CoreCast: How Core/Edge Separation Can Help Improving Inter-Domain Live Streaming

Loránd Jakab[a,*], Albert Cabellos-Aparicio[a], Thomas Silverston[b], Marc Solé[a], Florin Coraş[a], Jordi Domingo-Pascual[a]

[a] *Universitat Politècnica de Catalunya, Barcelona, Spain*
[b] *The University of Tokyo, Tokyo, Japan*

Abstract

The rapid growth of broadband access has popularized multimedia services, which nowadays contribute to a large part of Internet traffic. Among this content, the broadcasting of live events requires streaming from a single source to a large set of users. For such content, network layer multicast is the most efficient solution, but it has not found wide-spread adoption due to its high deployment cost. As a result, several application layer solutions have been proposed based on large-scale P2P systems. These solutions however, are unable to provide a satisfactory quality of experience to all users, mainly because of the variability of the peers and their limited upload capacity. In this paper we advocate for a *network layer* solution that circumvents the prohibitive deployment costs of previous approaches, taking advantage of the rare window of opportunity offered by the Locator/Identifier Separation Protocol (LISP). This new architecture, motivated by the alarming growth rate of the default-free zone (DFZ) routing table, is developed within the IETF, and aims to upgrade the current inter-domain routing system. We present CoreCast, an efficient inter-domain live streaming architecture operating on top of LISP. LISP involves upgrading some Internet routers and our proposal can be introduced along with these new deployments. To evaluate its feasibility in terms of processing overhead in networking equipment we have implemented CoreCast in the Linux kernel. Further, we compare the performance of CoreCast to the popular P2P streaming services both analytically and experimentally. The results show that CoreCast reduces inter-domain bandwidth consumption and that introduces negligible processing overhead in network equipment.

Keywords: LISP; multicast; live streaming

*Corresponding author
Email addresses: ljakab@ac.upc.edu (Loránd Jakab), acabello@ac.upc.edu (Albert Cabellos-Aparicio), thomas@hongo.wide.ad.jp (Thomas Silverston), msole@ac.upc.edu (Marc Solé), fcoras@ac.upc.edu (Florin Coraş), jordid@ac.upc.edu (Jordi Domingo-Pascual)

1. Introduction

The rapid growth of broadband access speeds has popularized multimedia services that nowadays contribute to a large part of the Internet traffic [1]. A considerable amount of multimedia content, such as popular live events commonly conveyed through IPTV, require live streaming from a source to a large set of users. For such content, network-layer multicast [2] is the most efficient solution. However, its deployment remains confined to some selected domains. Enabling inter-domain multicast requires upgrading a large subset of the existing routers and complex network management, tasks that amount to very high capital and operational expenditure [3].

As a result, during the last years, the industry along with the research community have designed several peer-to-peer (P2P) systems for streaming live events [4, 5, 6, 7, 8] to an ever increasing audience. These systems operate at the application layer, and create complex overlay networks to distribute multimedia content. Some of the commonly used systems are PPLive [7], TVAnts [8] and UUSee [9]. These applications are very popular and are being used daily by millions of Internet users [10].

Unfortunately, it has been shown that in practice these live streaming systems are unable to provide a satisfactory quality of experience at all times. As a consequence, the research community has thoroughly analyzed the streaming quality of these large-scale P2P networks in search for explanations [10, 11, 12, 13, 14, 15, 16]. Surprisingly, one of the main findings is that the streaming quality *degrades* as the number of peers *increases* if peak hours are considered [17]. Further, server capacity still plays a key role in these systems, and its insufficient supply leads to low streaming quality [17]. The main reasons behind these inefficiencies are: (*i*) limited upload capacity of the peers, who usually access the Internet through asymmetric links, and (*ii*) churn, peers may join/leave the system at any moment. Because of these reasons, application layer-based systems cannot guarantee a reliable streaming service, and cannot be considered as the long-term solution for broadcasting live multimedia content over the Internet. In this paper we advocate for a *network-layer* approach as the long-term solution.

We present CoreCast, a network-layer reliable live streaming protocol that avoids the high deployment cost of current approaches. In order to circumvent their prohibitive costs, CoreCast exploits a rare window of opportunity offered by the development and deployment of the Locator/ID Separation Protocol (LISP) [18], on top of which it is built. LISP is a new network architecture having an IETF working group devoted to its development and also enjoying support from Cisco. Its aim is to upgrade the current inter-domain routing system. The change is motivated by the alarming growth rate of the default-free zone (DFZ) routing table, listed as the most important problem facing the Internet [19]. In order to solve this critical issue, recent discussions within this Internet standards body suggested splitting the current IP address space into separate namespaces for identifiers and routing locators. The community generally agrees that this separation is a basic component of the future Internet,

and that the current routing system must be upgraded. Among the proposals rooted in this approach [18, 20, 21] LISP is the most advanced one and already counts with an experimental testbed (lisp4.net and lisp6.net) in the Internet. The deployment of LISP involves incrementally upgrading the border routers of all the autonomous domains present in the Internet. CoreCast extends LISP, providing live streaming capabilities, while only requiring explicit support at these selected routers. This way, CoreCast can be deployed along with LISP, avoiding the prohibitive deployment cost of IP multicast.

CoreCast has a push-based architecture, and its operations are based on two demultiplexing points and a caching mechanism. The streaming server transmits one header per subscriber and just one copy of the content for all of them. The stream is first cached and subsequently demultiplexed towards each client ISP. Then, each client ISP's border router caches and demultiplexes again the stream towards the subscribers. The main benefit of CoreCast is that it considerably reduces inter-domain bandwidth, compared to existing P2P or unicast mechanisms. Because of its architectural principles, CoreCast can offer a guaranteed and reliable live streaming service, since ISPs and content providers may easily establish Service Level Agreements. Furthermore, it is an ISP-friendly solution, since subscribers belonging to different ISPs do not exchange traffic as in P2P systems. This is a key benefit as ISPs have recently shown their concerns because of the large amount of traffic generated by P2P applications (e.g. BitTorrent) [22].

In order to evaluate the feasibility of CoreCast we have implemented it in the Linux kernel. Our experiments show that it introduces negligible processing overhead when compared to traditional unicast forwarding. Further, the state kept in the routers is in the order of a few kilobytes, and grows linearly with the number of streamed channels, while it is independent of the amount of subscribers. Using an analytical model, we show that the inter-domain traffic generated by CoreCast is lower than that generated by P2P live streaming applications for typical traffic parameters. We support the model with an analysis of the traffic generated by some of the most popular P2P live streaming applications, PPLive and TVAnts, during two very popular events. The traffic was captured in four different countries: Japan, France, Spain and Romania.

CoreCast was first introduced in [23], and in short, the main new contributions of this paper are: *(i)* the CoreCast architecture, *(ii)* analytical and measurement based comparison between CoreCast and P2P bandwidth requirements, *(iii)* an implementation for the Linux kernel and *(iv)* processing overhead analysis. The CoreCast implementation, along with a CoreCast packet dissector patch for the popular open source WireShark network analyzer can be found at `http://www.cba.upc.edu/corecast`.

