A Hybrid Peer-to-Peer and Client-Server Strategy for Multimedia Streaming

Samuel L. V. Mello*[1], Elias P. Duarte Jr.[2]

Departamento de Informática, Universidade Federal do Paraná

P.O. Box 19018 – Curitiba, PR 81531-980 Brazil

*[1]samuelmello@gmail.com; [2]elias@inf.ufpr.br

***Abstract*-Stream distribution is one of the key applications of the current Internet. As the number of users increases, the amount of bandwidth required at the streaming source server can become a bottleneck. Using peer-to-peer networks for content distribution is a realistic approach to avoid network traffic concentration and, at the same time, requires weak control of the distribution process by the streaming source. This work presents a hybrid system that uses peer-to-peer technology to assist the distribution of streaming data. The source server still participates in the distribution whenever necessary. The system was implemented and experimental results show an expressive reduction of the network traffic at the content source. Experiments also show the system behavior in the presence of peer crashes.**

Keywords- Multimedia Streaming; P2P Streaming; Internet Streaming

I. INTRODUCTION

The distribution of content such as files or live audio and video streams to a potentially large amount of users is one of the key applications of wide area networks, such as corporate WANs or the Internet. Traditionally, the client-server model is the usual approach for this kind of applications. As peer-to-peer networks [7] have become popular, new approaches have been proposed to leverage the efficiency and flexibility of these content distribution systems.

In the client-server model, the whole content is available at a server, from which it is transmitted to the clients. All transmissions necessarily rely on the server, which accumulates the upload costs and can become a performance bottleneck. When the content consists of large amounts of data, these properties are specially unwelcome. On the other hand, the client-server model offers to the content owner full control over the service provision to the clients. For instance if the content owner wishes to sell the service with certain guarantees, this can be directly done without relying on other parties. This property is specially desired when it is necessary to ensure that the performance requirements specified on a pre-defined SLA (Service Level Agreement) contract are met. Several examples of this client-server content distribution strategy exist, many of which are based on the Web itself, such as traditional on-line audio and video transmission systems [3].

Typically, a server can provide the content to several clients in parallel. As all clients receive the same content, they can build a peer-to-peer network and exchange the received data. In this way the server delivers just some part of the data to the client, which is able to obtain the whole content by exchanging data with other clients that have received different parts. Systems that use this approach, include BitTorrent [1].

The reduction of bandwidth usage at the server is specially attractive when used by network intensive applications such as large file transfers and multimedia streaming. Nevertheless, these two kinds of applications present significant differences in their content distribution processes. In file transfers, the whole file content is available since the beginning of the transfer and its size is fixed and well known. On the other hand, in multimedia streaming the content is produced at the server over time and needs to be available at the clients within certain time limits in order to be successfully used by an application that consumes the data at a fixed rate.

P2P networks for file sharing can rely on the fact that users may receive the content in any order. Furthermore the fact that the whole file content is available from the beginning. Thus, if several clients start downloading a specific file at roughly the same time, they may receive different parts of the file and later exchange those parts in order to completely receive the original file. In multimedia streaming the data must be received in a certain order, and “old” data is not useful for clients. This fact has an important consequence: the number of content parts that are of interest to clients is substantially reduced to parts of a relatively small buffer. Another difference between multimedia streaming and traditional file transfers is that streams may not have a known end. So, the process may continue indefinitely and the data source must keep sending data as long as the transmission continues.

In this work we present a multimedia content distribution system that uses a hybrid peer-to-peer and client-server approach. A server is responsible to create and maintain the content, which is temporarily stored in a transmission buffer and then split in small parts that are transmitted to a group of nodes. These nodes interact between themselves and with other nodes that did not receive the data directly from the server to set up forwarding agreements to exchange data. The forwarding agreements are renewed from time to time to adjust to network changes. As the clients need to retrieve the complete buffer within a certain time limit, and the peers that forward the data may fail, there is a mechanism that allows data to be retrieved directly from the server, whenever any buffer part has not been received within a specified time window. We performed experiments that show the impact of the system in terms of bandwidth requirements as well as its behavior in presence of failures.

Section 2 describes the proposed system in detail. Section 3 presents the experimental results, and Section 4 points to related work. Section 5 concludes the paper.

II. THE PROPOSED HYBRID STREAMING SYSTEM

The distribution of multimedia streams to a potentially large number of clients requires the transfer of large amounts of data. Peer-to-peer strategies are efficient in this scenarios because they create alternative data paths, thus not all data flows are concentrated at the source, which could otherwise become a performance bottleneck, restricting the maximum number of clients served. By using a peer's upload capacity the bandwidth requirements at the server are thus reduced.

