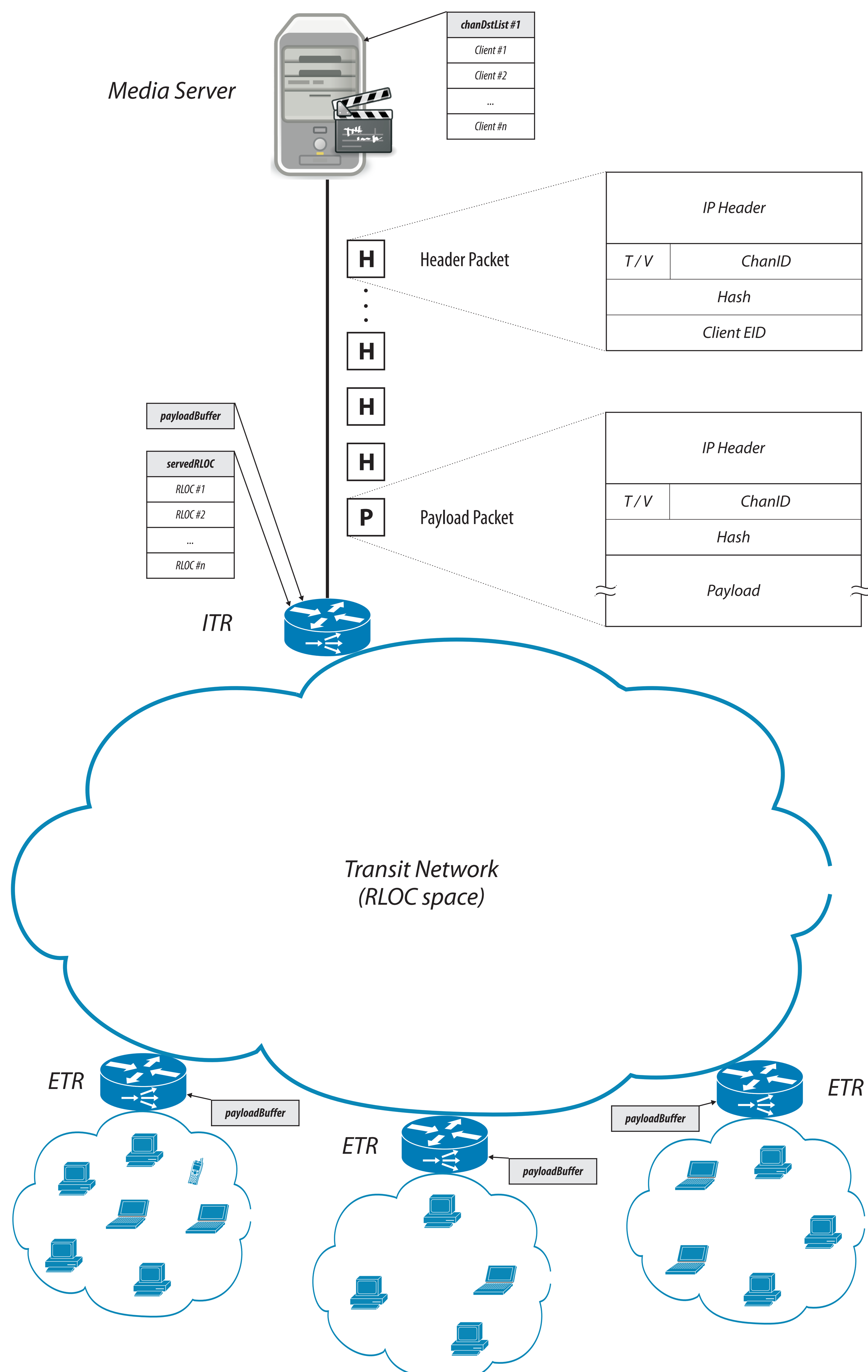
CoreCast | Efficient Live Streaming in the Core-Edge Separated Internet

Loránd Jakab, Albert Cabellos-Aparicio, Jordi Domingo-Pascual
Universitat Politècnica de Catalunya

Main idea: take advantage of the likely introduction of the Locator/Identifier Separation Protocol to "fix" multicast

Design goals: simple live streaming protocol, with minimal state, incrementally deployable

Gains: important bandwidth savings when clients cluster well in autonomous systems (ASes)



Media Server

For each multimedia stream (channel) that it is broadcasting, the media server 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. Each domain reserves an EID with local scope only for *CoreCast* streams. 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, *CoreCast* uses the reserved EID to trigger examination of its protocols fields, and leave all other packets in the fast path. For each channel, the media server 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 to the reserved special *CoreCast* EID. After the payload packet is sent, the media server iterates through *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. 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: $MaxClients = (C \times T) / 8H = (C \times P) / (H \times BW)$, where C is line rate in bits per second, T is time between payloads in seconds, P is payload size in bytes, H is header packet size in bytes, and BW the bandwidth required by the stream in bits per second. *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.

