# SeRViSO: A Selective Retransmission Scheme for Video Streaming in Overlay Networks

Carlos Eduardo Lenz, Lau Cheuk Lung and Frank Augusto Siqueira
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina - Florianópolis - Brazil
lenz@inf.ufsc.br, lau.lung@inf.ufsc.br, frank@inf.ufsc.br

## ABSTRACT

Several works proposed methods to make video streaming scalable over the number of clients, avoiding the linear growth of bandwidth requirements for the media source node. Some are based on overlay networks built on top of the IP protocol and distribute content between overlay partners. In this way the clients share their bandwidth, reducing the burden on the source node. Similar to data-oriented proposals, this work breaks the media into segments which are requested from partners when available. Novel in this technique is the explicit handling of losses with a selective retransmission mechanism based on H.264 content, controlled by estimated decoding importance of packets.

## Categories and Subject Descriptors

C.2.2 [**Computer Communication networks**]: Network Protocols — *Applications, Routing protocols*; C.2.4 [**Computer Communication networks**]: Distributed Systems — *Distributed applications*

## General Terms

Video Streaming, Distributed Multimedia

## Keywords

peer-to-peer, application level multicast, data-driven overlay networks, live video transmission

## 1. INTRODUCTION

During the last few years the increase of processing power available in devices made possible noteworthy advances in video compression. Connection technology also evolved (3G, WiMax, etc) offering home users higher data rates, wider coverage areas, more speed, lower price and more providers. Such technological advances start to make possible multimedia programming consumption through the Internet, challenging the television broadcasting method.


*SAC'10* March 22-26, 2010, Sierre, Switzerland.


The traditional client-server content distribution model has proven to be economically not viable, because it demands the server bandwidth to grow linearly with the number of clients: each client requires bandwidth equal to the media bitrate, which cannot be shared. One solution would be IP Multicast [2] or mBone which, however, are not available across the Internet: they are either unsupported by routers or disabled by administrators due to its costs and risks. In the literature, authors seek to overcome this problem with multicast protocols built on the application layer, to provide video services without imposing very high bandwidth demands on servers. Among the alternatives, some are built around a tree which guides how data is pushed [5], from the root to the leaves. Others are overlay networks without a selected structure, but with free association among nodes, where each part of the content is requested from a partner which has already received it. Those proposals, nonetheless, do not discuss retransmission due to losses, whether it should happen or not, and when.

This work starts with a data-driven overlay network, similar to existing ones, and builds a retransmission mechanism to handle lost packets. It establishes packet priorities and specifies a selection algorithm around them, so that unimportant packets may not be retransmitted. The priority and selection is based upon H.264 features: the NALU header, detailed in section 3. Other types of packets, such as audio, are given a fixed priority, although a later work may handle them differently. By focusing on the most important parts of the media, the selective retransmission can improve the playback, specially when the network is losing more packets than can be recovered. While currently SeRViSO is pull-only (see section 2), hybrid techniques may also be employed. The tests show live media streaming networks may benefit from this technique, by reducing the overhead caused by retransmissions up to 40%, without degrading the playback beyond specified by the algorithm (around 10%).

Section 2 overviews some of the literature for live video transmission, and digital video characteristics are summarized in section 3, together with some techniques used in other video distribution systems. The problem targeted by this article and the proposed solution are described in section 4. Details about the prototype built and the tests performed are discussed in section 5. Finally the section 6 brings the final conclusions.

## 2. RELATED WORK

The first application layer multicast proposals were based on distribution trees. ZIGZAG [5] builds an administrative

organization with all nodes, where control messages help balance the network after node arrivals, departures or failures. This structure is composed of a tree of node clusters, limited in size to avoid overloading of the internal nodes. In this way, the end-to-end delay to the leaves does not vary too much because their tree-depth is the same. The head node of each cluster takes the information of its subordinates and answers to the higher level cluster. If a node belongs to a level $n$ cluster, it is also the head of one child cluster in each level below. So each node belongs to clusters in a series of levels, except for nodes only on the leaves. The video server is the head node of the root cluster in the highest level.

The multicast tree is built with the help of the administrative organization, but without following its topology. Each node sends data from its highest position in the tree to subordinates in a child cluster where it is not head. Also the server has to send the data to its subordinates in the head cluster. Delay and latency are kept small given the network size, as with all tree-based techniques, but it is susceptible to departures of failures, because a group of nodes end up without the content until the tree is repaired. Another drawback is the higher load on the internal nodes because the leaves do not contribute with their bandwidth.

