CCN-TV: a data-centric approach to real-time video services

Vincenzo Ciancaglini*, Giuseppe Piro †, Riccardo Loti*, Luigi Alfredo Grieco† and Luigi Liquori*
*INRIA - Sophia Antipolis (France)
Email: {vincenzo.ciancaglini, riccardo.loti, luigi.liquori}@inria.fr
†DEE - Politecnico di Bari (Italy)
Email: {g.piro, a.grieco}@poliba.it

***Abstract*—Content-Centric Networking (CCN) is a promising data-centric architecture, based on in-network caching, name-driven routing, and receiver-initiated sessions, which can greatly enhance the way Internet resources are currently used, making support for a broader set of users with increasing traffic demands possible. The CCN vision is, currently, attracting the attention of many researchers across the world, since it has all the potential to become ready to the market, to be gradually deployed in the Internet of today, and to facilitate a graceful transition from a host-centric networking rationale to a more effective data-centric working behaviour. At the same time, several issues have to be investigated before CCN can be safely deployed at the Internet scale. They include routing, congestion control, caching operations, name-space planning, and application design. With reference to application-related facets, it is worth noticing that the demand for TV services is growing at an exponential rate over time, thus requiring a very careful analysis of their performance in CCN architectures. To this end, in the present contribution we deploy a CCN-TV system, capable of delivering real-time streaming TV services, and we evaluate its performance through a simulation campaign based on real-world topologies.**

***Keywords*-Content-Centric Networking, Multimedia Services, Live Streaming**

I. Introduction

Due to the relevant importance that content sharing applications are going to play in the upcoming future [2], [3], the Content Centric Networking (CCN) rationale [4] has been proposed as a possible way to drive the current *host-centric* Internet paradigm towards a novel *content-centric* behaviour. It is based on in-network caching operations, receiver initiated data exchange, hierarchical content naming, and native support to security and privacy.

In a CCN, contents are split in chunks which are requested using opposite *Interest* messages, generated at the client side. Each *Interest* is then routed until it reaches a node which has, in its own cache, a copy of the requested item. Then, this copy is sent, as a *Data* message, back along the path the *Interest* had gone through. Intermediate nodes can cache the *Data* before forwarding it to the next node (more details on the CCN working behavior is provided in Sec. II).

Since its born, the CCN vision has gained a warm attention from both scientific and industrial communities to discover the bounds of its real potential from different perspectives. Many studies have focused on modeling and designing caching strategies and data-transfer performance such as in [5]-[10]. In that direction, it is now clear that the cache size may have a major impact on the overall performance of a CCN even if finding an optimal caching strategy is still an open problem to address. With respect to congestion control issues, instead, recent studies shown as the classic additive increase multiplicative decrease algorithm, at the foundation of TCP, could be inherited by CCN, provided that some countermeasures are employed to limit unfairness issues that could arise among contents with different popularities [11], [12]. Another very relevant topic in CCN research covers routing operations, which are essential to properly drive the dissemination of receiver generated *Interest* packets. To this end, the adoption of Bloom filters appears a promising solution [13], [14] that merits further investigations. Starting from this premise, it is evident that all facets of CCN are going to be afforded in an ebullient panorama of activities that cover both the underlying mechanisms within the protocol architecture and the design of content oriented applications and services. With reference to application-related features, it is worth to notice that the demand for TV services is growing at an exponential rate over the time [2], thus requiring a very careful analysis of their performance in CCN architectures. A preliminary study presented in [15] addresses time shifted applications only, whereas live streaming operations are currently under investigation as testified in the interesting contribution [16]. To complement the research efforts of the community in a so relevant domain, the present manuscript is intended to design a complete CCN-TV system encompassing all the main facets of typical live streaming video services. The proposed CCN-TV has been tested through a solid simulation campaign based on real topologies. To this end, the *ccnSim* simulator [17] has been properly tailored to our purposes by adding window based flow control, handling of playout delay and real-time data, advanced logging functions, links with bandwidth constraints, and data session bootstrapping. Simulation results shown that in-network caching seems to play a minor role in live streaming video services, mainly because cached data looses its utility after the deadline is expired. On the other hand, the way CCN handles client requests for TV contents helps improving the performance of the system with respect to a plain IP infrastructure.

The rest of the paper is organized as follows: Sec. II illustrates the core concepts of CCN. Sec. III describes the CCN-TV system we propose in this paper, which is then evaluated in Sec. IV. Finally, Sec. V summarizes the main findings of this contribution and draws future research.

II. Basic background on CCN

Internet usage has undergone a radical change during the last ten years: content-sharing applications are now dominant whereas the IP architecture still provides a connection-less service among remote hosts [3]. Users ask for contents, looking for *what* they intend to retrieve from the Internet while the language spoken by the underlying IP infrastructure provides answers on *where* a packet should be sent. This mismatch is actually overcome by a number of workarounds at different levels of the protocol stack, which, indeed, limit the overall efficiency of the Internet.

