Performance Evaluation of Next Generation Content Delivery Proposals

Philipp M. Eittenberger, Klaus M. Schneider, Udo R. Krieger
Faculty of Information Systems and Applied Computer Science
Otto-Friedrich-University,
D-96047 Bamberg, Germany
Email: udo.krieger@uni-bamberg.de

***Abstract*—This study investigates two innovative content delivery proposals of next generation Internet, namely Libswift and CCN. We present results of a measurement campaign performed on a realistic network testbed to evaluate and compare their performance in relation to reference measurements obtained by the traditional client/server approach. The results underline the expected superior performance of these new architectures for many use cases. But we were also able to identify several severe weaknesses of their particular implementations. In particular, we could observe a heavy performance degradation under certain conditions, i.e., mainly for connections characterized by high delay and high packet loss. Furthermore, we investigate adaptations to increase their performance and we evaluate a prefetching mechanism for CCNx to improve its performance under these adverse conditions.**

***Index Terms*—Swift, content centric networking, performance evaluation, prefetching**

I. Introduction

Nowadays, the majority of Internet traffic is caused by content distribution, i.e., image, audio and video data [6]. A new network paradigm named *Content Centric Networking* (CCN) or *Information-Centric Networking* (ICN)[1] [7] has been proposed which shifts the focus to content instead of hosts. The main point of the paradigm states that users are more concerned about *what* content they want to retrieve, rather than *where* this content is located [13]. ICN promotes universal caching at each network node, which exploits the fact that the price of storage capacity has decreased faster than the price of network bandwidth and is continuing to fall in the future. This caching mechanism proves to be highly beneficial for content that is created once and accessed many times. The content-centric paradigm is not as new as recent research would suggest. On the contrary, the concept of "what instead of where" has been around long before the term *Content Centric Networking* emerged.

We may take as one example *content distribution networks* (CDN) which are usually deployed on Internet-scale. In a CDN multiple, widely distributed servers provide content under a single namespace. CDNs work well due to the fact that in most usage models clients only have read access to the data, which are written by the content providers. Therefore, depending on the frequency of changing the content, a read access occurs far more often than a write access. Thereby, a traffic reduction can be achieved by replicating the data among distributed data storages and linking clients automatically to the nearest caching instance. Then the data has to cross longer, more expensive links only in case of a write access. It makes this concept particularly useful if the content is rarely written into the storage, but often read. Observe that content cannot be cached constantly due to the limited cache size of the routers. Therefore, cache management strategies come into play. Compared to traditional distribution schemes with one server or a server farm, CDNs provide better load balancing capabilities and fault-tolerance. Downsides of CDNs are given by high installation and maintenance costs. Usually, they are application-specific and require planning in advance. The deployment of CDNs is only affordable for larger companies. End users can use their services to distribute content, but they have to accept restrictions from the providers, like additional advertisement or content censorship. In contrast, the *peer-to-peer* (P2P) approach gives end users the power to share content with millions of others without requiring the expensive infrastructure of CDNs. This scalability is achieved by the fact that each peer contributes to the network by providing resources, e.g., by uploading data. The performance of P2P systems relies on the download to upload ratio of the peers as well as on the ratio between seeders (peers contributing to the network) and leechers (peers contributing very little or nothing). Although this concept provides benefits for content providers, it can cause problems for ISPs, because P2P applications are normally not network-aware. This means that such systems do generally not regard the underlying network infrastructure, which can lead to an inefficient usage of the core network.

In the following we investigate the performance of two new upcoming proposals for next generation Internet: *Swift* [1] and *CCN*. We present the results of our measurement campaign performed on a small scale network testbed. The measurement campaign has been deliberately conducted on a rather small scale that already allows the detailed study of the implementation performance. Transmission artifacts of the particular application can be investigated without the need for large scale experiments. The obtained results indicate a performance degradation under certain conditions, e.g., for

[1] In this article we will use the acronym ICN when the concept of content centric networking is meant, while CCN will be used for the concrete protocol published by Van Jacobson *et al.* [13].

connections that are characterized by high delay and high packet loss, a typical characterization of a cellular network. Thereby, we take interest in the point of view of a single user in a wireless scenario with its obviously encountered packet loss. It means that this is not a study on the general applicability of these approaches to the Internet, as there are already other studies investigating this item. But it is more or less a reality check of these new proposals and the current state of their implementations from the point of view of a single user.