2. LISP Background

This section presents a basic overview of LISP [24, 18]. The reader already familiar with this protocol can safely skip it. As mentioned before, the main

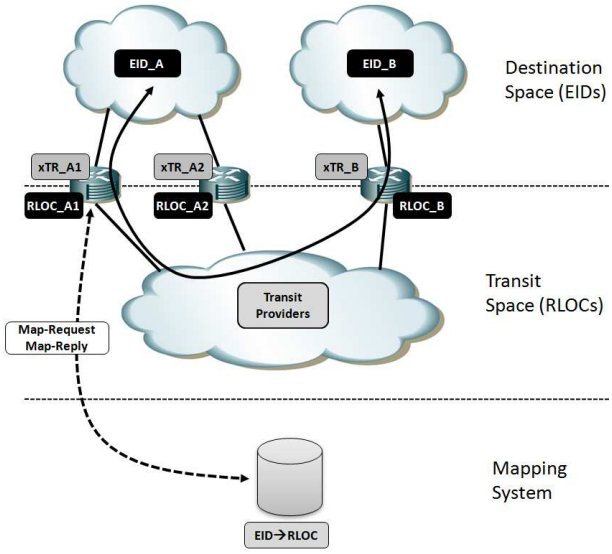


Figure 1: LISP architecture

drivers of this proposal are the scalability issues of the current Internet's routing infrastructure. The proposed solution lies in the separation of the address space of end hosts (Destination Space) from that of the transit network (Transit Space). Fig. 1 illustrates this separation. An additional plane, the Mapping System is required in order to facilitate the "glue" between the two addressing spaces.

In order to decouple the identifiers of nodes from their location, LISP introduces *Routing LOCators* (RLOCs), and *Endpoint IDentifiers* (EIDs). RLOCs are addresses used by network elements in the Transit Space, and define where a destination node is to be found in the routing topology. EIDs represent the identity of the node, regardless of its location, and are used as addresses in the Destination Space. In order to be incrementally deployable, and with no changes in end systems, RLOCs and EIDs are both using the IP address space.

When a packet is sent in the LISP-enabled Internet, it travels within the autonomous system (AS) using currently deployed mechanisms, until it reaches the border router. This router is called the *ingress tunnel router* (ITR) in LISP terminology because it is the ingress point to the tunnel towards the border router of the destination AS, the *egress tunnel router* (ETR). Since a border router can implement both functions, we will use the term *tunnel router* (xTR) for this kind of device. Consider Fig. 1 for example. A host with EID_A wants to send a packet to EID_B. It reaches xTR_{A1}, which takes the destination address (EID_B), looks it up in the mapping system, which returns $RLOC_B$. xTR_{A1} encapsulates the packet in a LISP header, sends it to xTR_B, which decapsulates it and then gets delivered to the destination.

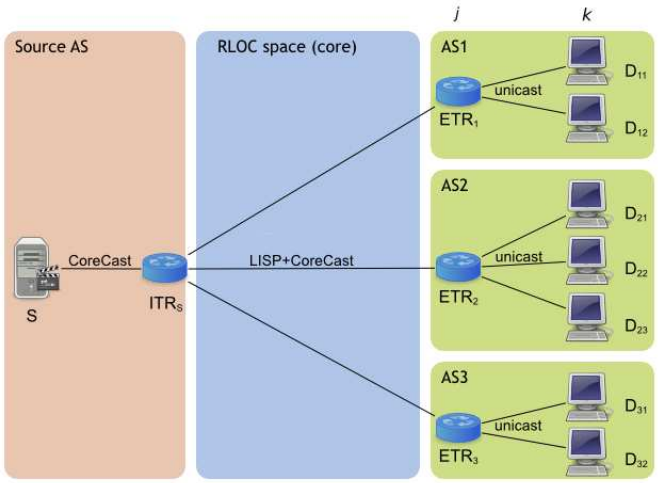


Figure 2: CoreCast architecture

3. The CoreCast Protocol

3.1. Overview

CoreCast is a simple, one-to-many multicast protocol with low deployment cost, incrementally deployable, exploiting features introduced by LISP in order to reduce redundant traffic. To implement CoreCast, a only small number of modifications to the current LISP specification are required.

Consider the following scenario: a broadcaster has a large number of clients (denoted by k) for live streaming content or an IPTV channel. These clients are dispersed over a number of j Autonomous Systems (ASes). When using unicast, the same content has to be sent k times by the source node S to reach all clients. Using CoreCast, the content is sent once to the ITR of the source node (ITR_S) along with a list of destinations, which in turn sends one copy to each of the involved ETRs ($ETR_1 \ldots ETR_j$), and these ETRs send out one copy to each destination node inside of their respective ASes. See Fig. 2 for an example of a small CoreCast streaming deployment with 7 clients distributed in 3 ASes. On each inter-domain link, the multimedia data travels only once, so in the example case from the figure, the core will see it only 3 times instead of 7.

Note that authentication, authorization, and accounting (AAA) is not part of the CoreCast protocol, the source node S should handle that at the application layer, using a framework of its choice. CoreCast is only concerned with the efficient transmission of the actual multimedia streams.

3.2. CoreCast Packet Types

CoreCast differentiates between two types of packets: *payload packets*, which carry the multimedia payload and *header packets*:

Payload Packets Contain the *hash* of the payload, which is the identifier used later by the caching and demultiplexing points, the *length* of the payload and the *payload* data itself.

Header Packets The CoreCast protocol data unit (content after the IP header) of this type of packets contains the *destination EID* of the client, and the *hash* that identifies the payload that has to be sent to it.

3.3. Source Node

As previously mentioned, S implements AAA in a separate, out-of-band framework (e.g., HTTPS web login, or a custom protocol over TCP), and performs the appropriate mechanism for each connecting client. For each multimedia stream (channel) that it is broadcasting, S maintains a data structure called `chanDstList` which contains a list of EIDs that are currently receiving that stream. After a successful connection, a new client is added to the requested channel's destination list. A client uses the same framework to signal departure from the channel, triggering removal from `chanDstList`. Departure may be caused by switching to a different channel, either on S or a different source, or simply stopping to watch the stream. In order to account for clients not signaling departure, but no longer using the stream (due to system crash, network error, etc.), the AAA framework can implement a heartbeat protocol as well. Clients would be required to send periodically a control packet to show continued interest in the stream. A large interval of several minutes would take care of removing dead clients, while addig very little overhead.

Each domain reserves an EID space with local scope only for CoreCast streams, where an EID designates a channel. This is required because the first demultiplexing point is located on a busy border router, which should forward regular packets at line speed, and only work with CoreCast packets in the slow path. Since the destination EID is always examined, the reserved EID range is used to trigger examination of the CoreCast protocol fields, and leave all other packets in the fast path.