Data-driven overlay networks is a new approach based on the data being segmented and pulled from different partners. When a partner fails, others are selected for transmission, which makes these techniques more resilient to departures and failures. There are no leaves, so all nodes share their bandwidth. Neither there are trees or an specific network structure, the connection graph may be described as a mesh. Nodes periodically send media segments availability reports to partners, so they can request it. This process leads to higher delay in relation to pushing techniques, where data is automatically sent. DONet [9] works like this, where a gossip protocol is used by nodes to know a subset of all nodes in the overlay and a few of them are selected to establish partnerships. Suitable segment size should be around 1 second media, based on its bitrate. A scheduling process [9] selects among partners which one should send each segment. Hybrid techniques were proposed later to avoid the drawbacks of previous approaches – tree-based push or mesh-based pull. They use the mesh for data-pulling, providing failure-resilience, but opt for push whenever possible, be it through an auxiliary partial tree or not. In mTreebone [6] a tree is used to push data and non-stable (i.e, new) nodes are kept as leaves. After a departure or failure, lost data is requested using the mesh until the tree is restored. Segments that are not received through push reach a window which trigger pull requests. GridMedia [8] adopts another hybrid approach, but does not have an auxiliary tree. Data is pushed after negotiation among partners, repeated periodically, but late packets are again pulled from partners.

## 2.1 Selective Retransmission Techniques

A semi-reliable multicast protocol is proposed in [1], where frame type and current loss rate are used to evaluate if a loss is NACKed to the neighbors. I frames are always recovered, and if the loss rate is less than 40 and 20%, P and B frames are respectively recovered too. A retransmission scheme for wireless networks, based on congestion risk and frame importance is detailed in [4]. In order to avoid overloading the network, less important frames are not retransmitted.

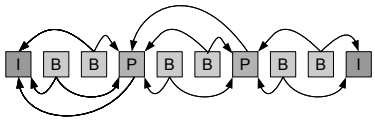


**Figure 1: GOP - Group of Pictures**

# 3. DIGITAL VIDEO

Previous push and pull proposals are format independent because they handle the media as an opaque data sentence, at the cost of losing opportunities for optimization, specially for retransmission.

Most current digital video coding standards split the pictures into groups (Figure 1), and to better apply different compression techniques, frames can be coded as intracode (I), predictive (P) and bidirectional (B). An I frame is self-contained, while P allows better compression since it requires information from a previous I or P frame. Finally, B requires information from both a previous frame and a frame ahead, having the greatest compression. So a lost I frame breaks the decoding of an entire Group of Pictures, which relies on it, a lost P frame affects the related Bs and the next P, while a lost B has little impact.

The H.264 standard has changed some details and introduced a few variants of the I and P frame types, but this work does not treat the variants differently from the original types. Noteworthy is the new Network Abstraction Layer Unit (NALU), a packet to delimit a smaller part of the data stream. It has one byte header where five bits identify the data type and two bits rate its importance for decoding subsequent NALUs. Each NALU can have a slice of any frame type, most commonly a raster scan sequence of blocks, a slice partition or a picture or sequence parameters set [7], which contain information necessary for decoding one or more pictures. The partitions, also new to H.264, are listed below from the most to the least important, considering the data they contain. There are 3 partition types: A – the smallest one, has headers and permits partial reproduction, B and C – contain P or B frame textures.

# 4. SERVISO

For a live multimedia transmission system, reception on time for display is more important than reliability, since small losses at 24/30 frames per second are less noticeable than late (so lost) frames. So retransmission of data close to its deadline is useless and further delays later packets. As the proposals in section 2 use unreliable protocols for transmission but do not explicitly handle losses, the following options might be considered:

- Do not send NACKs and assume that the network is reliable enough for losses to be insignificant. No such guarantee exists for the Internet;
- Every loss results in a NACK, which deny the reception on time priority explained before;
- A few packets are selected for recovery, which is the approach adopted in this paper.

A prototype was built to test the SeRVISO model, which means *Selective Retransmission Video Streaming Overlay.*

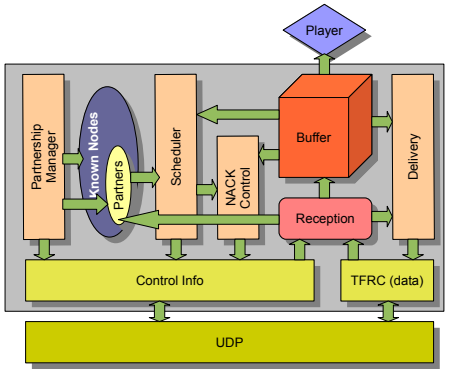


**Figure 2: Components.**

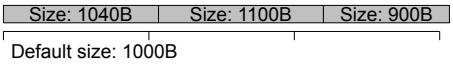


**Figure 3: Segments vary in size.**

Figure 2 shows its components, which are described next. In this work the term *element* will be used to designate a data block of variable size, which contains a NALU or an audio packet. Data transfer is augmented with information about elements, so when elements are lost the NACK algorithm can select which ones to request again. Unlike [9], the data segment does not have constant size to avoid breaking elements on the borders between segments (Figure 3).

