A Critical Perspective Towards CCN

Radu Alexandru Badea[1,*], Eugen Borcoci[1] and Radu Lupu[1], Jordi Mongay Batalla[2]
[1]*Politehnica University of Bucharest, Telecommunications Dept., Bucharest, Romania*
[2]*Warsaw University of Technology – Poland*
[*]Corresponding author (E-Mail : rbadea@elcom.pub.ro)

***Abstract* - In the last years, the internet network research field has witnessed a plethora of new architectures and improvement proposals. One of the more remarkable is the Information / Content Centric Networking (ICN / CCN) design. While quite acclaimed at introduction time, several conceptual and implementation related problems were soon observed by some researchers. This paper will broadly describe the basic ideas of CCN and then analyze at least two issues that in our opinion could affect the proposed architecture. The paper will investigate, in the context of CCN, topics like *self-similar network traffic*, the *Least Recently Used (LRU) Caching Algorithm performance evaluation* and *Cache reuse probability estimation*.**

***Keywords* - Content Centric Networking (CCN); caching; self similar network traffic; LRU; cache reuse estimation;**

I. INTRODUCTION

In the current and future internet it is recognized that more and more the information / content is of primary interest and not its location. Therefore new approaches appeared, some of them revisiting the fundamentals of the TCP/IP architecture. The most representative are the so called Information/Content Centric networking. While notions of CCN are present in research projects done over the last 15 years (TRIAD[1], DONA[2], etc.), the most recent and more complete proposal for an CCN architecture was issued by Van Jacobson et. al.[3][4] as described briefly next. The core idea behind CCN is making the "transit network" content aware. Users requesting content items are usually not interested in their location. For such services, the content name (WHAT) is put in front; its location (WHERE) is secondary and should be transparent to the end user. From the networks point of view, routing is done by considering "Data Name" as replacing the classic internet routing where "Data Location" (IP address) is the first routing information considered. However, a mapping between the two should finally exist. In CCN exists two types of packets, called "Interest" and "Data". An User, wanting to reach a certain content, generates to the network an "Interest" packet, containing the data "Name". Any Node, receiving the "Interest", can answer with a "Data" packet, if its (cached) content matches the expressed "Interest". Otherwise the user request is forwarded towards the Content Source (where the data is initially stored) - each involved router device with CCN capabilities will inspect its own Cache and look either for an identical request, issued by the same or another user - at a previous moment in time - or even better, the requested packet is already cached at that router. In this last case, that router will transmit the packet directly to the requesting part, avoiding so the longer path to the Content Source.

The contribution of this paper consists in analyzing the caching process in CCN context, while considering the traffic characteristics..

Section II is presenting results obtained by different research groups (Park / Willinger, Leland / Taqqu etc.) related to *traffic self-similarity* and its influence on data caching at routing devices. In Section III, the paper authors are evaluating the *LRU caching strategy* on a desktop machine, in order to establish the required processing time for different object (packet) loads. Section IV will analyze the *router cache reuse probability* and the paper authors will propose an evaluation model for its estimation. Finally, in Section V, conclusions are drawn and future work aspects are noted.

II. SELF-SIMILARITY AND CCN

Many studies [5][6][7][8] have been dedicated to the research and understanding of web traffic structure and modeling in general. One important conclusion that arouse is the fact that network traffic is rather *self-similar* [18] in nature. For network traffic study, statistical self similarity is of much more interest. In this case, the "snapshots" taken at different time (or space) scales will resemble each other, but rather through their (second order) statistical parameters (that capture values as burstiness and variability [5]). Correlations "at a distance" in time are still nontrivial, decreasing polynomially and not exponentially (as in memoryless processes). These nontrivial correlations are also called long range dependence [5]. The most important cause for self-similar traffic is the heavy tailed distribution of file and object sizes in distributed systems [5][9]. This has direct implications on aggregated network traffic. The effects on queuing on routers are such that buffering is not very effective in conditions of self similar traffic, the performance analysis works suggesting that buffer capacity at routers should be kept small, increasing at the same time the link bandwidth [5]. To mention is also the study done by Jon M. Peha from Carnegie Mellon University. In his research [10], he investigates another interesting option regarding self-similar causality, namely the possibility that protocol retransmission mechanisms are causing or at least reinforcing this phenomenon.