The so-called *Future Internet* represents a family of possible solutions to the aforementioned issues, embracing novel communication models that can better accommodate and fulfill users' requirements related to efficiency, security, support to mobility, and integrated media experience [3].

At the present stage, many valid proposals for the Future Internet exist, such as the Publish Subscribe Internet Routing Paradigm, the 4WARD NetInf project, and the Cache-and-Forward Network Architecture, the Data-Oriented Network Architecture and the CCN approach [4], [18], having different levels of compatibility with the IP paradigm.

Among them, the CCN vision looks promising since, besides being "data-centric", it can be gracefully integrated with today's IP-based Internet. In a CCN, all content is unambiguously identified by a hierarchical name, allowing users to retrieve information without being aware about the physical location of servers (e.g. IP address). Also, communication is receiver-driven and based on content chunk exchange, name-based routing, and self-certifying packets [4].

Nevertheless, the real performance bounds of a CCN and the actual benefits it can bring to the Internet are still not entirely known, mainly because there are many open issues that surround the CCN architecture, such as those related to: (i) routing, (ii) congestion control, (iii) strategy layer design, (iv) name space definition, (v) semantic layer, (vi) accurate models, and (v) fairness among heterogeneous applications and contents having different popularities.

As specified before, CCN communications are driven by the consumer of data and only two types of messages are exchanged (namely *Interest* and *Data*). A user may ask for a content by issuing an *Interest*, which is routed within the CCN towards the nodes in posses of the required information, thus triggering them to reply with *Data* packets.

The routing operations are performed by the strategy layer only for *Interest* packets. *Data* messages, instead, just follow the reverse path to the requesting user, allowing every intermediate node to cache the forwarded content.

CCN adopts a hierarchical structure for *names*, which leads to a *name tree*. In particular, it is formed by several components, each one made by a number of arbitrary octets (optionally encrypted), so that every *name prefix* identifies a sub-tree in the name space. An *Interest* can specify the full name of the content or its prefix, thus accessing to the entire collection of elements under that prefix.

Finally, since contents are exchanged based on their names, multiple nodes interested in a particular data can share it using multicast suppression techniques over a broadcast medium. Analyzing a CCN node, it is possible to identify three main structures [4].

- the *Content Store* (CS): a cache memory that can implement different replacement policies as Least Recently Used (LRU) and Least Frequently Used (LFU);
- the *Forwarding Information Base* (FIB): is similar to an IP FIB except for the possibility to have a list of *faces*[1] for each Content Name entry, thus allowing *Interest* packets to be forwarded towards many potential sources of the required Data;
- the *Pending Interest Table* (PIT): is a table used to keep track of the *Interest* packets that have been forwarded upstream towards content sources, combining them with the respective arrival faces, thus allowing the properly delivery of backward Data packets sent in response to *Interests*.

When an *Interest* packet arrives to a CCN node, the CS is searched to discover whether a data item is already available as an answer to be sent immediately back to the requesting user. Otherwise, the PIT is consulted to find out if others *Interest* packets, requiring the same content, have been already forwarded towards potential sources of the required data. In this case, the *Interest*'s arrival face is added to the PIT entry. Otherwise, the FIB is examined to search a matching entry, indicating the list of faces the *Interest* has to be forwarded through. At the end, if there is not any FIB entry, the *Interest* is discarded.

On the other hand, when a Data packet is received, the PIT table comes into play, which, keeping track of all previously forwarded *Interest* packets, allows to establish a backward path to the node that requested the data.

III. CCN-TV architecture

Unlike Video-On-Demand, real-time video distribution has to deal with a specific class of problems to ensure the timely delivery of an ordered stream of chunks. Video chunks have to be received in playing order and within a given time interval (the *playout delay*), before they are actually played, thus "expiring". A chunk not delivered before its expiration will result in degradation of the rendered

[1]In CCN it is used the term "*face*" instead of the "*interface*" because packets are also exchanged between application process, besides being forwarded only over real network interfaces.

video, impacting the end user Quality of Experience; the extent of the video degradation may vary depending on the video codec and the type of the lost frame. To solve these challenges, client nodes implement a receiving buffer queue where the chunks are stored in order, that is emptied while the video is being played. Therefore, any chunk not received before its playing instant becomes useless. To reduce the chance of chunk loss several mechanisms can be put in place to retransmit requests for chunks close to expiration. Furthermore, in modern codecs, such as H.264 [19], there are different types of video frames, encoded using intra-frame or inter-frame techniques, each having a different level of importance. For example, the so called I-frames, derived using intra-frame compression techniques, actually represent a full video image, providing a fundamental reference for subsequent inter-frame encoded images.