## 4.1 Overlay construction and dynamics

Previous models [9, 8] built its network and handled network dynamics in a similar way, but not completely. First, a replicated group of Rendezvous are used exclusively to help managing the network, neither transmitting nor exhibiting the media. When a node wants to join the network it sends an ***ENTER*** message to one of the Rendezvous. It stores the network address of the node in its alive node list and shares it with the other Rendezvous. Then it takes a subset of the alive list and answers to the node with a ***NODES*** message. For the node to be kept as a member of the network, it keeps sending the ***ENTER*** message from time to time to signal that its network connection is still alive, and also to discover new nodes. The node providing the video performs the same operations, so the Rendezvous can not identify it. There are tree ways the Rendezvous can remove a node from the alive list:

- The node sends a ***LEAVE*** message to the Rendezvous;
- After a timeout $t_1$ without messages from the node;
- Some node stops receiving messages from one of its partners, after a timeout ($t_2 < t_1$) it sends a ***LEAVE*** message for the failed partner to the Rendezvous.

Each node keeps a local list of the network members it knows, which is a subset of the entire network. Once the list is full, the oldest elements are removed, but if it is too small the node searches for new nodes. A new node can go into the list if:

- A message of any type is received from the node;
- A ***NODES*** message from the Rendezvous included it;
- If the list is still small, an empty ***NODES*** message is sent to one of the nodes in the list, requesting it to send some nodes of its list, as the Rendezvous would send.

Before transferring the video data, a node has to establish partnerships with some of the known nodes from its local list (**Partnership Manager** component). So ***PARTNER*** messages are sent to random nodes in the list, which are answered to confirm it. The requested node accepts if it has not already reached its maximum number of partnerships. A single bit is used to distinguish a request from a confirmation ***PARTNER*** message.

## 4.2 Data Transfer

After a partnership is established, both nodes start sending each other reports on segments availability in **Buffer** component through ***BMAP*** messages. A segment is considered available if the NACK algorithm does not ask for any element. The reports are limited in size, like [9], and the scheduling algorithm is also the same, but over a limited range of segments ahead, so the transmission of segments far ahead or that already lost is deadline does not delay the ones important for exhibition (**Scheduler** component). Given the list of partners and the corresponding segments to request, the node sends each selected partner the appropriate ***REQUEST*** message, which is similar to the ***BMAP*** message and a set bit means the segment is requested.

The partner receives the request and prepares the ***DATA*** messages (**Delivery** component). The messages are sorted by the start position of the contained data, so NACKed elements are delivered first. This queue can also be flushed of messages related to segments not requested in the last received ***REQUEST***. Each ***DATA*** message contains a set of sequential elements, along with the following information:

- Start position and size of the data interval;
- Coded details about the segment elements that are not in the message, to supply information for the NACK algorithm when a packet is lost:
  - Element start position and size;
  - Element type: the NALU type or 'OTHER' otherwise (it is probably audio);
  - A bit set when the supplier does not have the element.

But ***DATA*** messages are not delivered directly if they are small, instead they are packed into larger ***MULTI*** messages. Upon dequeuing, ***DATA*** or ***MULTI*** messages are encapsulated in a TFRC (*TCP Friendly Rate Control Protocol* [3]) packet, while messages of other types are small and sent directly. This protocol calculates the maximum throughput to provide congestion control based upon tree measures (in SeRViSO each host is controlled individually):

- Packet size. So small ***DATA*** messages result in a throughput too small for real time video streaming;
- Losses reported through TFRC *FEEDBACK* messages;
- Round trip time, estimated with the help of the *FEEDBACK* messages.

