APPLICATION OF POLICY BASED INDEXES AND UNIFIED CACHING FOR CONTENT DELIVERY

Andrey Kisel
Alcatel-Lucent

Abstract

Caching technologies are used to improve quality of experience while controlling transit and transport costs. Two common caching technologies are considered transparent caching (TC) and on-net content delivery network (CDN). We illustrate cache avoidance, analyze and model two common approaches - content hash indexes and policy based indexes - to minimize the impact. We show how the use of policy based indexes and applying advanced features of IP routing can be used to merge on-net CDNs and TC together into a single unified caching system with superior and quantifiable improvement in cache efficiency, while providing resilience and simplifying operations.

INTRODUCTION

The content industry is being reshaped by exponential video traffic growth, rapid proliferation of connected devices and delivery formats, and surging consumer demand for video from many different online sources. To stay competitive, MSOs need solutions that can cost effectively deliver pay TV content and Internet traffic with high quality of experience while optimizing transit and peering costs.

The majority of the growing Internet traffic originates from outside of MSO's networks and from Content Providers with whom they do not have business agreements to deliver content. Internet traffic is diverse by nature, and while video is the largest type by volume, software updates and Web traffic constitute substantial proportions.

MSOs are using caching technologies to improve quality of experience while controlling delivery costs for own video services and Internet traffic. They use transparent caching for unmanaged over-the-top Internet content and on-net content delivery network for managed or premium content they own or distribute for partners. Currently both solutions operate independently from each other.

There are several fundamental challenges that MSOs are facing with unmanaged OTT content: cache avoidance, a technique used by content providers to prevent MSOs from caching their content, different traffic types that needs to be cached, and maintaining high network resilience where caching is deployed. In this paper, we illustrate and model the impact of cache avoidance on TC and introduce two common approaches - content hash indexes and policy based indexes - to minimize this impact. Using traffic data from field trials and through numeric modeling we compare the two approaches and show how the use of policy based indexes improves cache efficiency and reduces latency for any traffic type. We then discuss application of advanced features of IP routing to resiliently deploy TC into greenfield and existing on-net CDNs creating a single unified caching system with improved resilience and simplified operations.

IMPROVING CACHE EFFICIENCY

Cache Avoidance

Caching is intended to provide distributed and scalable content delivery from inside MSO networks, optimizing network savings while maintaining a consistently high QoE. Cache efficiency is typically measured in cache-hit rate, the ratio of Bytes delivered form cache storage B_C versus total Bytes delivered B_C.

$$R_{CH} = {}^{B_C}/_{B_T} \quad (1)$$

HTTP cache control headers such as “Cache-Control", "Etag", "Date", "Last-Modified" are commonly used to improve cache efficiency and are well defined by RFC 2616. However, some content providers do not make their content objects easy to cache by either obfuscating object URLs, using unique object URLs for the same objects, setting incorrect values for cache control headers such as “Cache-Control", "Etag", "Date", "Last-Modified" or a combination of these methods, even if objects are cacheable by nature. While there may be valid reasons for using such approach, such as load balancing, often there are other considerations such as promoting deployment of content provider’s own caching inside MSOs networks. Regardless of the reason, this approach is known as cache avoidance, and there are two broad types of cache avoidance based on using semi-dynamic or fully dynamic URLs.

Semi-dynamic URLs are content URLs containing different object requests for the same content object. Different hostnames may also be used, for example, for load-balancing. A common feature of semi-dynamic URLs is extensive use of dynamic query string parameters and random parameters attached to object URLs to make it unique, even if the object name is the same. Semi-dynamic URLs make traditional on-net CDN caching un-efficient because multiple URLs point to the same content. An example of a semi-dynamic URL from NetflixTM is presented below:

http://31.55.163.113/737473630.ismv/range/34270098-34927096?c=gb&n=25127&v=3&e=1391197153&t=2EWCIyTIS201F4kw3AAyYWfUhb8&d=silverlight&p=5.rjKPKTOOlAKs2xPbICBh007uSde_EIHN1XbhqYSxeAs&random=976648079 (2)

The URL includes ‘random’ query string parameter making content request unique among different users requesting the same video object. In other approaches object name, ‘737473630.ismv‘ in our example, may be unique for the same video objects, and the video object may be only identifiable by static query string parameters inside semi-dynamic URL.

