Resource Usage Prediction for Application-Layer Multicast Networks

Genge Béla and Haller Piroska
Department of Electrical Engineering
"Petru Maior" University of Târgu Mureş
Târgu Mureş, Mureş, Romania, 540088
{bgenge,phaller}@engineering.upm.ro

***Abstract*—We propose a memory and CPU usage prediction model for application-layer multicast networks. The predicted values are used in the distribution of end-hosts to overlay-hosts. Such a distribution enables us to limit the maximum resource consumption for a given node, leading to an efficient utilization of overlay-hosts and, finally, to an increased overall performance of the system. The model parameters are determined using training sets gathered from measuring resource consumption while distributing end-hosts to overlay-hosts. Both end-hosts and overlay-hosts are running on PlanetLab nodes, a testbed that provides researchers a real environment where nodes can become unreachable, network bandwidth can fluctuate and node processing capabilities can drop dramatically. Using the determined parameters, we show that our proposal can be used to estimate memory and CPU usage and to efficiently distribute end-hosts to overlay-hosts.**

***Keywords*-application-layer multicast; predictive; MIMO-model; memory; CPU;**

I. Introduction

Because of technical and administrative issues that IP multicast networks had to deal with in the past [1], they have been replaced by application-layer multicast [2] networks. The popularity of application-layer multicast (i.e. ALM) comes from its ease of implementation over the already existing and well-established Internet protocols that require little or no modifications in existing routers. One of the main applications of ALM is in the field of group communication.

For several years now group communications have been receiving significant attention from both the industry and scientific communities [3], [4]. The main goal of group communication is to enable the exchange of information between group members that can be located across the entire globe. Historically speaking, the first multicast applications were implemented over the IP layer, also known as *IP multicast* [5]. However, after nearly a decade of research in the field of IP multicast, it was never fully adopted because of several technical and administrative issues [1].

Later, there have been several proposals for other multicast implementations that would be easier to deploy over the already existing and well-established Internet protocols and would require little or no modifications in existing routers. Such a survey of existing solutions was provided by El-Sayed et al [2].

One of the directions that has been clearly adopted over the last few years is ALM, which implements the multicast functionality at the application layer. The main goal of ALM is to construct and maintain efficient distribution structures between *end-hosts* (i.e. EHs). These structures are constructed using an *overlay* network providing the necessary infrastructure for data transfer between end-hosts. The overlay must ensure a scalable and reliable group communication in a dynamic environment where end-hosts can join and leave sessions at will.

Existing research on ALM focuses on constructing the overlay network using the closest node principle [6], [7], [8], head of structure principle [9], [10], [11] or simply using a random distribution [12]. Their main goal is to provide an overlay structure that optimizes a given set of criteria. However, as shown in our previous work [13], data transfer between EHs using the constructed overlay is also affected by the distribution of EHs to *overlay-hosts* (i.e. OHs). Data transfer is thus affected by network latency that must be measured and used as a reference by a central distribution node, called *monitoring-host* (i.e. MH) to efficiently distribute EHs.

In our previous work [13] we have used the measured latency between each EH-OH pair for an efficient EH distribution. However, we have not considered other key factors such as memory or CPU usage that can affect, over time, the overall performance of sessions. In this paper we use a model predictive approach for estimating memory and CPU usage in the distribution of EHs. By doing so, we can limit the number of distributed EHs to each OH based not only on a maximum EH count but based on the consumed resources. In order to validate our proposal we use PlanetLab [14], a testbed that provides researchers a real environment where nodes can become unreachable, network bandwidth can fluctuate and node processing capabilities can drop dramatically.

The paper is structured as follows. In Section II we provide a brief overview of the overlay topology. Our proposed predictive model is described in Section III. In Section IV we provide an evaluation of our proposal. We end with a conclusion and future work in Section V.

II. Overlay Topology and Distribution Algorithms

The considered overlay topology is shown in Fig. 1, where we have illustrated the presence of 3 host types:

- End-hosts (i.e. EH);
- Overlay-hosts (i.e. OH);
- Monitor-hosts (i.e. MH).

The complete graph model has several advantages over hierarchical ones. First, there is no need for implementing complex routing algorithms [15], which greatly simplifies the implementation and functionality of the overlay. Second, maintaining routing tables is not more complex than maintaining connections with all the other nodes. As a downside of this topology, there is a large number of connections that must be maintained, which grows exponentially with the number of OHs. However, the simplicity of the routing algorithms between OHs makes this topology a great candidate for using it as a leaf component in hierarchical topologies [16], [17].