A wide variety of applications may take advantage of the reduction of bandwidth demands at the server. Examples include popular Internet video hubs, but can be restricted to a single network, one for instance that allows users at a campus network to receive a video stream efficiently. The features of the underlying networks may vary greatly, and for all cases the decrease of network usage at the source server creates clear advantages.

In the present work a hybrid client-server and peer-to-peer system is introduced that presents an advantage in terms of the reduction of network requirements at the server, but as peers are unreliable entities in the sense that they may leave the network at any point in time, the proposed system includes a mechanism that allows clients to obtain data from the server whenever it is needed.

A. *Proposed System Description*

The proposed system is composed of a server and a set of clients. The server is responsible for producing the data flow that is streamed to the clients and to help them to find one another, as well as to transmit some copies of the content so a set of peers which exchange the obtained parts to get the complete data.

The clients are responsible for obtaining the stream and playing it back to user. Also, they serve parts of the stream to other clients forming a peer-to-peer network. Each client connects to other clients which are actually peers to try to retrieve all parts of the stream and if, by any reason, it is not able to retrieve some part, the server will provide the missing part, preventing any service interruption to the user.

The multimedia stream is split into slices of fixed size that have sequential identifiers. The slices are produced and consumed at a fixed rate. Just after producing each slice of the stream, the server splits it in *blocks* of constant size to be sent to a group of clients. Each block in the slice is numbered with its position in the slice.

Clients make forwarding agreements among themselves. When a client receives a block, it is retransmitted according to the forwarding agreements in place. Each forwarding agreement refers to blocks of the same position in the slice and is valid for a certain amount of slices, being optionally refreshed after that. All forwarding agreements established in the system are valid for the same duration and start at slices whose sequential numbers are multiples of the agreement duration. The first slice of each agreement is called the *base-slice* of the agreement. If the client is unable to establish agreements with other peers for any reason, it still can establish an agreement with the server.

Figure 1 depicts the blocks affected by a given forwarding agreement. In the example, the node receives from node X blocks at position 3, starting at slice 20 and considering an agreement duration of 10 slices.

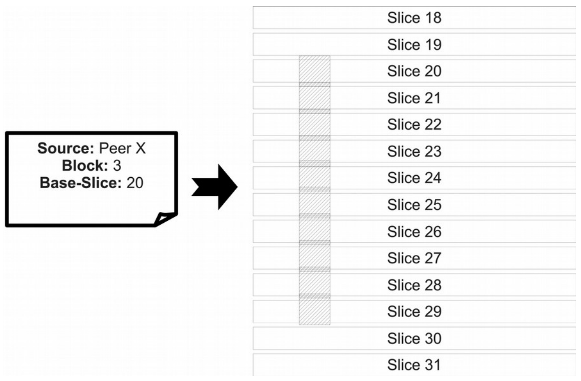

Fig. 1 Forwarding agreement example.

Figure 2 shows an overview the transmission process. At the server, the media producer creates the content and splits it in slices (1). Each slice receives a sequential identifier and then is split in blocks (2). In the example, each slice is split in four blocks. The blocks are then transmitted over the network up to the client (3). This transmission may occur directly from the server to the client or through any number of peers. Each block may take a different route to the client. After receiving the blocks, the client rebuilds the stream slice (4) and plays it back to the user (5).

The stream is produced and split into slices at the server by the media producer. The rate in which slices are created and its size are configurable. Just after the creation, each slice is sent to clients according to current forwarding deals and then stored in a buffer. The server uses this buffer for possible retransmission requests from clients. The slices are held in this buffer for at least the duration of one forwarding agreement.

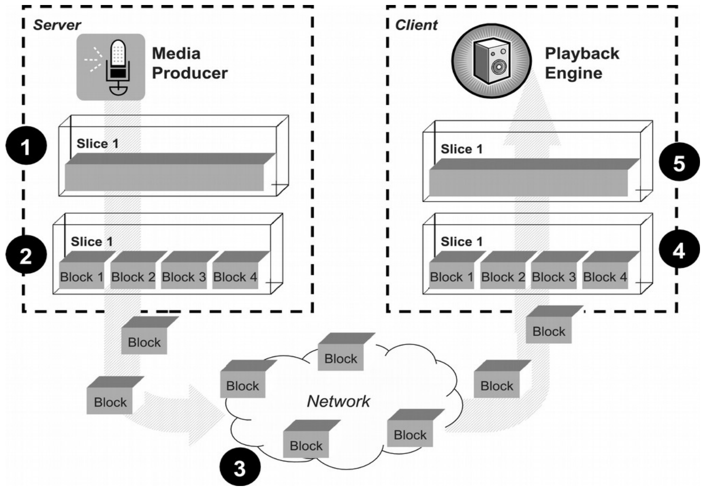

Fig. 2 Data flow.