II. Architectures Proposed For Next Generation Internet

Current protocols that follow a content centric paradigm, like BitTorrent, Gnutella and current CDN designs, are usually located at the application layer of the TCP/IP stack, which allows a rapid deployment. In the following we will look at two protocols at lower layers, which may come to a wider deployment in the future: Swift and CCN. The selection of both projects is motivated as follows. Despite the fact that there already exist many ICN projects, CCN has been chosen since it has already gained a large momentum. Regarding P2P systems, there is a plethora of them. Yet, Swift is, to the best of our knowledge, the only one trying to establish the P2P paradigm at the transport layer. Furthermore, Swift is used in the reference implementation of the IETF draft on the *Peer-to-Peer Streaming Peer Protocol* (PPSPP) [5]. It has already shown a comparable performance to highly optimized, "old-fashioned" BitTorrent clients [11].

A. Swift

Swift [1], also called Libswift[2], is a transport layer protocol that is used to distribute content among a swarm of peers. It provides functionality similar to BitTorrent, while its implementation is more lightweight with a substantially fewer number of lines of code. Swift supports the communication protocol *peer exchange* (PeX), which allows peers to directly exchange lists of active peers with each other. This eliminates the need for a central tracker server. Swift was built around the abstraction of atomic datagrams in contrast to flow centric schemes like TCP. Each data packet is stored permanently and it is possibly useful for other peers independent of its received order. Swift uses Merkle hash trees for datagram-level integrity checks. They help to keep the chunk size small, while also reducing the data overhead necessary for the hashes [4]. The implementation Libswift has been analyzed recently and its performance has been found comparable to current BitTorrent clients, even though, the technology is relatively new and not as optimized as current BitTorrent implementations (see [11]). Petrocco *et al.* [11] also stress Libswift's low startup time and promote it as alternative to current P2P content dissemination systems.

[2]Talking about the protocol, we use the term *Swift* whereas *Libswift* is used when we consider the implementation.

B. CCN

Content Centric Networking (CCN) or Networking Named Content (NNC) [13] is a network layer protocol. It tries to solve the problems of CDNs and P2P networks by proposing a *clean-slate* architecture based on the content-centric paradigm. In contrast to CDNs or web caches, it provides content caching at all network elements, supporting every application layer protocol and content of all users [7]. CCN packets contain content names instead of source and destination addresses. There are two CCN packet types: *interest* packets to retrieve data from the next CCN router and *data* packets that contain the actual data. The data packets hold a signature that is created by a hashing algorithm to provide data integrity and contain the public key of the publisher, which can be used to check the authenticity of the data. CCN shifts host-based security to content-based security, such that data can be received securely from caches of CCN routers that might be compromised. In CCN data are named hierarchically similar to HTTP URLs. Besides the globally routable and organizational name of the content, the name of the data contains information on its version and segmentation. It can be used by clients that do not need the latest version of a packet. They can retrieve it from a nearby router without contacting the original source for more recent versions. The infrastructure of CCN routers differs from the one of current routers. It is necessary because CCN implements routing based on hierarchical names instead of IP addresses. The name *face* is used to accent that not only physical network interfaces are used for communication, but also virtual local interfaces or *faces* between processes. The *content store* contains all cached data packets of the CCN router. The application of caching techniques is necessary to decide which packets will get discarded once the cache has reached its limit. The *forwarding information base* (FIB) uses a longest-prefix matching. It is similar to current routing tables, but can hold more than one destination face per content prefix. The *pending interest table* saves interests and adds requesting faces to its list, if a packet is requested by multiple neighbor routers before being delivered. CCN can be deployed incrementally on top of the IP network, but it only reaches its full potential when it is implemented on a high percentage of core routers. Recent research has already proposed design principles to achieve high performance with Internet-scale CCN forwarding [15].