With this in mind in CCN-TV we considered a network of nodes requesting different real-time video streams, identified by a *channelID*, served by one or more broadcast server.

Unlike canonical UDP/TCP-based streaming, in CCN-TV each video chunk, identified by a progressive *chunk number*, has to be requested individually, via a dedicated *Interest*.

Although this might look costly at a first sight, CCN's routing mechanisms ensure that *Interests* for the same chunk do not propagate twice along the same routing path (unless under specific conditions, as explained in Sec. III-B), and the caching strategy implemented by every node can make sure that *Interests* for the most popular contents are served before going through the whole routing path. Moreover, the *Interest/data* exchange allows for a natural flow control mechanism, where each node can request for new chunks just when the old ones have arrived.

Herein we thoroughly describe the design rationale and all the details of the CCN-TV system this paper targets. Specifically, in what follows, we present: the bootstrap phase, the flow control strategy, and the management of retransmitted *Interest* packets.

A. Channel bootstrap

One challenge we are faced with in a real-time scenario is to bootstrap the channel to be received. Bootstrapping a channel involves the operations of finding a routing path to the nearest channel provider and locating the first valid chunkID of the video stream. Due to video codec requirements, the video stream can be visualized only once the first I-frame has been received. Therefore, a client has to first gather from the server the first chunk (and the corresponding chunkID) of the last generated I-frame. To do so, it sends an *Interest* packet for the URI: `[domain]/[channelID]`, with the *Status* field set to *BOOTSTRAP* and the *Nonce* field set by the client, as in in Sec. III-D. An *Interest* with Status = BOOTSTRAP would travel unblocked until it reaches the first good stream repository (i.e. a node who can provide a continuous real-time flow of chunks, not just cached ones).

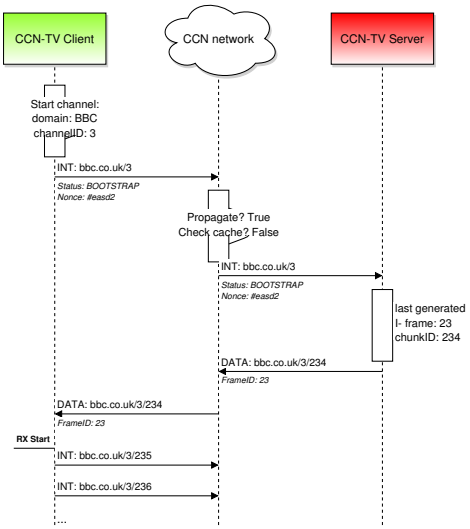


Figure 1. Bootstrap handshake

To this *Interest*, the server responds with a data message in the format `[domain]/[channelID]/[chunkID]`, with *chunkID* being the first chunk of the last generated I-frame, and the corresponding *Frame ID* field value. Upon receipt, the node starts asking for subsequent chunks, using the sliding window mechanism detailed in Sec. III-B. The use of a *nonce* (a uniquely generated identifier) in the *Interest* URI allows the *Interest* to propagate to the server without being blocked along the routing path, as every bootstrap *Interest* for the same channel has a different nonce. It also avoids the retrieval of the data response from the cache of an intermediate node; the risk, in this case, is the retrieval of a bootstrap data message for a given channel from a cache containing an already expired chunk of an I-frame.

B. Flow control

From the moment a node receives the bootstrap data message, it can initiate the sliding window mechanism to request the subsequent chunks in an optimal way. Each node has a windows of size *W* to store W pending chunk. We define *pending chunk* a chunk whose *Interest* has been sent by the node, and the window containing the pending chunks a *Pending Window*. Together with the chunkID, we store in the pending window other information, such as the timestamp of the first request and the timestamp of the last retransmission. Whenever a new data message is received, the algorithm described in Fig. 2 runs over the Pending Window, to perform the following operations:

1) Purge the Pending Window from all the chunks who are expired, i.e., who have already been played, to free new space in the sliding window.
2) Retransmit all chunks that have not been received for a given timeout (onward denoted as *windowTimeout*.
3) Transmit, for each slot that got freed by the received or expired chunks, the *Interest* for a new one.

```
1: procedure SENDINTERESTS(PW, W, WinT, Now, LC)
2:     # Remove all expired Interest
3:     for all CID in PW do
4:         if CID is expired then
5:             remove CID from PW
6:         end if
7:     end for
8:     # Resend stale Interests
9:     for all CID in PW do
10:        if lastTx(CID) < Now − WinT then
11:            resend(Int(CID))
12:            lastTx(CID) ← Now
13:        end if
14:    end for
15:    # Send Interests for new chunk
16:    NNC ← W − size(PW)
17:    for i ← 1, NNC do
18:        send(Int(LC))
19:        lastTx(LC) ← Now
20:        add LC to PW
21:        LC ← LC + 1
22:    end for
23: end procedure
```

Figure 2. Sliding window algorithm

Furthermore, the same operations are performed if a node doesn't receive any data for at least *windowTimeout* seconds, in which case, all the *Interests* for non-expired chunks in the Pending Window are retransmitted, together with new chunks if new slots have been freed due to expired chunks.