Fully dynamic URLs completely randomize object name and remove query string parameters, a combination of which may uniquely identify video object, form object URLs. For example, two URLs below requested the same object:

http://www846.megavideo.com/files/d008f8c759a4f4b3f07ccef7ea7588a4/
http://www763.megavideo.com/files/f66f936f7fc39f1bb9524f49c5f54184/ (3)

Caching Policy and Content Hash Indexes

In order to improve TC R_{CH} two main approaches have been developed: content policy index (content policies) and content hash index (content hashing). Content policy is a set of TC instructions how to group similar hostnames and identify the same content objects being delivered from each hostname using different semi-dynamic URLs. Essentially the policy defines parts of a URL that can be omitted and parts that shall be used for unique content identification - cache index. Content policy relies on presence of unique static parts or parameters inside each URL identifying content object. For example, a policy to cache semi-dynamic URLs (2) is illustrated below:

```
[policy-netflix]
match url regex http://([0-9]{1,3}\.){3}[0-9]{1,3}/(?<chunk>\d+\.ismv)\?c=\w+&n=\d+&v=\d+&e=\d+&t=(?<t>[\w|-]+)&d=(?<device>\w+)&p=5\.(?<p>[\w|-]+)
cache_index = netflix-$e-$chunk-$device (4)
```

The policy effectively defines that TC can use a combination of chunk name, device name and ‘e’ parameter to uniquely identify content, omitting the remaining of the URL. Content policy can identify content objects

without waiting for any response from the Origin server, and therefore content delivery can start faster improve time to first byte T_{FB}, a time between sending request and receiving the first byte of response. Content policies can efficiently cache objects identified by semi-dynamic URLs.

Alternative approach uses content hash indexes. Content hash is a hash computed form the first Bytes of object, and used to uniquely identify the complete object. In order to compute the hash, TC passes original object URL to the Origin, waits for the object delivery to start, computes hash of the first Bytes, and checks whether the hash matches hashes for content objects already stored in the in the cache. If the match is found, TC takes over object delivery and disconnects the Origin. Typically the size of data to compute hash is measured in the number of IP packets, for example, often 10 IP packets are used to identify video objects. Multiple hashing algorithms can be utilized, for example, MD5 (RFC1321) or SHA-1 (RFC3174) is often used.

Content hash indexes can efficiently cache content identified by both semi-dynamic and fully-dynamic URL. However, the proportion of fully dynamic URLs in the total traffic volume observed in a field trial is negligible, as discussed below, and they are considered instead an alternative to content policies when caching content identified by semi-dynamic URLs, rather then a supplement. Content hash indexes cannot improve T_{FB} because TC has to wait for the Origin's response, and are less efficient in caching smaller object where a larger proportion of the object needs to be used for hash computation and therefore received from the Origin. We will compare efficiency of both approaches in the next section using object distribution inside traffic volume obtained from field trial.

Cache Efficiency

We will use two characteristics when analyzing cache efficiency: R_{CH} (1) and P_{CT} – is the number of bytes required to compute hash index and B_T is defined in (1).

$$P_{CT} = \left(1 - {}^{B_H}/_{B_T}\right) \quad (5)$$

Figure 1 illustrates percentage of video traffic excluded form caching if content hash indexes are used for $B_H = 15000$ Bytes, (10 IP packet, 1500 B each), 2s Smooth streaming and 10 sec HLS segments.

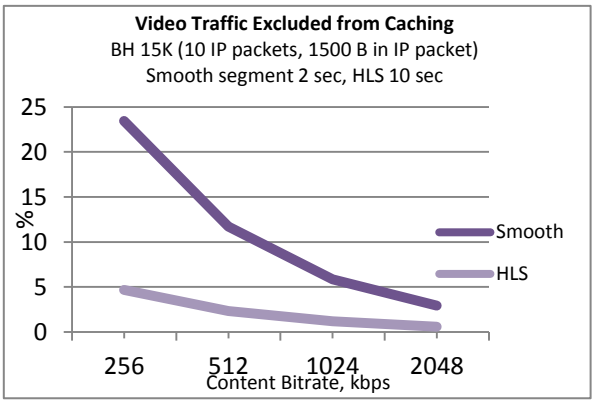

Figure 1. Video traffic excluded from caching.

As illustrated on Figure 1 more traffic is excluded from caching for lower video bitrates, decreasing cache efficiency for networks with such traffic profile, for example, serving large number of mobile clients.

