LEMP: Lightweight Efficient Multicast Protocol for Video on Demand

Panayotis Fouliras, Spiros Xanthos, Nikolaos Tsantalis, Athanasios Manitsaris
University of Macedonia
Egnatias 156,
540 06 Thessaloniki, Greece
{pfoul, it0187, it0157, manits} @uom.gr

ABSTRACT
In this paper, we propose a new scalable application-layer protocol, specifically designed for data streaming applications with large client sets. This is based upon a control hierarchy of successive levels for the clients, has minimal overhead with constant number of messages per client, and is robust to client and network failures, making it suitable for wireless environments. The video server bandwidth utilization is also significantly reduced. We present an analysis and simulation results, showing that LEMP is near optimum in terms of performance.

Keywords
Video-on-Demand, multimedia, application-layer, protocol, chaining

1. INTRODUCTION
The advance of communication technology has spawned a large number of services, previously too expensive or impossible to access for the average user.

However, there are still services that require a large amount of bandwidth and the associated cost has not allowed them to achieve widespread use. Video-on-Demand (VoD) is one such service, composed of at least three entities, namely the video server, the customer's client computer and the intermediate network.

There are several conflicting requirements: Less bandwidth per video means more video streams (*virtual channels*) available per video server. Also, due to the nature of the Internet, it is not easy to avoid jitters by establishing isochronous virtual channels between the server and the client. The memory space occupied by a single video is often larger than the available memory on the client. Moreover, in a typical video service the client expects additional capabilities, such as fast forward, pause and rewind, not to mention interactive video.

Many researchers have worked on these issues the past few years and have proposed several interesting ideas, such as patching [1], skyscraper broadcasting [5], bandwidth skimming [6], SVD [7] and greedy disk-conserving broadcasting [11], all of which try to minimize the duration of broadcast or the number of additional server channels for the same video. Others try to utilize client memory in a simple but very hard to implement way (e.g. chaining [4]).

Other proposals have slightly different goals or assumptions, such as Variable Bit Rate (VBR) broadcasting [10], [12], [13], [14], lossy network environments [2], or are extensions of existing techniques for distributed VoD service [8].

Due to the nature of the Internet, multicasting is not a realistic option in a large scale. Hence, the key problem we address in this paper is how to minimize overall video server network bandwidth, while simultaneously maintaining the latency of service to client requests minimal, for popular videos, using unicasting. We assume client bandwidth slightly higher than the playback rate for incoming traffic. The number of video channels per client is upwards bounded by $b \geq 1$.

The clients can be of any type in terms of computing power and buffer size. Furthermore, the clients can join or leave a broadcast at any time, either by choice or due to the problematic nature of the network. These assumptions are quite realistic both for workstations and mobile clients (e.g. PDAs).

Based on these assumptions, we propose the use of clients both as passive receivers of videos, as well as partial video servers for other clients through the use of their buffers. This is not a novel approach [4], [9], [15], 16], but the organization of the underlying access control mechanism is, since we propose a semi-hierarchical overlay, against the tree-like arrangements proposed by other researchers (e.g., [16]). We also propose a different mechanism for the join and departure of clients, with emphasis on fault-tolerance and self-recovery and the reduction of control overhead and simultaneous server channels.

The rest of the paper is organized as follows. In section 2 we formulate the problem. In section 3 we present our proposal (LEMP). In section 4 we analyze its performance, showing that LEMP is near optimum, scalable and resistant to failures. Our conclusions follow in section 5.

2. PROBLEM FORMULATION
For simplicity we assume one video server S, containing a set of videos, with D the duration of each video. C is the set of all clients, with C_m the set of clients requesting the same video m up to a certain time point. The cardinality of these sets is n_c and n_{cm}, respectively. The buffer size available at each client, expressed in playing time, is $d < D$.

There is no limit on the amount of clients which can make requests, except that only one request per client may be outstanding or served at any moment. Client requests are denoted by r_{jm}, where $1 < j \leq n_{cm}$ and may arrive at any time. The server tries to serve them at discrete successive time points, t_i, t_{i+1}, …, so that $t_{i+1} - t_i \leq t_w$. The latter (t_w) is a constant that depends on the amount of time a client is willing to wait for service, before it decides to withdraw its request.