For each channel, S divides multimedia data into chunks of payload in such a way, that packet size is maximized, but the MTU is not exceeded on the path to the destination. For each chunk, it first sends a packet with the payload, setting the destination address in the IP header to the channel's reserved EID. After the payload packet is sent, S iterates through the `chanDstList`, and sends a header packet for each of the destinations listed. The process is then repeated at regular time intervals, determined by the bandwidth required for the stream. For example, sending a 384 Kbps stream, broken down into fixed sized payloads of 1200 bytes would require a payload packet to be sent every 25 ms. Optionally, the source could combine the list of destinations and instead of a header packet for each destination, send up to MTU sized packets with the list

of destinations to the ITR. However, this would add complexity that is unlikely to be implementable in hardware at line speed, thereby actually reducing the number of clients supported by the router.

The above mechanism sets an upper limit of how many destinations CoreCast can support. This limit is function of the transmission medium's capacity, the bandwidth required by the stream, and payload size:

$$MaxClients_{CC} \simeq \frac{C \cdot T}{8 \cdot H_{CC}} = \frac{C}{BW} \cdot \frac{P}{H} \tag{1}$$

where C is line rate in bits per second, T is time between payloads in seconds, P is payload packet size in bytes, H is header packet size in bytes, and BW the bandwidth required by the stream in bits per second. The same limit in case of unicast transmission is:

$$MaxClients_{UC} = \frac{C}{BW} \tag{2}$$

CoreCast's gain in terms of maximum number of supported clients depends on the ratio between the payload size and the header size: P/H. Using these formulae the content provider can do capacity planning based on the expected number of clients, and add network and/or server capacity as needed.

For our example in Fig. 2, the source would send one CoreCast payload packet to the ITR followed by 7 CoreCast header packets for each PDU that has to be transmitted, placing destination EIDs D_{ij} into the header.

When requesting a channel, the client software informs S using the previously mentioned out-of-band AAA protocol if its domain supports CoreCast. For domains without a CoreCast capable ETR, the source will send regular unicast packets.

3.4. Ingress Tunnel Router

The ingress tunnel router is the first of the two CoreCast stream demultiplexing points. In order to process CoreCast packets, it maintains a `payloadBuffer` data structure, which reserves one record entry for each supported channel. The entry contains the hash of the payload, the payload itself, and a pointer to a `servedRLOC` buffer. This buffer is created for each payload on arrival, and tracks the locators of the ETRs which already received payload data (see Fig. 3). When all clients from the domain of the locator have been served with the current payload, the buffer is freed. To avoid keeping too much state, the ITR keeps only one payload for each channel in the `payloadBuffer`.

Algorithm 1 shows the packet processing mechanism in the ingress tunnel router. When a *payload* packet is received, the ITR identifies the channel using the reserved destination EID in the IP header, checks if the hash in the CoreCast header matches the locally calculated hash of the payload, and then overwrites the old payload record in the `payloadBuffer`, also destroying the associated `servedRLOC` buffer and allocating a new one. No packet is sent upon receving a payload packet.

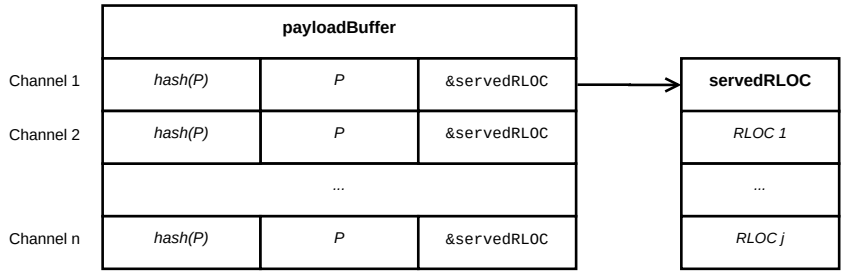


Figure 3: ITR Memory Structures. There is one `payloadBuffer` entry for each channel; a `servedRLOC` buffer is allocated for each payload on arrival and released after all clients have been served

For each CoreCast *header* packet, the ITR extracts the client EID and the payload hash from the CoreCast header and checks for the existence of the associated payload data in the `payloadBuffer`. Then, it looks up the locator of the client EID, which is an operation already provided by LISP (see § 2). If the locator is listed in the `servedRLOC` buffer associated to the payload, the ITR forwards the header to the ETR, doing the usual LISP encapsulation. In the case no payload was yet sent to a particular locator, a payload packet is generated before forwarding the header packet.

Algorithm 1 ITR Demultiplexing

```
if type = Payload then
  /* Payload packet */
  if hash = hash(payload) then
    Store to payloadBuffer
  else
    Drop
  end if
else if type = Header then
  /* Header only */
  if hash ∈ payloadBuffer then
    if RLOC ∈ servedRLOC then
      Send header to RLOC
    else
      Send payload to RLOC
      Send header to RLOC
      Store RLOC in servedRLOC
    end if
  else
    Drop
  end if
end if
```

Take the example in Fig. 2. The ITR would store the payload, then send

one copy to ETR_1 followed by headers for D_{11} and D_{12}, another copy to ETR_2 and headers to destinations in AS_2, and finally a copy to ETR_3 and headers to destinations in AS_3.

In the case when the AS of the source node does not offer CoreCast protocol support in the ITR, but allows LISP encapsulated packets to be sent from the inside, the functions of the ITR could be performed by S itself: it could send one LISP encapsulated payload to each locator and LISP encapsulated headers for the destinations. For this, it would need access to the LISP mapping system used by tunnel routers, which is allowed by the LISP specification.

Note that CoreCast is a unidirectional *live* streaming protocol; as such there is no feedback from the clients about lost packets and no retransmissions are being made. The expected deployment of the source node close to the ITR reduces the packet loss probability to a minimum. On the other hand, the path between the ITR and ETRs is more prone to losses, and loss of single payload packet will affect all users of the destination AS. To alleviete this problem, the ITR can be configured to interleave an additional payload packet after every n header packet going to the same ETR. That way the loss of a payload packet affects a reduced set of clients.

3.5. Egress Tunnel Routers

The ETR is the second and last demultiplexing point, and it works similar to the ITR, storing the payload for each received stream. But instead of forwarding headers, it has to expand them to regular unicast packets that get delivered within the AS to their final destinations, by retrieving and adding the corresponding payload data from the `payloadBuffer`. The packet processing mechanism in the ETR is presented in Algorithm 2.

Algorithm 2 ETR Demultiplexing

```
if type = Payload then
  /* Payload packet */
  if hash = hash(payload) then
    Store to payloadBuffer
  else
    Drop
  end if
else if type = Header then
  /* Header only */
  if hash ∈ payloadBuffer then
    Send reconstructed unicast packet to destination
  else
    Drop
  end if
end if
```

To complete our example from Fig. 2, ETR_2 stores its copy of the payload, then for the following headers sends a unicast IP packet with the payload to D_{21}, D_{22}, and D_{23}.