One consequence is the fact that "unexpected" events such as packet bursts are possible. Subtle effects as long range dependency imply traffic correlations over "long" periods of time. The presented studies have shown that for many cases, buffer increase on network elements will not improve significantly the overall performance. Especially the effect of packet bursts on CCN caching has to be more carefully analyzed. Since CCN has at its core, among other things, a

978-1-4799-2385-4/14/$31.00 ©2014 IEEE

caching (buffering) infrastructure, it remains to be shown by the CCN proposal authors [3][4] how the presented aspects affect the CCN architecture as a whole. Regardless of the caching mechanism, for CCN to work and work correctly in the proposed manner, there are many assumptions that have to be fulfilled in order that the whole chain of storage, forwarding and processing behave as expected. We will regard at some of these constraints in the next paragraphs.

III. LEAST RECENTLY USED (LRU) CACHING EVALUATION

The authors of CCN suggest as caching strategy the "LRU" (Least Recently Used) algorithm, as follows: "To maximize the probability of sharing, which minimizes upstream bandwidth demand and downstream latency, CCN remember arriving Data packets as long as possible (LRU or LFU replacement)" [4 - Page2]. There are some performance studies done for different caching types and for LRU at least, there seem to be some performance problems, in the sense that implementing a "true" LRU pattern is computationally expensive [11]. To get a better understanding of what "expensive" means in this context, we have implemented a little performance test application, with the goal to give us a broad estimation of the amount of processing needed for an LRU algorithm. The code is partially based on the open source project that can be found at [12].

The caching implementation is based on a linked list (to store the removal order) and a dictionary structure (for object storage). Besides that, a timer is testing all "n" milliseconds if Cache cleanup (due to time expiration of individual objects) has to be performed.

We have done 3 different tests, as can be seen in "Fig. 1". The "SizeBased" and "TimeBased" ones are adding values to the Cache (*no reading*). We add in these scenarios 100.000 objects (strings of 36 bytes as the one given as example in the program window) - single threaded. After 10.000 objects are added to the Cache or 100ms elapsed (depending which event occurs first), each new object add will replace an older (the oldest) cache item. Note please that each object in the storage is a {"key:value"} pair. So, for instance, if the Cache has a size of 3 elements as follows : {"1:a", "2:b", "3:c"} and we add a 4th item - {"4:d"} - the new Cache will be {"2:b", "3:c", "4:d"}, with {"1:a"} being removed from the storage. For the SizeBased test, there is no data timeout. The last test (Threading) is measuring the performance in case of random multithreaded access. In this situation, several threads are randomly *writing and reading* data from the Cache, *the stored values being shorter (several bytes long) random strings*. Also, since this test is multithreaded, the application can take advantage of the multi-core processor. "Table I" is presenting results for several Object / Cache combinations. The test was run on an Intel i3 (2.6GHz CPU) system with 8GB of RAM. For 10.000, 100.000 and 250.000 Objects, we have generated results for objects of 36 bytes and 3600 bytes in size.

We should now interpret the resulted data. The first observation that can be done is that processing time has an approximately linear increase with the number of objects. Further, object size has almost no performance influence (increasing object size 100 times is affecting processing time

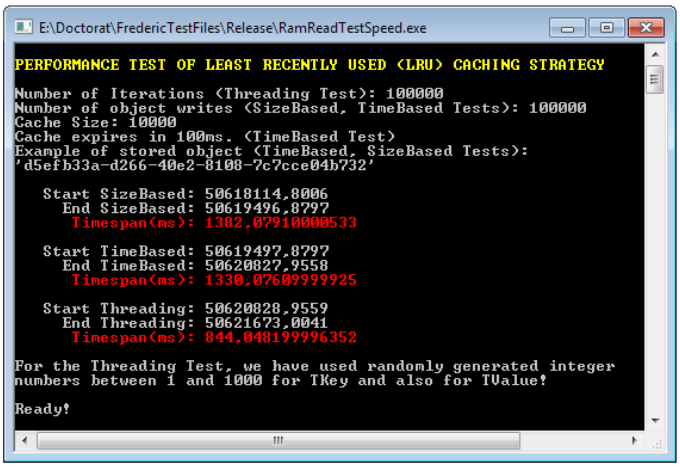

Figure 1. Performance Test of LRU Caching Strategy

only marginally). The last observation can be explained by the fact that objects are stored in a "Dictionary", that implements a sort of hash table, with constant seek time. More, desktop compilers, as the one used for this sample, are optimized for dynamic memory allocation, used at object creation. Probably the situation closest to reality is the 250.000 / 3600 combination, since it yields a Cache work value of about 1000MB. For this case, processing time is between 2 and 3 seconds for our tests. Also, the adopted implementation, with dynamic memory allocations and linked list has some performance limitations on real time embedded systems, as mentioned in [13][14].

