Content Delivery Networks and Video Delivery

Elmar Stellnberger, Hermann Hellwagner

Department of Information Technology, University of Klagenfurt, Austria
Seminary in Multimedia Communication 623.700, 2019S
estellnb@elstel.org

Abstract. Today´s large web services like Youtube or Facebook deliver their content via so called Content Distribution Networks (CDNs) which care about routing and caching the content provided by one or more origin servers up to the end users located all over the world. Today more than half of the video content is delivered by Content Distribution Networks. Netflix and Youtube alone account for more than half of the peak downstream traffic within North America. Large CDNs usually deploy a hierarchy of caches with a tree like structure comprising parent caches and several edge caches near the clients which can provide fast access and low start-up latency for the user. Usually requests are redirected through the DNS system the way that authoritative name servers of the CDN redirect requests to the nearest edge server. Servers are usually concentrated in clusters in a few strategic locations around the world. A two fold mapping first maps a request for a certain object to a cluster centre and then to an individual machine in a cluster. Specific algorithms and data structures like Bloom filters which can space and time efficiently record a set of objects with a low false positive rate are employed to meet the algorithmic challenges within a CDN. Different cache admission and eviction policies have been proven to be best suitable for different kinds of video content. One of the best performing caching strategies thus divides a cache into partitions for different kinds of content.

Keywords: routing, overlay, DNS-resolving, content aware caching, cache hierarchies, algorithmic basis for CDNs: bloom filters, stable marriage, hashing, leader election.

1. Introduction

Initially the internet was built as a network which connected hosts for military and scientific applications. Today’s internet is developing more and more towards a content centric network at least for the majority of the users [7]. This means that we access, generate and share content without caring where our content is hosted. Content centric access was an initial feature of the world wide web though in today’s web a certain website is often no more hosted on a specific server in the web. Instead the content is delivered via a so called Content Distribution Network (CDN) which employs several edge caches near the user to speed up delivery and very often also a hierarchy of caches including parent caches which increase the origin offload. The content provider does only need to host one or more origin servers while the CDN takes care of transporting and caching the content up to the last mile to the user. The origin offload can be measured as the factor by which the CDN can reduce the traffic the origin server has to serve.

Besides the caching overlay, CDNs also provide routing and security overlays [2]. An overlay can be seen as a virtual network built on top of a physical network which provides virtual resources and new features to the user of the network. While Video on Demand is a content type which is very well cacheable, live videos can not be cached at all and thus have very different requirements. Live videos can thus profit the most from a routing overlay which speeds up delivery. Dynamically generated content like content generated by an interactive application is also largely uncacheable and thus requires a routing overlay. However other approaches like edge computing where the application logic is replicated upon multiple servers may be more promising when such an application needs to be scaled at a large scope.

In a routing overlay, the traffic is routed over several servers of the CDN rather than directly sending it over the internet. The advantage is that alternative paths for the content can be exploited because the core internet services tend to route traffic with the same destination always over the same path. This can lead to speed-ups of more than the 2.5-fold in normal operation for distances as large as between the USA and Asia and to speedups of more the 1.5-fold within North America. The larger the distance, the more possibilities are there for traffic optimization and thus the higher the net gain is. In case of catastrophic events routing overlays can offer the highest benefits [9,10]. This can be the failure of a large Internet Service Provider (ISP) like the nine hour lasting WorldCom outage in 2002 or failing of ISPs to peer with each other like Cable and Wireless in 2001 for financial reasons. At an undersea cable cut between Europe, the Middle East and South Asia (the SEA-ME-WE 4 cable in 2010) regular internet suffered severe slowdowns while the routing overlay could bypass bottlenecks with just a minimal performance degradation. This was due to the fact that a routing algorithm that always selects the same path for the same destination can not distribute the traffic upon multiple small links and does thus heavily rely on big backbones. Video delivery does profit the most from duplicating data streams among so called reflectors and later on reuniting the traffic on edge servers to compensate for packet losses [9,10].