The goal is to utilize as much of the available memory and bandwidth of the clients that are already being served by the server. Therefore, if at least one client receives the same video at successive time points with time difference $t_w < d$, it is possible to form a "chain" of successive video streams that serve all client requests up to the present time. Thus, at some time point t_i, there are n_{cm} clients for the same video grouped in i levels, namely L_1, …, L_i. The server is always at level L_0 and broadcasts to the clients of level L_1.

Each client has only one channel for video reception and only b channels for video broadcasting at slightly larger than the playback rate. These are the *data* or *video* channels.

All clients use unicasting to broadcast video; hence, b is some positive integer, depending on the upper limit of video channels per client.

It is possible for clients to fail, withdraw or operate in a lossy network environment. Consequently, such pipelines would break and the system should somehow try to remedy the situation. Therefore, a solution must satisfy the following characteristics:

- Be simple and fast to adapt quickly to the changing circumstances
- Minimize simultaneous video server channels for the same video
- No client must wait longer than t_w for service
- Manageable network traffic per client
- Speedy recovery for client or network failures
- Minimal requirements regarding client computing power

3. PROPOSED SOLUTION

3.1 The Control Hierarchy

The LEMP protocol arranges clients into a hierarchy of i levels, where $0 < i \leq \lceil D / t_w \rceil$. The main operation creates and maintains this hierarchy.

Contrary to other proposals [16], the data and control paths are different: The data path follows a tree-like arrangement where a client at level L_i provides a set of up to b unicast streams to a group of clients at level L_{i+1}. These streams do not have to be synchronized; they may be transmitting different parts of the buffer content.

The control path is twofold: All the clients at level L_i are organized in a star-like structure. One of the clients at each level L_i is the *Local Representative* (LR_i). This client together with other *LRs* from the rest of the levels communicates with the video server forming a control topology of a star, keeping overall communication minimal. The rest of the clients at level L_i maintain and exchange control information with their respective LR_i. This arrangement allows quick response in the event of client failures.

3.2 Arrangement of Clients

From time t_i to $t_{i+}t_w$ the server receives client requests for the same video, which it groups into the level L_i. The arrangement is not random; the end-to-end latency of the path between a client and the server is used as criterion to select the *Local Representative* for this level (LR_i) and all other clients are placed closest to it. The second closest client is selected as the *Backup Local Representative* (BLR_i).

This happens because LR_i is the only client for level L_i communicating with the server under normal conditions; hence, an effort is made to select the one closest to the server in terms of end-to-end latency. Similarly, the *BLR* is selected in order to replace quickly a failed *LR*.

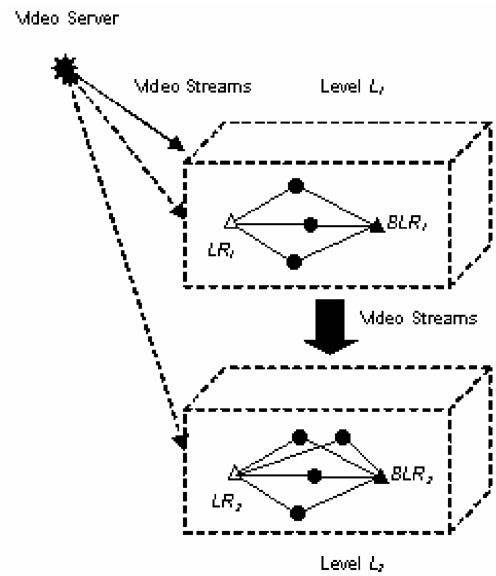


Figure 1. Hierarchy under LEMP

The *LRs* and the server form a star with the server at the center. The total communication load on the server for this set of clients is relative to maximum number of levels $\lceil D / t_w \rceil$ This arrangement has the advantage that the server can detect *LR* problems quickly and is more reliable than any other scheme with message hopping from *LR* to *LR* until the server is reached [4], [15], [16]. After all, *LRs* are clients that can withdraw at any moment without notice. This hierarchy is depicted in Fig. 1, for the first two levels of clients.

Assuming there are n_{i-1}, n_i and n_{i+1} clients at levels L_{i-1}, L_i and L_{i+1} respectively, the server divides the clients at level L_i in n_{i-1} equal-sized subgroups, if possible, assigning each subgroup to a client at level L_{i-1}. This, progressively, forms a tree structure, used for video streams.

Finally, each client v at level L_i communicates for control purposes with its respective LR_i, requiring only one control message under normal conditions.

3.3 Protocol Operations

Under LEMP there are three phases for any client: *Join, Work* and *Leave*.