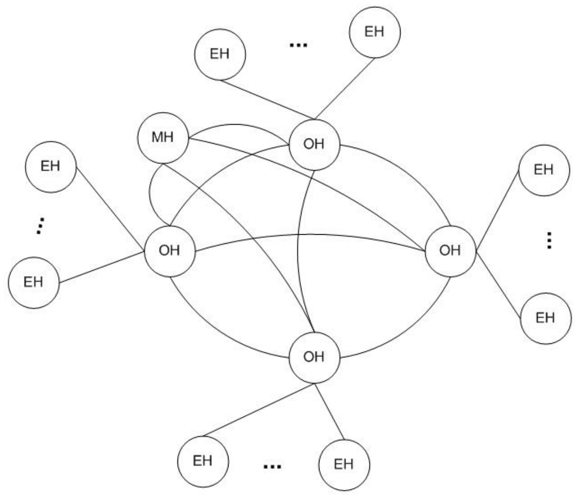

Figure 1. Multicast topology

EHs are the producers and consumers of data transferred by the overlay. MHs are used to monitor the load of each OH (e.g. memory and CPU usage) and to distribute EHs to the least loaded OH. We identify two types of EHs: *measuring EHs* and *streaming EHs*. *Measuring EHs* denote EHs that connect to OHs in order to measure network latency. These measurements are then sent to the MH that runs the distribution algorithm presented in our previous work [13]. The MH then sends back the selected OH to which EHs connect and request resources. Through this last step, EHs become *streaming EHs*.

In this paper we provide a predictive model for resource allocation. The implementation of this model is used in the process of distributing EHs to OHs based on resource usage criteria. This way we are able ensure an efficient utilization of OHs and, finally, to an increased overall performance of the system.

As mentioned before, the role of the MH is to distribute EHs to OHs. The algorithm we used in our previous work to distribute EHs relies on connection latencies measured by EHs to each OH (i.e. Alg. 1). These are then sent to the MH that chooses an OH such that the overall graph latency has a local optimal value. We use a local optimal value instead of a global one because from our simulation results this approach runs in the order of milliseconds, while the global optimal algorithm runs in the order of minutes for several thousand EHs and several hundred OHs.

Algorithm 1 EH distribution algorithm - current version

```
Let $Req$ be the set of OH - measured latency pairs $(oh, l)$
Let $Chosen_{OH}$ be the set of chosen OHs
Let $eh_{OH} = OH_1$ be the OH chosen for this EH
Let $l_{min} = MAX\_VAL$ be the minimal computed latency

{Search for the OH that minimizes the overall graph latency}
for all $(oh, l) \in Req$ do
  Let $l = Latency(Chosen_{OH} \cup \{oh\})$
  if $l < l_{min}$ then
    $eh_{OH} = oh$
    $l_{min} = l$
  end if
end for

{Save the chosen OH for next EH distributions}
$Chosen_{OH} = Chosen_{OH} \cup \{eh_{OH}\}$
```

Alg. 1 does not consider previously distributed EHs. This can lead to overloaded OHs and to an overloaded overlay that can significantly affect the performance of group communications. The chosen OH (i.e. $Chosen_{OH}$) for each EH must be selected in such a manner that we also take into consideration previous distributions that can later require more memory and CPU, also affecting the round-trip latency.

Alg. 2 is an improved version of Alg. 1 because it also considers memory and CPU usage of selected OHs in the distribution process. It additionally uses the totally ordered set $(Min, \leq)$ to store in an ordered manner the OHs that have been found on the way to the minimal latency graph. The elements of Min are pairs of OH and latency values, ordered by the value of the computed latency. Let $Min_i =< eh, l >$, with $Min_i \in Min$. Then $Min_i^1 = eh$ and $Min_i^2 = l$.