Fig. 2 details the implemented algorithm; for the purpose of brevity and readability, the variable names have been contracted: *PW* is the Pending Window, *W* is the aforementioned system parameter, indicating how many *Interests* should a node have ongoing, *WinT* is the window timeout, after which *Interests* in the Pending Window are resent, *Int* is a new *Interest* message, *CID* is a chunkID in the pending window, *lastTx* is the transmission time of the most recent *Interest* for a given chunkID, *LC* is the chunkID of the most recent requested chunk and *NNC* is the number of new chunks to request, after the pending window has been purged.

To provide a further insight, we reported in Fig. 3 an example of the conceived sliding window algorithm, in which we have set the value of W to be equal to 3.

C. Interest routing

As described in Sec. II, CCN nodes along the routing path of an *Interest* will stop the propagation of said *Interest*, if they have previously routed another *Interest* for the same resource, and the correspondent data has note been sent back yet; instead, they will simply update their Pending Interest Table adding the face from where this newcomer *Interest* was originated, so to reroute the data back recursively along the path the *Interest* has gone through.

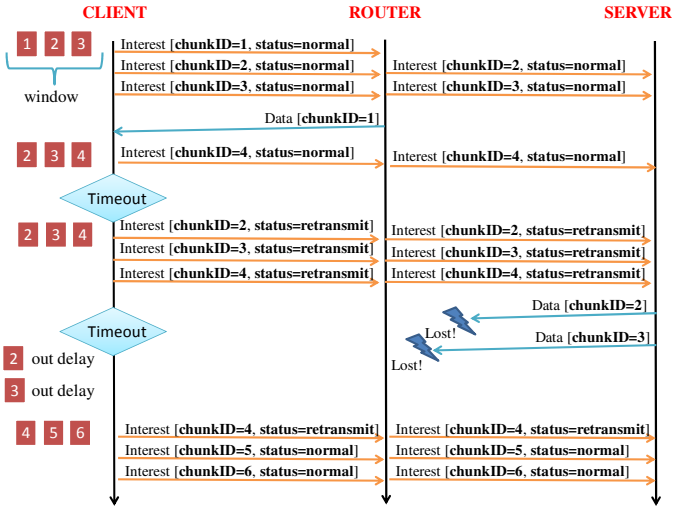


Figure 3. Sliding window example

This mechanism ensures flow control and limits the propagation of duplicate *Interests*, in case several nodes in the same network are watching the same channel.

However, to make the *Interest* retransmission mechanism effective, a retransmitted *Interest* needs to propagate all the way up to the server, or to the first node with the desired chunk in cache. Therefore, retransmitted *Interests* carry the *Status* field set to *Retransmission* to mark if the *Interest* is a retransmission or not, and each node along the routing path propagates the *Interests* marked as retransmitted, thus skipping the usual CCN mechanism, unless the correspondent chunk is found in the cache.

Table I
MESSAGES USED IN CCN-TV

Packet type	Field	Content
Chunk Interest	*Content Name*	[*domain*]/[*channelID*]/[*chunkID*].
	nonce	Used only for the bootstrap phase.
	Publisher Filter	Not used.
	Status	*Bootstrap*, *Normal*, *Retransmission*.
Chunk Data	*Content Name*	[*domain*]/[*channelID*]/[*chunkID*].
	Publisher ID	Optional.
	Signature	Optional (for increased content authentication).
	Stale Time	Set to the *frame time* of the frame the chunk belongs to.
	Frame ID	ID of the frame the chunk belongs to.
	Data	The request video chunk binary data.

D. CCN-TV messages

As detailed above, additional functionalities required by the system for real-time video streaming are implemented on top of existing CCN data and *Interest* messages via new fields carrying the required additional information.

However, should the situation require the system to conform to classical CCN messages, all additional fields can be easily replaced by additional fields in the content name.

Tab. I shows how we made use of the classical CCN message fields, together with the new fields and their use.

In particular, CCN-TV *Interests* carry an additional *Status* fields marking if the *Interest* is a *bootstrap Interest* (Section III-A), a normal one or a *retransmission* (Section III-C).

Concerning CCN data message, we extended the messages with an additional field, i.e., *Frame ID*, containing the ID of the frame to which the embedded chunk belongs to.

