# IMPROVING CUSTOMER EXPERIENCE THROUGH COOPERATIVE IN-HOME CACHING AND PRE-POSITIONING (CIHCP)

John Jason Brzozowski
Jan van Doorn
Comcast Cable

*Abstract*

*While discussions of content delivery often center on backbone architecture, a substantial portion of the cost of delivering content from Internet peering points to subscribers is spent on Access Networks.. Content Delivery Networks (CDNs) have done a good job optimizing backbone traffic, but to date, there has been less innovation around optimizing traffic flows at level of Metro and Access Networks, which represents a major opportunity.*

*ISP networks, including Metro and Access Networks, are scaled for peak usage, and are underutilized at off-peak times. In this way, user demand and usage is to some degree, predictable.*

*In this paper we explore a potential method of cooperative in-home caching and pre-positioning. This approach seeks to leverage the predictable nature of user demand to reduce the highest peaks, by pre-caching content in the home during the traffic valleys. In this paper we explore only the potential technical function of a cooperative in-home caching and prepositioning approach. Related issues such as cache management, telemetry, and application interfaces would be the subject of future work on this topic.*

## INTRODUCTION

Traffic patterns in ISP IP networks are very predictable, with high peaks at primetime and low valleys late at night / early morning. See Figure 1 for a typical example of traffic in an ISP's network segment. *In most ISP networks it is not uncommon for traffic peaks to represent three times their corresponding valleys.* Obviously, the network has to be scaled to at least the highest peak in these normal traffic patterns. Specific events, like a software or a game release, can spike these peaks even higher: 1.5x the normal daily peak is not unheard of, so for the ISP to maintain quality of service in those cases it would need to scale the network much higher than even these daily peaks. It follows that there is unused capacity in the ISP network for a very large percentage of the day.

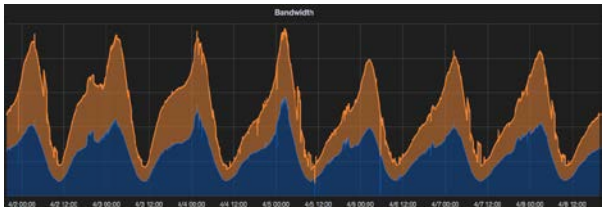


*Figure 1: Typical traffic pattern*

CDNs do important work in reducing backbone or transit traffic but that is typically the smallest portion of the network cost and they are often not in the optimal position in the network. One of the goals and ongoing challenges has been to move content delivery to be as close to the end user as possible. In many cases the deepest (meaning closest to the customer) caches are outside of the ISP network. Recently, many ISPs have started deploying CDN caches deeper in their networks. Even when deployed in the ISP network, the caches require extensive Metro resources and are still "above" the CMTS driving most of the costs of delivering the bytes to the customer (i.e. below the CMTS [1].)

Individual bandwidth usage is, to a certain degree, predictable as well. For IP video, a

viewer who scheduled a DVR recording for a certain show is most likely going to watch that show. After seeing episode 1 through 4 of a series, it is likely that episode 5 is next. Some OTT video providers have a "My List" or "Watch List" feature that the viewer can use to queue things to watch later – the videos queued here are likely to be watched by this person. For software releases, a person who downloaded MacOS 10.11.3 is likely going to download version 10.11.4 of that same OS shortly after it is released. Similarly, a household that has several devices of the same make and model, i.e. iPhone6, is likely to download the same update to each and every device. Other possible use cases of cooperative in-home caching and pre-positioning could be cloud-based backups that can be scheduled for upload overnight instead of when children are actively gaming online and pre-positioning of popular game release, or updates, gradually in advance of the official release date.

Individuals and families also often have unused storage and computing resources in their home. The household PC is often sitting idle, as is the game console's storage. Those, along with retail devices, like Network Attached Storage (NAS) appliances, represent opportunities to leverage large amounts of idle in-home storage.

In this paper we try to make use of the above observations by exploring on-demand and pre-caching of content at low traffic times in an in-home cache; for the purposes of this paper it is assumed that the cache is customer-owned and maintained equipment, which cooperates with either the CDN or the client. However, the technological concepts detailed here apply equally to operator-provided in-home cache appliances.

In all cases, these models are premised on a cooperative, opt-in approach that would involve the participation of customers, content providers and ISPs.