Finally CDNs are known to effectively cope with Denial of Service Attacks simply due to their sheer large size where enough resources are available to keep servers up and running. Companies employing third-party Content Delivery Networks like Netflix and many smaller companies can rely on a large pool of resources where resources can be relocated effectively within a short time when a respective demand is given. This does not only apply to peak usages called flash crowds which are commonplace in content delivery [4] but even more to DoS attacks. It can be far harder if not impossible for a singleton service provider to provision sufficient overcapacity for such peak usage events.

Besides their size CDNs can profit from a bundled and shared security expertise [2]. Security personnel will keep the network free of known and new vulnerabilities and a security team can maneuver, if required together with the content provider, against ongoing attacks. The origin may be shielded against external traffic only allowing for the cache and overlay servers of the CDN. It is also noteworthy that there are certain peculiarities that can be used to differ between flash crowds and DoS attacks [7]. In a real flash crowd the number of clients dramatically increases while DoS attacks are usually carried out by a few clients and a small percentage of content (typically less than 1%) is responsible for a very large fraction of requests (up to 90%, i.e. Zipf-like distribution). An effective countermeasure could be to block access only for the clients carrying out the DoS attack.

In the following article we will first introduce load balancing and request redirection which does mostly rely on DNS-resolution. Then we will present admission and eviction policies for caches, analyse cache hierarchies where there are not only edge caches but also parent caches as well as content aware caching which separates the cache into multiple partitions for different types of contents. With Bloom filters, the Gale-shapley algorithm, hashing and leader election we also tell about the algorithmic foundations of CDN technology. In the conclusion we give and outlook and touch PA-CDNs (peer assisted CDNs). CDNs can be combined with P2P (Peer-to-Peer) solutions to achieve significant traffic savings for popular content when many users are online.

2. DNS-resolution and Request Redirection

If you visit a site in the internet, the respective application which is typically your browser will issue a DNS-query to resolve the domain name of the visited site. Most internet providers provide users with their own recursive local LDNS (Local Domain Name Server) which in turn asks the authoritative name servers of the CDN providers to resolve the DNS query. The authoritative name servers will respond with a server near the user which is up and running. The mapping is thereby twofold. First the request will be assigned to a server cluster near the client taking into account the distance between client and server and secondly local load balancing will assign to a

specific server within the cluster which guarantees a short response time and is thereby not already overloaded. Global load balancing does not only take the physical distance into account but also current connectivity measured by latency, packet loss and throughput as well as overall server load in the cluster. The LDNS does typically cache results from the authoritative name servers. In order to be able to reassign clients to another server the TTL field (Time to Live) for caching needs to be set appropriately.

In the traditional approach the authoritative name server does only know the IP of the LDNS but not the IP of the client when it needs to decide for the assignment of a respective edge server. Things can get really bad if a global public resolver like the OpenDNS server 208.67.222.222 is used. Then there is no hint at all where on the world the client resides. The problem can be mitigated by selecting a local DNS server. For public resolvers like OpenDNS and Google Public DNS the client-LDNS distance has a median of 1028 miles while the circumference of the earth amounts to 25.000 miles. Private LDNSes usually have a far better distance of 200-300 miles though over a quarter of the population in countries like Brazil, Australia and Argentina has to use LDNSes with a distance of 4500 miles. When we assign IPs for domain names we do actually have to consider the worst case since a service should be available all over the world and at best under any circumstances of operation.

To mitigate the problem of opaque DNS queries an extension to the DNS protocol the EDNS0 client-subnet extension has been introduced [6]. By making use of these extensions the authoritative name servers get to know the client IP address of the requester. The request is usually issued with a /24 prefix of the 32bit IPv4 and often responded upon with a /20 prefix for caching. A system that rolls out the EDNS0 extension for better DNS responses is called an end user mapping system. Results have shown that for clients with an LDNS distance of about 2000 miles the RTT (round trip time) which is the best indicator for download time can be reduced by one half and the time to first byte including the server response time by 30%. The number of DNS queries a server has to answer may rise from 870K to 1.17 million. Latency is an important issue since a 1-second delay in page response time can result in 7% reduction in web service subscriptions. 25% of the users may abandon a page if it takes longer than 4 seconds to load [3]. Even a few 100ms increase in page download time can decrease revenues [6]. Search engines also honour fast response times.