Figure 3 shows the message flow during client start up. To make it simpler, each slice is composed by only two blocks in the example. When the client starts, it registers itself with the server (1). The server accepts the connection and adds the client to a list of active clients. After that, it sends back to the client configuration data about the multimedia stream (2). The configuration data is then used to set up the client's playback engine and includes, for instance, the rate in which a slice should be consumed and the size of each block. Afterwards, the client requests information about other peers and the server responds with a group of randomly chosen clients from the active client list (3).

In addition to help clients find each other, the server also takes part in the data distribution transmitting some copies of the content to a group of clients. The server holds a list of forwarding agreements that indicates to which clients it will send newly created slices. Upon a new client arrival, the server creates forwarding agreements for the whole slice up to the next agreement dealing round (4), so the client can start receiving the stream quickly.

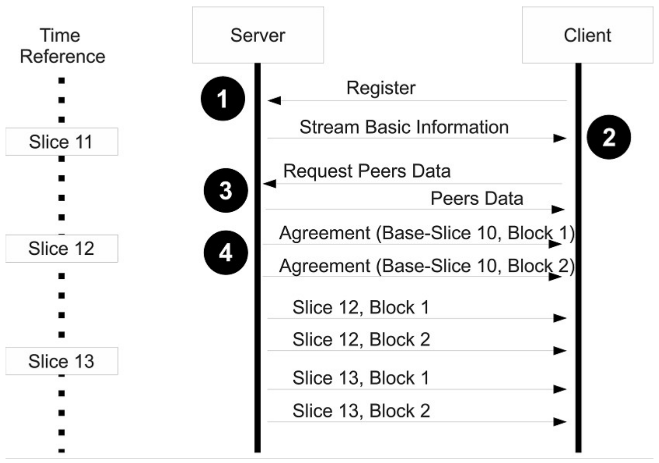

Fig. 3 Message flow during client start-up.

The server must send the content to clients at least once, so at least one peer will have each part and the content can be successfully rebuilt by the peers. The largest the number of copies sent willingly by the server to the peers, the easier it is for each peer to find others that receive the parts it needs, in order to establish a new forwarding deal. In this way, the server chooses a number of clients to receive forwarding agreements. This number of clients usually is a fraction of the total number of connected clients, and the number of content copies transmitted is configurable. Peers are chosen based on how low the RTT (Round Trip Time) is.

During the time period covered by a forwarding deal, the server chooses the group of clients that will receive these forwarding deals in the next period. This is done in advance so peers have sufficient time to negotiate and establish forwarding agreements before the next base-slice gets transmitted. After choosing the group of peers that will receive the forwarding agreements the server notifies these peers, indicating which data each peer is going to receive. Then, the server notifies all connected clients that there are forwarding agreements available among them for the next time period and peers begin the phase in which they set up forwarding agreements for the upcoming period.

Each client holds a list indicating from which peer it receives which part of the stream. This information is also used by a monitoring process. Periodically, each peer exchanges monitoring messages with its neighbors. These messages contain approximate assessments of the delay between the creation of a slice at the server and the expected arrival time on the client, considering the current forwarding agreements. As each block may take a different route towards the peer, the delay for each block can be different, as shown in figure 4. The monitoring messages are padded to have the same size of one stream block, so the node can measure the time spent to retrieve a monitor message from a peer and use this time as an assessment of how long it would take to transfer a stream block from that peer. These assessments are used during the agreements phase.

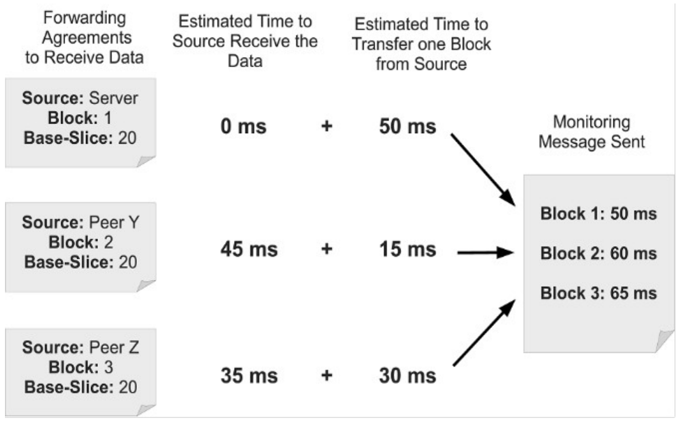
Forwarding Agreements to Receive Data
Estimated Time to Source Receive the Data
Estimated Time to Transfer one Block from Source
Source: Server Block: 1 Base-Slice: 20
0 ms
+
50 ms
Source: Peer Y Block: 2 Base-Slice: 20
45 ms
+
15 ms
Source: Peer Z Block: 3 Base-Slice: 20
35 ms
+
30 ms
Monitoring Message Sent
Block 1: 50 ms
Block 2: 60 ms
Block 3: 65 ms

Fig. 4 Monitoring message.

