consumption.

As future work, we expect that our strategy could serve as a base for studying name-based routing protocols. Being a suggestion based mechanism, it is feasible to adapt it to manage content among nodes, to predict popularity and to route content to destination.

Acknowledgement

The authors would like to thank the *Conseil Régional de Lorraine* for its financial support.

References

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2011-2016," Cisco, Tech. Rep., feb 2012.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658941

[3] C. Dannewitz, "NetInf: An Information-Centric design for the future internet," 2009.

[4] N. Fotiou, G. C. Polyzos, and D. Trossen, "Illustrating a publish-subscribe internet architecture."

[5] [Online]. Available: http://www.ccnx.com

[6] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "Multicache: An overlay architecture for information-centric networking."

[7] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of ccn-caching trees," in *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I*, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 78–91. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008780.2008789

[8] D. Rossi and G. Rossini, "Caching performance of content centric networksunder multi-path routing (and more)," Telecom ParisTech, Tech. Rep., 2011.

[9] ——, "Evaluating ccn multi-path interest forwarding strategies," in *CCNxCon*, 2012.

[10] D. S. M. Elisha J. Rosensweig and J. Kurose, "On the steady-state of cache networks," in *IEEE Infocom*, 2012.

[11] C. Fricker, P. Robert, J. Roberts, and N. Sbihi, "Impact of traffic mix on caching performance in a content-centric network," Feb. 2012. [Online]. Available: http://hal.inria.fr/hal-00666169

[12] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," in *Networking (1)*, 2012, pp. 27–40.