While Figure 1 illustrates theoretical impact of content hashing on HTTP Adaptive streaming, to consider practical impact on all Internet traffic requires introduction of object distribution. For simplicity of practical simulations we split object sizes into bands, and define object distribution as

$$P_{ONi} = \frac{N_{O\,i}}{\sum_{k=1}^{M} N_{Ok}} \quad (6) \text{ and}$$

$$P_{OVi} = \frac{B_{T\,i}}{\sum_{k=1}^{M} B_{Tk}} \quad (7),$$

where P_{ONi} is a percentage of objects inside *i-th* band in relation to total number of objects, P_{OVi} is a percentage of traffic inside *i-th* band in relation to total traffic volume N_{Oi}- number of objects inside *i-th* band, B_{Ti}-size of all objects inside *i-th* band and M- the number of

bands. Figure 2 illustrates observed object distribution for a large NAR ISP.

On the next step let's consider P_{CT} gains - ΔP_{CTi} for object distribution presented in Figure 2. Figure 3 illustrates gains in the percentage of cacheable traffic for content

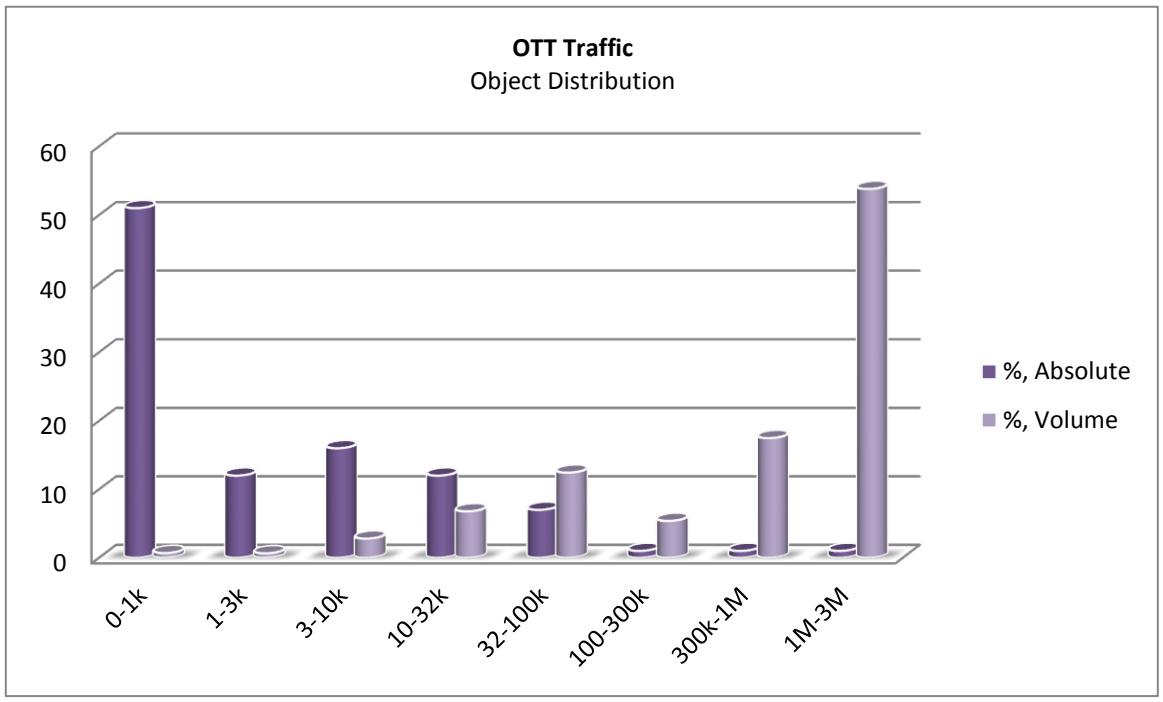

Figure 2. Object distribution for Internet (port:80) traffic.