**Algorithm 1** selectMissingElements(missing : list, { sumWeightsAvailable, sumWeights } : real, sumBytesAvailable : integer)

```
1: ACCEPTABLE_RATE ← 90%
2: MINIMUM_BYTES ← 0.7 × SEGMENT_SIZE
3: sort_by(missing, elementWeight, 'descending')
4: selected ← ∅
5: for all element ∈ missing do
6:    weight = elementWeight(element)
7:    if weight >= 3 then
8:       selected ← selected ∪ {element}
9:       missing ← missing − {element}
10:      sumWeightsAvailable ←
         sumWeightsAvailable + weight
11:      sumBytesAvailable ←
         sumBytesAvailable + length(element)
12:   end if
13: end for
14: while missing ≠ ∅ ∧ (sumWeightsAvailable /
    sumWeights < ACCEPTABLE_RATE ∨
    sumBytesAvailable < MINIMUM_BYTES) do
15:   element ← first(missing)
16:   selected ← selected ∪ {element}
17:   misssing ← missing − {element}
18:   sumWeightsAvailable ←
      sumWeightsAvailable + elementWeight(element)
19:   sumBytesAvailable ←
      sumBytesAvailable + length(element)
20: end while
21: return selected
```

## 4.3 Retransmission

When ***DATA*** messages are received the node sets the time stamp of the corresponding segment, so the NACK process can be triggered after a timeout. The NACK process also runs when a segment $i$ is incomplete but a segment $j > i$ has a newer time stamp. The node runs algorithm 1 (with element priorities set by algorithm 2) to select elements among the missing ones, joins them into greater intervals if possible, and requests them all as pairs $(startposition, size)$ into a ***NACK*** message.

A node might discover that a segment is available from one or more partners when the segment is already outside of the scheduling window, but ahead of the playback index (yet not too close). When this happens the node applies the desperate mode: breaks the segment into arbitrary intervals about the same size without knowledge of the elements inside it, and asks each partner a piece, using ***QNACK*** messages. They are similar to standard ***NACK***s, but answered by ***QDATA***s. These pairs of messages make it clear that the partner is not responsible for streaming the whole segment. A node also sends ***QNACK*** messages when the algorithm has selected an element which the partner has informed it does not have.

## 4.4 Optimizations and Restrictions

The origin node reports the availability of each node to a small number of partners to avoid the risk of transmitting them all. Due to the unavailability of previous works for the tests, an alternative was implemented to provide a baseline for the data and charts in section 5. In this *Traditional* mode, the prototype operates without bothering about the data layout of the media, as an standard data-driven overlay does. So it works in a simpler way:

**Algorithm 2** elementWeight(element)

```
1: sizeWeight ← MAX(10−log10(element.size),0) / 10
2: if not NALU then
3:    typeWeight ← 2
4: else
5:    type ← h264NaluType(element)
6:    if type ∈ {NON_IDR, IDR} then
7:       type ← h264SliceType(element)
8:    end if
9:    case type in
10:      when I, PARTITION_A : typeWeight ← 3
11:      when P : typeWeight ← 2
12:      when B, PARTITION_B, PARTITION_C :
         typeWeight ← 1
13:      when delimiter : typeWeight ← 0
14:      otherwise : typeWeight ← 2
15:   end case
16: end if
17: return MIN(sizeWeight + typeWeight, 3)
```

- The media is not processed for NALUs, as this information would not be used. Each segment is simply divided into eight fixed-size parts for transmission, so the ***NACK*** message is simpler;
- ***DATA***: contains only the data of a single part, as the information about the parts before and after it are not necessary for its NACK algorithm;
- ***NACK***: contains a bit-map, where each bit of a byte represents whether a part of the segment needs to be retransmitted or not. The index of the first segment is included as well;
- The NACK algorithm is also simpler: just try to recover everything. This is the second possibility for NACKs devised in section 4.

# 5. EXPERIMENT

The experiment was run on about sixty PlanetLab[1] nodes. One of them was both the media source and the Rendezvous (only one Rendezvous was used) as independent processes, while the other were clients. Information was saved during the experiment to feed the charts and the table below.

Tests include SeRViSO and the *Traditional* mode (section 4.4). Losses were forced before sending the packets to the network – but after TFRC have accounted them – for the following levels: 0 and 20%. Since TFRC uses the loss rate to limit the throughput, more aggressive protocols could take its place if higher loss rates are expected. Apart from congestion control of TFRC, each node limits its global throughput to 2 Mbps.

Since the *Traditional* mode was the basis of comparison instead of previous work, only the NACK algorithm is evaluated, not basic overlay features, like resilience to failures. As the prototype took as input pre-recorded MPEG-4 files with