Now let us come to the task of the authoritative name servers which need to map a client request to a respective edge server. The mapping for global load balancing may be done based on a variant of the Gale-shapley algorithm which finds a stable allocation [1]. Each request is identified by a so called mapping unit made up of the IP-prefix of the requester and the traffic class of the request which may be one of video, web content, applications or software download. The algorithm then associates each mapping unit with a server cluster which will need to serve the request while both of these values define a 2-tuple or pair.

Each map unit has a preference list for clusters based on connectivity and response time while each cluster gives preference to the nearest mapping unit. A stable mapping is a mapping where for every pair no more than one participant is allowed to be able to suggest another pair where its preferences are better fulfilled. The algorithm stems from finding optimal marriages between men and women where in every round all men ask the woman they prefer most. In the first round men begin to ask the women at the top of their preference list. Women accept the best offer according to their preference list. Consequently in the first round not all men get a woman and the algorithm continues to seek for men that are not yet engaged in a fiancé. Women may change to a more preferred partner at any time. The algorithm is of complexity $O(N^2)$ and men optimal because it always favours the preferences of the men when multiple stable allocations exist. The actual implementation is a bit different as each server cluster can accept a plethora of mapping units. Resource constraints can be modelled in so called resource trees which account for network and processing power constraints in a leaf-to-root path where different constraints can be set for different traffic classes like video, web and applications. On a new mapping the newly required resources are subtracted from the remaining constraints.

In the next step the request is allotted to a number of servers within the cluster based on the serial number. Different serial numbers are dispensed to different customers or content providers. The rationale behind this step is that a content provider always provides multiple objects which are usually fetched together. This way the user does not need to open different connections to different HTTP servers if he just wants to retrieve the files of a singleton web page. This mapping may be determined by a leader election algorithm as it is too important to rely on a singleton machine for its calculation.

The final mapping step onto a specific server is not performed until the object, i.e. the requested file is known which is generally not the case for the DNS-resolver which just knows the domain name. Besides DNS-redirection there are also other request redirection methods like HTTP redirect, URL rewriting and anycast. While a HTTP redirect incurs a certain runtime penalty and a penalty when a search engine wants to index a page URL redirecting seems faster though it requires the dynamic replacement of web content. However for a media manifest file as used for videos the URL can easily be replaced upon shipping that file. The last mentioned replacement method is the anycast: Some servers have the same IP address but the request is only forwarded to the nearest server. However this method can compared to DNS-resolving, not take care of the requested object either. It is feasible that an arbitrarily selected edge server asks a cache which is known to store the object and then just forwards the object without caching the object itself. The way to connect two times to a distant server via a HTTP redirect is usually longer than forwarding a request within the same cluster.

The final mapping of a specific object to a server that holds it is done by a hashing function. Hashing with this usage purpose is called ‘consistent hashing’. In case that a server may fail or for popular objects which need to be cached by many servers an alternative value for the hash function $h_0(x)$, $h_1(x)$ is required. This value is best obtained by an own function that performs overflow caching. Storing overflowed items consecutively can yield suboptimal results if two popular objects reside on nearby positions.

3. Cache Admission and Eviction Policies

The admission and the eviction policy is at the heart of caching. The admission policy controls when a new data item is accepted and thereupon held in the cache, i.e. we say it is admitted into the cache. Every time a new item is ingested into the cache an older item has to be replaced because caches have a fixed or at least a maximum size. The eviction policy now controls which item is removed from the cache before the new item can be inserted.

When we consider a hierarchy of caches, then popular admission strategies are Leave Copy Everywhere (LCE), Leave Copy Down (LCD, only copied one level down the hierarchy) and Move Copy Down (MCD, usually less useful) [3]. One of the most popular strategies is NHIT which caches an item not before it has already been seen n-times. This strategy takes the fact into account that there is almost always a long tail of infrequently accessed content by preventing the long tail to pollute the cache. According to statistics from Akamai, a popular CDN provider, nearly three-quarters of all objects were only accessed once and 90% of the objects were accessed fewer than four times during two days [1]. Objects which are only accessed once are jocularly called ‘one hit wonders’ [5].