When a client receives the server notification indicating that there are forwarding agreements available for the next period, it creates a list of neighbours for each block position with which it will try to establish forwarding deals. Each list is then sorted by the estimated time to receive the data from that peer, calculated as the sum of the time the peer itself will take to receive the data, as informed in the monitoring message, and the time spent to retrieve the monitoring message. Additionally, the client starts a timer to finalize the agreements phase.

B. Agreements Phase

The client begins the agreements phase for each block by sending an agreement request to the first peer in the neighbor list built for that block. If the peer accepts the request then the agreements phase is over for that block. The peers create a forwarding agreement valid for the requested block during the next period. If the peer rejects the request, the client then sends a request to the next neighbor in the list. Peers that rejected the request are reinserted at the tail of the list to be retried if no other peer responds positively. A node cannot accept an agreement to forward a block for which it does not have an agreement to receive, so this interval between retries is intended to allow the peer to obtain a forwarding agreement for that data, so that it can hopefully accept the request made at the next retry. This process is repeated until the client obtains forwarding agreements for all blocks or a timer expires indicating that agreements phase is over.

If at the end of the agreements phase the client was unable to establish forwarding agreements for any block, it sends a request directly to the server that always accepts all requests. In this case, along with the agreement request the node receives from the server information about more peers with which it can expand its neighbors list. So, in the next agreement phases the client will have a larger number of neighbors to try, increasing the chances of success.

When a client receives an agreement request, it checks if there is already a forwarding agreement established to receive the same block and base-slice. If there is such agreement, it verifies the number of peers to whom it has already agreed to forward the block to. Each client shall accept at least a certain amount of forwarding agreements. If this amount has not been reached yet, then the request is accepted, otherwise it is rejected.

Figure 5 depicts an example of a forwarding agreement establishment, executed by a client (peer A) after receiving the notification from the server. In the example, peer A receives a forwarding agreement from server for block 1, peer Y for block 3 and peer Z for block 2. Afterwards, peer A creates (1) a neighbor list for block 2 and another for block 3. These lists are composed of all neighbors that the peer has knowledge of, sorted by the estimated time to receive data from each neighbor. For block 2, peer A initially sends an agreement to peer Y (2), which is unable to accept the request, as it does not have an agreement to receive that block yet. Then, the request is sent to peer Z, the next on the list (3). Peer Z accepts the request and peer A adds the new forwarding agreement (4) to its list of forwarding agreements.

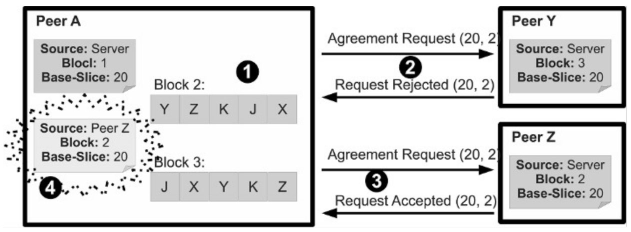

Fig. 5 Forwarding agreement establishment.

C. System Behavior in the Presence of Failures

All data received by the clients is stored in a buffer before being consumed by the playback engine at a fixed rate after an initial delay for jitter prevention. This initial delay is a constant value configured in the system.

In order to start consuming a slice, the client ensures that all blocks from the next slice have been correctly received. If any is missing, for example due to the failure of a peer from which it was supposed to be received, the client requests the missing block directly to the server. The server then transmits the missing block directly, allowing the slice to be rebuilt and the playback to continue seamlessly. The time instant at which this checking takes place is computed as a multiple of the round trip time to the server, so there should be enough time to retrieve any missing block.

Figure 6 shows this procedure. A failure prevents blocks with a certain position to be received after slice 7. The nodes from which blocks with other identifiers are received do not present any failure and keep working as expected. This makes slices from 8 to 10 to be incomplete in the buffer, with block 7 missing. When starting to consume each slice, the client requests the missing block of the next slice to the server, receives it and continues the playback seamlessly. This procedure is repeated for all slices up to the next agreements phase, when the client will establish forwarding agreements with fault-free nodes for all blocks.