IV. Simulation results

In this section, we will evaluate performances of the proposed CCN-TV architecture. To this end, we implemented it within *ccnSim*, i.e., an open source and scalable chunk-level simulator of CCN [17] built on top of the Omnet++ framework [20], dedicated to the evaluation of Video On Demand systems on top of CCN. By itself, *ccnSim* models a complete video distribution systems, with a high degree of fidelity concerning catalogs, requests and repositories distribution, and network topologies. Since, however, it did not support the real-time constraints required by our evaluations, it has been modified and improved in the following ways:

- we added support for links with bounded capacity and packets with a well defined size, which was missing in *ccnSim*, to be able and estimate the CCN behavior under some bandwidth constraints;
- due to the datarate channels, we implemented a transmission queue for each face of each node, in order to properly manage the packet transmission;
- we added the support for synthetic video traces, so to be able to transmit and receive chunk of real videos, and consequently being able to reconstruct the received video and evaluate its PSNR;
- due to possible expiration of *Interests*, we implemented a cleanup mechanism for each node's PIT, to avoid having in long term stale entries due to expired chunks;
- we improved and enriched the logging system, so to be able to record each node's received chunks and reconstruct the received video;
- we added more controls server-side, to send a data only for those chunks who have already been generated.

Furthermore, the following mechanisms, beyond the provided ones, have been implemented in the simulator:

- the sliding window mechanism described above, and all the related data structures;
- the *Interest* forceful propagation in case of retransmission;
- constant data reception, until a channel is changed.

The extended *ccnSim* simualtor is available at [21].

The aim of our study is to evaluate how the behavior of the CCN-TV system is influenced by (i) the amount of the network bandwidth dedicated to real-time streaming services, (ii) the *windowTimeout* adopted by the sliding window mechanism, (iii) the playout delay, and (iv) the cache decision policy.

We focus the attention on the GEANT network, which interconnects the European research and education institutions and it is composed by 22 routers [22]. Every node of the network is considered to be a direct CCN node, i.e. no TCP or UDP encapsulation is implemented. We assume the presence of only one small video streaming provider that offers 5 parallel real-time transmissions to remote clients. It is connected to one of the nodes forming the GEANT topology. In every simulation round, each video content is mapped to a video stream compressed using H.264 [19] at a average coding rate randomly chosen in the range [250, 2000] kbps. Clients, i.e., CCN nodes that download video contents from the server, are connected to remaining nodes (1 client per node). In order to catch the behavior of people watching TV, we modeled two groups of users: *faithful* and *zapping*. *Faithful* users are attached to one video channel for the whole simulation. *Zapping* users, instead, change frequently the channel among those offered by the server according to a Poisson process with parameter $\lambda = 0.0666$. Further, the channel selection process has been modeled considering that contents popularities follow a Zipf distribution. According to [23], the most of works presented in literature set the parameter α of the Zipf distribution in the range [0.6, 2.5]. In line with these common settings, we set α = 1. Once a client decides to watch a specific channel, it performs the bootstrap process described in the previous section and then starts sending *Interest* packets following the designed sliding window mechanism.

In our tests, we adopted the optimal routing strategy, already available within the *ccnSim* framework. According to it, *Interest* packets are routed towards the video server along the shortest path. On the other hand, three caching strategies have been considered in our study: *no-cache*, *LRU*, and *FIFO* [23]. When well known *LRU* or *FIFO* policies are adopted, we set the size of the cache to 100 chunks. The *no-cache* policy is intended to evaluate the performance of the CCN without using any caching mechanism.

The window size W has been set to 10, ensuring that faces of the server are almost fully loaded in all considered scenarios. Also, the transmission queue length associated to each face, Q, has been set, in order to be larger than

$$Q = 2 * L_c * P_D \tag{1}$$

where L_c and P_D represent link capacity and maximum propagation delay in the considered network topology. All simulation parameters have been summarized in Tab. II.

A. Interest generation process

As a first step, we investigate the impact that the sliding window mechanism has on the amount of sent *Interest*

Table II
SUMMARY OF SIMULATION PARAMETERS

Parameter	Value
Topology	GEANT with 22 routers
Link capacity	40 Mbps and 100 Mbps
Number of real-time service provides	1
Number of clients	21
Catalog size	5 files
Chunk size	10Kbytes
Video average bit rate	250kbps, 600kbps, 1000kbps and 2000kbps
W (window size)	10
Playout delay	10s and 15s
Window timeout	1s, 3s, and 5s
Caching strategy	No cache, LRU, and FIFO
Cache size	100 chunks
Client zapping behavior	50% fixed, 50% changing on average every 15s
Simulation time	60s
Number of seeds	5

packets, which is shown in Fig. 4. From these plots it is evident that the highest *windowTimeout*, the lowest the total number of *Interest* messages sent by end users. When the *windowTimeout* increases, the probability that a given client does not receive any chunks within such a time interval decreases and, as a consequence, also the number of retransmitted *Interest* packets decreases as well. As a further confirm of this result, Fig. 5 shows that the percentage of duplicated *Interest* packets increases when the *windowTimeout* decreases due to a high number of chunks that are unable to reach the client within the expected timeliness.