Note that the demultiplexing must not necessarily use unicast packets towards the final destinations. The domain may decide to use IP multicast internally instead, especially if live streaming services prove popular. IP multicast is not widely used in inter-domain scenarios, but it has been succesfully used for distributing TV channels inside an ISP [25].

Both the ITR and the ETR have to keep some state for CoreCast operations. In the case of the ETR the only CoreCast specific data structure is the `payloadBuffer`, the size of which depends only on the number supported channels. For example, supporting up to 1,000 channels, with a maximum payload size of 1200 bytes requires just over 1 MB of memory. The ITR adds one more data structure per channel, `servedRLOC`, to store the locator that received already the current payload. Each entry in this structure is either 4 bytes (IPv4 core) or 16 bytes (IPv6 core). For our example of 1,000 channels, clients distributed in up to 1,000 ASes and an IPv6 core would add an extra 16 MB to the memory requirements. We believe this is an acceptable amount of memory for a border router.

4. Analytical Evaluation

Along with architectural benefits, CoreCast lowers inter-domain bandwidth requirements for ISPs providing live streaming services to their subscribers. Existing methods to deliver these services include IP multicast, unicast and application layer P2P systems. To evaluate our proposal, we take P2P systems as base for comparison, because unicast is the most ineffiecient solution and multicast is difficult to deploy in inter-domain scenarios [3]. In contrast, P2P live streaming systems are in wide use today.

In order to compare the data rate of CoreCast with that of current P2P applications we developed a formula for the former and an estimation method for the latter. Furthermore, we focus the analysis on inter-domain traffic, mainly because it is more expensive for ISPs. In fact this is a key issue and recently ISPs have shown their concerns because of the large amount of inter-domain traffic generated by P2P applications [22].

In the case of CoreCast, we consider inter-domain the traffic sent from the ITR to the ETRs and intra-domain the one flowing from all ETRs to all individual clients. In the following, the terms *inter-domain traffic* and *transit traffic* will be used interchangeably; also, *intra-domain traffic* and *local traffic* is to be interpreted as the same.

The total transit traffic generated by sending one payload is as follows:

$$TT_{CC} = (H_L + P_{CC}) \cdot j + (H_L + H_{CC}) \cdot k, \tag{3}$$

where H_L is the size of a LISP header, P_{CC} is the CoreCast PDU size for a payload packet, H_{CC} is the size of a header packet, and j and k are the number of RLOCs and clients, respectively, as described in § 3. Thus the corresponding bandwith is

$$BW_{transit} = \frac{TT_{CC}}{T} \tag{4}$$

On the other hand the total local traffic is given by:

$$LT_{CC} = (H_I + D) \cdot k, \tag{5}$$

where H_I is the size of an IP header, and D is the size of one multimedia chunk.

$$BW_{local} = \frac{LT_{CC}}{T} \tag{6}$$

Due to the way P2P streaming applications work, the total traffic can only be determined if each participating node sends its statistics to a central server or we can capture at each node. The first approach is only available to the owners of the application software, while the second is unfeasible due to its scale.

Instead, we will show that CoreCast interdomain traffic is very close to the theoretical minimum, thus it is unlikely that any P2P network configuration produces less traffic of this type.

The expression for inter-domain traffic for a single payload in CoreCast has been computed in Equation 3. We will compute the inter-domain traffic for a single payload in a P2P network of the same size. Such a network will contain $k+1$ nodes (including the broadcaster) and $j+1$ ASs, since the broadcaster is assumed to be an independent AS.

If we model the P2P network as a graph, in order to cast the payload to all nodes, the graph must be connected. Some of these connections between nodes would be inter-domain connections. Let us denote by i the number of inter-domain arcs in the graph. Since all ASs must be connected, the minimum number of inter-domain arcs is j, so $i \geqslant j$.

The total transit traffic for a P2P (under LISP) can be written in terms of the i arcs as

$$TT_{P2P} = (H_L + P_{CC}) \cdot i. \tag{7}$$

As far as the transit traffic is concerned, CoreCast is more efficient than the P2P network whenever $TT_{CC} \leqslant TT_{P2P}$. Substituting by the corresponding expressions, we obtain the following equivalent formulation:

$$\frac{H_L + H_{CC}}{H_L + P_{CC}} \leqslant \frac{i-j}{k}. \tag{8}$$

The difference $i - j$ is the total number of inter-domain arcs in the P2P that are not really essential. To ease the interpretation of the results we rewrite this difference in terms of k, so that $i - j = \alpha \cdot k$. The parameter $\alpha \in [0, k-1]$ has a straighforward interpretation: it is the average number of non-essential inter-domain connections per node. Using α in Equation 8, CoreCast produces less inter-domain traffic than a P2P when

$$\alpha \geqslant \frac{H_L + H_{CC}}{H_L + P_{CC}}. \tag{9}$$

The parameter α depends on the particular P2P system and is difficult to estimate. However for reasonable values of header and payload sizes, only for very

small values of α is the inequality in Equation 9 not satisfied. For instance, the LISP header is 20 bytes long ($H_L = 20$), the CoreCast header has 60 bytes ($H_{CC} = 60$), and a popular payload size used by live streaming applications is 1200 bytes ($P_{CC} = 1200$, see [10]). In this case we have that $\alpha \geqslant 0.0656$. Any P2P system having more than 0.0656 non-essential inter-domain arcs per node would be less efficient than CoreCast in terms of bandwidth for a 1200 byte payload size. Even if we decrease the payload size to, e.g., 400 bytes, we get a very small $\alpha = 0.16$ lower bound, from which CoreCast is more efficient.

In the next section we present a measurement-based analysis of popular P2P applications from several vantage points, that help us providing plausible values for the parameters of Equation 9.

5. Experimental Evaluation

In this section we first describe the datasets collected for our evaluation and secondly we compare P2P live streaming bandwidth requirements to that of CoreCast.

5.1. Experimental Datasets

To obtain the experimental datasets needed for the evaluation we performed several measurement experiments using different P2P live streaming systems. We passively measured the network traffic of the application and saved the data for offline analysis. In particular, we collected three different datasets from a wide variety of well-distributed vantage points at two different live events:

Set 1: This dataset consists of traces collected at multiple different capture points situated in France and Japan, to obtain a global view of the P2P network. We passively measured the network traffic of TVAnts, which is a very popular P2P live streaming application.

Our measurement testbed is divided into two parts: one in France, the other in Japan. Since a large community of users employ P2P live streaming to watch live soccer games, we performed our measurement experiment during such kind of events that also exhibit a strong interest to be watched live. The measured event was a qualifying soccer match for the Olympic tournament with China vs. Vietnam on August 23, 2007.