3.3.1 Join Phase

Under *Join*, a client v requests a video from the server at some time $t_{jv} \in (t_{i-1}, t_{i-1}+t_w)$ and $1 \leq j \leq n_i$. Each request includes a time-stamp set by the server indicating t_{jv}. The server gathers all requests R_j and calculates the end-to-end latency between each client and itself, forming an ascending sorted list of clients as follows: The server sends a special probe message to each of the above clients and uses the respective reply message from the client to calculate the latency.

```
Procedure Join(t_i)
create a list l_i of all pending client requests R_i up to t_i
sort l_i in ascending order
    if ∃ client v at level L_{i-1} then
        calculate L_i
        send L_i to LR_i and BLR_i
        for client j=1 to n_i
            send to j the identity of
                    LR_i, BLR_i LR_{i-1}, BLR_{i-1}, parent(j)
        endfor
        form up to n_{i-1} subgroups of the n_i clients
        for client k=1 to n_{i-1}
            send to k the members_id of the kth subgroup
        endfor
    else
        schedule a new server channel for l_i
    endif
end
```

Figure 2: Basic Form of the Join of Clients

Next, the server determines whether there is already a broadcast to at least one client, currently receiving the first part of the video (level L_{i-1}). If none exists, a new broadcast is scheduled by the server; otherwise, the new level L_i and identity of LR_i are determined.

This information is sent to the LR_i and BLR_i of level L_i. Each client v_i only receives the identity of LR_i, BLR_i LR_{i-1}, BLR_{i-1} and its parent. Thus, the size of these messages is constant.

Finally, the server divides the clients at the new level L_i in up to n_{i-1} subgroups and sends this information to each parent at level L_{i-1}, and its child at level L_i. Thus, a forest of trees is formed, augments the data path (Fig. 2). If possible, LR_i, BLR_i and their neighbors are not assigned any children due to their additional administrative load.

This step concludes the *Join* phase. By now, each client at level L_i has the following information:

- An identity in L_i
- The identity of LR_i, BLR_i at its level
- Its parent in the data path
- It knows whether it is the LR_i or BLR_i for level L_i
- It knows the LR_{i-1} and BLR_{i-1} for its parent level

In addition, each client at level L_{i-1} knows its children in L_i.

3.3.2 Work Phase

During this phase, the clients at level L_{i-1} broadcast the video in their buffers to their respective children at level L_i. Apart from data, control information is exchanged in order to detect any possible problems.

First, all clients send periodically an *Alive* message to their respective *LR*. If no such message arrives to *LR* from any client v within a certain time interval t_δ, then v is considered to have failed. Each of these *Alive* messages includes the client's identity and load. Thus, a list of potential parents is formed, sorted according to their load, in case of regular parents fail.

Finally, LR_i periodically exchanges a special *LRAlive* message with the BLR_i and the video server, containing all information updates regarding the state of clients at the particular level. This is sent so that either can detect potential failure of its peer and synchronize control information.

3.3.3 Leave Phase

There are two cases: Under the first case, a client v that wishes to withdraw sends a *Quit* message to LR_i and also to its parent p and children. Under the second case, a client v no longer broadcasts video to its children and does not send *Alive* messages to its *LR*.

In both cases the p removes v from the list of its children. Furthermore, p stops broadcasting video to v. The *LR* updates its information, accordingly.

3.3.3.1 Orphans and Recovery

There are two problems: The first is that the children (of parent v) at level L_{i+1} are now *orphans*. Since they know LR_i, they send an *OJoin (Orphan Join)* message to LR_i. LR_i determines potential parents and replies by sending *ODirect (Orphan Direct)* messages, directing them to the appropriate new parents.

If LR_i has failed, the orphans try the same process with BLR_i.

If no new parent is found or both LR_i and BLR_i have failed at the same time, the orphans contact the server, which schedules a new broadcast to the orphans. Then the process continues as described above.

3.3.3.2 Uncertainty of Client Failures

The control communication pattern is fairly distributed and unreliable. It is possible that no *Alive* message by v reaches its respective *LR* within the time interval $t_\delta/2$. This is a partial failure: One or more network links have failed to deliver the *Alive* message, but client v operates properly.

In this case the *LR* simply deletes v from its list of potential parents, although it keeps waiting for *Alive* messages for another time interval $t_\delta/2$ (a total of t_δ). It is, thus, hoped that the link with v will operate again soon, in which case v is re-instated as an potential parent by the *LR*; otherwise, v is permanently deleted from its list.