Disk load is a key performance metric and often the bottleneck during peak usage time. The response delay can increase significantly during peak load traffic [4]. Besides this the wear out is a drawback when using solid state drives because the content of memory cells can only be changed a limited number of times [3]. NHIT caching can reduce disk load for more than 90% without affecting the hit ratio significantly [4].

Besides the n for the n-th hit, the reset period for NHIT caching is an important parameter. The number of hits are only counted dating back up to the reset period. NHIT caching is usually

implemented by a data structure called Bloom filter which we will cover in a following section. While 90% of objects had an inter-arrival time of less than 6-hours the Bloom filter reset period was set in an experiment to 6 hours which is a relatively large value [4]. Consequently the optinmal N for NHIT caching turned out to be N=4 for small objects and N=2 for larger objects. A shorter period would incur a lower N.

The NHIT strategy may be combined with LCD by set intersection. Instead of computing NHIT independently on every machine (LCE), the object is only considered one level down the cache (LCD). Other strategies like probationary admission which admits a new item with a given probability value seem less efficient at least as long as the probability value remains fixed. A probability may be beneficial when there is not enough space to keep all objects for a given strategy [3].

The most well known eviction policies are Least Recently Used (LRU) and Least Frequently Used (LFU). LRU has been shown to outperform LFU by 5-10% for varying cache sizes. However there are more sophisticated strategies like Segmented Least Recently Used (SLRU) and Adaptive Replacement Cache (ARC). With SLRU the cache is partitioned into a probationary part where objects are placed on the first admitted access and a protected part which assimilates objects that are already in the probationary part and generate a hit. Similarly, before eviction an object falls back from the protected part into the probationary part. One hit wonders can never progress into the protected part and thus may only occupy a limited amount of resources. The difference between SLRU and ARC is that with ARC the sizes of the segments become adjusted at runtime. Each segment has a ghost list of the same length as the segment. For objects on the ghost list the objects themselves are no more stored but only their hashes. Object hashes stay a while on the ghost list (LRU) after eviction before they become forgotten. When an object shall be included in the probationary part and it is already on the ghost list, the size of the probationary part is increased by one because it was too small to still hold the forgotten object. The same applies for the protected part.

As SLRU and ARC are eviction policies and NHIT is an admission policy, both strategies can be combined. Besides the traditional implementations of SLRU and ARC, ghost lists can also be used for faster admission. An element seen in the ghost list of the probationary part could directly be assimilated in the protected part in order to save space in the probationary part and in order to keep the probationary part smaller. When using ARC two different ghost list lengths for resizing and assimilation may be used. Usually for ARC the size of the ghost list is as long as the list of objects stored in the part of the cache corresponding to the ghost list. A further well used extension to ARC is to use a time-to-live (TTL) value for the ghost-lists in order not to keep too old and outdated values.

A caching hierarchy typically includes memory, SSD and hard drives with their own parameterization or implementation of admission and eviction policies on each level.

4. Cache Hierarchies

Caches are usually organized in hierarchies. What can not be found in the edge caches where the user issues his request, is forwarded to parent caches. If it can not been found in the parent caches, the origin server is queried. As parent caches should be able to find things not present in edge caches their size needs to be respectively larger than the size of the edge caches. At Youtube cache sizes of 1GB, 10GB, 100GB, 1TB and 10TB have been considered for each level [3].

There are basically two different ways in how to construct a hierarchy of caches: Geo Split Caching and Object-based Split Caching [5]. In geo split caching you have a traditional tree like setup while for object-based split caching each server at one level is connected with every server at the level above. That way an object that is not found in one cache can be requested also from a more distant server. However this can deteriorate the response latency. Latency is the key

performance measure to consider. Answering by a cache near the user improves latency and thus performance. An improvement in the hit rate does also lead to an equivalent improvement in the latency if the object can be answered from the same cache. This is not the case for object-based split caching when distant caches are consulted. An improvement in sever load and ultimately in origin offload can be best achieved by object-based caching which observes the best hit rates while the improvement for the overall latency is not as distinct.

A worthwhile alternative is a combination of geo and object-based split caching called hybrid caching. Some objects possess geographically local popularity and best fit geo splitting while the remaining objects can be split object based. Hybrid caching performs best with respect to latency but incurs further complexity. Leaf caches have to keep track of the popularity of the content requested to answer in case of a cache miss. Caches on the same level do not interact in both models. This is different for the servers of a data center which keep to be synchronized and which act as one single cache.