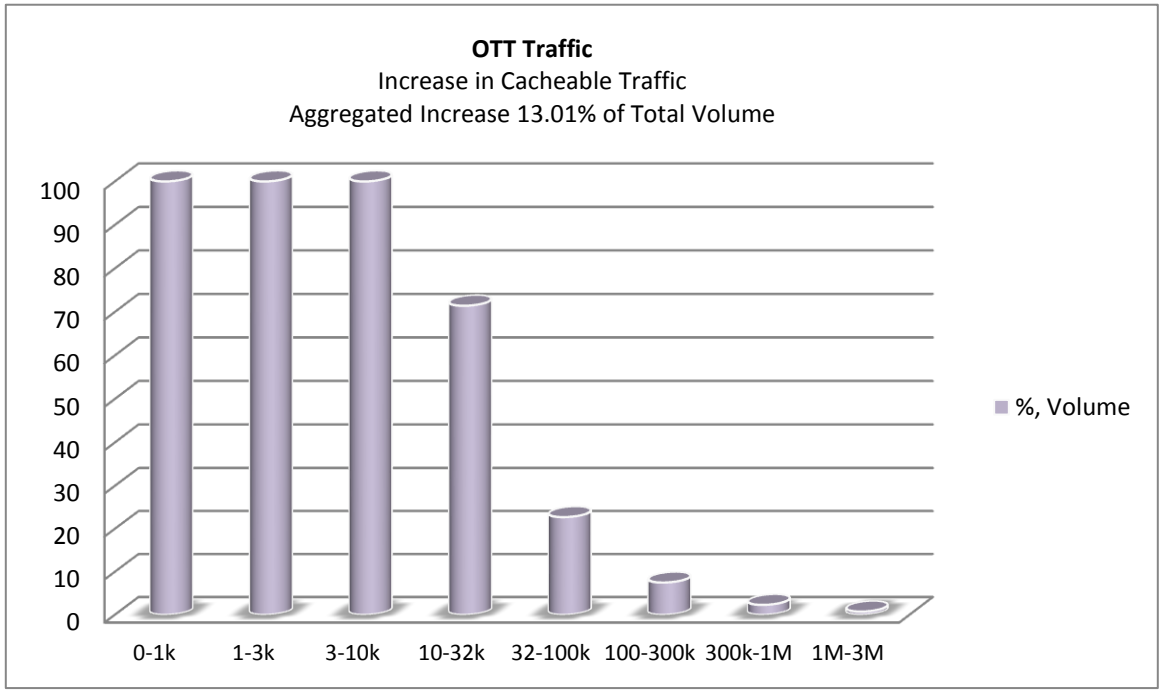

Figure 3. Increase in cacheable traffic.

policies for each of object bands, and aggregated gain P_Σ across all bands factoring into account relative contribution of traffic inside each band into total the volume (illustrated on Figure 2).

When calculating aggregated gain we applied unified object distribution inside each band for simplicity.

$$\Delta P_{CTi} = \begin{cases} 100\%, & B_H \geq \overline{B_{Oi}} \\ (B_H / \overline{B_{Oi}}) \times 100, & B_H < \overline{B_{Oi}} \end{cases} \quad (8),$$

where $\overline{B_{Oi}}$ is the average object size for *i-th* band, B_H is introduced in (5) $P_\Sigma = (\frac{1}{100}) \sum_{i=1..M} P_{OVi} \times \Delta P_{CTi}$ (9),

where P_{OVi} is introduced in (7) and ΔP_{CTi} is introduced in (8).

As illustrated by Figure 3 overall increase in cacheable traffic for object distribution in Figure 2 is 13%. The increase depends on object distribution and will be larger for bigger proportion of smaller objects. Figures 2 and 3 show two interesting trends. First, in terms of object numbers, the majority of objects are small, therefore any caching solution that can deliver small objects with better T_{FB} and lesser latency would improve QoE. Policy based indexes reduce T_{FB} for over 80% of traffic objects more then content hashing. Second, in terms of traffic volume, larger objects dominate traffic and make main contribution to R_{CH}, however, the proportion of smaller objects (e.g. below 100 kB) is non-negligible.

So far, we considered the gains in cacheable traffic by using policy indexes instead of content hash indexes. Let's consider impact of using content hashing on cache hit rate R_{CH}. R_{CH} is a linear function of B_C, and if portion of the traffic is excluded from caching, B_C and R_{CH} will linearly decrease as illustrated on Figure 4. The rate of decrease will be higher for larger potentially achievable cache R_{CH}, therefore content hash indexes impact most the better performing caching systems.

$$R_{CH} = R_{CH\,MAX}\left(1 - {}^{P_{EXC}}/_{100}\right) (10),$$

where $R_{CH\,MAX}$ cache hit rate when policy indexes are used, P_{EXC} percentage of traffic excluded from caching.