A comprehensive in-home caching strategy including content pre-positioning is mutually beneficial to multiple stakeholders across the Internet ecosystem. The primary beneficiary is the consumer, who will realize the benefits of massively improved performance, efficiency, and a heightened customer experience. Content owners who elect to participate can benefit by way of increased delivery efficiency by transmitting fewer copies of the same content to a single household, and off-peak content delivery infrastructure, which manifests lower costs and improved customer satisfaction. Broadband network operators with the ability to deliver content off-peak can especially benefit, by way of overall network performance and capacity management. Tests have illustrated that a basic in-home caching solution minimally yields a 300% increase in download performance when in- home caching is utilized. The evolution of in-home caching to include off-peak content pre-positioning introduces the additional benefit of delivering content when network utilization is lower, which helps to utilize valuable network resources when demand is at its lowest.

The rest of this paper is organized as follows: First we examine the different types of HTTP proxies in use today, and how CDNs utilize them. Then we explore the options available to insert the in-home proxy into the client's request path. Next, we describe a prototype implementation that was built to help think through the problems and options. We conclude with final thoughts and suggested future work.

## HTTP/1.1 AND PROXY/CACHES

This section is an abbreviated version of a section of the Traffic Control documentation written by one of the authors of this paper [2]. The main function of a CDN is to proxy requests from clients to origin servers and

cache the results. To proxy, in the CDN context is to obtain content using HTTP from an origin server on behalf of a client. To cache is to store the results so they can be reused when other clients are requesting the same content. There are three types of proxies in use on the Internet today: The "reverse proxy," the "forward proxy," and the "transparent proxy." They are described below.

Reverse Proxy

A reverse proxy acts on behalf of the origin server. The client is mostly unaware it is communicating with a proxy and not the actual origin. The typical use case for the reverse proxy is a CDN - to the end user, a CDN appears as a reverse proxy because it retrieves content from the origin server, acting on behalf of that origin server. The client requests a URL that has a hostname which resolves to the reverse proxy's IP address and, in compliance with the HTTP 1.1 [3] specification, the client sends a `Host:` header to the reverse proxy that matches the hostname in the URL. The proxy looks up this hostname in a list of mappings to find the origin hostname; if the hostname of the Host header is not found in the list, the proxy will send an error (`404 Not Found`) to the client. If the supplied hostname is found in this list of mappings, the proxy checks the cache, and when the content is not already present, it connects to the origin the requested `Host:` header maps to and requests the path of the original URL, providing the origin hostname in the `Host:` header. The proxy then stores the content for this URL in cache and serves the contents to the client. When there are subsequent requests for the same URL, a caching proxy serves the content out of cache thereby reducing latency and network traffic.

For a reverse proxy to be injected into the path of delivery the content owner changes the delivery URL to not point to the origin server, but to the CDN, and the CDN is configured to map this delivery URL back to the origin.

Forward Proxy

A forward proxy acts on behalf of the client. The origin server is mostly unaware of the proxy; the client requests the proxy to retrieve content from a particular origin server. The typical use case for the forward proxy is an organization trying to save on Internet resources, either bandwidth or address space. In a forward proxy scenario, the client is explicitly configured to use the proxy's IP address and port as a forward proxy. The client always connects to the forward proxy for content. The content provider does not have to change the URL the client obtains, and is unaware of the proxy in the middle.

For a forward proxy to be injected into the delivery path, the client is configured with the forward proxy information, and often, for what domains to use it.

Transparent Proxy

In the Transparent proxy case, neither the content owner nor the clients are aware that the proxy is in the path. The traffic is intercepted by putting the cache directly in the path as a router, or by a router with a special configuration to intercept traffic and redirect it to the proxy. The typical use case of the transparent proxy is a school or business that wants to force all Internet traffic through the proxy to be able to block some sites (parental controls) and to share Internet resources.

For a transparent proxy to be injected into the path, a network administrator in the layer 3 network path needs to configure the interception.

Even though caching and proxying are distinctly different functions, in the rest of this paper we will refer to a *cache* as the

combination of these two functions in one device or piece of software running on a multi purpose computer.