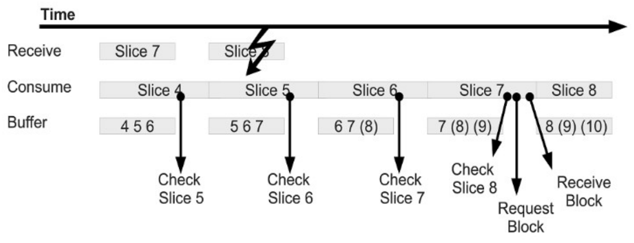

Fig. 6 System behavior in the presence of failures.

III. EXPERIMENTAL RESULTS

This session describes experimental results obtained with a prototype implementation of the proposed system. The prototype was implemented with the Java language. Messages were implemented as plain objects that were packed using standard Java serialisation and transmitted over TCP/IP connections. Although the default Java serialisation mechanism adds some bytes of overhead, the strategy was good enough to demonstrate the main characteristics of the proposed system.

All network operations were wrapped by a class that provides artificial bandwidth limits and collects statistics of the amount of data sent and received. These artificial bandwidth limits allow the simulation of several clients in the same host, making it easier to run the experiments.

All experiments involved the transmission of a 128 Kbps stream. Each slice was composed by 8 blocks of 4 KBytes each, consumed each 2 seconds. The forwarding agreements lasted for 10 slices and the server sent forwarding agreements to 40% of the nodes after sending the fifth slice of each forwarding agreement. The number of block copies transmitted varied in the experiments. All clients had an artificial bandwidth limit of 1024 Kbps. The experiments were run several times and the results are representative of the range obtained.

The rest of this section describes the three experiments, showing the impact of the proposed system on the bandwidth usage at the server, the influence of the amount of copies sent by the server on the upload demand for the peers and then the system behavior in the presence of failures.

A. Impact on the Bandwidth Requirements at the Server

The first experiment shows the bandwidth usage at the server with different settings for the number of copies the server willingly forwards and the number of clients. There were experiments with 1 to 5 copies of the content for 8, 16 and 32 clients. Each execution lasted for 180 seconds. The values were measured at the server and consider all data sent, including control messages. In all cases, the playback interruption rate was below 1%, that is, more than 99% of the slices were available for playback at correct time.

The graph of figure 7 and table I present the results. For each number of clients, the total amount of data sent by the server in executions configured for 1 to 5 copies is shown. In figure 7, the last column of each column group represents the theoretical minimum amount of data to be transmitted using a pure client-server approach, not considering control messages and headers.

TABLE I BYTES SENT AT THE SERVER X NUMBER OF COPIES SENT.

Copies	8 Peers	16 Peers	32 Peers
1	5637 KB	8049 KB	12809 KB
2	5620 KB	10803 KB	15249 KB
3	6125 KB	13486 KB	17982 KB
4	6380 KB	16186 KB	20710 KB
5	6880 KB	18853 KB	23297 KB
Pure Client-Server	23040 KB	46080 KB	92160 KB

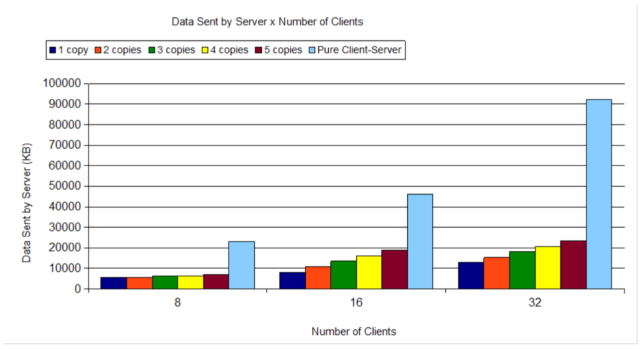

Fig 7 Bytes sent at the server x Number of copies sent.

B. Influence of the Number of Copies Sent by the Server and the Upload Requirement of the Peers

As the number of copies the server willingly sends to the clients increases, the peers need to upload less data from each other. The graph in figure 8 and table II show the average amount of data uploaded by each peer as the number of copies sent by the server vary. The values shown are the mean computed on the data sent by all peers, including bytes used for control and monitoring. It is possible to note that for 16 and 32 clients there is a slight reduction of the value. For 8 clients the value remains nearly constant, as the copies sent by the server are distributed among only 40% of the clients and using these parameters, a system with only 8 clients does not take advantage of the additional copies sent by the server.