During the experiment, all the PCs were running the TVAnts P2P live streaming application and WinDump to collect the packets. All the seven traces we collected have the same duration of 2:45h. This duration is slightly larger than a soccer game (105 minutes) because we wanted to capture all the events that may occur at the beginning or the end of the game. The traces show very similar properties: their average size and number of packets are 2.5 GB and 3 millions of packets respectively. All the traces count approximately 1,850 distinct IPs. More than 95% of the IPs of a trace are also present in the other traces. This suggests that we captured the full population of peers in the measured event. A detailed description of this dataset can be found in [26]. Throughout the paper we refer to the individual traces in this dataset as FR1, FR2, FR3, FR4, JP1, JP2 and JP3.

Set 2: To obtain this dataset we performed a measurement experiment using the PPLive application, developed and popular especially in China. The event monitored was the memorial service of Michael Jackson, on July 7, 2009, during which we passively measured the network traffic of the application and saved the data for offline analysis in two different vantage points: our network lab at the Technical University of Catalonia in Barcelona, Spain and a home broadband network in Cluj-Napoca, Romania. The traces are denoted with ES and RO respectively in the following. This was a very popular live event, to the extent that the channel broadcasting it was advertized by the PPLive client to the users with popups, and attracted a large number of users.

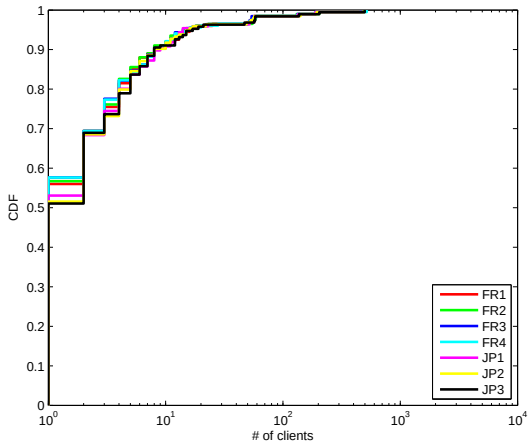
We identified 23,713 and 3,252 unique IP addresses in the RO and ES traces respectively, during the 2:43h and 3:23h timeframe we captured traffic. The RO trace is 40 minutes shorter, because traffic capture started later into the event. We attribute the difference in population to the restrictive firewall of the university, which caused a significant decrease in the number of different IP addresses observed in the trace. Additionally, multimedia traffic in the former is transported predomimantly over UDP, while the latter over TCP. This seems to corroborate the assumption of the firewall, as TCP connection tracking is more reliable on stateful firewalls.

5.2. AS Distribution Analysis

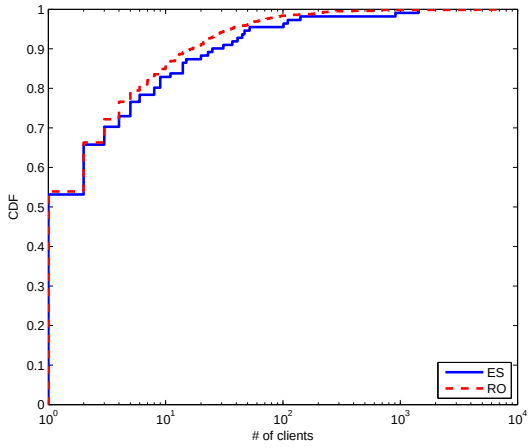
Recalling from Equation 9 CoreCast saves bandwidth compared to existing solutions when these P2P applications use more than 0.16 (α) inter-domain links per node on average. In order to better understand this parameter we study the amount of IPs contacted by each monitored node, and how these IPs are clustered among ASes. They are directly related to the amount inter-domain links used by each node. Besides the α parameter, the efficiency of CoreCast heavily depends on how clients viewing the stream are distributed into ASes. The worst case scenario for CoreCast is when each AS contains a single client, the best case being all the clients clustered in the same AS.

The 7 traces from *Set 1* had very similar characteristics, contacting about 1,850 distinct IP addresses, distributed among approximately 200 ASes. This results in almost 10 clients per AS on average, but the distribution of clients is not uniform. Fig 4(a) plots the cumulative distribution of the number of clients in ASes. We can see that for all data sets we have just over 50% of the ASes with a single client. These ASes would neither benefit from Corecast, nor be negatively impacted by its deployment. But there are a few domains containing a large number of clients, the largest population exceeding 500 viewers. Using CoreCast in these domains would result in a 500-fold decrease of expensive inter-domain traffic.

For the RO trace in *Set 2*, the 23,713 clients were distributed in 1291 ASes according to the CDF in Fig. 4(b). It is worth noting that the first three ASes (all of them from China) captured 12,160 clients, which is more than half the total number of clients. These autonomous systems would have benefited the most from using CoreCast. In the ES trace, clients cluster even more in the top ASes: the top two (which are the same as for RO) contain 72% of all viewers.



(a) Dataset 1



(b) Dataset 2

Figure 4: Number of clients per autonomous system

According to these results, P2P clients use tens of inter-domain links on average, suggesting that $\alpha \gg 0.16$, a conservative lower bound for CoreCast's efficiency computed in § 4. Particularly the ASes that include a very large amount of clients would strongly benefit from the use of CoreCast.

5.3. Bandwidth Comparison

In this subsection we aim to estimate the total amount of inter-domain traffic saved with respect to P2P live streaming systems, based on actual measurement

data. We also provide a comparison to unicast, using today's common parameters.

Typical servers today are equipped with 1 Gbps network cards, with some high end models supporting 10 Gbps interfaces. A common data rate for multimedia streams is 384 kbps. The maximum number of supported clients by CoreCast and unicast are given by Equations 1 and 2. A server with a 10 Gbps capacity, broadcasting a single channel, considering $P = H_L + P_{CC} = 20 + 1200$, $H = H_L + H_{CC} = 20 + 60 = 80$, will support $397,135$ clients when using CoreCast. Unicast would limit that number to only $26,041$, an order of magnitude smaller. Note that large content providers could horizontally scale to millions of viewers using several servers in a datacenter with 40 Gbps or 100 Gbps connectivity.

When comparing CoreCast bandwidth consumption to that of P2P applications, we need the average payload size, inter-packet arrival times and length of the broadcast to calculate the total CoreCast traffic. But to get the total P2P traffic, we either have to capture at each participating node, or have the client software on each node reporting statistics to a centralized measurement host. Since the first method is unfeasible due to its scale, and the second is only available to the owners of the P2P application, we need an alternative approach. To this end, we considered the *Set 1* data, captured using the TVAnts client software, and estimated the total traffic of a hypothetical, but plausible P2P system. To build this system, we assume that all peers ($c_{1...k}$) have a similar traffic profile. By traffic profile we refer to how the amount of bytes downloaded from the peers is distributed among them. This assumption seems reasonable when considering Fig. 5, because the cumulative distribution functions for the 7 capture points almost completely overlap.