4. PERFORMANCE ANALYSIS

As described earlier, there are at most $O(\lceil D/t_w \rceil)$ possible time-slots at which client requests may belong, requiring a separate video channel for their service.

The criteria for good performance are:

- Minimization of messages per client
- Minimization of server channels for the same video
- Minimization of the overhead for failure recovery
- Minimization of time for error recovery

Under LEMP, the number of video server channels for a video m range from one (optimal case when at least one client per time slot) up to $\lceil D/2s \rceil$ (worst case when client requests arrive every two time slots). The overhead to assign the respective subgroups of clients, *LRs* and *BLRs*, under normal conditions, is derived from procedure *Join* (Fig. 2).

With n_i clients at level L_i, two messages of length $O(n_i)$ must be sent from the server to LR_i and BLR_i; another $O(n_i)$ messages of constant length must be sent to the ni clients; finally, $O(n_i\text{-}1)$ messages of length $O(b)$ must be sent to the parents of these clients. Since b is usually very small and no immediate acknowledgement is required by the clients, the only overhead is the time to transmit these messages. On average, this overhead time must be smaller than t_w plus the average time required for video streams to be transmitted from the parents to the new clients. The former is relatively easy to estimate, whereas the latter depends on the current state of the underlying network. If this condition is not met, failures are considered to take place.

In case of any parent failure at level L_i, the worst case for the amount of messages per orphan is 3, namely to *LR*, *BLR* and the server. This is exceptionally low compared to similar figures reported elsewhere [18].

The only exception to the analysis above is the *LR* at each level *i*. This has a higher burden than the rest of the clients, since it has to receive the initial level information by the server. It also needs to select and propose new parents to orphans. In the worst case these messages (*ODirect*) are as many as the clients at the next level *i*+1, the *BLR* and the server, which adds up to $O(n_{i+1}+2)=O(n_{i+1})$ messages.

All clients are $n_{cm}=\sum_{k=1}^{i} n_k$ for *i* levels. In the worst case the clients at level L_i are:

$$n_i \le b^{(i-1)} * n_1$$

Hence, the messages for each LR_i in the worst case are:

$$O(b^i * n_1)$$

Finally, the new levels are initially calculated by the server and thereafter in two extreme cases: Either both LR_i and BLR_i or the complete level L_i of clients have failed.

In the first case the server selects two of the remaining clients of that level as the LR_i and BLR_i. In the second case the server assumes the responsibility of broadcasting the video to all the clients of level L_{i+1}.

Of course, if all parents at every second level fail, the server reverts to a batching strategy, with $\lceil D/2t_w \rceil$ channels to accommodate the orphans, although not for the full duration of the video [3].

Based on the discussion above, we see that for many clients, additional server channels are required only in the case of massive client faults at the same level. If only partial faults take place and the clients are evenly distributed at each level, only as many as n_1 (clients at level L_1) video streams are required.

In order to better evaluate LEMP's performance we used the GT-ITM generator [20] to create 10,000 node transit-stub as our underlying network topology. Routing is determined using the shortest path algorithm. We assumed that the video play-back rate was constant. Links were considered to have adequate bandwidth.

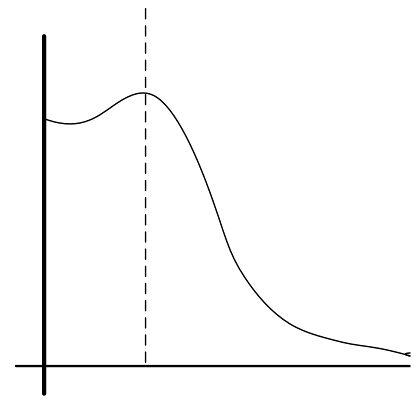


Figure 3. Probability Distribution for *b*

We performed a simulation with the following characteristics:

- The popular video *m* has duration D_m=100 minutes, with t_w=3 minutes. The assumption for t_w is realistic and far better than in [19].
- The client requests arrive at the server, following a Poisson distribution with λ=10 for the duration of t_w.
- The total number of clients is n_{cm}=1,000; each client is randomly placed in the network.
- Each client has only one incoming link for video reception; the number *b* of outgoing links was determined according to four different scenarios: $1 \le b \le 3$ and $b \in [0, 4]$ under the Gaussian probability distribution (see Fig. 3).
- There is a probability $0 \le p_f \le 0.5$ that clients will quit or fail.