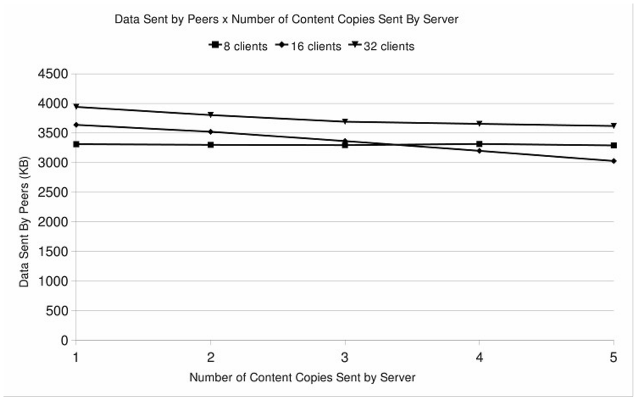

Fig 8 Bytes sent by peers x number of content copies sent by the server.

TABLE II DATA SENT BY PEERS X NUMBER OF CONTENT COPIES SENT BY THE SERVER.

Copies	8 Peers	16 Peers	32 Peers
1	3313 KB	3639 KB	3943 KB
2	3302 KB	3523 KB	3805 KB
3	3297 KB	3366 KB	3692 KB
4	3317 KB	3201 KB	3657 KB
5	3292 KB	3029 KB	3620 KB

C. System Behavior in the Presence of Failures

Another experiment shows the system behavior in the presence of peer failures. The failure model considered is crash. The configuration used was identical to the previous experiment and the server willingly sends 3 copies of the content. The network was composed by 32 peers and, during the execution 16 randomly chosen clients fail. Clients that were receiving data from the nodes that failed used the retransmission mechanism to request the missing blocks directly to the server in order to continue content reproduction without interruption. Even in the presence of failures affecting half of the network nodes, the playback interruption rate for failure-free nodes remained below 1%, that is, more than 99% of the slices were available for playback in time. The retransmission of missing blocks increased the bandwidth usage at the server until the next agreement period. The graph in figure 9 shows the impact of these retransmissions on the server's bandwidth usage.

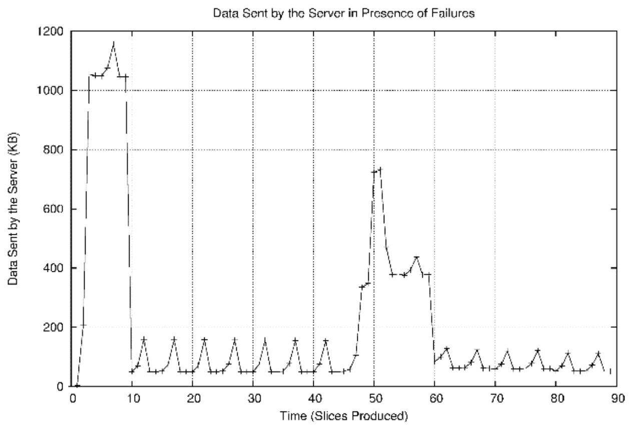

Fig 9 Bytes sent by server in the presence of failures.

It is possible to note in the graph a large amount of data transmitted in the beginning of the transmissions up to the time instant slice 10 is created. This high volume reflects the initial forwarding performed by the server just after the connection of a new client. In this experiment, all clients started at the same time, in the very beginning of the execution. Nevertheless, in real scenarios clients are expected to enter the system according to other distributions, usually Poisson, which would reduce this initial data volume.

Clients begin to take advantage of the established forwarding agreements at the creation of slice 10. Then, the system presents a stable behavior up to the failure that takes place near slice 45. During this period, it is possible to note some small peaks caused by monitoring and control messages. After the period in which the system was under the effect of the failure, near slice 60, it is possible to note that those peaks are even smaller, as the number of active clients exchanging control and monitoring messages with the server was reduced.

The failure happens near the time of the production of slice 45, but the effects on the server are observed around the time of production of slice 48. During this interval clients consumed buffered data. When the buffer was about to get empty, clients requested the missing data to the server, and this caused an increase on bandwidth usage. The system was under effect of the failure until the time of production of slice 60, when new forwarding agreements became active.

IV. RELATED WORK

Related work includes several P2P multimedia streaming strategies. Reference [12] presents CoolStreaming, a live video streaming P2P system. The authors describe the development of the system and the several challenges for making such a system perform well. Usually pure P2P strategies do not offer QoS guarantees, such as [7] in which peers are selected based on their measured availability. In [2] a scalable pure P2P strategy is presented which is based on gossip.