As expected, the *playout delay* has a minor impact on the number of generated *Interest* messages, which, as is known from the theory on sliding window mechanisms [24], can be only influenced by window size (W) and *windowTimeout*.

Also, caching operations do not have any significant impact on the number of generated *Interest* messages. The main reason being that chucks stored in cache memories lose their effectiveness after their deadline is expired.

On the other hand, the link capacity greatly influence the *Interest* transmission rate. From Fig. 4 emerges, in fact, that the number of *Interest* lowers when the capacity of links decreases. This is because a limited bandwidth reduces the quota of received chunks, thus preventing a rapid advancement of the sliding window. In other terms, this result proves, once again, the effectiveness of the sliding window mechanism in CCN.

B. QoS and QoE

The first important parameter that describes how CCN-TV settings affect the quality of service offered to end users is the chunk loss ratio, which represents the percentage of chunks that have not been received in time (i.e., before the expiration of the *playout delay*) by clients.

From Fig. 6, showing the chunk loss ratio measured in all considered network scenarios, we note that *playout delay* plays a fundamental role. When the *playout delay* increases, in fact, the client could receive a *Data* packet within a longer time interval, thus reducing the amount of chunks discarded because out of delay. On the other hand, a slight increment of the chunk loss ratio can be registered by increasing the *windowTimeout*. If the client retransmits an *Interest* packet after long time, there is the risk that the *Data* packet will be reached by the destination after the expiration of the *playout delay*. In addition, we note that a reduction of the link capacities leads to a higher number of lost chunks, due to increased latencies induced by network congestion.

It is very important to remark that the presence of the cache can guarantee only a small reduction of the chunk loss ratio. With our study, we found that, in the presence of real-time flows, the cache does not represent an important CCN feature. On the other hand, we noticed that the PIT plays a more relevant role. In fact, in presence of live video streaming services, clients that are connected to a channel request the same chunks simultaneously. In this case, a CCN router has to handle multiple *Interest* messages that, even though sent by different users, are related to the same content. According to the CCN paradigm, such a node will store all of these requests into the PIT, waiting for the corresponding *Data* packet. As soon as the packet is received, the router will forward it to all users that have requested the chunk in the past. According to these considerations, the use of the cache will not produce a relevant gain of network performances. Indeed, the PIT helps reducing the burden at the server side by avoiding that many *Interest* packets for the same chunk are routed to the server.

To conclude our study, we have computed Peak Signal to Noise Ratio (PSNR), which is nowadays one of the most diffused metrics for evaluating user satisfaction, together with interactivity level, in real time video applications [25]. Results shown in Fig. 7 are in line with those reported for chunk loss ratio (the PNSR is higher in the same case in which the chunk loss ratio is lower). Again, link capacity greatly influences the quality of the TV service provided to users. According to [26], the obtained PSNR values can be translated to a Mean Opinion Score (MOS) not less than 4, corresponding to satisfactory quality for almost all users.

V. CONCLUSIONS AND FUTURE RESEARCH

In this work, the effectiveness of TV services in a CCN has been investigated. To this end, the *ccnSim* simulator has been modified to add several relevant features such as window-based flow control, handling of playout delay and real-time data, advanced logging mechanisms, and data session bootstrapping. Preliminary results reported herein clearly show that the most relevant CCN feature to TV services is the management of *Interest* packets through the PIT data structure. In fact, such a mechanism limits the number of requests for the same chunk at the server side for multiple clients watching the same TV channel, thus decreasing

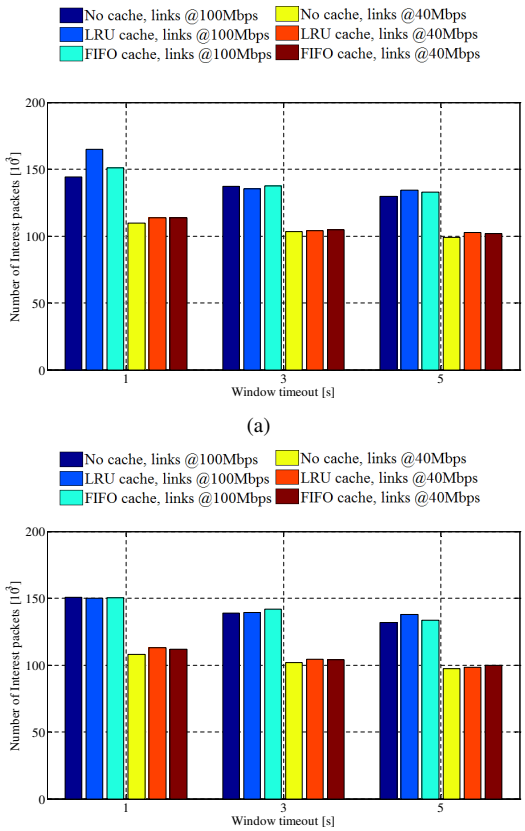


Figure 4. Total number of *Interest* packets sent by clients with playout delay of (a) 10s and (b) 15s.