III. Performance Evaluation

Regarding the transport network many scientific studies have already underlined the performance benefits of ICN approaches. In our evaluation we are therefore not interested to investigate the presented proposals at a large scale, but we want to evaluate the performance from the point of view of a single user. In particular, we look at typical users that are nowadays connected by a wireless network link. It is characterized by relatively high packet loss and high delay. Therefore, we use Swift and CCN in mixed scenarios with LAN and WLAN nodes and compare their file transfer performance to a classical client/server transmission. The measurements shall

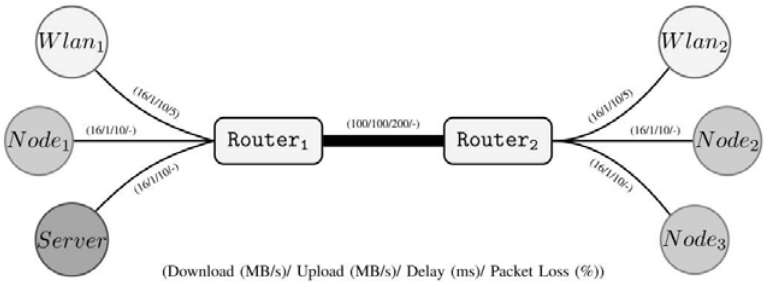

Figure 1. Topology of the testbed

shed light on the performance benefits of these technologies in such a scenario. They have to outweigh their deployment and maintenance costs to achieve a wider deployment. More dramatically speaking, if the investigated proposals are not able to prove their superiority to other already established approaches, they will not be deployed. Regarding the experiments we have used version 0.6.1 of the CCN prototype called CCNx [2] and the latest release of Libswift [1]. All experiments were conducted with the C implementation of CCNx (a Java version is also available). Unless otherwise stated, at least 6 test runs have been conducted for every test set-up.

A. Experimental Set-Up

The experiments were conducted on *Emulab* [14], a network testbed facility available for research purposes. Emulab enables its users to build real network topologies and to specify link characteristics to conduct realistic experiments. For each node of a topology one can decide which operating system is used. Hence, we have created our own operating system image based on Ubuntu 10.04 LTS. This image comprises the necessary software for CCNx, Libswift and packet capture. In every experiment we have synchronized the start time by the programs *emulab-sync* or *clusterssh* to minimize uncontrolled deviations.

Regarding the network model of our performance evaluation, we have chosen a simple dumbbell topology (see Figure 1). The link in the middle of the network, which symbolizes the core network of the Internet, has a high bandwidth and high latency to emulate a high physical distance and multiple hops between both routers. Furthermore, we have two access networks connected by the "core" link. One access network comprises the source of the video file, the so called *Server*, and two "clients". The other access network only consists of clients. The clients are either connected by a LAN (simply called *Nodes*) or connected by a WLAN. In a more realistic scenario the core network would consist of much more routers, but we think that for our purpose (the point of view of a single user) such a simple topology is already sufficient to evaluate the key metrics of interest.

We have defined the following three link metrics for the experiments: *link bandwidth*, *link latency* and *link packet loss*. The bandwidth has been modeled according to a regular ADSL connection with 16 Mbps downstream and 1 Mbps upstream. The packet loss varies on both types of clients to emulate a LAN (no packet loss) and a WLAN (higher probability of packet loss[3]) links. During each measurement we have transferred a video file of 7.1 MB, which is regarded as the typical average data volume of a YouTube video flow [3]. For each test run we have measured the download completion time and the upload/download volume regarding the following three technologies:

- **Rsync**: Rsync is used in a very popular current CDN implementation. YouTube uses it to synchronize the data of its video caches [8]. We use Rsync as representative of the old client/server communication paradigm and do not make use of its more advanced features, like incremental backup or replication. It is just applied as reference measurement to obtain an upper bound that the completion time of the other implementations should definitely not exceed. In the experiments the completion time has been analyzed in a simultaneous manner, i.e., each node started the download simultaneously and in a choreographed manner, where each download was started one after another.
- **Libswift**: As Libswift is a P2P protocol, the upload and download volume of each host is also of interest as opposed to, e.g., client/server approaches like Rsync. In Libswift a file request requires the file hash and the hostname of one peer in the swarm, which must not necessarily be the tracker server. The peers were configured to stay in the swarm and to provide bandwidth after they finished their downloads.
- **CCNx**: We have measured the completion time for normal, clean state transmissions (i.e., without cached content) and once the content was already cached at the routers Router$_1$ and Router$_2$. To request a file in CCNx, only the data name, in this case */video.mkv*, is needed. The local CCNx daemon creates an interest packet and sends it to its router, which will either forward the interest or, if possible, reply with a data packet. An additional measurement has been conducted on CCNx with the host