## CDN INTEGRATION

An in-home cache needs to be inserted into the CDN delivery path (with the cooperation of the CDN of course). In this section, we describe the options that were evaluated. Before we do that, we first look at how a CDN normally gets inserted into the request path. The client will do a DNS address lookup of the hostname of the content URL, and in most cases the CDN is authoritative for the domain-name used in the content URL. The CDN can do one of 2 things. For short-lived sessions, the CDN will return an IP address of an edge cache that is closest to the local DNS (LDNS) server that handles the DNS resolution for it. This is often referred to as *DNS content routing*. DNS content routing is low-latency, but has some disadvantages: the CDN doesn't actually know the client's IP address, only the IP of the LDNS server, and the client could have a remote LDNS configured, such as Google's public DNS [4], moreover, the CDN only knows the hostname of the content URL, not the path. To address this, some CDNs will send the IP address of a content router in response to the DNS query of the hostname in the content URL. The client will now connect to this content router and this router will send a HTTP 302 redirect to the client based on the actual client IP address *and* the requested content – a much more informed decision that enables better localization and content affinity. This is often referred to as *HTTP content routing.* DNS content routing is usually preferred for objects and images in web pages, since there are usually many of these in a web page, and the added latency of the 302 redirect is too costly. For video and game or software downloads, the added latency is not that much overhead compared to the expected download time, and the added precision is worth the extra round trip to the content router.

Next, we look at the options of inserting an in-home cache into the client request flow.

1) Domain Name System

In this case, the CDN authoritative DNS server will respond with a CNAME to a `.local` address that the in-home cache advertises using mDNS, and has a reverse proxy rule for. The biggest advantage of this approach is the simplicity, but that is also it's drawback – there is no way the CDN can determine what client is getting the DNS response (only the DNS caching server or LDNS server knows), so the CDN will have to send this response to everyone asking for the address of this name, regardless of presences of the local proxy. HTTP only allows for A/AAAA records in resolving hostnames, and while you can send multiple IP addresses in the answer to an A/AAAA query, these answer records don't allow for any prioritization. This means we can't send a DNS response to instructs the client to first try the in-house cache, and if that fails, then try the CDN cache higher in the network.

2) HTTP 302 Redirect

With HTTP 302 redirect, we can target individual clients, and we can send the 302 to a URL that "points to" the in-house cache. The advantage here is that we now can only redirect to the local cache if it is running and "registered" with the CDN, and only for content that is known to be cached, since the registration could tell the CDN content router what content is pre-cached. The disadvantage is that the CDN now needs to keep track of all the clients that have an active caching proxy in their home, as we as the content that is pre-cached. At scale, this could be a challenge.

3) Transparent Proxy

Transparent caching could be an interesting option, but we are looking to do this in cooperation with the content owner, and, with the advent of HTTPS in standards [5, 6] and practice [7] it seems that transparent proxies

are becoming more and more difficult (if not impossible) to implement without major security compromises.

4) DNS Intercept
In this scenario, the DNS request for the CDN domain is intercepted by software on the in-house cache using ARP (Address Resolution Protocol) spoofing, special software on the router, or even on the LDNS server itself. For domains that have content pre-cached, a CNAME response is sent to the client that points to a .local name the cache is advertising using mDNS. Another variant of this method is the Smart DNS proxy method, where the client is configured with LDNS servers that send the proxy's IP address in response to a DNS query for certain origin / CDN domains. Finally, interaction between the local in-home cache and the local DNS server can also be leveraged to update the local DNS with the IPv6 and/or IPv4 address of the in-home cache. Often the home gateway or wireless gateway in the house is configured as the LDNS server for all downstream clients, and it simply forwards all DNS requests to the ISP's LDNS servers. This local LDNS server in the gateway could be modified to be authoritative for a domain that is not `.local` (and thus doesn't have the mDNS quirks and drawbacks), for instance `cache.lan.` When the in-home cache pre-caches content for a domain, it updates the gateway and the CDN serving this domain with that information, and this update would ensure that the LDNS server in the gateway is able to resolve the CNAME response that was provided by the authoritative CDN DNS server locally.

With the growing and pervasive deployment of Domain Name System Security Extensions (DNSSEC) it is important to note that some DNS approaches outlined here may no longer be viable or the use of the same may become increasingly complex. Cooperative in-home caching and pre-positioning that properly integrates every player in the content ecosystem could conceivably incorporate the use and support of DNSSEC including any local DNS infrastructure in the home. However, it is important to note that the use of DNSSEC may not be strictly required by local DNS. The detailed analysis of DNSSEC and content caching and pre-positioning are out of scope for this document.