The algorithm starts by storing all $< eh, l >$ pairs to the Min set. These represent the possible OHs to which the EH could be distributed. However, instead of choosing the OH with the minimal latency value, as in case of Alg. 1, we continue with searching an OH that provides the

Algorithm 2 EH distribution algorithm - improved version

```
Let $Req$ be the set of OH - measured latency pairs $(oh, l)$
Let $Chosen_{OH}$ be the set of chosen OHs
Let $eh_{OH} = OH_1$ be the OH chosen for this EH
Let $l_{min} = MAX_{VAL}$ be the minimal computed latency
Let $(Min, \leq)$ be a totally ordered set of minimum latency OHs
Let $found = 0$ be a flag denoting that we have found a proper OH

{Search and store the OHs that minimize the overall graph latency}
for all $(oh, l) \in Req$ do
  Let $l = Latency(Chosen_{OH} \cup \{oh\})$
  if $l < l_{min}$ then
    $Min = Min \cup \{< OH, l >\}$
    $l_{min} = l$
  end if
end for

{Search for the OH that also has a minimum MEM and CPU usage}
for $i = \overline{1, |Min|}$ do
  if $Pred_{MEM}(Min_i^1) < MAX_{MEM}$ then
    if $Pred_{CPU}(Min_i^1) < MAX_{CPU}$ then
      $eh_{OH} = oh$
      $found = 1$
      @EndFor
    end if
  end if
end for

{In case we could not find a proper OH (with low MEM and CPU usage), assign the OH with minimal latency}
if found = 0 then
  $eh_{OH} = Min_1^1$
end if

{Save the chosen OH for next EH distributions}
$Chosen_{OH} = Chosen_{OH} \cup \{eh_{OH}\}$
```

smallest latency value and the lowest memory and CPU usage. We use the $Pred_{MEM}$ and $Pred_{CPU}$ functions to denote the prediction of memory and CPU usage in case of an additional EH and the MAX_{MEM} and MAX_{CPU} values as constants to denote the maximum allowed memory and CPU usage.

The flag $found$ is used to signal if there was an OH found that satisfies the mentioned conditions. If no such OH is found, the EH is distributed to the OH that provided the minimal latency value.

In this section we have simply provided an improved distribution algorithm that also takes into consideration the predicted memory and CPU usage. Throughout the next sections we continue with constructing the prediction models and evaluating them on several PlanetLab nodes.

III. Predictive Model

In order to predict memory and CPU usage for a single OH based on the number of connected EHs, we start from the following general *MIMO* (i.e. *Multiple Input Multiple Output*) model:

$$\begin{aligned} mem(t+1) &= a_1 mem(t) + b_1 meh(t) + c_1 seh(t), \\ cpu(t+1) &= a_2 cpu(t) + b_2 meh(t) + c_2 seh(t), \end{aligned} \tag{1}$$

where $meh(t)$ is the measuring EH count and $seh(t)$ is the streaming EH count. The model parameters are determined by minimizing the sum of the squared errors. More formally, we minimize the functions:

$$\begin{aligned} J(a_1, b_1, c_1) &= \sum_{t=1}^{N} (mem'(t+1) - mem(t+1)), \\ J(a_2, b_2, c_2) &= \sum_{t=1}^{N} (cpu'(t+1) - cpu(t+1)), \end{aligned} \tag{2}$$

were $mem'(t+1)$ and $cpu'(t+1)$ denote the measured values with a number of measured values of $N+1$.

In order to quantify the predicted values for a given number of streaming EHs, this model is parameterized:

$$\begin{aligned} P_{mem}^{\alpha}(t) &= a_1 mem(t) + b_1 meh(t) + c_1(seh(t) + \alpha), \\ P_{cpu}^{\alpha}(t) &= a_2 cpu(t) + b_2 meh(t) + c_2(seh(t) + \alpha), \end{aligned} \tag{3}$$

where α denotes the number of additional streaming EHs. By using this model, when the MH receives α requests after update t, it calculates the predicted values based on the previously reported ones with an additional α number of streaming EHs. However, there can be several requests received between t and $t+1$, each one requiring a new set of resources to be allocated. The total number of requests between t and $t+1$ is denoted by β_t. For each new request we also consider previous requests after the last update, resulting the following quantified predicted values after update t:

$$\begin{aligned} Q_{mem}(t) &= \sum_{i=1}^{\beta_t} (P_{mem}^{\alpha_i}(t) - mem(t)) + mem(t), \\ Q_{cpu}(t) &= \sum_{i=1}^{\beta_t} (P_{cpu}^{\alpha_i}(t) - cpu(t)) + cpu(t). \end{aligned} \tag{4}$$