ITR

The ITR is the first of the two *CoreCast* stream demultiplexing points. In order to process *CoreCast* packets it maintains a *payloadBuffer* data structure, which holds the current payload data of each stream. To avoid keeping too much state, only one payload per stream is stored at a time, identified by *hash(payload)*. Each of the entries points to a structure storing the list of RLOCs (*servedRLOC*), which have already received the current payload. For each *CoreCast* header packet, the ITR identifies the channel using the reserved destination EID in the IP header, extracts the client EID and the hash from the *CoreCast* header and looks up the RLOC associated to the client EID. This lookup of the client EID to destination RLOC is a function already provided by LISP. Using the hash from the *CoreCast* header, it checks for the existence of the associated payload data in the *payloadBuffer*, checks if the RLOC has already received the payload using the *servedRLOC* list and forwards the header to the ETR, doing the usual LISP encapsulation. In the case no payload was yet sent a particular RLOC, a payload packet is generated before forwarding the header packet.

ETRs

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*.

Processing Overhead Evaluation

Extra operations

- hashing & store (payload packets)
- hash lookup (header packets)
- mapping lookup (LISP specific)

Implementation

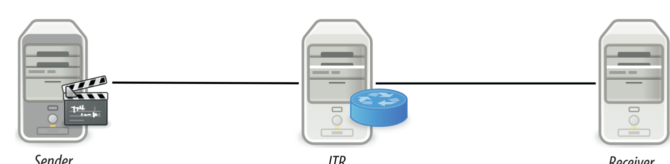
Sender: Packet stream generator with raw sockets

ITR: Linux 2.6 kernel module – Netfilter hook

Hash function used: SHA1

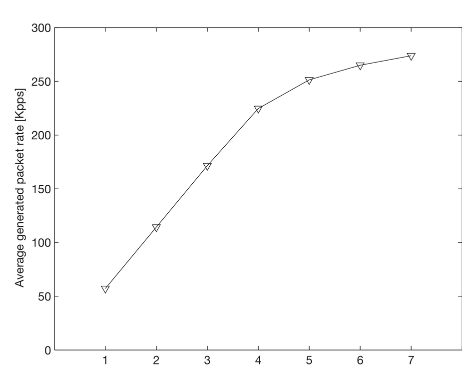
Experimental setup

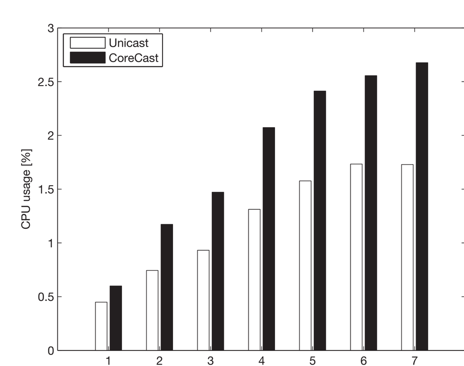
- P4, 3 GHz machines with Intel Gigabit cards (Debian GNU/Linux, kernel 2.6.26)
- 7 sets of experiments with *n x 1429* clients each set (we call *n* scaling factor)
- 20 repetitions/iteration averaged



Results

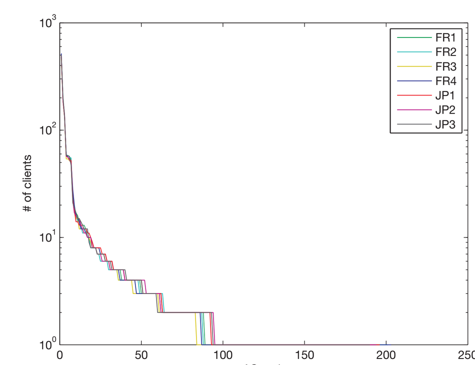
As expected, *CoreCast* incurs higher CPU usage that simple unicast packet forwarding. However, the increase is about 52% on average. Taking into account that our implementation is not optimized compared to the unicast forwarding code in the Linux kernel, which has been tuned for several years we can safely conclude that the processing overhead of the *CoreCast* is not a problem for a router. A production router could easily have software or hardware optimizations implemented for the new operations introduced by *CoreCast*.