5) HTTP Intercept / 302
A cache can be inserted by "snooping" the connection that carries the HTTP GET request, then returning a 302 to the cache on the connection without the knowledge of the destination server of the HTTP GET request. This is what Laguna [8] does. This has the same restrictions as method 3.

6) None – Client configuration and forward proxy
In this and the next option, we don't cooperate with the CDN, but instead with the browser. The browser is configured to use the in-home proxy/cache as a forward proxy for certain domains, or for all domains. The drawback of this system is the client configuration, and the fact that now the proxy is in the client path for all requests in the configured domain.

7) None - Proxy Auto Configuration (PAC)
This option is very much like 6, but here, we use the in-home proxy/cache as the PAC server, and it can insert domains to dynamically use the in-home cache, as the content is pre-positioned. Since most PAC systems do not have an auto update, and the proxy configuration only gets loaded on client start, this option does not seem to add much over option 6.

Notes about mDNS

The multicast Domain Name System, (commonly referred to as mDNS) [9] is a system for advertising and resolving host names to IP addresses (as well as services locally) in small networks that don't have a local name server. It is most often used with

zero configuration services like DNS service discovery (DNS-SD) [10] or Apple's Bonjour. In this case we use the simplest form of mDNS. We only use it to do hostname advertisement, and to advertise A or AAAA records, with no associated services.

Using mDNS carries some drawbacks. During testing, we found that some applications assume it is used in conjunction with DNS-SD, and won't work when you just use it for hostname resolution. Specifically, we were not able to have a CNAME in the unicast DNS system point to a `.local` mDNS announced record, and have browsers "follow" that CAME. This severely hampers some of the DNS based CDN integration options.

In an IPv6 capable system the mDNS announcements in the CDN integration options could be replaced with real DNS AAAA records that point to the IPv6 address of the in-home cache. This has the advantage of not requiring mDNS, but adds some scaling and other requirements to the DNS system overall, specifically for authoritative DNS server operators which also tend to be content owners and/or CDN providers.

## THE PROTOTYPE SYSTEM

The three options explored in the prototype system outlined in this paper are option 2, cooperation with the CDN, a combination of option 2 and 5, where we 302 redirect to a unicast announced DNS with gateway modification, and option 6, cooperation with the content consumer.

The sequence of events when using the system in CDN cooperation mode is depicted in figure 2. Steps 1 through 5 are only done in low traffic times, while the rest of the steps can be done at peak or low traffic times.

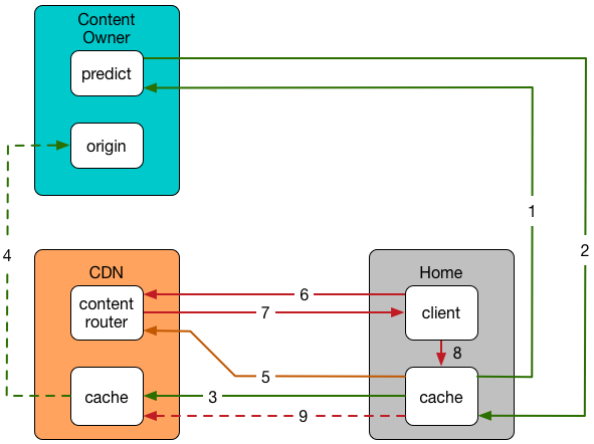


*Figure 2: in home pre-caching system with CDN/content owner cooperation*

1) The in-home cache registers with the content owner(s).
2) The prediction engine at the content owner instructs the in-home cache to pre-cache an (or set of) object(s).
3) The in-home cache fetches the content from the CDN and caches it.
4) On miss, the cache fetches the content from the origin server.
5) The in-home cache registers the pre-cached content with the CDN's content router. In an IPv4 system, the registration will likely arrive at the CDN's content router using the same IPv4 NAT address that the client will use. In an IPv6 system, it will need to tell the CDN content router the delegated prefix. At this time, the in-home cache also adds an mDNS announcement of the content URLs hostname with a .local appended (<original FQDN>.local).
6) The client requests the content from the CDN
7) The CDN content router, knowing that this content is pre-cached at the in-home cache, redirects the client to <original FQDN>.local
8) The client fetches the content from the in-home cache
9) On miss, the in-home cache fetches the content from the CDN cache.