[3]Both WLAN nodes experience a mean packet of loss of 5 %. This is quite high for WLAN, but seems reasonable for a cellular network (see [9]).

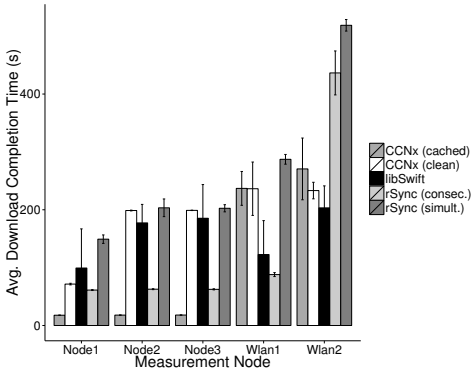

Figure 2. Average download completion time of all tested technologies on all nodes with confidence intervals at a confidence level of 95%

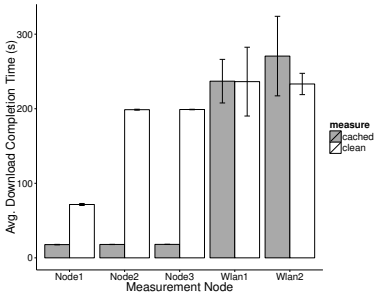

Figure 3. CCNx download completion time with confidence intervals at a confidence level of 95%

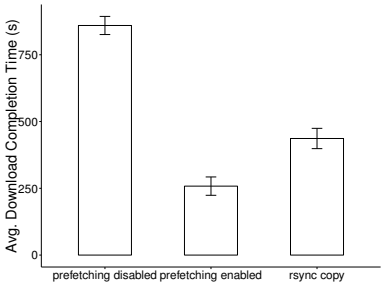

Figure 4. CCNx prefetching mechanism: Download completion time from Server to $Wlan_2$ with and without prefetching in comparison to Rsync's consecutive file transfer with confidence intervals at a confidence level of 95%

$Wlan_2$ as single client. It encounters a high latency in the core network and a high packet loss on the last hop. In the first step, we have checked the regular completion time of the host in a clean state scenario, where the caches were cleaned after each run. In the second step, the effects of our proposed prefetching mechanism were recorded. The results of this measurement are described in Section IV.

B. Download Performance

a) Rsync: Regarding the Rsync tests we have copied the video file from the host Server to all clients $Node_1$, $Node_2$, $Node_3$, $Wlan_1$ and $Wlan_2$. In the first test the file transfer started at the same time on all nodes. In the second "choreographed" test setup each client transferred the file synchronized one after another. Both tests were run six times and the mean download completion times are shown in Figure 2 with confidence intervals at a confidence level of 95 %. While $Node_1$, $Node_2$ and $Node_3$ perform equally in the consecutive test, $Node_1$ is significantly faster in the simultaneous test. As the only difference between those nodes is given by the latency, this can be explained by TCP's congestion control mechanism. It allows the node observing less delay to use a higher share of the bandwidth. The WLAN nodes, which encounter a high amount of packet loss, need much longer to download the file. The download completion time of $Wlan_2$, which has a high latency and high packet loss is particularly noteworthy. In both test scenarios it takes significantly longer to complete the download than on any other node. Our tests have shown that a regular file copy deals well with high latency and high packet loss on its own, but very poorly when high latency and high packet loss are combined.