Next, we will evaluate the traffic passing a standard router each second. Consider a device that is (de)multiplexing 10 x 1Gbps links to 1 x 10Gbs ("Fig. 2"). On each 1Gbps link, we have roughly a throughput of 100MB(ytes)/sec. If we presume that for each input port (physical connection) of the device, we have buffer memory sized to 1000MB RAM, we can make following computation:

1000MB (RAM) / [(100MB/sec) *0.7] => 15 seconds, to fill the buffer at 70% (peak) traffic on the link, by storing all incoming packets (data). After (at most) 15 seconds, the content in the buffer must be replaced, or new arriving content will not be stored any further (due to lack of space). That is, ***we have a timeframe of 15 seconds to reuse the cached data.*** The 15 seconds above would be possible ONLY under "ideal" conditions, where the writing overhead is virtually zero.

Cache processing time will depend on the selected number of objects to be stored. For instance, if we consider the average payload internet packet size being 1500 Bytes, that implies about 600.000 packets (objects) to reach the 1000MB discussed earlier. In this case, processing time will go towards 5 to 6 seconds, with an average processing throughput of 600.000/6 = 100.000 packets/sec., or 100.000 (packets) * 1500 (Bytes/packet) = 150 Mbytes/sec. At least for this solution, fewer but larger packets are better. On the other hand, if caching is to be done on the faster 10Gbps link, processing time has to be drastically increased in order to cope with data throughput. In that situation, memory will be filled in below 1 second in the worst case scenario. Please notice that in our test

Table I
LEAST RECENTLY USED (LRU) CACHING ALGORITHM - EVALUATION RESULTS

Number of Objects	Cache Size (Objects)	Object Size (Bytes)	Size-Based (ms)	Time-Based (ms)	Threading (ms)
10.000	1.000	36	132	125	105
10.000	10.000	36	81	74	84
10.000	10.000	3600	79	82	85
100.000	10.000	36	1382	1330	844
100.000	100.000	36	748	715	1427
100.000	100.000	3600	735	730	1443
250.000	250.000	36	1769	2714	2679
250.000	250.000	3600	1796	2731	2639
1.000.000	100.000	36	12626	12424	9927
1.000.000	1.000.000	36	7359	12636	9511
10.000.000	1.000.000	36	124688	125058	103180
10.000.000	10.000.000	36	86726	130417	150392

environment, no further processing was done, besides the simplest one, to insert and remove packets (objects). The overhead generated by decision making algorithms applied to received data, will surely increase processing time even more.

What conclusion can we point out from the presented calculations? Our test suggests that caching of large amounts of real time data with LRU-like algorithms has some performance penalties associated. Processing time is rather high, for usual buffer capacities being several seconds long. The cached packet reuse timeframe is also (severely) limited, being in the order of seconds or below.

IV. CACHE REUSE PROBABILITY ESTIMATION

The next topic that we will analyze is the likelihood of Cache content being reused among several requests. We are interested especially in high volume data, such as streaming media. For this purpose we will look a little bit closer to "youtube.com" statistics. We can get some insight about "youtube" traffic by using something like "youtube.com trends" [15]. This tool allows us to see *what* videos (by number of views over the last days) are most popular at the moment, and more important, to see *where* they are popular. The first thing to be noticed is that videos tend to be grouped by regional (US) and cultural (Europe) factors. In other words, while in the US we see a high degree of correlation (about 80% of top 10 videos for the given moment in time, are shared among regions, even remote ones), in Europe the situation is different. For no country group we find an overlapping of video preferences exceeding 30% (that is, 3 videos out of top 10 of each country, shared by at least 3 countries), and those 30% are for Germany-Switzerland-Austria constellation, where the three countries speak the same language and have a strong cultural affinity.