Another similar option is the use of unicast DNS. This approach shares many similarities with that of the mDNS-based solution. The authoritative DNS server for the origin content must still be able to detect and reply accordingly to end user and in-home cache requests separately. An example sequence of events in this option is show in figure 3:

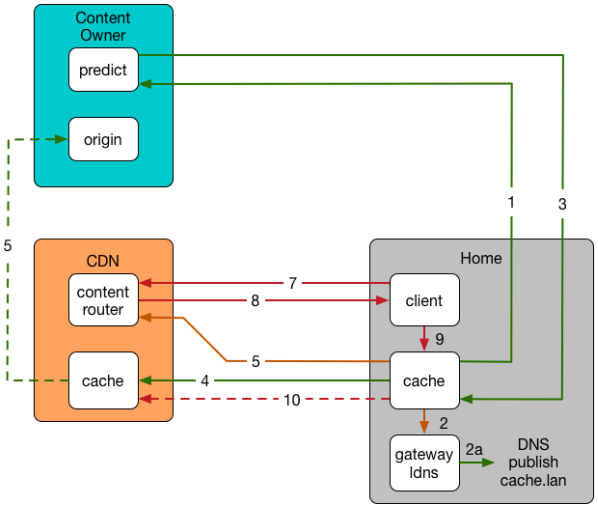


*Figure 3: option 2 and 5 combined*

1. The in-home cache registers with the content owner(s).
2. The in-home cache must publish itself to the local, unicast DNS server in the customer's premises. This could very well be an API or dynamic update to the DNS server running within the Wi-Fi router or a standalone DNS server. The in-home cache would update DNS with an FQDN that is resolvable locally (i.e. `cache.lan.`) and that would be used by the authoritative DNS server as a CNAME to redirect content requests to.
3. The prediction engine at the content owner instructs the in-home cache to pre-cache (a set of) objects. This is the same whether unicast DNS or mDNS is used.
4. The in-home cache fetches the content from the CDN and caches it.
5. On miss, the cache fetches the content from the origin server.
6. As part of the registration the in-home cache can for example be added to an authoritative view that will allow it to properly resolve DNS for the content origins, instead of the resolution that will redirect content requests to the local, in-home cache by default. In an IPv4 system, the registration will likely arrive at the CDN's content router using the same NAT address that the client will use. In an IPv6 system, it will need to tell the CDN content router the delegated prefix. At this time, the in-home cache is expected to be resolvable using a local FQDN, i.e. `cache.lan.`
7. The client requests the content from the CDN
8. The CDN content router, knowing that this content is pre-cached at the in-home cache, 302 redirects the client to `cache.lan.` with the same path. Conversely, the CDN's authoritative DNS servers may use CNAME resource records to re-direct the client content request to the local, in-home cache.
9. The client fetches the content from the in-home cache
10. On miss, the in-home cache fetches the content from the CDN cache.

Separately, option 6 has also been implemented and assessed. This is the option where the content consumer is cooperating with the in-home caching system.

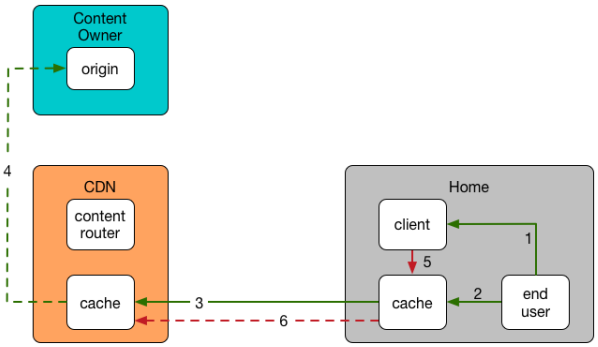


*Figure 4: in home pre-caching with content consumer cooperation*

1) The end user configures the client to use the in-home cache as a proxy for certain domains.
2) The end-user tells the in-home cache what to pre-cache, or the in-home caches uses the "watch list" for the content to pre-cache.
3) The in-home cache pre-caches the content.
4) On miss the CDN cache fetches the content from the origin.
5) The client requests the content with the original CDN hostname, but the client will use the configured forward proxy, and thus go through the in-home cache.
6) On miss, the in-home cache fetches the content from the CDN cache.

Candidate requirements for the end-to-end system are listed below:

*Secure*
Security and privacy are of paramount importance for a system like this. This applies equally to content consumers and creators.