When estimating resource allocation such as memory and CPU, we must also take into consideration that the allocation is done sometime in the future. This means that for the following several updates received from the OH, EHs do not have any resources allocated yet. However, we must consider that some resources will be allocated in the future, but we also must consider that EHs can decide simply not to connect to the designated OH, which translates simply to resources not being allocated. This means that we must use an exponential forgetting factor λ_{mem} and λ_{cpu} for older predicted values. The current estimated resource allocation for memory and CPU is calculated using the following set of equations:

$$\begin{aligned} C_{mem}(t) &= mem(t) + \sum_{k=1}^{t-1} \lambda_{mem}^{t-k} \Delta Q_{mem}(k), \\ C_{cpu}(t) &= cpu(t) + \sum_{k=1}^{t-1} \lambda_{cpu}^{t-k} \Delta Q_{cpu}(k), \end{aligned} \tag{5}$$

where $\Delta Q_{mem}(k)$ and $\Delta Q_{cpu}(k)$ denote the difference between predicted and current values, such that:

$$\begin{aligned} \Delta Q_{mem}(k) &= Q_{mem}(k) - mem(k), \\ \Delta Q_{cpu}(k) &= Q_{cpu}(k) - cpu(k), \end{aligned} \tag{6}$$

with $\Delta Q_{mem}(k) = \Delta Q_{cpu}(k) = 0$ for $k \le 0$. In control theory, the values of λ_{mem} and λ_{cpu} are usually chosen between 0.95 and 0.999 [18].

In real implementations the value of memory and CPU usage can not grow above certain values (e.g. 100%). We incorporate this into our model using δ_{mem} and δ_{cpu}, denoting the maximum possible values for memory and CPU usage, respectively. The real predicted memory and CPU usage are denoted, respectively, by R_{mem} and R_{cpu}, defined as follows:

$$\begin{aligned} R_{mem}(t) &= \begin{cases} C_{mem}(t), & if\ C_{mem}(t) < \delta_{mem}, \\ \delta_{mem}, & otherwise. \end{cases} \\ R_{cpu}(t) &= \begin{cases} C_{cpu}(t), & if\ C_{cpu}(t) < \delta_{cpu}, \\ \delta_{cpu}, & otherwise. \end{cases} \end{aligned} \tag{7}$$

IV. Model Evaluation

Because of its global deployment, nodes from PlanetLab have a heterogeneous hardware setup and a homogeneous software setup. Each node is running a Fedora Core-based Linux with multiple virtual servers that host user programs. Because of the heterogeneity of the hardware, in order to evaluate the proposed model, model parameters $a_1, b_1, c_1, a_2, b_2, c_2$ must be determined for each OH. In this section we present the model evaluation for a single OH, which can be used as a reference for evaluating it on other OHs. In our setup, each OH transmits frames (i.e. JPEG encoded images) of variable size (i.e. between 10000 and 20000 bytes) to each connected streaming EH with a frame-rate of 5 frames/second. Frames are received from another OH that provides the JPEG encoding of the images.

The calculated parameters are given in Table I. By using these values we can predict the next memory and CPU usage, as shown in Fig. 2.

Table I
Model parameters

a_1	b_1	c_1	a_2	b_2	c_2
0.99999	0.00012	0.00004	0.72439	0.27876	0.01106

The predicted values are calculated by the MH for values received from a single OH using $mem(t+1)$ and $cpu(t+1)$ as defined in equation 1. The *update count* denotes the number of updates received from the OH. Based on the previous values (i.e. $mem(t)$, $cpu(t)$ and $meh(t)$) and the current streaming EH count (i.e. $seh(t)$), we calculate the predicted $mem(t+1)$ and $cpu(t+1)$ shown in Fig. 2.

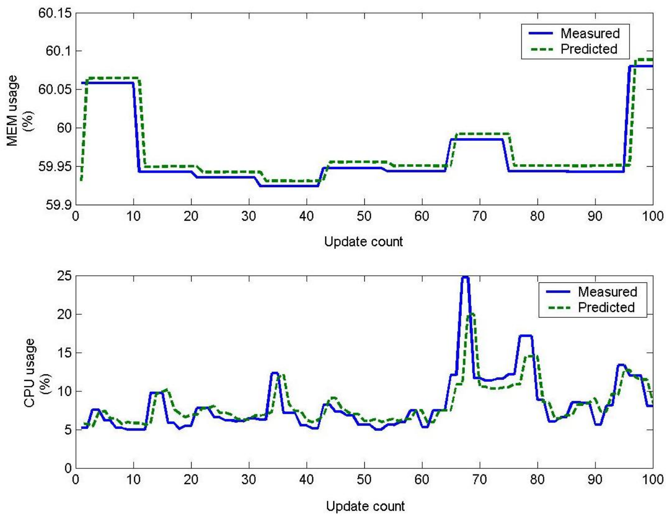

Figure 2. Prediction of next memory and CPU usage

We calculate the medium percentage error using the following equations:

$$\begin{aligned} err_{mem} &= \frac{\sum_{t=1}^{N-1} \frac{|mem'(t+1) - mem(t+1)| * 100}{mem'(t+1)}}{N-1}, \\ err_{cpu} &= \frac{\sum_{t=1}^{N-1} \frac{|cpu'(t+1) - cpu(t+1)| * 100}{cpu'(t+1)}}{N-1}. \end{aligned} \tag{8}$$