Following geographic area related traffic estimations can be done, based on the findings presented hereto. The most popular global videos (youtube) have about 1 Million daily views on average, each. If we consider a time span of 12 hours when these videos are watched, we can do the following computations, assuming a relatively uniform distribution of requests:

time_in_seconds = (12 hours) * (3600 sec) = 43.200 seconds.

average_views_per_second = 1.000.000/43.200 = 23 views/sec

So, each second another 23 people start streaming the video in question. Consider next simplifying assumptions:

- Geographic Area are the USA
- Select only top 300 urban areas with population over 100.000 (and 300.000 average)
- We assume a high speed DSL line per 10 inhabitants for the mentioned urban areas.

Combining the above information, we can state the following:

For an average city of 300.000 persons, we have 30.000 high speed DSL lines capable of video streaming. Because of different time zones (continental US has 4 main time zones) we assume that at any given moment, no more than half of the internet connections (persons) will be able to receive the video. So, from the 300 urban areas, only 150 will be considered as potential candidates for concurrent streaming. Following computation results:

30.000 * 150 = 4.500.000 potential connections where our streaming requests could originate.

To cover these subscribers, we need about 50.000 * 1Gbps lines (we have aggregated the traffic from 100 subscribers, assuming a line speed of 10Mbits/sec/subscriber => 100 *10Mbit = 1Gbps). The number of lines was also rounded up to 50.000, introducing a 10% safety margin.

The 23 request/sec for our video, that we have computed before, will arrive randomly to one of the 50.000, 1Gbps (aggregated) endpoints (routers). For cached data to be reused, following conditions and limitations exist simultaneously:

- After a packet has been cached, there is a small finite time to reuse the data, as described in section III (in our example, about 8 seconds - i.e. *time overlapping*)

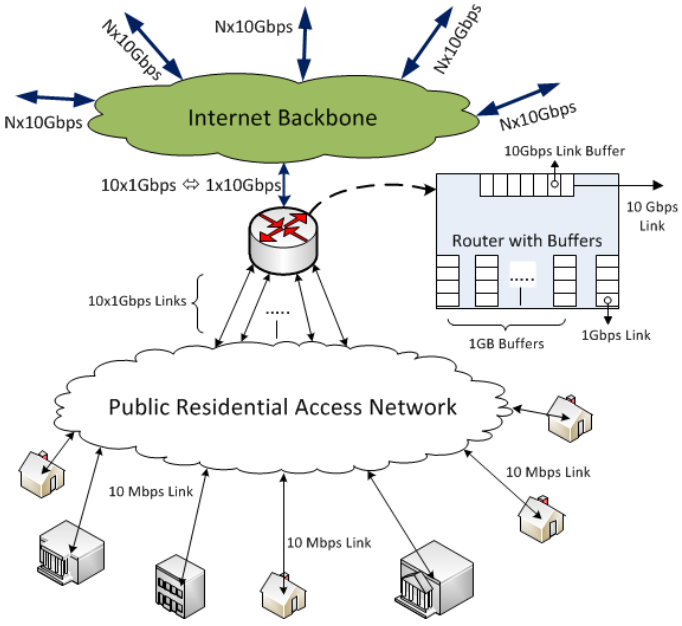

Figure 2. Core Router with Caching Services

- A new request is looking for EXACTLY the cached data (i.e. *data overlapping*)
- The new request is hitting the router where our data is cached (i.e. *path overlapping*)

In order to estimate the data reuse probability among requests, we will construct following scenario, as described next.

Let us "quantize" the *request target space* into slices of one second each. So, for example, a video file with a total length of one minute will be represented by our model as 60 samples of 1 second each. Further, we assume that at a time moment , name it "t0", we begin to observe traffic on the 50.000 routers from above. We want to track the traffic for one defined file, name it "FILE A", and from "FILE A" we are interested only in the first sample, "SAMPLE 1" (of 1 second length). To simplify further our start conditions, we assume also that at moment "t0" we do not have memorized "SAMPLE 1" anywhere in the network under observation. Now, as we have found out earlier, we also assume an average of 23 requests/sec for this file (so, this is an highly popular file).