Content policies offer noticeable

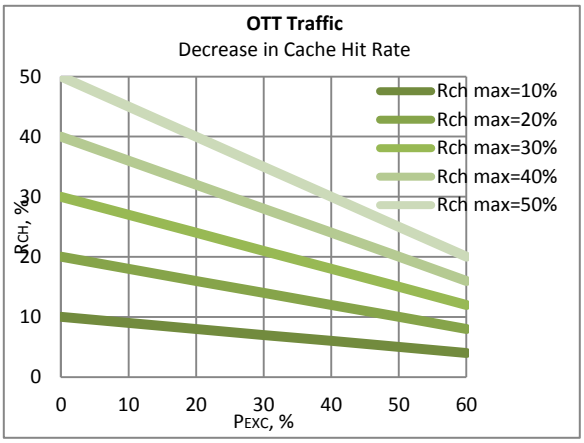

Figure 4. Cache hit rate

decrease.improvement both in increased volume of cacheable traffic and in cache-hit rates. The improvement is dependent on the traffic profile inside ISP's network. Traffic profile from a large NAR ISP yield 13% improvement in cacheable traffic and derived 6% gains in cache hit rate. Another important characteristic, often overlooked when analyzing transparent cache efficiency, is caching of small objects and improved T_{FB}. Content policies enable to improve T_{FB} for 80% of more objects, which otherwise would have slipped through caching using content hash indexes. While an argument can be made that content hashing can cache objects identifiable by fully dynamic URL, we have not observed traffic with fully dynamic URL among top 10 sites contributing over 80% of traffic volume in the field trial, therefore on their own content hashing is a weaker alternative to content policies.

Content policies can be configured to initially contact Origin, similar to content hashing, if, for example, preferred for operational reasons. However, content

policies allow initial contact with the Origin to be made without initiating content delivery by using 'If-Modified-Since' HTTP header, preserving cache efficiency. While another argument can be made about management of content policies, we observed that content policies are relatively static, and none of them needed updates after initial tuning likely due to fairly static format of content URLs used by Content Providers. Moreover, policy updates could be fully automated. Therefore both arguments do not disprove superior cache efficiency of content policies, or introduce noticeable barriers against applying content policies.

RESILIENT DEPLOYMENTS

Application of Policy Based Routing

Traditional deployment of transparent caching relies on using policy based routing (PBR) to divert all or HTTP (port:80) traffic to the cache. This approach places the cache on data path, and therefore resilience is one of the main considerations for MSOs. Load-balancers and *N+1* or *N+N* redundancy are typically used for resilience, although at extra costs and complexity. We will consider alternative approaches using advanced functions already available in IP routing. The approaches rely on the ability of IP routing to reroute traffic from failed caches using conditional redirects, or ability to duplicate traffic, letting the Origin deliver objects instead of a failed cache. The approaches do not require any modifications to the cache, for example, adding additional protocols or signaling.

First approach is based on using conditional redirects applied to policy based routing. Figure 5 illustrates this approach together with logical modifications of a routing plain.

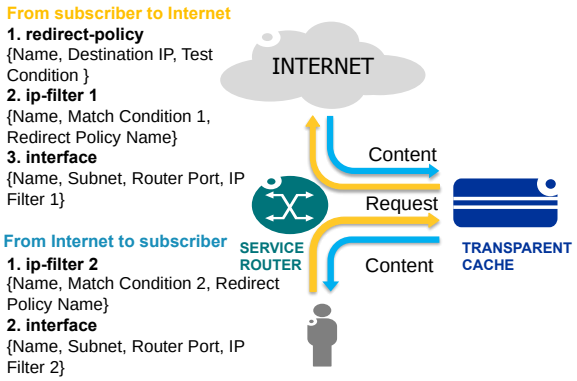

Figure 5. Resilience based on PBR with conditional redirects.

Let's consider logical modification to the routing plain for resilient handling of subscriber traffic first. A redirect policy is added to route traffic to the *Destination IP* of the cache if *Test Condition* is passed. *Test Condition* verifies cache availability, and most of the routers can test ICMP ping, HTTP GET or SNMP messages. Second, *ip-filter1* is created incorporating redirect policy and filtering traffic passing *Match Condition 1*, for example, match condition '*protocol:TCP, destination port:80'* would effectively diverting to the cache only HTTP traffic that it can potentially cache. Removing other traffic types from the cache frees cache's resources for caching instead of analyzing and returning without caching other traffic types. Next, the *ip-filter 1* needs to be applied to the router port handling traffic from subscribers.