For the predicted memory usage we get $err_{mem} = 0.02537\%$ and for the predicted CPU usage we get $err_{cpu} = 21.035\%$. This notable difference between err_{mem} and err_{cpu} is due to the oscillation of measured CPU values.

As we can see from Fig. 2, the model given in equation 1 can predict the next memory and CPU usage, but it can not predict the effect of current requests on future resource usage. This is because it does not take into consideration previous EH requests. This model is extended with knowledge on previous EH requests in equations 5 and 7. The prediction becomes more realistic, taking into consideration not only the current requests, but previous requests and their effect on memory and CPU usage.

The predicted values for the later enriched model are shown in Fig. 3. These correspond to $R_{mem}(t+1)$ and $R_{cpu}(t+1)$ and are calculated for a single OH. Requests are sent by EHs to the MH that predicts the future memory and CPU usage. In practice $\alpha_i = 1$ (from equation 1) because each request is received independently from others, and the value of β_t (from the same equation) equals the number of requests between two updates. Thus, in this case $max(\beta_t) = 4$, where $t = \overline{1,400}$, with a total number of 400 updates. Because the memory and CPU usage are expressed in percentage $\delta_{mem} = \delta_{cpu} = 100$ (from equation 7).

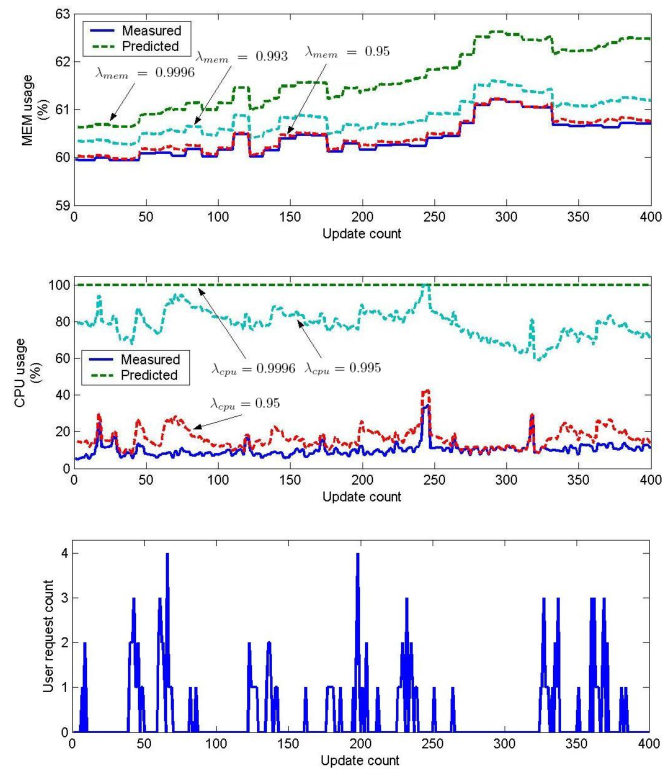

Figure 3. Prediction of future memory and CPU usage for different λ_{mem}, λ_{cpu} values and EH request count

In Fig. 3 we can also see the changes in prediction for different λ_{mem} and λ_{cpu} values. In order to determine the best value for λ_{mem} we use an empirical approach. As shown in the same figure, EH requests between updates 33 and 71 have an effect on memory usage only after update 100. A value for $\lambda_{mem} = 0.95$ leads to a rapid phasing out of previous predictions, while a value for $\lambda_{mem} = 0.9996$ leads to a greater impact of previous predictions. A much more reasonable prediction results for $\lambda_{mem} = 0.993$. The same procedure is used to determine the value of λ_{cpu}. Thus, we estimate that a value of 0.95 for λ_{cpu} would be appropriate for predicting CPU usage.