In our scenario, we want to maximize the cache hit probability, and establish an upper bound of cache efficiency. We want to determine the relative probability that at a moment "t0 + Delta(t)" a new test request will hit a router with cached data ("SAMPLE 1").

Each new second, starting from "t0", another 23 requests for "SAMPLE 1" (the first slice of our popular file) hit the network. We will name such a bunch of requests, a "group" (later, "cache group"), giving it also an ordinal number, starting at 1. Once again, in order to maximize cache hit probability for our test request, we assume that each request from a "group" will arrive at a different router, where the corresponding server response ("SAMPLE 1") is cached. That is, each second, 23 new and different routers will cache the packet "SAMPLE 1". In "Fig. 3" we can see this illustrated. At moment "t0+1s" the first 23 routers will cache "SAMPLE 1". This cache "group" is named "cache 1" on the picture. The sample will stay in the router buffer for 8 seconds, when it is removed. Each square on a line in the image represents the cache of a "group" of 23 routers that have cached "SAMPLE 1", and the evolution in time of this cache for the next seconds. Thus, we have:

- At moment "t0+1s", "cache 1" will store "SAMPLE 1" for the next 8 seconds (till "t0+9s")
- At moment "t0+2s", "cache 2" will store "SAMPLE 1" for the next 8 second (till "t0+10s")
- At moment "t0+10s", "cache 1" kicks off "SAMPLE 1" since time limit for this packet has been reached. (We assume a cache duration of no more than 8 seconds for a packet).
- Also at moment "t0+10s", "cache 10" is caching "SAMPLE 1".
- On the columns we see the aggregated caching for "SAMPLE 1" at moment "t0+k". For example, at "t0+5s", 5 different router groups (23*5 = 115 machines) have cached "SAMPLE 1"

From time evolution of the aggregated cache presented in the image, we see that for the first eight seconds the overall storage of "SAMPLE 1" increases with each new arriving bunch of requests. At "t0+1s", only 23 routers have cached "SAMPLE 1". At "t0+9s", we have "SAMPLE1" stored by 9*23 = 207 routers, where our overall cache also peaks. After "t0+9s", the caching of "SAMPLE 1" cannot be further increased for the same request arrival rate, since for each new 23 requests, a group that previously stored "SAMPLE 1" will reach its time limit and erase the packets - at best, the peak number of storing routers will be preserved, with a router space shifting. This is marked on the image with the two blue columns. At "t0+9s", {cache1...cache9} store "SAMPLE 1", at "t0+10s", {cache2...cache10} store the same packet.

Now, we consider our test request for "SAMPLE 1" arriving at moment "t0+9s". The probability mass function is also assumed to be the uniform distribution, that is, each router out of the group of 50.000 is an equal probable target for the request. If we define an "event" (in the statistical sense), as being the subset of the sample space of 50.000 routers (caches), that contains the ones that stored at moment "t0+9s" the packet "SAMPLE 1", the probability of this "event" is given by**: Pr = 207/50.000 = 0.4%**.

As a conclusion, we can now state that under the assumptions of our model, the **probability of cache reuse for a packet is about 0.4%**.

Next, we should consider the limitations of the presented model, that can influence the outcome probability for an "event", in both directions (increasing or decreasing it).

- The model considered an total of about 9 Million high speed DSL (HS-DSL) lines for the whole US. If we take the data presented by the World Bank [16], there are 0.28 HS-DSL/100 people, implying about 300 * 0.28 = 84 Million HS-DSL lines, so the "real" figures are almost an order of magnitude higher than our estimate. This difference will evidently impact the number of routers, in an increasing sense.

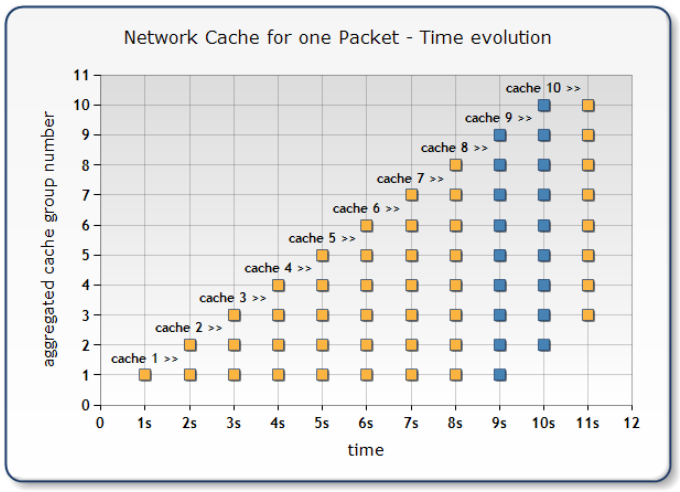

Figure 3. Network Cache - Time Evolution for "SAMPLE 1"