The metrics used for LEMP evaluation are:

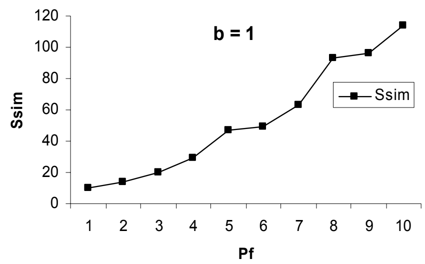
- The number of maximum simultaneous server channels S_{sim} over constant and variable maximum values for *b*; this should be minimal.
- The failure probability (p_f) impact on the performance of LEMP, measured as a function of S_{sim} over p_f.
- The total network load in control messages.
- The worst time t_{res} for an orphan to resume video reception.

The first three metrics are calculated by the simulation. The calculation for the last metric is straightforward:

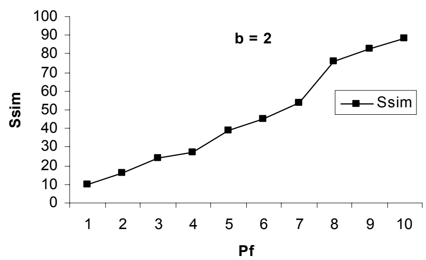
$$t_{res} = t_p + t_{LR} + t_{BLR} + t_S,$$

where t_p is the time for a client to realize that it has become an orphan, t_{LR} the time to request a new parent from the *LR* and not get a reply, t_{BLR} is the same for the *BLR* and t_S the same for contacting the server and start receiving video from it. In our case $t_{res} < t_w \approx 3$ minutes, which is a realistic value.

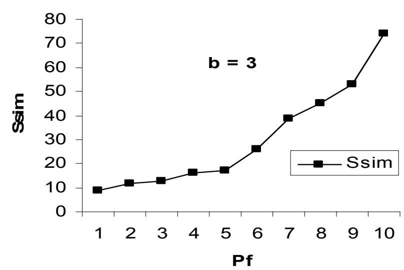
From Fig. 4 we see that the maximum simultaneous server channels are, on average, very few compared to the total number of clients – approximately 50 for 1,000 clients. This holds even for a significant percentage of failures (p_f = 50%) and improves as *b* increases. Such performance results in an exceptionally reduced load on the video server, which represents the only potential bottleneck in such systems. Comparing to pure unicasting (1,000 streams) this represents a 20 times improvement and is quite realistic, since 50 channels with an average of 150kbps/channel would require a total of 7.5Mbps.



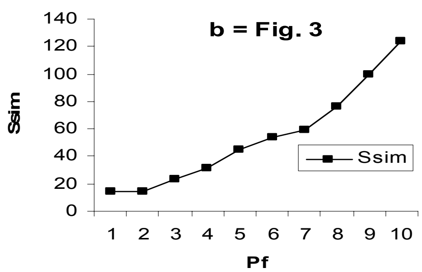
(a)



(b)



(c)



(d)

Figure 4. Max. Simultaneous Server Channels (S_{sim}) over *b*

Furthermore, if multicasting were applied on successive batches of client requests, we would need as many video channels as the batches. This is $\lceil D/t_w \rceil = 34$ for our example and represents the optimal case. Our results compare favorably, with only 1.5 times the optimum number of channels.

In addition, LEMP can operate with a very low number of broadcasting channels per client. This is emphasized in Fig. 4(d), where a high proportion of clients does not broadcast content to others at all. Such an assumption is realistic, given that there can be clients which cannot or do not wish to participate fully in such an arrangement.

Furthermore, Fig. 4(d) answers one more question: What happens under realistic, varying values for *b*, as the number of failures increases either due to faults of the parents or due to increased delays appearing in the underlying network? The answer is that LEMP operates efficiently up to the point where these failures become very high ($p_f \geq 70\%$).

For such an evaluation to be complete one must measure the load imposed on the network by LEMP. This load is approximately the same for different values of *b*, when the percentage of failures p_f is very small.