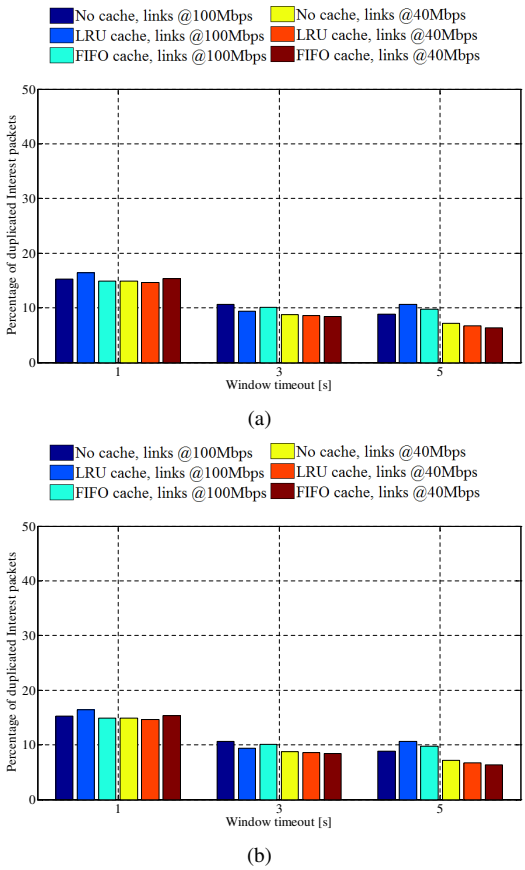


Figure 5. Percentage of duplicated *Interest* packets sent by clients with playout delay of (a) 10s and (b) 15s.

the link and the computational load at the server. Further research will explore: wider scenarios with many users and available channels, advanced optimization techniques for TV services in CCN (including routing and congestion control), and the adoption of scalable video coding. Moreover, a more deeply investigation on the relevance of the cache and the PIT in live TV services will be also conducted.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] "Cisco visual networking index: Forecast and methodology, 2010-2015," White Paper, Cisco, Jun. 2011.

[3] B. Ahlgren, P. A. Aranda, P. Chemouil, S. Oueslati, L. M. Correia, H. Karl, M. Sollner, and A. Welin, "Content, connectivity, and cloud: ingredients for the network of the future," *IEEE Commun. Mag.*, vol. 49, no. 7, Jul. 2011.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM CoNEXT '09*, 2009.

[5] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Int. Teletraffic Congress, (ITC)*, 2011.

[6] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *ACM SIGCOMM workshop on Information-centric networking (ICN '11)*, 2011.

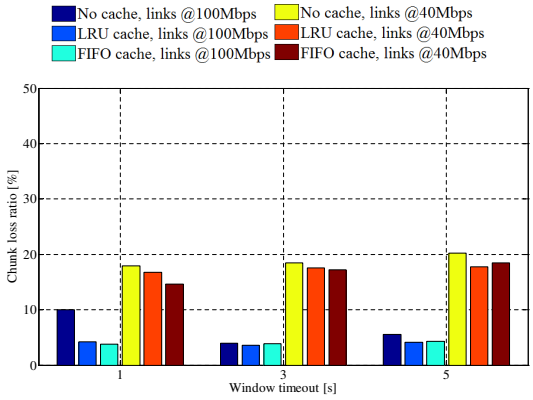
[7] M. Varvello, I. Rimac, U. Lee, L. Greenwald, and V. Hilt, "On the design of content-centric manets," in *Int. Conf. on Wireless On-Demand Network Systems and Services, (WONS)*, 2011.

[8] G. Carofiglio, V. Gehlen, and D. Perino, "Experimental evaluation of memory management in content-centric networking," in *IEEE ICC*, 2011.