- The average connection speed is, for the US [17], about 6 Mbps , this is 60% of our estimate. Accordingly, traffic at aggregated routers of our model will represent more users per Gigabit, working towards a decrease of router number, if we want to maintain the 1Gbps reference measure for a caching router.
- We have presumed uniformly distributed (equal probable) requests for the router sampling space (Not to be confused with traffic generated by these requests, that is self similar as presented before!). While this could be true for high traffic content (popular videos as in our example), we suspect it is unlikely for lesser popular content. This has to be investigated further.
- Files (videos) with 1 million views a day (23 request/second), as we have used for our model, are, even for youtube standards, the top of the top - we find them among the 5 or 10 most popular videos. The rest, also very popular entries (say Top 100) , are at least 10 times below this number of requests a day. The consequence of this finding is that cache hit probability of 0.4% is plausible only for highly demanded content, decreasing rapidly with popularity decrease of content.
- Requests have also been uniformly distributed over the whole time frame of 12 hours, with an average of 23 requests/second. This is certainly not accurate. There will be time intervals with requests several times higher than our average, but also other intervals with substantially lower number of requests. This will produce a cache hit oscillation, with periods of increased and periods of decreased hit probability.
- The caching routers were intentionally considered to be at most several hops away from the subscriber (hence the 1Gbps aggregated traffic, from one or two hundred subscribers). Caching, to work well, has to be as close as possible to the "cache user". If we move the caching facility towards internet core (where inter-AS links exist), the probability of cache reuse will increase due to the smaller router space. On the other hand, caching time will also decrease due to the higher connection speeds (we pass from, say, 1 Gbps per port to 10 Gbps per port), leading to a falling reuse probability (the two aforementioned measures work somehow in opposite directions with regard to caching). Besides that, the further we are from caches "end user", the less relevant becomes caching compared to direct source request (that is, to download the content directly from the provider without any caching at all).

V. Conclusions and Future Work

This paper has presented a number of challenges that a modern, content aware network technology like CCN, has to master. Are the long term benefits prevailing against its drawbacks? Is the proposed architecture realistic in its assumptions? What potential problems could arise?

We have tried to highlight a few of these situations where, in our opinion, CCN has to bring more detailed and up to the point answers.

As mentioned in the paper title and also in the introductory chapter, besides our own experiments and contribution to the subject (Chapters III and IV), we have also mentioned results obtained by other researchers, that support our critical view about the CCN storage model (Chapter II). Since router caching is a fundamental element of CCN, the topic has to be analyzed from several points of view, one being traffic self-similarity, with the conclusions obtained by prestigious research groups. It was not our intention to investigate this relationship further than the mentioned conclusions.

In Section III, we have established that, at least for the assumed simulation environment, processing time is likely to be in the order of seconds and the timeframe where cached Content can be reused is also, at most, several (dozen) seconds long (depending on buffer size, link speed and processing overhead). After this time expires, a stored packet has either to be replaced with new Content or the arriving packets will not be cached any more till memory is freed.

Section IV has found a cache reuse probability (under the assumptions of the constructed model) for a packet close to 0.4% , for very high-demand Content, our estimation being that this value will decline fast for lesser popular data, making CCN router caching actually useless for most practical situations.

Our model has not as primary target the accurate computation of a reuse probability, rather an estimate of it. We have selected scenario parameters to generate a favorable value for the resulting probability. In fact, as discussed in the last part of chapter IV, most real world situations will likely pose harder challenges. Even if our prediction is wrong by an order of magnitude, the conclusions are still valid, a reuse probability of 4% for high demand Content is still very low by any standard. Here, a further, more detailed study is necessary, to reveal the influence of the aforementioned model constraints.