5. Content Aware Caching

If we observe different content categories, we can see that they can behave very differently with regards to their caching properties. While music is being popular over weeks, news will be mostly popular on the same day. Evidence has shown that for different content categories not only the caching parameters need to be varied but also that different strategies can outperform others in another field of use [3]. A study has found that for music videos ARC/NHIT performed best while for videos about entertainment and people and blogs SLRU/LCE performed better [3]. The same study from Youtube furthermore lists which admission and eviction policy performs best on which hierarchy level and for which cache size [3]. SLRU and ARC have always performed better than mere LRU/LFU. An independent evaluation that did not consider content aware caching found out that LCD performed best so in case of doubt it may still be a good candidate to consider since it does not copy too aggressively like LCE and does not remove anything from the cache like MCD [13].

As different caching strategies perform best for different types of content it is obvious to divide the cache into multiple segments for each content type and to apply different strategies with different parameters for each category. For news a much faster Bloom filter reset period may be appropriate than for music (NHIT-caching) while early video popularity alone is unsuitable for predicting the future popularity.

The dimension of segments for individual categories may be adjusted due to request count (smaller object) or to request size and frequency (better suitable when dealing with larger objects like video). As the popularity changes over time and even diurnally it will be necessary to readjust the segment sizes frequently. Resizing the segment sizes by a predicted value should usually perform better than a historic resizing with hindsight [4].

ACDC (Adaptive Content-Aware Designed Caching) is a varied approach where a common probationary part is used for all categories and respective protected parts for each segment [4]. When a new content item is admitted to the cache an older one needs to be evicted. However this is not decided upon a prediction but upon the current state of matters. When it comes to shrink a partition different strategies have been investigated: Smallest ghost list (SGL) proposes to shrink the division with the smallest time-to-live equipped ghost list since the ghost list is rarely used. ARC would at least not extend a segment with this property as often. Largest ghost list (LGL) does the opposite and shrinks the division with the largest ghost list based on the rationale that with a large ghost list items are evicted more often and thus do not deserve caching. Which approach will perform better does of course depend on the value of the TTL property. With a sufficiently small TTL value it may be more favourable to pick SGL since a small ghost list will not be polluted and indeed signify little used content. With a larger TTL it may be more favourable to consider LGL. Then there are also the relatively smallest ghost list (RSGL) and the relatively largest ghost list (RLGL) strategies which compare the size of the ghost list to the segment size. A study at Youtube

has found the RLGL to be most appropriate for most strategies so it was used for shrinking the cache in addition to maintaining a minimum division size because of the observed convergence of smaller division sizes to very small values [3]. Initially the probationary division is assumed to shrink. We would suggest to use SGL when comparing two categories that performed best with SGL like sports and comedy. Besides this one could try to make divisions with different best opted strategy comparable by comparing a respective value to a mean observed value by the division of numbers. The mean value can be calculated by adopting the same strategy as for the value the mean value is compared to for all divisions. The temporary adoption of a different strategy merely takes place in order to calculate a mean value for comparison. Perhaps these measures could mitigate the convergence of low sized categories to an even lower size better than just a minimum division size.

Evidence has shown that ACDC performs best for cache sizes of more than 10GB. Content aware caching was found to best improve the hit rate while the effect of NHIT on disk writes was the largest [4]. ACDC could also lower the user-perceived latency and the number of disk write operations as it seems to be more stable with regards to its caching results [3]. The hit rate of segmented content-aware caching predicted with ARIMA (Autoregressive Integrated Moving Average) was observed to improve from 91.8% to 97.8% by 6% in comparison to content-oblivious caching. This is equivalent to a 73% in reduction of cache misses (cache miss reduction = 1 – low-misses/high-misses = 1 – (1-goodhitrate)/(1-badhitrate)). The hit rate improved with ACDC from 77% to 91% by 14% over non content-aware caching which amounts to a 61% reduction in cache misses (see table 2 in [3]). The more the cache hit rate approaches to 100% the higher will be the reduction in cache misses.