b) Libswift: In the Libswift test scenario we have measured the download completion time and additionally the total amount of downloaded and uploaded data for each host. The download volume of all peers was found as expected. It comprises the video content of 7.1 MB and a small additional amount of packet header overhead. The upload volume differed much stronger among the peers. The server has the highest upload volume and the WLAN nodes provide nearly no upload capacity to the network. This can be explained by the fact that the WLAN nodes were the slowest to download the data and after completion all other nodes were already done. Therefore, they could not provide any data chunks to the overlay network. An interesting anomaly is given by the difference in the upload volume between $Node_2$ and $Node_3$, which is not significant, but distinguishable. This difference is mainly due to the fact that it is not possible in Emulab to synchronize the start of the experiment precisely on all hosts. When the start time differs even only a fraction of a second, for instance, $Node_2$ starts shortly before $Node_3$, the data will be transferred over the expensive link to $Node_2$ and then partly from $Node_2$ to $Node_3$. Hence, $Node_2$ would have a higher upload volume in this scenario. As shown in Figure 2, the overall performance of Libswift was relatively stable across all nodes and has shown the best performance on the nodes with packet loss. Libswift behaves similar to Rsync for nodes with high latency, but much better for nodes with high packet loss, especially for the host $Wlan_2$, which also encounters a high delay. This is a consequence of its substitution of the flow abstraction by the atomicity of datagrams and the P2P data dissemination process. When a peer received a data chunk, it provides this data to the rest of the network. Therefore, slower hosts experience a performance boost by faster ones. Moreover, the data is brought "closer" to the peers. Thereby, the host $Wlan_2$ can receive data from $Node_2$ and $Node_3$ without using the "slow" link to the Server.

c) CCNx: The CCNx test scenario has been similar to the ones described before. We conducted six runs with a *clean state*, i.e., we cleared caches after each run, and six runs with the content *cached* at the routers. The clean cache scenario would be encountered for new content, which is currently released, or for content that has been already discarded due to the cache management policy. The results are illustrated in Figure 3. The completion time on hosts with no packet loss were significantly lower in the cached scenario, where these hosts could use their full download bandwidth of 16 Mbps. However, caching did not bring any benefit on links

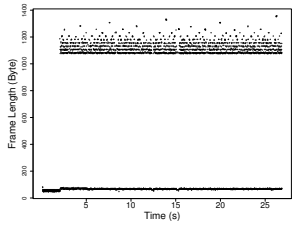

Figure 5. Libswift: Packet length over time

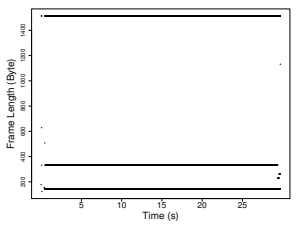

Figure 6. CCNx: Packet length over time

with high packet loss, where the difference between cached and clean state was found to be insignificant. Interestingly, the average download completion time on node $Wlan_2$ has even been higher when the content was already cached as opposed to the clean state measurements. Figure 2 shows that CCNx outperforms all other technologies, once the content is cached at the routers and a lossless link is available. Apart from the caching benefits, the superior performance of CCNx is also due to its *local broadcasting* capability. If a node searches for content, it broadcasts its interest packets on all faces. Any node that receives an interest packet and has the corresponding data may respond with a data packet [13]. Thus, CCNx employs a P2P mechanism to increase the content dissemination performance locally. On lossy links CCNx is still better than a simple file copy mechanisms, but the performance is much worse compared to Libswift, regardless whether the content has been cached or not.

C. Transport Layer Issues

To answer the question, if these new approaches use the available resources effectively at the transport level, we have investigated the packet length distribution for exemplary Libswift and CCNx sessions. The time series of the packet length process of an exemplary file transfer by Libswift is shown in Figure 5. Actually, the size of the Ethernet frames is plotted, i.e., using the term *packet* in this context, we are referring to an *Ethernet frame*. Three separate levels of packet lengths are visible. Smaller packets at about 70 bytes are used for peer discovery and maintenance of the connection, i.e. requests, acknowledgments, etc. The larger packets contain a data block of 1 KB and header information. The headers contain the local availability, acknowledgments for previously received packets and requests for new data blocks and vary in size [11]. The data block size of one kilobyte is chosen to prevent fragmentation. It is necessary due to the variable length headers, but it leads to a varying size of the Ethernet frames. As the size of the header overhead per packet remains the same, packets smaller than the maximum transmission size have a comparatively higher overhead.