PROMISE [23] is a P2P streaming system that is based on the CollectCast communication service. CollectCast monitors the state of peers and dynamically reconfigures the system, switching from active senders to standby senders to improve the global performance of the system. A key contribution is that senders are selected also taking into account the underlying network, including the topology. The approach proposed in the present paper is different from pure P2P strategies because the server remains responsible for eventually sending the stream if it is not received by clients from its peers up to a certain threshold in time. Furthermore the server still interacts with all clients at least once every at every round in which peer agreements are established.

Several strategies based on BitTorrent have appeared. References [2, 3] describe modified versions of the BitTorrent protocol for continuous stream distribution; reference [5] also presents a system that is similar to BitTorrent but employs network coding on the stream. The work in [10] describes adaptations to the BitTorrent protocol itself so that it can support live streaming.

Another common approach is to rely on a multicast tree for delivering the stream, such as in [6]. Some strategies such as [1] focus on VoD (Video on Demand), in which the user can execute functions on the stream such as pausing or fast forwarding. Reference [4] introduces a pure P2P system based on BitTorrent for VoD.

A number of papers on the performance of P2P streaming systems have been presented. In [8] the Skype telephony protocol is evaluated. In [11] the focus is on the evaluation of VoD systems that employ a peer-to-peer strategy. Finally, in [13] a monitoring system based on buffer maps is proposed as the basis with which the quality of network-wide streaming can be inferred.

More recent related work includes other hybrid strategies, but they often involve either CDN (Content Distribution Networks) [14] or clouds [17]. In [15] the authors investigate replica placement in a hybrid CDN-P2P network. A novel replication strategy is proposed that employs a dynamic mechanism to optimize the number of replicas and their placement. In [24] the authors aim at developing a cost-effective hybrid CDN-P2P architecture, in the sense that as little as possible CDN resources are employed guaranteeing that the media quality is not compromised. Peers are assumed to have a limited contribution capacity, and thus they do contributed, but in a fair and limited basis. Several of these hybrid CDN-P2P multimedia systems are commercial. LiveSky [16] is an example. The authors describe as one of their main concerns the integration with existing CDN infrastructures. The main problem of these strategies is that employing a CDN is expensive and depends on the provider network bandwidth and load. Furthermore the CDN must have its service located close to the clients, which usually are spread across multiple administrative domains. Note that this solution also poses an additional complexity to set up, not only because contracts must be signed before the system is used, but also because depending on the location of the potential users they may be hard to configure.

In [17] the authors investigate different aspects of cloud-based multimedia services, including processing, storage, and distribution. The authors propose a cloud architecture to provide this type of service. This architecture includes high-performance clusters at the edge of the cloud. The multimedia distribution strategy is similar to that provided by a CDN. In [18] an architecture is proposed that merges private and public clouds and is intended for commercial use. The architecture includes APIs for the cloud, containing built-in functions for automatic QoS calculation, which permits negotiating QoS parameters. In other works [19, 20] the authors use cloud-based services to adjust the content to mobile users, depending on client device capabilities and network conditions.

Other P2P streaming systems rely on the network topology and/or technology to make streaming decisions. In [21] the authors propose a P2P streaming system focused on mobile ad-hoc networks, targeting environments that present a high churn rate, frequent mobility and multiple-hop wireless links. In [22] the authors introduce a system targeted at scenarios with peers distributed across several different ISPs. The proposed system uses the ISP network information to favor intra-ISP peering, reducing the inter-ISP network traffic.

V. CONCLUSIONS

In this paper we presented a hybrid peer-to-peer and client-server multimedia streaming distribution system. Clients receiving the content establish forwarding agreements for the parts of the stream they do not have, avoiding the concentration of network traffic on the content source. The proposed hybrid system allows the stream source server to send data directly to the clients that are unable to retrieve it from other peers. An experimental evaluation shows that the system provides an expressive reduction of the bandwidth requirements at the server and that the system is able to support the failure of nodes without playback interruption for nodes that remain fault-free

As future work, we plan to execute large scale experiments using larger number of peers and higher data rates. We also plan to run experiments using different network topologies, including clients with asymmetric bandwidth. Also, new policies can be implemented for choosing the clients that will receive the forwarding agreements, adapting to the network topology. Finally, we can run experiments that compare the proposed approach with peer-to-peer systems that are based on multicast trees for distributing the content.

ACKNOWLEDGEMENTS

This work was partially supported by the Brazilian National Research Council (CNPq) grant 309143/2012-8.