Resilient handling of return traffic from the Internet requires similar logical steps (Figure 5) with main differences that *Match Condition 2 = protocol:TCP, source port:*80 selects HTTP traffic based on source port and the new filter needs to be applied to the router port handling return traffic from the Internet.

In case of transparent cache failure, the traffic is routed to the upper caching layers or to the Origin; therefore they would deliver more content until failed cache is restored. Field simulations in a large NAR ISP network showed that conditional redirects applied to policy based routing did not cause any noticeable network outage following

simulated cache failure, and from the user experience, there were no disruptions to unicast streaming services from Amazon and YouTube, no disruption in constant ping, or in ability to surf Web. Therefore, policy based routing with conditional redirects offered a robust and cost efficient deployment for transparent caching without need to introduce extra network functions, for example, load-balancers or *N+1/N+N* redundancy.

Application of Traffic Mirroring

Let us consider alternative approach to achieving resilience where conditional redirects are not available. The approach is based on feature of IP routing to duplicate traffic, or 'port mirroring', and requires the cache to intercept delivery for the content objects it decides to serve. The 'port mirror' approach is illustrated on Figure 6.

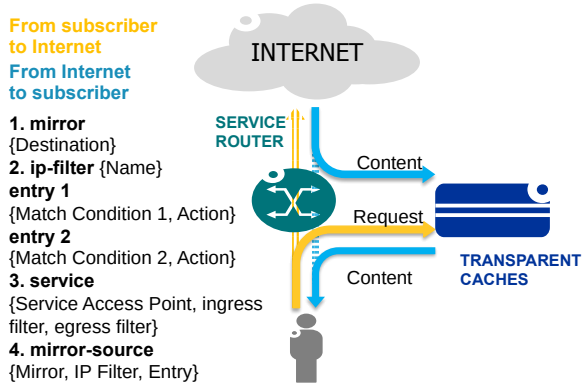

Figure 6. Resilience based on traffic mirroring.

Logical modifications of a routing plain for 'port mirror' based resilience requires creation of a mirror destination point connected to TC, a filter forwarding (*Action=forward*) HTTP traffic to and from the Internet, for example where '*Match Condition 1 = protocol:TCP, destination port:80'* and '*Match Condition 2 = protocol:TCP, source port:80'*, a service applying the filter to the ingress and egress Internet traffic, and defining filtered traffic as a source for previously defined mirror destination point.

With 'port mirror' approach both TC and Content Origin can see object requests, and TC needs to take over delivery for objects it has in the cache. Figure 7 illustrates flow diagram how TC can intercept delivery of content objects.

As shown on Fig. 7, initial request is sent to both Origin and TC, and there is a race between the Origin and the cache when replying to the request. The cache needs to win the race for successful delivery, which can be achieved by inserting HTTP 302 Redirect, message spoofing the Origin and instructing
the client to reconnect directly to the cache (or it's peer), and followed by instructions to the client to close current TCP connection to the Origin, e.g. by sending TCP FIN.

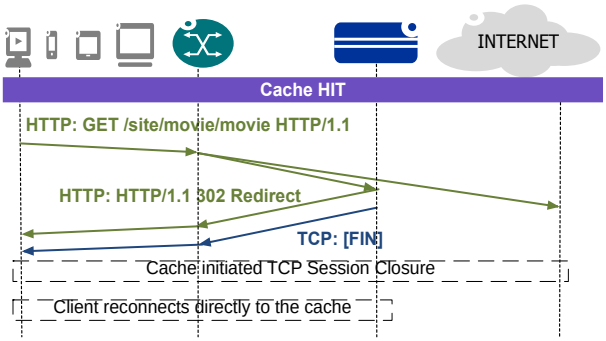

Figure 7. Flow for TC intercept of object delivery.

A cache failure in 'port mirror' approach would not affect requests delivered to the Origin because the cache is no longer on the path and resilience is achieved natively. Recovery mechanism for clients with in-progress object or video delivery during the cache failure is the same as in PBR approach, the client needs to re-establish TCP connection and re-request object being requested but not received when the cache failed, therefore practical testing of the user experience obtained for PBR mode equally apply. Similarly to PBR mode, in case of transparent cache failure, the traffic is routed to the upper caching layers or to the Origin.