As depicted in Figure 6, CCNx shows a much cleaner packet length distribution than Libswift. Like Libswift, CCNx uses UDP as default transport protocol in its C implementation and relies on IP fragmentation. Regarding the IP fragmentation, CCNx uses a hard-coded maximum transmission unit (MTU) of 8800 bytes. Per default each CCNx interest packet is answered by three full-size data packets and one smaller data packet. The first three frames have a UDP payload of 1472 bytes and a size of 1514 bytes determined by the MTU of an Ethernet frame (apart from jumbo frames). The fourth data packet has a frame size of 332 bytes. This means that in total a payload of 4706 bytes is transmitted for each interest request. Due to the reliance on IP fragmentation, it makes sense to limit the number of fragmented packets since the loss of any fragment requires to retransmit all of them. However, it is not clear why the fourth data packet is not filled up to the Ethernet MTU limit. The efficiency of the implementation could be improved by filling all four frames to the maximum size and thereby, reducing the total number of transmitted packets by more than 19 % in customer premises networks. This seems like a minor problem that could be easily fixed. But the question remains why is it not properly done from scratch.

But even the right choice of the chunk size would not mitigate the problem of the performance degradation in scenarios with packet loss. As already noted by Salsano *et al.* [12], CCNx sessions experience a severe performance degradation while encountering packet loss. This degradation is mainly due to the used Go-back-N ARQ mechanism. One viable approach could be the application of more sophisticated fragmentation schemes. They would allow us to reduce the number of CCNx interest request/reply interactions drastically by increasing the chunk size. For instance, advanced forward error correction techniques like Fountain codes would enable a much higher window size without the need for re-transmissions due to packet loss.

IV. Prefetching in CCNx

In our experiments we have noticed that current file transfer mechanisms like Rsync perform very poorly when faced with high latency and high packet loss at the same time. P2P systems will behave in similar way if no local peers are available, and CCNx also, if cached data are not available at the routers (see Figure 3). To address the described problem, we have investigated the following prefetching mechanism. If high latency and additionally high packet loss is detected, a CCN router near the client, which has high latency but low packet loss on the link to the server, requests larger data chunks in a pipelining fashion. The client will then retrieve the data from the router nearby, which is much faster, since the link provides high packet loss but low latency. The proposed prefetching mechanism targets content that is currently not cached, i.e., it has been just released or it has already been discarded due to the cache management policy of the routers. This test has used parts of the topology shown in Figure 1 with $Wlan_2$ as the client, Server as the data source and both routers, but no other client nodes (which explains the different average download times). The measurement results in Figure 4 show that there is a significant speed gain by the proposed approach compared to tests without prefetching on a non-cached CCNx router and regular file transfer by Rsync.

Moreover, the prefetching mechanism may also prove to be beneficial for ISPs since it reduces the traffic on the core network caused by re-transmissions. Applying this mechanism, the re-transmission of lost packets from the client have to reach only the CCN router next to the client. Thus, the load on the core network is reduced. This may provide an incentive for ISPs to encourage the deployment of CCN. There are two obvious use cases for this mechanism. In the first scenario the client is able to interact with its serving CCN router. It would be true for clients in home or company WLANs that use a CCN router as gateway. In this scenario the client could simply request the gateway router to retrieve the content. This could be implemented by a special interest packet that tells the router to prefetch a larger data chunk. In the second scenario the client has no possibility to interact with the CCN router. The ISP could still enable this kind of prefetching since it has the incentive to minimize the traffic on the core network, that is created by a high amount of re-transmissions arising from the client. Obviously, security and trust mechanisms play an important role in this scenario. For instance, denial-of-service attacks are possible. When, for instance, a client requests the prefetching of data, that it does not need, undesirable traffic is created on the core network. However, this vulnerability is also inherent to the current CCNx design as forged, "false" interest requests are not limited. This can be used to attack and overload the routing infrastructure by resource exhaustion.