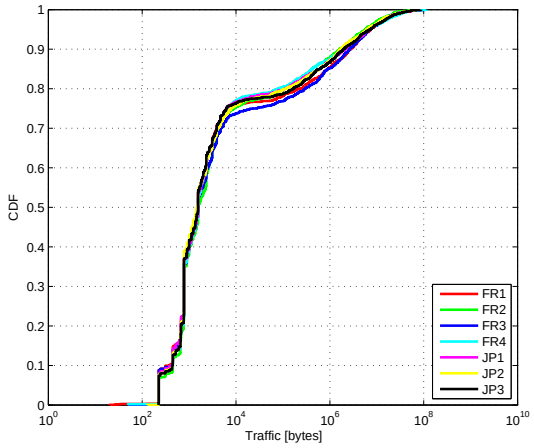


Figure 5: Cumulative distribution of the amount of traffic exchanged with peers from the 7 monitoring points of *Set 1*: they are almost identical

To estimate the total traffic, we determine the video download traffic (in bytes) of the monitored client in the trace from all contacted peers. We then create a two-dimensional traffic matrix with both columns and rows represented by peers c_i, and fill the row for c_1 with the data determined from the trace. Rows $c_2 \dots c_k$ are random permutations of the same values. Iterating through each element c_{ij} of the matrix, knowing the IP addresses of clients we determine if the communications was transit or local, and sum the amount of bytes to corresponding counter.

Trace	**IPs**	**ASes**	**TT** [GB]	**LT** [GB]
FR1	1855	209	1940	252
FR2	1865	204	1472	195
FR3	1769	201	1761	228
FR4	1888	207	1974	268
JP1	1856	201	1754	226
JP2	1863	197	1645	215
JP3	1878	194	1840	248
CoreCast	1853	202	131	895

Table 1: Estimated transit (TT) and local (LT) traffic of CoreCast, and a hypothetical P2P network, based on *Set 1*

Table 1 shows the results for dataset 1. Column one specifies the trace collection point (except last row, representing CoreCast), while columns two and three describe the number of unique IP addresses and ASes per trace respectively. Columns TT and LT show the transit and local traffic, calculated with the algorithm described in the previous paragraph. It is worth noting here that we are not asserting that Table 1 presents an estimation of the total traffic produced by the applications running on the monitored networks. Instead, the values can be thought as the traffic generated by a representative P2P application, which is reasonable according to Fig. 5. Using formulae 4 and 6 we also calculated the equivalent CoreCast traffic (last row), the results suggesting remarkable savings in terms of transit traffic with respect to P2P systems. On the other hand, local traffic volume increases when CoreCast is used. Since local traffic does not incur the same high costs as transit traffic, this should not be a problem for the destination domain. Moreover, using multicast internally would bring traffic to the same level as the transit traffic, resulting in an improvement over P2P even for local traffic. Note that P2P cannot be optimized by using network layer multicast, while it has been shown [25] that multicast works well inside autonomous systems.

Fig. 6 shows the cumulative distribution of the previously estimated P2P and CoreCast inter-domain traffic per autonomous system. As our analysis in § 5.2 showed, approximately 50% of domains have just one client. We can see in the figure that for these domains the bandwidth requirements of the P2P application and that of CoreCast are similar, the latter being slightly less efficient. Since CoreCast sends a separate payload packet and then a separate header packet,

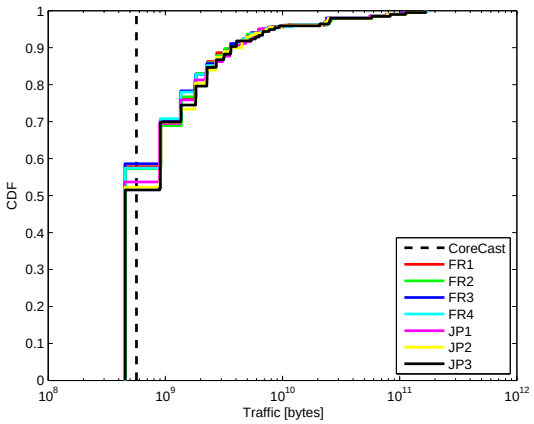


Figure 6: Cumulative distribution of the video download traffic per AS. For the slightly over 50% of ASes with only one peer CoreCast has a small overhead, but performs better in all other cases

this overhead is to be expected.

The rest of the domains however save considerable amounts of traffic with CoreCast, and 10% of the them, in the case of this hypothetical P2P network, consume over an order of magnitude less of bandwidth (best case scenario is a 300-fold deacrese). Considering that 10-fold increases in interconnect technologies are coming at a deacreasing rate (3 years from 100Mbps to 1Gbps, 4 years for 1Gbps to 10Gbps and only draft standard 100Gbps in 8 years), this is a very important improvement.

In summary, the reversal of preference for intra-domain traffic in the case of CoreCast is beneficial for ISPs, because inter-domain traffic is more expensive. Additionally, while not all domains would benefit from CoreCast, 10% of domains could have reduced traffic with more than an order of magnitude for our datasets.

6. Implementation

Because CoreCast introduces additional operations in the packet forwarding path on supported routers, it is important to quantify the processing overhead caused by these operations, as excessive overhead would discourage CoreCast adoption. In the following we quantify the increase in CPU load caused by CoreCast, compared to unicast forwarding.

6.1. Testbed

In order to test the processing overhead of the CoreCast protocol, we have implemented it as a Linux 2.6 loadable kernel module. This kernel module

Figure 7: Testbed. S generates CoreCast traffic, ITR processes the packets, and CAP captures the resulting traffic

creates a Netfilter hook, which receives all IP packets before they are routed and applies Algorithm 1 to them.

Due to the hashing, hash lookup and EID-to-RLOC lookup, CoreCast incurs an overhead compared to simple unicast packet forwarding. To quantify this overhead, we set up a small testbed with 3 machines (see Fig. 7): one for generating packets (S), one for acting as a CoreCast capable ITR and one to count received packets (CAP). All machines were the same hardware configuration: 3GHz Pentium 4 processor, 1GB of RAM and 2 on-board Intel Gigabit Network controllers. On the software side, they were running Debian GNU/Linux, with a 2.6.26 version of the Linux kernel. All machines were running only the bare minimum of the services necessary to run the experiments.

The first machine, S, was running a CoreCast traffic generator, which we implemented using raw sockets. For our experiments, we used the generator to send a 384 Kbps CoreCast stream with 1200 byte payloads every 25 ms, each payload packet being followed by header packets corresponding to a client list. To have realistic load on the ITR, in terms of distribution of clients among ASes, we used the client list from a trace with 1429 clients. The total duration of the stream was set to 30 seconds. The traffic generator also implements a scaling factor, which we use to gradually increase the load on the ITR. The scaling factor is a multiplier, specifying the up-scaling ratio for the total number of clients. The generator sends header packets to the number of clients present in the client list, multiplied by the scaling factor, using the same AS distribution.