*Open*
The system must be based on open standards, and must completely comply with the same. Ideally key elements of the solution itself may be open sourced and/or include open APIs (application programming interfaces) to enable wide spread adoption and seamless integration. Some of the CDN integration options that intercept traffic are disqualified by this requirement, since they create traffic patterns that violate the specification.

*Non-intrusive*
The system must be seamless to the end-user once installed, and even more so if failures were to occur; there must be no impact to the customer experience. The last thing we want is for a failure in this system to prevent viewers from accessing content they would've otherwise been able to access.

*Scalable*
A CDN often serves many millions of users simultaneously. A cooperative in-home caching and pre-positioning system needs to scale to the same numbers.

*Content-Aware*
Given the fact that a large proportion of the Internet traffic these days is video, the system should understand popular ABR video formats. It should understand, for instance, the most-used manifest file formats, so that the in-home cache can be instructed to cache a video asset just by giving it the manifest URL.

<u>The in-home cache requirements:</u>

Requirements for the in-house cache include:

*Portability*
The software should run on as many operating systems and environments as possible, with as few dependencies as possible. While a hardware appliance may be utilized, the in-home caching system should be able to run on consumer-provided hardware, such as a personal computer or Network Attached Storage (NAS) with sufficient capacity.

*Lightweight*
The software should be able to run on small systems as a background daemon with little or

no noticeable impact to performance of other tasks.

*User-Friendly*
The software should be easy to install and configure. Any end-user should be able to install and configure it in minutes, and no additional management or maintenance should be necessary.

Given these requirements we decided to implement the in-home cache using the Go language. The Go language is supported on many platforms, and has the big advantage that the binaries produced are self-contained, and not dependent on libraries or other external files, and don't require installation of any packages. See github.com [12] for the source code for the prototype in-home cache and CDN integration implementation cited in this paper.

Predicting what to pre-cache

Predicting what to cache is easy when we cooperate with the consumer and/or content owners – the user will simply tell the system what to cache either by directly interfacing with the in-home cache, or by keeping a watch list up to date. Consumers, in cooperation, with the content owners and broadband operators can further enhance the in-home caching experience. People consuming content may optionally share additional information with the prediction engine that can be used to more accurately predict future video selections.

In the case that the content owner is providing the pre-cache instructions, the content owner can use its own usage data to create the pre-cache rules. How to do this in the most optimal way is out of the scope of this paper.

## CONCLUSION

Over the years, the way people use the Internet has rapidly evolved -- from web pages and email, to rich applications, video, and interactive content. This transformation has dramatically changed how players across the Internet ecosystem engineer their infrastructure, products, and services. Modern applications and rich content has pushed the limits of modern software and networking technologies resulting in new, innovative approaches to support the delivery of the same. Innovation and engineering in these areas has been largely focused on increasing efficiency, lowering costs, and ultimately improving the customer experience.

The approach documented in this paper is intended to take these innovations to new levels. A cornerstone principle of this work has been to assess, technically, how close content can be positioned to the consumer, and what benefits can be expected from such close proximity. A material and intended byproduct of this work has been to develop a proof-of-concept, intended to derive data that can be used to quantify the performance and efficiency of in-home caching and pre-positioning. As cited in this document, the performance improvements are non-trivial, as are the benefits for the broadband network -- and ultimately the customer experience.

In summary, the principles outlined in this document, when done cooperatively, represent distinct, measurable benefits to every stakeholder across the content ecosystem. Cooperative in-home caching and pre-positioning effectively represents a win-win-win scenario for content consumers, owners, and for those who facilitate delivery of the same including ISPs.

## REFERENCES


[1] http://traffic-control-cdn.net/

[2] [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,

Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, http://www.rfc-editor.org/info/rfc2616

[3] https://developers.google.com/speed/public-dns/

[4] Dorwin D., Smith J., Watson M., Bateman A. “Encrypted Media Extensions”, W3C Working Draft, March 2016, https://www.w3.org/TR/encrypted-media/

[5] [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, http://www.rfc-editor.org/info/rfc7540

[6] Dan Goodin, “It wasn’t easy, but Netflix will soon use HTTPS to secure video streams”, April 2016, http://arstechnica.com/security/2015/04/it-wasnt-easy-but-netflix-will-soon-use-https-to-secure-video-streams/

[7] https://github.com/concurrentlabs/laguna

[8] [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, http://www.rfc-editor.org/info/rfc6762

[9] [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, http://www.rfc-editor.org/info/rfc6763

[10] https://golang.org/