V. Conclusions

In this paper we have investigated two content delivery proposals of next generation Internet, Swift and CCN. To provide a first insight into the actual benefits of these new transport platforms, we have evaluated the performance of their current prototypes in a realistic network scenario. We could identify some of their weaknesses, e.g., the fragmented packet length distribution of Libswift or that CCNx is obviously not beneficial for fresh, not yet cached content. Additionally, we have highlighted some of their strengths, for instance, the superior performance of Libswift on wireless links and of CCNx subject to already cached content when no packet loss is encountered. Both protocols are implemented below the application layer within the TCP/IP stack. This makes the deployment complicated and tedious since a great amount of software and hardware has to be replaced to implement these proposals. Therefore, Swift and CCN will probably take some time to reach a wider distribution, if at all. However, they both provide benefits which could be worth the effort of further optimizations. We found that Libswift is a lightweight alternative to BitTorrent at the transport layer. The fact that it is located at the transport layer may slow down deployment, but its small resource footprint and its ease-of-use may prove to be beneficial. CCN is a more ambitious project due to its operation at the network layer. It was originally designed to replace the IP stack [13], but it can also be deployed incrementally as an overlay technology. CCN aims to provide availability and performance benefits through content caching at practically all participating network elements. However, there are some open problems that need to be overcome first. A recent study by Perino and Varvello [10] states that CCN is not ready for an Internet-scale deployment yet, but only on ISP- or CDN-scale due to current hardware and software limitations. Another study by Ghodsi *et al.* [7] questions not only CCN but the whole paradigm of intrinsic and ubiquitous content caching, stating that it will probably not bring the expected performance benefits. However, the paper mentions other advantages such as the better security model. As our study did not aim to address scalability issues, we do not try to generalize our findings, neither to an (inter/national) ISP-wide scale, nor to the whole Internet. However, we have found that from the perspective of an end user caching at the current level of CCNx does not provide any benefits if links are encountering packet loss. Furthermore, if the content is not cached, the download performance of CCNx also seems to be suboptimal. Due to limited cache sizes this might happen quite often. Therefore, we have evaluated a prefetching mechanism for CCNx to cope with links exhibiting high latency and high packet loss. It has shown a significant performance gain and proves that right now CCNx is not tailored to the requirements of our mobile/wireless world.

References

[1] Libswift. http://libswift.org/.

[2] Project CCNx™. http://www.ccnx.org/.

[3] A. Abhari et al.. Workload generation for Youtube. *Multimedia Tools and Applications*, 46:91–118, 2010.

[4] A. Bakker. Merkle hash torrent extension: Bittorrent enhancement proposal 30. http://bittorrent.org/beps/bep_0030.html, March 2009.

[5] A. Bakker, R. Petroccoand, V. Grishchenko. Peer-to-peer streaming peer protocol (ppspp). IETF Draft. Expires: August 15, 2013.

[6] Cisco Systems. Cisco visual networking index: Forecast and methodology, 2011–2016. May 2012.

[7] A. Ghodsi, et al.. Information-centric networking: seeing the forest for the trees. In *HotNets*, 2011.

[8] O. Heckmann. Youtube scalability - lessons learned. In *MMB & DFT - Industrial Talk*, 2012.

[9] C. Lumezanu, et al.. The effect of packet loss on redundancy elimination in cellular wireless networks. In *IMC*, 2010.

[10] D. Perino and M. Varvello. A reality check for content centric networking. In *ICN*, pp. 44–49, 2011.

[11] R. Petrocco, et al.. Performance analysis of the libswift p2p streaming protocol. In *P2P*, 2012.

[12] S. Salsano, et al.. Transport-layer issues in information centric networks. In *ICN*, pp. 19–24, 2012.

[13] Van Jacobson, et al.. Networking named content. In *CoNEXT*, pp. 1–12, 2009.

[14] B. White, et al.. An integrated experimental environment for distributed systems and networks. In *OSDI*, pp. 255–270, 2002.

[15] H. Yuan, T. Song, and P. Crowley. Scalable ndn forwarding: concepts, issues and principles. In *ICCCN*, 2012.