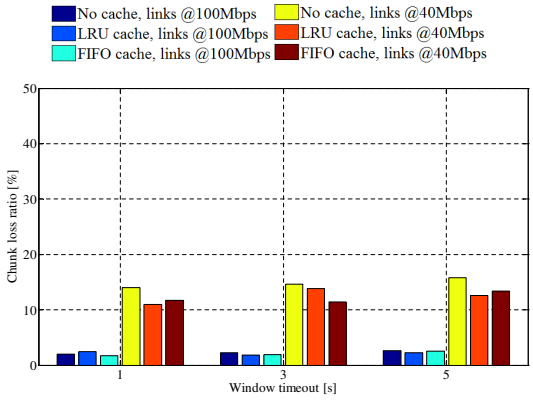
[9] M. Tortelli, I. Cianci, L. A. Grieco, G. Boggia, and P. Camarda, "A fairness analysis of content centric networks," in *Proc. of Int. Conf. on Network of the Future, NOF*, Paris, France, 2011.

[10] D. Rossi and G. Rossini, "On sizing CCN content stores by exploiting topological information," in *IEEE INFOCOM, NOMEN Worshop*, 2012.

[11] L. A. Grieco, D. Saucez, and C. Barakat, "AIMD and CCN: past and novel acronyms working together in the Future Internet," in *Capacity Sharing Workshop 2012 (CSWS'12) co-located with ACM SIGCOMM CoNEXT 2012.*, Dec. 2012.

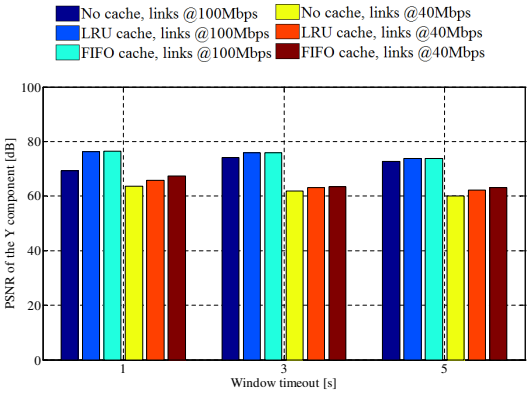


(a)

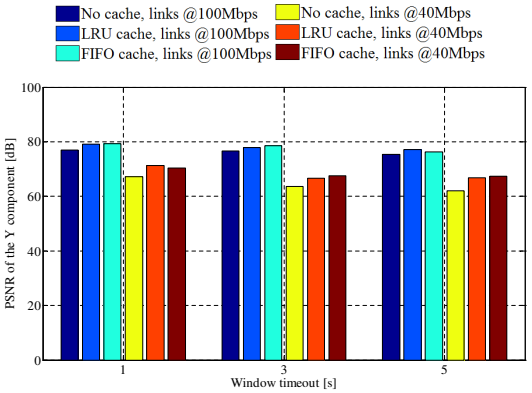


(b)

Figure 6. Chunk loss ratio with playout delay of (a) 10s and (b) 15s.



(a)



(b)

Figure 7. PSNR of the Y components of received videos with playout delay of (a) 10s and (b) 15s.

[12] G. Carofiglio, M. Gallo, and L. Muscariello, "Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks", *ACM SIGCOMM, ICN12 workshop*, 2012.

[13] M. Tortelli, L. A. Grieco, and G. Boggia, "CCN forwarding engine based on bloom filters," in *Proc. of ACM Int. Conf. on Future Internet Technologies, CFI*, Seoul, Korea, Sep. 2012.

[14] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "Dipit: A distributed bloom-filter based pit table for CCN nodes," in *21st International Conference on Computer Communications and Networks (ICCCN)*, 2012, pp. 1–7.

[15] Z. Li and G. Simon, "Time-shifted TV in content centric networks:the case for cooperative in-network caching," in *Proc. of IEEE ICC*, Jun. 2011.

[16] H. Xu, Z. Chen, R. Chen, and J. Cao, "Live streaming with content centric networking," in *Proc. 3rd Int. Conf. on Networking and Distributed Computing, Hangzhou, China*, 2012.

[17] G. Rossini and D. Rossi, "Large scale simulation of ccn networks," in *In Algotel* 2012.

[18] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, 50(7), pp. 26–36, 2012.

[19] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. on Circuits and Systems for Video Technology*, 13(7), pp. 560–576, Jul. 2003.

[20] Omnet++ home page. [Online]. Available at: http://www.omnetpp.org/

[21] Ccn-tv webpage. [Online]. Available at: http://telematics.poliba.it/ccn-tv/

[22] "Geant project website," [OnLine] Available at: http://www.geant.net/.

[23] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," in *Technical report, Telecom ParisTech*, 2011.

[24] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th edition. Addison-Wesley Publishing Company, 2012.

[25] G. Piro, L. Grieco, G. Boggia, R. Fortuna, and P. Camarda, "Two-level Downlink Scheduling for Real-Time Multimedia Services in LTE Networks," in *IEEE Trans. Multimedia*, vol. 13, no. 5, Oct. 2011, pp. 1052–1065.

[26] J. Ohm, *Multimedia Communication Technology*. Springer, USA, 2004.