6. Bloom Filters and Leader Election

Bloom filters are a data structure which can store a set [1], [4]. There is no possibility to enumerate the set (except by iterating over the total universe of all possible entries) but there is an efficient method to test for set membership. Bloom filters are implemented by a hash function which leads to a certain false positive rate if hash values collide. To keep the false positive rate low an object is not only hashed once but multiple times. Only if all k entries for k subsequent tests in the bitfield yield a one then the element is assumed to be part of the set.

Bloom filters are used for implementing NHIT caching, a process which is also called cache filtering. Another application purpose is cache summarization where the Bloom filter stores which elements are present in which cache. For this usage purpose, the variant of counting Bloom filter is used since object insertion as well as deletion need to be supported. Bloom filters are a space efficient method to store sets and can easily be exchanged between servers. It is desirable to know the optimal k for a filter with m bits when inserting n elements. The probability that after inserting one element B[h(o)]=0 is (1-1/m) and after inserting k times n elements it is $(1\text{-}1/m)^{kn}$. The probability p of a false positive is therefore:

$$p = \left(1-\left(1-1/m\right)^{kn}\right)^k \approx \left(1-\mathrm{e}^{-kn/m}\right)^k, \quad k \approx (\ln 2)(m/n), \quad p \approx 2^{-k} \approx 2^{-\frac{m\ln 2}{n}}$$

With these formula the optimal number of insertions k for a given m and n may be determined. Sometimes a smaller k is chosen in order to improve runtime efficiency. It is also interesting to calculate the desired m by supplying the maximum tolerable false positive rate p and a given number of elements n that we want to insert from the last equation.

Bloom filters used for cache filtering use a reset period after which the filter is reset to zero in order not to reflect too outdated access values. In order not to lose all historic values two or more historic Bloom filters can be used. New values will always be inserted into the most current filter and the oldest filter will be forgotten when a new freshly zeroed filter is inserted into the head of the FIFO queue. For membership testing all Bloom filters in the FIFO need to be considered.

Leader election strategies are often used in the context of CDNs when it comes to load balancing or as mentioned before for the assignment of servers to serial numbers of content providers when determining on which servers in a cluster a given web page should be cached [1]. The replication of the decision making process may be necessary to avoid a single point of failure.

Before leader election takes place the participating servers, which are part of a predetermined candidate set, periodically broadcast their own health values as well as health values of servers within their reach. Health values may depend on the connectivity of a server, its load and responsiveness as well as whether the server already has a sufficiently recent data set to compute the required results. If the current leader has not been heard of for a given timeout or also simply after a given period of time a new leader is elected.

Leader elections differ in the facet whether in case of a network partition a leader is elected for each partition (at-least-one semantic) or if a leader is elected at all in case of a partitioned network (at-most-one semantics). A network partition occurs if there is no sufficiently stable interconnection between two parts of the network. While most times an at-least-one semantic is required the at-most-one semantic may be useful in case of when two different versions of a result would lead to adverse effects and when using a slightly older result is still tolerable. This may be the case for a network connectivity map where the same data should not be applied to different versions of the map.

7. Outlook and Conclusion

Today a few top content publishers account for a majority of the Internet traffic. CDNs are used for large web services and are predicted to deliver 71% of the global internet traffic by 2021. Different web applications like video streaming, web pages and social networks depend on CDNs [3]. As basic research was already carried out decades before the first CDNs were built, CDNs are nowadays a very well established technology. While some large companies like Google, Facebook and Microsoft implement their own in-house CDNs there is a number of third-party content delivery networks like Limelight, Akamai, Level 3, MaxCDN or ChinaCache that can be made use of for mid-size or smaller web services [8]. CDNs accomplish high scalability, faster content delivery, better availability, performance and origin offload. A precise caching infrastructure lowers the amount of redundant traffic over the web. CDNs improve caching, routing and security. Recent years have witnessed tremendous growth in video traffic on the Internet as a result of higher broadband data rates, proliferation in smart handheld devices and affordable unlimited data contracts offered by Internet Service Providers (ISPs). However even CDNs can be stressed by the peak usage demands of video delivery.