We performed tests with scaling factors ranging from 1 (simulating 1429 clients) to 7 (10003 clients), repeating each test 20 times and averaging the results. To avoid artifacts that may be caused by caching or unexpected interface buffering behavior, the tests were interleaved as follows: we did one test with scaling factor 1 first, followed by 7, 2, 6, 3, 5, 4, then repeated this process 20 times.

On the ITR we used the `sar` command to monitor the CPU usage during each 30 second test. For the baseline unicast forwarding the CoreCast kernel module was not loaded, and packets were routed without any kind of packet mangling or filtering. The tests were repeated with the kernel module loaded, to emulate the CoreCast forwarding model, for a total of 280 experiments. Since LISP is still not implemented for Linux, we used a static EID-to-RLOC mapping

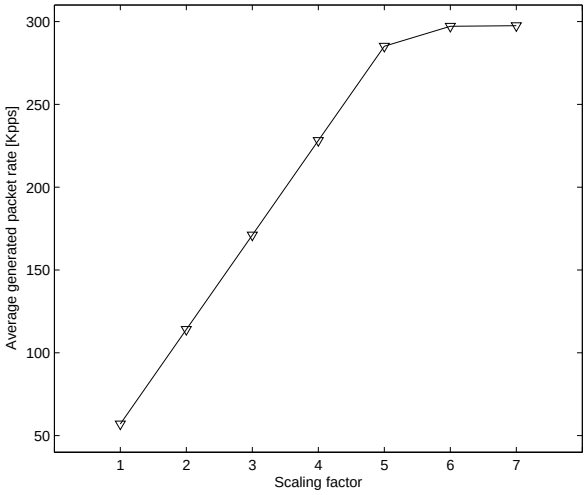


Figure 8: Average generated packet rate on S

cache, which was built based on the AS distribution from the Veetle trace.

Finally, we captured all CoreCast packets on CAP, to determine packet loss.

6.2. Experimental results

Using the algorithm presented in the previous subsection, we obtained increasingly larger values for the number of packets sent per second by S when increasing the scaling factor (see Fig. 8). The increase was linear up to a scaling factor of four (equivalent of 5716 clients), at which point the hardware limitations of S caused a slowdown of the increase in packet rate. Figure 9 shows the evolution of the CPU usage on the ITR for both Unicast and CoreCast forwarding, with progressively increasing the scaling factor. The CPU usage shows a strong correlation with the packet rate in both cases.

As expected, CoreCast incurs higher CPU usage than simple unicast packet forwarding. However, the increase is about 52% on average. To find out what causes this significant increase, we selectively disabled the hashing, hash lookup and EID-to-RLOC lookup functions of the ITR CoreCast kernel module. SHA1 hashing surprisingly did not incur any measurable overhead, compared to the unicast forwarding. The Linux kernel includes several highly optimized cryptographic functions, among them SHA1, which explaines why it performed so well. Please not that other systems include similar optimizations, and on a production CoreCast capable router this hash could be implemented in hardware, basically eliminating the overhead caused by hashing.

We determined the increase in packet processing load was cased by EID-to-RLOC lookups. Our module implements a naive linear search algorithm,

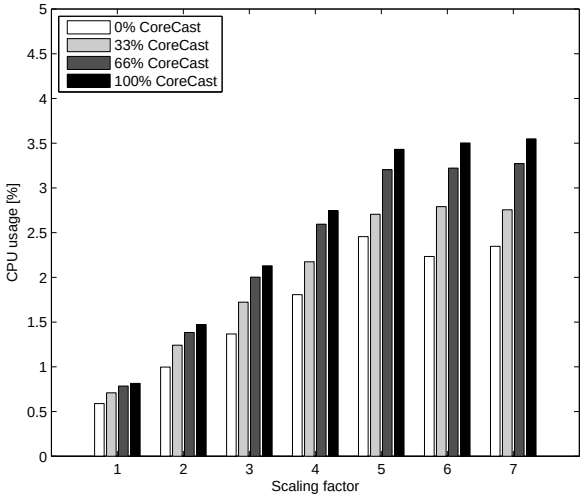


Figure 9: CPU usage on *ITR*

which is inefficient for this task. However, a production router would use either expensive Ternary Content-Accessable Memory memory for this purpose, or implement highly optimized radix tries for lookups. Moreover, EID-to-RLOC lookups are a mandatory function of any LISP router, independently of CoreCast, which means that our proposal adds virtually no measurable overhead to packet processing.

7. Related Work

CoreCast is a network-layer protocol that offers reliable live streaming and that works on top of LISP. In today's Internet, live events are transmitted over the Internet using either traditional unicast or large P2P networks such as [4, 6, 8, 7, 9] (see a survey in [5]). These P2P systems use complex application-layer overlay networks and their performance has been largely analyzed by the research community [10, 11, 12, 13, 14, 15, 16, 17]. The main findings are that they cannot offer an acceptable quality of experience to all the users all the time [17]. This is mainly because the limited upload capacity of the peers and churn, users joining/leaving the overlay at any time. Consequently, P2P streaming systems cannot be considered as the long-term solution for the transmission of live multimedia over the Internet. CoreCast is a pure network-layer approach built of top of LISP that can offer reliable transmission of live multimedia content. Because of its architecture, CoreCast allows content providers and ISPs to establish Service Level Agreements.

The second approach to transmit live content over the Internet is traditional unicast. Since unicast suffers from severe scalability issues, it is often used in combination with Content Delivery Networks (CDN) such as [27, 28, 29]. These networks deploy geographically distributed servers in the Internet, replicating the multimedia content. This content is usually distributed using P2P networks [30]. Finally, the users receive the content from the closest copy. This approach can be considered as a hybrid solution, where first the content is distributed using an overlay (application-layer), and then the users access this content directly using unicast (network-layer). Within this context, CoreCast can also operate in conjunction with CDNs, helping to distribute the content from the central servers to the distributed delivery servers, significantly reducing the overall bandwidth consumption, and hence, decreasing the CDN's operational costs.

CoreCast can also be seen as an explicit IP multicast solution [2]. The most similar approach to CoreCast is Xcast [31]. This protocol operates as follows: data is sent to a special destination address and all routers in the data path have to know how to handle that special multicast address to reach subscribed clients. Xcast explicitly specifies all destinations in its protocol header. This significantly limits the use of Xcast to small groups, thus it cannot be used for broadcasting popular events.

Finally, it is worth to mention that LISP also incorporates a multicast protocol [32], which extends traditional IP multicast [2] to LISP. It requires support at the core routers as well, in addition to the border routers. As IP multicast, this leads to very high deployment costs [3], and that's the main reason why IP multicast has not found wide-spread deployment. In contrast, CoreCast is intended to be shipped with LISP (not LISP multicast), and LISP requires upgrading only the domain's border routers.