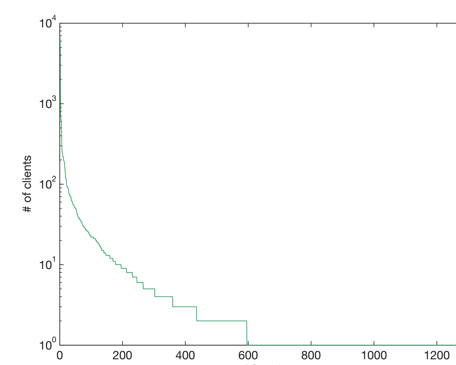




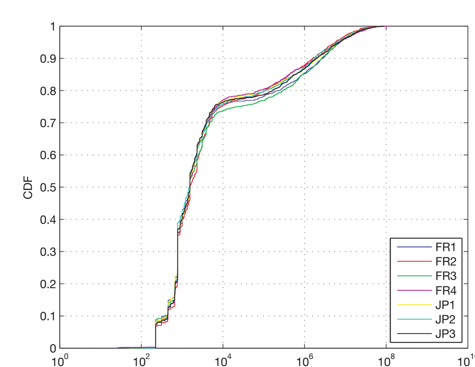
Bandwidth Savings

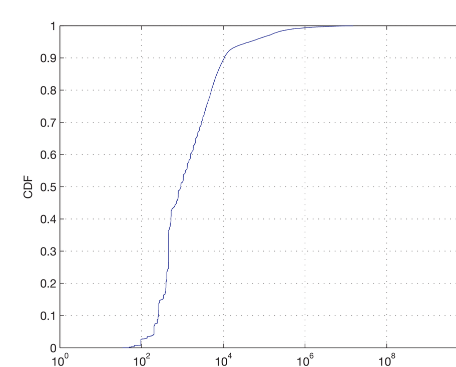
- Depends on the distribution of clients in AS-es: the more clients/AS the bigger the savings
- We used 2 P2P live streaming traffic trace sets for comparison
- **Set 1**: collected at multiple capture points in France and Japan using TVAnts – qualifying soccer match for the Olympics, China vs. Vietnam on August 23, 2007
- **Set 2**: collected at the Technical University of Cluj-Napoca during the memorial service of Michael Jackson, using the PPLive client software on July 7, 2009
- Distribution of clients in ASes can be seen in the figures below





- A few top ranking ASes would have benefited greatly if using *CoreCast*, because the large number of clients per AS
- In Set 2, the first 3 ASes were holding over 50% of the total 23713 clients
- Clients where traffic was captured had the following traffic distribution:





- To estimate savings we assumed all clients have the same traffic profile
- We estimated intra- and inter-domain traffic exchanged in the P2P overlays
- Then, we calculated the traffic generated by *CoreCast* for the same stream, which is about 130 GB in the inter-domain case and 900 GB for intra-domain traffic

Estimated bandwidth usage in the P2P overlay

Trace	IPs	ASes	IDT [GB]	IAT [GB]
FR1	1855	209	3881	505
FR2	1865	204	2944	391
FR3	1769	201	3523	456
FR4	1888	207	3948	536
JP1	1856	201	3509	452
JP2	1863	197	3291	431
JP3	1878	194	3681	496

Summary

- Window of opportunity to add scalable multicast support to the future Internet
- Important bandwidth savings: reduces the ISP's access link bandwidth consumption by an order of magnitude compared to existing P2P streaming services
- *CoreCast* is a simple protocol: zero-configuration, small amount of state
- Small processing overhead
- Because of its architecture, it allows independent Service Level Agreements to be established between Content Providers and ISPs. The latter property implies that it can be considered a reliable live streaming solution, for which content providers or ISPs can charge subscription fees.
- The only equipment that has to upgraded to support *CoreCast* is the same that has to support LISP – core routers untouched

Future Work

- Deploy *CoreCast* in PlanetLab and perform scalability experiments
- Implement and test *CoreCast* in the live LISP testbed (see *lisp4.net*)