Nonetheless there are still challenging areas for current research efforts which can complement the traditionally established CDN technology. Software Defined Networks (SDN) can be used to exchange current connectivity values on the internet which is important for routing overlays as well as global load balancing. The ALTO protocol can be used to query such data from ISPs while extensions have been proposed to also communicate the needs of a CDN back to the network [11,12]. When it comes to implement web applications with a rich application logic, the caching service of traditional CDNs may not be sufficient so that technologies like edge computing will need to be considered. Some CDNs use cloud technology which employs a distributed model of virtualization.

Finally by combining traditional CDNs with peer-to-peer (P2P) networks the benefits of both worlds can be combined. The good thing about P2P networks is that they have the self-scaling property. The more peers there are in the swarm the more resources become available. P2P networks can complement traditional CDNs for popular content when a sufficient number of peers are online while detriments like missing Quality of Service can be levelled by the traditional CDN infrastructure. Leading companies like Akamai, ChinaCache or Xunlei nowadays deploy PA-CDNs (peer assisted CDNs) [8]. Challenges like the heterogeneity of resources, start-up delay in video delivery, clients hidden behind firewalls and incentives for user participation need to be addressed by PA-CDNs. P2P networks constitute just a very different approach to serve content and may thus also be seen as content-centric networks.

References

[1] B. M. Maggs, R. K. Sitaraman, “Algorithmic Nuggets in Content Delivery”, ACM SIGCOMM Computer Communication Review, vol. 45, no. 3, pp. 52-66, July 2015.

[2] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, M. Jain, “Overlay Networks: An Akamai Perspective”, In Pathan, Sitaraman and Robinson (eds.), Advanced Content Delivery, Streaming and Cloud Services, John Wiley & Sons, 2014.

[3] C. Koch, J. Pannmüller, A. Rizk, D. Hausheer, R. Steinmetz, “Category-aware Hierarchical Caching for Video-on-Demand Content on YouTube”, In Proceedings of the 9th ACM Multimedia Systems Conference (MMSys ‘18), pp. 89-100, June 2018.

[4] M. Z. Shafiq, A. R. Khakpour, A. X. Liu, “Characterising Caching Workload of a Large Commercial Content Delivery Network”, In Proceedings of IEEE INFOCOM 2016, April 2016.

[5] A. Rizk, M. Zink, R. K. Sitaraman, “Model-based Design and Analysis of Cache Hierarchies”, In Proceedings of the 2017 IFIP Networking Conference, June 2017.

[6] F. Chen, R. K. Sitaraman, M. Torres, “End-User Mapping: Next Generation Request Routing for Content Delivery”, In Proceedings of ACM SIGCOMM 2015, Aug. 2015.

[7] A. Passarella, “Review: A Survey on Content-Centric Technologies for the Current Internet: CDN and P2P Solutions”, Computer Communications, vo. 35, issue 1, pp. 1-32, Jan. 2012.

[8] N. Anjum, D. Karamshuk, M. Shikh-Bahaei, N. Sastry, “Survey on Peer-Assisted Content Delivery Networks”, Computer Networks 116 (2917) 79-95.

[9] K. Andreev, B. M. Maggs, A. Meyerson, J. Saks, R. K. Sitaraman: “Algorithms for Constructing Overlay Networks for Live Streaming”, arXiv preprint arXiv: 1109.4114v1, Sep 2011 / March 2018.

[10] K. Andreev, B. M. Maggs, A. Meyerson, R. K. Sitaraman: “Designing overlay multicast networks for streaming”, Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures, pages 149-158. ACM, 2003.

[11] S. Chen et al., “ALTO Implementations and use Cases: A Brief Survey”, Internet Draft (Work in Progress), July 2018. https://www.ietf.org/id/draft-chen-alto-survey-00.txt.

[12] S. Ellouze, B. Mathieu, T. Lemlouma, “A Bidirectional Network Collaboration Interface for CDNs and Cloud Services Traffic Optimization”, In Proceedings of the IEEE Conference on Communications 2013, pp. 3592-3596, June 2013

[13] J. Dai, Z. Hu, B. Li, J. Liu, B. Li, “Collaborative Hierarchical Caching with Dynamic Request Routing for Massive Content Distribution”, In IEEE INFOCOMM 2444-2452, 2012