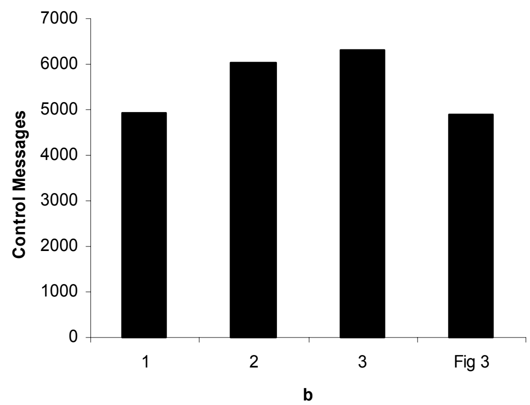


Figure 5. Control Messages over *b* with $p_f = 40\%$

However, as can be seen in Fig. 5, the network load increases with *b*: When *b* is high, a client failure creates more orphans, which cause the generation of more messages until they are assigned to new parents. Therefore, we see that when *b* follows the distribution of Fig. 3, LEMP is both efficient in terms of network and server load, even with a significant percentage of failures.

5. CONCLUSIONS AND FUTURE WORK

We have proposed LEMP, a new multicast application layer protocol for VoD, utilizing the available buffer of clients, in a lossy environment, leading to better server and overall network utilization. LEMP imposes only one control message per client under normal operation and up to three messages per client when its parent fails.

We ran a detailed simulation under different parameters, which showed that the maximum simultaneous number of server channels is very low, even under significant percentage of client failures, at approximately 1.5 times the optimal case (pure multicasting). This is one of the most important metrics, since the video server represents a potential bottleneck. Furthermore, the network overhead in terms of recovery control is very low.

Also, LEMP is scalable, quite robust and relatively easy to implement, since it is less complex or demanding for clients compared to other proposals. An added advantage is that operations such as Fast-Forward and Rewind can easily be included, due to the inherent nature of the protocol. Work is in progress to incorporate this feature as well as implement it in a real network environment.

6. REFERENCES

[1] Kien A. Hua et al, Patching: A Multicast Technique for True Video-on-Demand Services. ACM Multimedia 1998, 191-200.

[2] A. Mahanti et al, Scalable On-Demand Media Streaming with Packet Loss Recovery. SIGCOMM'01, August 2001, 97-108.

[3] Jack Y. B. Lee, UVoD: An Unified Architecure for Video-on-Demand Services. IEEE Communications Letters, Vol. 3, No. 9, September 1999, 277-279.

[4] S. Sheu, K. Hua, W. Tavanapong, Chaining: A Generalized Batching Technique for Video-On-Demand Systems. Proceedings of ICMCS'97, 1997, 110-117.

[5] K. Hua, S. Sheu, Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. ACM SIGCOMM'97, 1997, 89-99.

[6] K. Hua, Y. Cai, S. Sheu, Patching: A Multicast Technique for True Video-on-Demand Services. ACM Multimedia'98, 1998, 191-200.

[7] D. Eager et al, Optimal and Efficient Merging Schedules for Video-on-Demand Servers. ACM Multimedia'99, 1999, 199-202.

[8] Min-You Wu et al, Scheduled Video Delivery for Scalable on-Demand Service. ACM NOSDAV'02, 2002, 167-175.

[9] James Z. Wang, Ratan K. Guha, Data Allocation Algorithms for Distributed Video Servers. ACM Multimedia 2000, 456-458.

[10] C. Loser et al, Distributed Video on Demand Services on Peer to Peer Basis. Intl. Workshop on Real-Time LANS in the Internet Age (RTLIA 2002).

[11] D. Saparilla, K. Ross, Periodic Broadcasting with VBR-Encoded Video. Proceedings of the IEEE Infocom, 1999, 464-471.

[12] Lixin Gao et al, Efficient schemes for broadcasting popular videos. Multimedia Systems, Springer-Verlag, Vol. 8, 2002, 284-294.

[13] M. Tantaoui et al, Interaction with Broadcast Video. ACM Conference on Multimedia (SIGMM 2002).

[14] Yanping Zhao et al, Efficient Delivery Techniques for Variable Bit Rate Multimedia. Proceedings of the MMCN 2002.

[15] Kien Hua, JungHwan Oh, Khanh Vu, An adaptive video multicast scheme for varying workloads, Multimedia Systems. Springer Verlag 2002, Vol. 8, 258-269.

[16] Kien Hua et al, Overlay Multicast for Video on Demand on the Internet, ACM SIGAPP (SAC), 2003.

[17] S. Banerjee et al, Scalable Application Layer Multicast, ACM SIGCOMM, 2002.

[18] D. A. Tran et al, ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming, IEEE INFOCOM, 2003.

[19] Yang Guo et al, P2Cast: Peer-to-peer Patching Scheme for VoD Service, ACM WWW, 2003.

[20] E. Zegura et al, How to model an internetwork, Proc. IEEE Infocom, April 1996.