REFERENCES

[1] B. Cohen, “Incentives Build Robustness in BitTorrent”, *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[1] M. Zhang, L. Zhao, Y. Tang, J.G. Luo, and S.Q. Yang, "Large Scale Live Media Streaming over Peer-to-Peer Networks through the Global Internet," *ACM Workshop on Advances in P2P Multimedia Streaming*, 2005.

[2] Helix Community, http://www.helixcomunity.org/. Accessed in April 2014.

[3] C. Dana, D. Li, D. Harrison, and C. N. Chuah, "BASS: BitTorrent Assisted Streaming System for Video-on-Demand," *IEEE Workshop on Multimedia Signal Processing*, 2005.

[4] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr, “Overcast: Reliable Multicasting with and Overlay Network,” *Fourth Symposium on Operating System Design and Implementation (OSDI)*, 2000.

[5] C. Gkantsidis and P. Rodriguez, “Network Coding for Large Scale Content Distribution,” *IEEE/INFOCOM 2005*, 2005.

[6] S. Androutsellis-Theotokis and D. Spinellis, “A Survey of Peer-to-Peer Content Distribution Technologies,” *ACM Computing Surveys*,

2004.

[7] S. Baset, and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," *IEEE International Conference on Computer Communications*, 2006.

[8] X. Liao, H. Jin, Y. Liu, LM. Ni, and D. Deng, "AnySee: Peer-to-Peer Live Streaming," *IEEE International Conference on Computer Communications*, 2006.

[9] A. Valvianos, M. Hiofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications," *IEEE International Conference on Computer Communications*, 2006.

[10] B. Cheng, X. Liu, Z. Zhang, and H. Jin, "A Measurement Study of a P2P VoD System," *International Workshop on Peer-to-Peer Systems*, 2007.

[11] S. Xie, B. Li., G.Y. Keung, and X. Zhang, "CoolStreaming: Design, Theory, and Practice," *IEEE Transactions on Multimedia*, Vol. 9 No. 8, 2007.

[12] X. Hei, Y. Liu, and K.W. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *IEEE Journal of Selected Areas in Communications*, Vol. 25, No. 9, 2007.

[13] I. Ha, S. Wildman, J. Bauer. "P2P, CDNs, and hybrid networks: the economics of Internet video distribution," *Int. Telecommunications Policy Rev.*, Vol. 17, No. 4, pp. 1-22, 2010.

[14] M. Garmehi, M. Analoui, M. Pathan, R. Buyya. "An economic replica placement mechanism for streaming content distribution in Hybrid CDN-P2P networks," *Computer Communications*, Vol. 52, No. 1, pp. 60-70, 2014.

[15] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, B. Li. "Livesky: enhancing CDN with P2P," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*. Vol. 6, No. 3, pp. 1-16, 2010.

[16] W. Zhu, C. Luo, J. Wang, S. Li. "Multimedia Cloud Computing," *IEEE Signal Processing Magazine*, Vol. 28, No. 3, pp. 59-69, 2011.

[17] I. Trajkovska, J. S. Rodriguez, A. M. Velasco. "A novel P2P and cloud computing hybrid architecture for multimedia streaming with QoS cost functions," *Proceedings of the international conference on Multimedia. ACM*, pp. 1227-1230. 2010.

[18] S. Chang, C. Lai, Y. Huang. "Dynamic adjustable multimedia streaming service architecture over cloud computing," *Computer Communications*, Vol. 35, No. 15, pp 1798-1808, 2012.

[19] C. Lai, H. Chao, Y. Lai, J. Wan. "Cloud-assisted real-time transrating for http live streaming," *Wireless Communications, IEEE* , Vol. 20, No. 3, pp.62,70, 2013

[20] J. Kuo, C. Shih, C. Ho, Y. Chen. "A cross-layer approach for real-time multimedia streaming on wireless peer-to-peer ad hoc network," *Ad Hoc Networks*, Vol. 11, No. 1, pp. 339-354, 2013.

[21] N. Magharei, R. Rejaie, I. Rimac, V. Hilt, M. Hofmann. "ISP-Friendly Live P2P Streaming," *IEEE/ACM Transactions on Networking*, Vol. 22, No. 1, pp. 244-256, 2014.

[22] M. Hefeeda, A. Habib, B. Botev, D. Xu, B. Bhargava, "Promise: Peer-to-Peer Media Streaming Using Collectcast," *The 11th ACM International Conference on Multimedia* 2003, Berkeley, CA, 2003.

[23] D. Xu, S. S. Kulkarni, C. Rosenberg, H.-K. Chai, "Analysis of a CDN-P2P Hybrid Architecture for Cost-Effective Streaming Media Distribution," *Multimedia Systems*, Vol. 11, No. 4, 2006.