[1]http://www.planet-lab.org/

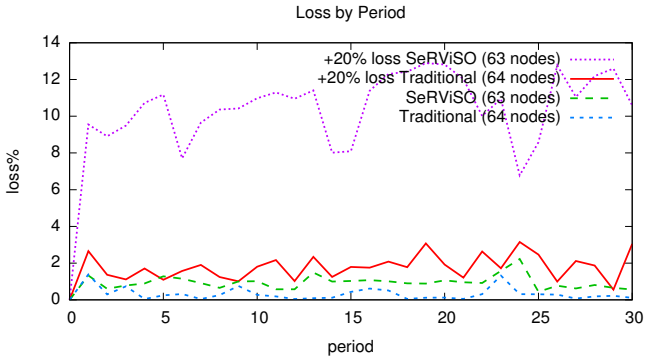


**Figure 4: Loss Rate**

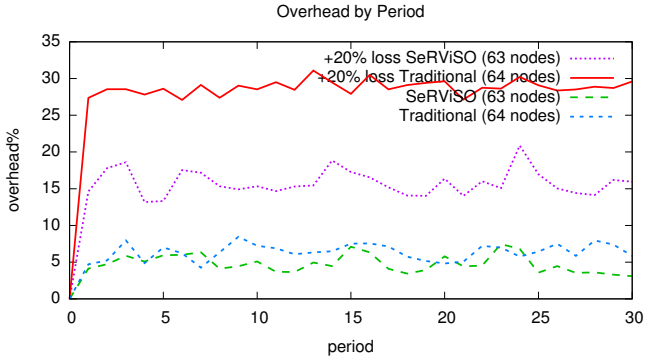


**Figure 5: Retransmission Overhead Rate**

| Mode | Loss (%) | Retransmission (%) |
|---|---|---|
| Traditional | 0.32 | 6.03 |
| SeRViSO | 0.96 | 4.62 |
| +20% loss | | |
| Traditional | 1.78 | 22.34 |
| SeRViSO | 10.61 | 13.64 |

**Table 1: Average Rates**

H.264 content, the file headers must not be lost for the decoding to happen, requiring the same reliable transfer that the traditional mode does. But in this PlanetLab experiment they were skipped because the nodes have no GUI and it would change the measured statistics. The media lasts for thirty minutes and its bit rate is 222 kbps. After the first segment is available, each node waits ten seconds before starting the playback, to have a buffer of a few seconds, as in [9].

Figure 4 shows the measured loss rate for both modes (SeRViSO and traditional) and inflicted loss rates (0 and 20%), for the thirty minutes test, while Figure 5 has the relative overhead caused by retransmissions, also for both modes and loss rates. Table 1 has the averages for the whole sessions. From the first chart and the table, loss rate is negligible, except for the SeRViSO mode with 20% inflicted loss rate, which was around 10% as designed by the NACK algorithm. Also, some of the local maximums and minimums are more or less inverted into the charts, a by product of the algorithm. When important parts of the stream are lost instead of less important ones, respectively I and B frames, there is less margin for tolerance and more data must be retransmitted. The second chart shows that up to 40% of the retransmissions are unnecessary if SeRViSO is used.

# 6. CONCLUSION

This article presented a data-driven overlay network for live H.264 video streaming, adapted to perform selective loss recovery. When a node detects that a segment of the media was not successfully received, it triggers an algorithm that selects which lost packets to NACK. First, it evaluates weights for the lost and received packets, based on their size and the importance of the data contained. Then it selects all highly important missing packets (I frames). It also selects the missing packets with higher weights, until the sum of the weights of all packets which were successfully received or selected for retransmission reaches 90% of the sum of packet weights in the same segment. The tests have shown that it tolerates the expected amount of losses, while requiring much less retransmissions (up to 40% less). The algorithm was found appropriate to deal with losses and to adapt to the networks conditions.

**Acknowledgment**

This work is supported by CNPq (Brazilian National Research Council) through processes 472754/2008-4.

# 7. REFERENCES

[1] C. Bortoleto, L. Lung, F. Siqueira, A. Bessani, and J. da Silva Fraga. A semi-reliable multicast protocol for distributed multimedia applications in large scale networks. In *Proceedings of the 8th International Conference on Management of Multimedia Networks and Services*, page 109, 2005.

[2] S. Deering. Host Extensions for IP Multicast (RFC 1112). *Internet Engineering Task Force*, 1989.

[3] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control Protocol (RFC 3448). *Internet Engineering Task Force*, 2003.

[4] Á. Huszák and S. Imre. Content-Aware Selective Retransmission Scheme in Heavy Loaded Wireless Networks. *Wireless and Mobile Networking*, 248:123–134, 2008.

[5] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2.

[6] F. Wang, Y. Xiong, and J. Liu. mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *27th International Conference on Distributed Computing Systems*, page 49, 2007.

[7] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[8] M. Zhang, J. Luo, L. Zhao, and S. Yang. A peer-to-peer network for live media streaming using a push-pull approach. In *Proceedings of the 13th ACM international conference on Multimedia*, pages 287–290, 2005.

[9] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *Proceedings of IEEE Infocom*, volume 3, pages 13–17, 2005.