V. Conclusion and Future Work

We proposed a model for predicting memory and CPU usage in ALM environments. The prediction model uses memory and CPU reports, measuring and streaming EH count to generate the next predicted values. However, using only these elements is not enough for a correct estimation of resource consumption. This is why we also use previous prediction values multiplied with a forgetting factor. By doing so, the time required for EHs to connect to OHs and to be allocated all the necessary resources is also taken into consideration.

Our proposal allows us to predict memory and CPU usage for each new EH. Implementations must previously determine model parameters based on a measured training set for each OH. However, we consider that an adaptive approach - where model parameters are determined at run-time for each OH and are continuously calculated - would be more appropriate for larger overlays. As future work we intend to extend the model proposed in this paper with an adaptive component that automatically calculates both model parameters and the values of λ_{mem} and λ_{cpu}. Also, we intend to expand on the set of monitored and predicted resource allocations and use a weighted function for automatically distributing EHs to OHs.

References

[1] C. Diot, B.N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture", IEEE Network Magazine, Vol. 14, No. 1, 2000, pp. 78–88.

[2] A. El-Sayed, and V. Roca, "A Survey of Proposals for an Alternative Group Communication Service", IEEE Network, Vol. 17, No. 1, 2003, pp. 46–51.

[3] F. Bacelli, A. Chaintreau, Z. Liu, A. Riabov, and S. Sahu, "Scalability of Reliable Group Communication Using Overlays", Proceedings of INFOCOMM, 2004.

[4] S. M. Venilla, and V. Sankaranarayanan, "Threat Analysis for P-LeaSel, a Multicast Group Communication Model", Asian Journa of Information Technology, Vol. 7, 2008, pp. 64–68.

[5] S. Deering, and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANS", ACM Transactions on Computer Systems, Vol. 8, No. 2, 1990, pp. 85–111.

[6] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure", 3rd Usenix Symp. Intl Technol. and Syst., Mar. 2001, pp. 49–60.

[7] Z. Xu, C. Tang, S. Banerjee, S.J. Lee, "RITA: Receiver Initiated Just-in-Time Tree Adaptation for Rich Media Distribution", ACM NOSSDAV, Monterey, CA, 2003, pp. 50–59.

[8] Y. Zhong, S. Shirmohammadi, and A. El Saddik, "Measurement of the Effectiveness of Application-Layer Multicasting", IEEE IMTC, vol. 3, Ottawa, Canada, May 2005, pp. 2334–2339.

[9] L. Mathy, R. Canonico, and D. Hutchison, “An Overlay Tree Building Control Protocol”, 3rd Int’l. Workshop Net.Gr. Commun., London, UK, Nov. 2001, pp. 78–87.

[10] B. Zhang, S. Jamin, L. Zhang, “Host Multicast: A Framework for Delivering Multicast to End Users”, IEEE INFOCOM, vol.3, New York, USA, Jun. 2002, pp. 1366–1375.

[11] D. A. Tran, K. A. Hua, and T. T. Do, “A peer-to-peer architecture for media streaming”, IEEE J. Sel. Areas Commun., Vol. 22, No. 1, Jan. 2004, pp. 121–133

[12] Y. Chu, S. G. Rao, and H. Zhang, “A Case for End System Multicast”, IEEE J. Sel. Areas Commun., Vol. 20, No. 8, Oct. 2002, pp. 1456–1471.

[13] P. Haller, and B. Rakosi, “Optimal server distribution in multimedia communication”, In the Proc. of the 4th International Conference on RoEduNet, 2005, pp. 142–147.

[14] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, “Operating System Support for Planetary-Scale Network Services”, Networked Systems Design and Implementation, 2004.

[15] T. L. Huang, and D. T. Lee, “A distributed multicast routing algorithm for real-time applications in wide area networks”, Journal of Parallel and Distributed Computing, Vol. 67, Issue 5, 2007, pp. 516–530.

[16] W. Jia, W. Tu, and J. Wu, “Hierarchical Multicast Tree Algorithms for Application Layer Mesh Networks”, Networking and Mobile Computing, LNCS, Vol. 3619, 2005, pp. 549–559.

[17] W. Yong, W. Seng, and H. Xianying, “A new Hierarchical Application Layer Multicast algorithm for large-scale video broadcasting”, In the Proc. of the 2nd IEEE International Conference on Computer Science and Information Technology, 2009, pp. 610–613.

[18] M. Haeri, A. H. M. Rad, “Adaptive model predictive TCP delay-based congestion control”, Journal of Computer Communications, Vol. 29, 2006, pp. 1963–1978.