For example, not assuming a relatively uniform distribution of requests, but taking micro-region behavior into account,

could predict more accurately cache reuse for routers deployed to specified geographical areas. Especially urban areas with their fast growing connection bandwidths and subscriber numbers have to be closely investigated considering also other factors such as subscriber density, QoS, etc.

At the same time, there could be imagined application patterns where path caching is potentially useful. If we think about messaging services like Twitter or Facebook, where many small messages are produced, but with a high probability to be reused by their target group (the receivers and posters themselves), with the appropriate caching strategy, service providers could obtain performance improvements, moving demanded content closer to their users.

One main conclusion is that high volume, real time data caching on the (router) path, regardless if CCN aware or in a classical sense, is not suited to improve drastically traffic parameters. Some side effects could even have adverse effects. The topic has to be researched more in depth, taking many more aspects into account.

The overall conclusion could be that data caching is not well suited to be done in a "short term" fashion (seconds), an efficient and performance oriented strategy will need to store reusable Content a longer (indefinite) time as for example proxy servers and CDNs do. In a next paper we will present an alternative caching strategy, based on "smart" edge routers.

ACKNOWLEDGMENT

This work has been partially supported by the Research Project DISEDAN, No.3-CHIST-ERA C3N, 2014- 2015.

REFERENCES

[1] Stanford TRIAD CCN Project - http://gregorio.stanford.edu/triad/
[2] DONA - Data Oriented Network Architecture - http://www.eecs.berkeley.edu/Research/Projects/Data/102146.html
[3] CCNx - Jacobson - http://www.ccnx.org/
[4] V. Jacobson, D. Smetters, et.al. - Networking Named Content / PARC / SIGCOMM 2009 http://conferences.sigcomm.org/co-next/2009/papers/Jacobson.pdf
[5] Kihong Park and Walter Willinger - Self-Similar Network Traffic: An Overview, http://www.cs.purdue.edu/nsl/intro-ss-chap.pdf
[6] W. Leland, M. Taqqu, W. Willinger, and D. Wilson - On the selfsimilar nature of ethernet traffic - In Proc. ACM SIGCOMM '93, pp: 183–193, http://www.ece.ucdavis.edu/~chuah/classes/EEC274/refs/94LTW-selfsimilar.pdf
[7] J. Beran, R. Sherman, M. S. Taqqu and W. Willinger - Long range dependence in variable bitrate video traffic - IEEE Transactions on Communications 43, 1995, pp: 1566–1579.
[8] M. T. Lucas, D. E. Wrege, B. J. Dempsey, and A. C. Weaver - Statistical Characterization of Wide-Area IP Traffic - Proc. 6th IEEE Intl. Computer Communications and Networks Conf., Sept. 1997, pp. 442-7.
[9] Kihong Park, Gitae Kim, Mark Crovella - On the relationship between file sizes, transport protocols, and self-similar network traffic. http://www.cs.bu.edu/faculty/crovella/paper-archive/icnp96.pdf
[10] Jon M. Peha - Carnegie Mellon - Protocols Can Make Traffic Appear Self-Similar - http://users.ece.cmu.edu/~peha/self.pdf
[11] Nimrod Megiddo and Dharmendra S. Modha, Adaptive Replacement Cache (ARC), Computer (Volume:37, Issue:4) April 2004, pp:58 - 65 http://dbs.uni-leipzig.de/file/ARC.pdf
[12] LRU Codeplex Code - http://lrucache.codeplex.com/
[13] Remy Mahler - Dynamic memory allocation in embedded systems http://blog.bbv.ch/2012/07/16/usage-of-dynamic-memory-in-a-real-time-system/
[14] David Crocker - Dynamic Memory Allocation in Critical Embedded Systems http://critical.eschertech.com/2010/07/30/dynamic-memory-allocation-in-critical-embedded-systems/
[15] youtube trends - http://www.youtube.com/trendsmap
[16] World Bank - Fixed broadband internet subscribers 2011 http://data.worldbank.org/indicator/IT.NET.BBND.P2
[17] Sam Byford - Average connection speed 2011 http://www.theverge.com/2012/5/1/2990469/average-global-internet-speed-drop-us
[18] Self-Similarity: http://en.wikipedia.org/wiki/Self-similar_process