However, port mirror approach has drawbacks. First, some client may either not support HTTP 302 or being prevented by local security setting from following 302 off domain for the initial object requests. Second, the race condition means that if there is small latency between the client and the Origin the cache may not win all races reducing P_{CT} and overall R_{CH}. Impact of P_{EXC} on R_{CH} follows the same linear function as for applying content hashing and as illustrated on Figure 4. The main difference is that in this case traffic would be excluded form caching due to lost races with the Origin rather then due to the need to hash first bytes of each object.

Unified Caching

The PBR with conditional redirect or port mirror based resilience enable MSOs to build robust distributed unified caching architecture where transparent caches are deployed close to network edges caching both OTT and CDN content, while CDN appliances are deployed closer to the network core. Both architectures provide caching resilience without relying on lab-balancers allowing upper-layer CDN or the Origin serve content requests instead of failed transparent caches.

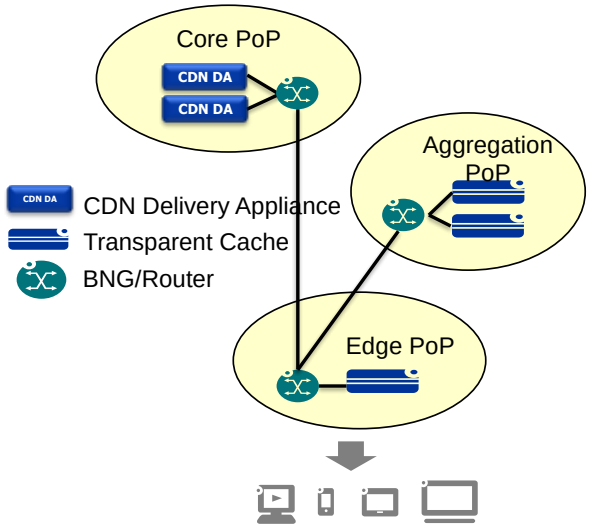

Fig. 8 Resilient unified caching.

Center the captions under each illustration and make the text large enough so that captions are easy to read.

CONCLUSIONS

We considered two fundamental challenges for MSOs when caching unmanaged OTT content: cache avoidance and maintaining network resilience, and introduced two approached to deal with cache avoidance: content policy indexes and content hash indexes. Content policies show improvement in volume of cacheable traffic and in cache-hit rates and are a stronger alternative to content hashing. The improvement is dependent on traffic profile in an ISP network. For traffic profile observed in one of the NAR ISPs policy indexes yield 13% improvement in cacheable traffic and 6% gains in cache hit rate. Other parameters affecting QoE, for example, caching of small objects and time to first byte are also discussed. Content policies enable to improve T_FB for 80% more of objects compared to content hash indexes for the traffic profile from the same ISP. That traffic would otherwise would have slipped through caching, and although 80% of objects do not translate in equal volume of cacheable traffic in Byte terms, it contributes to QoE improvement.

Further we introduced two approaches to maintaining network resilience without need for deploying extra functions in networks like load-balancers. The approaches used advanced features readily available from network routing of underlying networks: PBR with conditional redirect and port mirror. Both approaches enable MSOs to build robust distributed unified caching architecture where transparent caches are deployed in distributed network locations. However, port mirror approach is restricted to client that can support HTTP 302 redirect or are allowed to follow off domain redirects, and the race condition may reduce P_CT and overall R_CH if the latency in the network is small.

In conclusion, using PBR with conditional redirect and policy based indexes enables MSO to benefit from a more robust unified caching solution for premium and OTT content resulting in improved cache efficiency and resilience, reduced latency and simplified operations.

REFERENCES

1. Introduction to Content Delivery Networks. White paper. September 2011, Velocix, Alcatel-Lucent.
2. Velocix CDN, A Vision For The Future of Content Delivery. Strategic White Paper. September 2012. Velocix, Alcatel-Lucent.
3. Velocix unified caching. Strategic White Paper. Dec 2012. Velocix, Alcatel-Lucent.

ABBREVIATIONS

BNG	Border Network Gateway
CDN	Content Delivery Network
HLS	HTTP Live Streaming
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPTV	IP Television
MD5	Message Digest
MSO	Multiple-System Operator
NAR	North America Region
OTT	Over The Top
PBR	Policy Based Routing
PoP	Point of Presence
QoE	Quality of Experience
RFC	Request For Comments
SHA	Secure Hash Algorithm
TC	Transparent Caching
TCP	Transmission Control Protocol