8. Conclusions and Future Work

In this paper we have presented CoreCast, a reliable network layer live streaming protocol. The main improvements over existing solutions are architectural: it is a simple protocol that provides one-to-many multicast in a future core/edge separated Internet. CoreCast circumvents the high deployment cost of previous network layer approaches because it works on top of LISP. Note that there is considerable consensus in the routing research community that the current routing system must be upgraded, and splitting the current address space is seen as the most scalable solution [19]. CoreCast has been implemented to operate on LISP, however it could operate on other proposals [20, 21] with minor modifications. Therefore, CoreCast's low deployment cost is not limited to LISP, but to any core/edge separation protocol deployed in the Internet.

The CoreCast architecture enables network planning for ISPs, because they can estimate the resources necessary for supporting a given number of streams. This further allows service level agreements between content providers and service providers, ensuring proper delivery of the content for the former and opening a new business opportunity to the latter.

Another contribution of the paper is the Linux implementation of CoreCast. Using this implementation we saw a 52% increase in CPU usage when comparing unicast and CoreCast forwarding. The increase was determined to be caused by the EID-to-RLOC lookup function, which is part if the LISP protocol, and will be offered by future LISP routers. This operation is easily optimizable in hardware and will likely have negligible overhead in production equipment.

Additionally, our analytical model, combined with measurement data, suggests that compared to reasonable P2P live streaming systems, CoreCast produces significantly less inter-domain traffic. The gains depend on the distribution of clients among autonomous systems, with as much as 300-fold decrease observed in our datasets.

As future work we plan implementing CoreCast in Cisco's existing LISP testbed (lisp4.net and lisp6.net).

Acknowledgements

This work has been partially supported by the Department of Innovation, Universities and Enterprise of the Generalitat of Catalonia under scholarship number 2006FI-00935 and under grant 2009SGR-1140.

References

[1] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, S. Tewari, Will IPTV ride the peer-to-peer stream?, IEEE Communications Magazine 45 (6) (2007) 86–92.

[2] S. Deering, Host extensions for IP multicasting, RFC 1112 (Standard) (Aug. 1989).

[3] C. Diot, B. N. Levine, B. Lyles, H. Kassem, D. Balensiefen, Deployment issues for the IP multicast service and architecture, IEEE Network 14 (1) (2000) 78–88.

[4] X. Zhang, J. Liu, B. Li, T.-S. P. Yum, CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming, in: Proceedings of the 24th Conference on Computer Communications (IEEE INFOCOM '05), 2005.

[5] J. Liu, B. Li, J.-Q. Zhang, Adaptive video multicast over the Internet, IEEE Multimedia 10 (1) (2003) 22–33.

[6] Sopcast.
URL `http://www.sopcast.com/`

[7] PPLive.
URL `http://www.pplive.com/`

[8] TVAnts.
URL http://www.tvants.com/

[9] UUSee.
URL http://www.uusee.com/

[10] X. Hei, C. Liang, J. Liang, Y. Liu, K. W. Ross, A measurement study of a large-scale P2P IPTV system, IEEE Transactions on Multimedia 9 (8) (2007) 1672–1687. doi:10.1109/TMM.2007.907451.

[11] X. Hei, Y. Liu, K. W. Ross, Inferring network-wide quality in p2p live streaming systems, IEEE Journal on Selected Areas in Communications 25 (9) (2007) 1640–1654.

[12] S. Ali, A. Mathur, H. Zhang, Measurement of commercial peer-to-peer live video streaming, in: Proceedings of Workshop in Recent Advances in Peer-to-Peer Streaming, 2006.

[13] T. Silverston, O. Fourmaux, Measuring P2P IPTV systems, in: Proceedings of NOSSDAV '07, 2007.

[14] B. Li, S. Xie, G. Y. Keung, J. Liu, I. Stoica, H. Zhang, X. Zhang, An empirical study of the Coolstreaming+ system, IEEE Journal on Selected Areas in Communications 25 (9) (2007) 1627–1639.

[15] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, X. Zhang, Inside the new coolstreaming: Principles, measurements and performance implications, in: Proceedings of the 27th Conference on Computer Communications (IEEE INFOCOM '08), 2008, pp. 1031–1039.

[16] T. Silverston, O. Fourmaux, A. Botta, A. Dainotti, A. Pescapé, G. Ventre, K. Salamatian, Traffic analysis of peer-to-peer IPTV communities, Elsevier Computer Networks 53 (4) (2009) 470–484. doi:http://dx.doi.org/10.1016/j.comnet.2008.09.024.

[17] C. Wu, B. Li, S. Zhao, Diagnosing network-wide P2P live streaming inefficiencies, in: Proceedings of IEEE INFOCOM '09 Mini-conference, 2009.

[18] D. Farinacci, V. Fuller, D. Meyer, D. Lewis, Locator/ID Separation Protocol (LISP), draft-ietf-lisp-05, work in progress (Sep. 2009).

[19] D. Meyer, L. Zhang, K. Fall, Report from the IAB Workshop on Routing and Addressing, RFC 4984 (Informational) (Sep. 2007).

[20] C. Vogt, Six/one router: A scalable and backwards compatible solution for provider-independent addressing, in: Proceedings of the 3rd International Workshop on Mobility in the Evolving Internet Architecture (MobiArch '08), 2008.

[21] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang, L. Zhang, Towards a new Internet routing architecture: Arguments for separating edges from transit core, in: 7th ACM Workshop on Hot Topics in Networks (HotNets '08), 2008.

[22] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, A. Silberschatz, P4P: Provider portal for applications, in: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, 2008.

[23] L. Jakab, A. Cabellos-Aparicio, J. Domingo-Pascual, CoreCast: Efficient live streaming in the core-edge separated Internet, in: ACM SIGCOMM '09 poster session, 2009.

[24] D. Meyer, The locator identifier separation protocol (LISP), The Internet Protocol Journal 11 (1) (2008) 23–36.

[25] M. Cha, P. Rodriguez, S. Moon, J. Crawcroft, On next-generation telco managed P2P TV architectures, in: Proceedings of the 7th International Workshop on Peer-to-Peer Systems (IPTPS '08), 2008.

[26] T. Silverston, Peer-to-peer video live streaming: Measurement experiments and traffic analysis, Ph.D. thesis, Université Pierre et Marie Curie (Sep. 2009).

[27] Akamai Technologies.
URL `http://www.akamai.com/`

[28] EdgeCast Networks.
URL `http://www.edgecast.com/`

[29] LimeLight Networks.
URL `http://www.limelightnetworks.com/`

[30] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM Computing Surveys 36 (4) (2004) 335–371.

[31] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, Explicit Multicast (Xcast) Concepts and Options, RFC 5058 (Experimental) (Nov. 2007).

[32] D. Farinacci, D. Meyer, J. Zwiebel, S. Venaas, LISP for Multicast Environments, draft-ietf-lisp-multicast-02, work in progress (Sep. 2009).