COMPUTATIONAL PROBLEMS OF ELECTRICAL ENGINEERING

Vol. 2, No. 2, 2012

MODELS AND MECHANISMS OF IPTV VOD TRAFFIC BALANCING

Mykhailo Klymash, Yasser Hayali, Orest Lavriv
Lviv Polytechnic National University, Ukraine
yasser.hayali@live.com

Abstract: Models for traffic balancing that enable to evaluate the current IPTV traffic balancing mechanisms are suggested in this paper. Also, an analysis of each of these mechanisms is performed using the IPTV traffic model.

Keywords: IPTV, traffic balancing, modeling.

1. Introduction

The low cost of broadcasting over the Internet, and the reachability of the Internet broadcast compared to conventional broadcasting transmitters and equipment; have made Internet Protocol television (IPTV) the promising future of television and radio, and the choice of many providers nowadays.

Generally, IPTV is a term that is applied to the delivery of traditional TV channels, movies, and video-on-demand content over a private or public network. From the end user's perspective, IPTV should look and operate just like a standard pay TV service with the addition of new features and services.

Typically, IPTV servers stream live channels by multicasting traffic to the connected clients, to minimize the load on the servers while streaming to large numbers of clients simultaneously, thus handing the responsibility of packet delivery to the network infrastructure. In the case of Video on Demand (VoD), a unicast connection has to be established between the client and the server to stream the requested media. This leads to the conclusion that the server load is directly proportional to the number of connected VoD clients. Therefore, a number of VoD streaming servers that should run simultaneously to respond client requests should be estimated statistically from the number of subscribers, and the research of a mechanism that ensures balancing the distribution of client requests to the VoD servers and fault tolerance, crucial and importante to IPTV.

Currently, few mechanisms of balancing traffic exist. This work is dedicated to find out how these mechanisms handle IPTV traffic.

In order to determine the response of a system implementing a certain balancing mechanism, firstly, we need a model of IPTV traffic. This model should easily be incorporated into simulators and closely resemble an IPTV source.

Models for the traffic balancing mechanisms are also required to mathematically analyze the response of these mechanisms when fed with the IPTV traffic model.

2. IPTV Video Traffic Model

Typically, High Definition (HD) TV content is encoded using MPEG-4 (H.264), which has a high compression ratio and results in highly variable data rates (VBR) for the compressed video.

The main task of video compression is to remove spatial and temporal redundancy within each frame and between consecutive frames to maintain a bandwidth. A continuous video stream is sampled into a sequence of frames as the input to the encoder. After encoding, frames are emitted periodically comprising group of pictures (GoPs). Each GoP contains an I-frame and a number of P- and B-frames. For example, with MPEG codec, the generic GoP is I B1 B2 P1 B3 B4 P2 B5 B6 P3 B7 B8. The first frame in each GoP is an I-frame, which is intra-coded without reference to any other frames. The following P-frames are both intra-coded and inter-coded with respect to the previous P- or I-frame. The remaining B-frames are also intra-coded and inter-coded, and they use both the previous and following P- or I-frames as references.

For the purposes of this paper, the two-level Markovian traffic model that is proposed by Fengdan Wan, Lin Cai, and T. A. Gulliver is used. The model considers both spatial and temporal correlation in MPEG encoded video sequences, so it can mimic the highly variable data rate (VBR) behavior of IPTV sources. The model contains the GoP-level Markov chain and the frame-level Markov chain, so it can capture both the inter-GoP and intra-GoP correlations.

a. GoP-level Markov chain

Frame size depends on the texture and motion complexity of the video content, or its spatial and temporal domain correlations.

In the spatial and temporal domains, we categorize the video into a number of levels, S and T, respectively. Thus, we can use $S \times T$ states to represent the correlation in both domains. Since the duration of a GoP is less than half a second, we assume that the spatial and temporal correlations of the video source in each GoP remain at the same level, or in the same state. Therefore, we can build a GoP-level discrete time Markov chain, with each state representing the temporal and spatial correlations of the video within that GoP.

The more states (larger S and T), the more accurate the model, but at the cost of computational complexity in

generating and applying the model. Experimental results show that choosing S = 3 and T = 3 provides a good trade-off between model accuracy and complexity. The three levels are denoted as low (L), medium (M) and high (H) correlation states.

The boundaries between states should be set appropriately, so that a limited number of states could be used to accurately capture the video statistics. According to experimental results with a few video streams, evenly dividing the frames into each state gives reasonably good results. That is, Pr(Si) = 1/N, where Si denotes one of the states, and N = S × T is the total number of states.

In the spatial domain, since only the I-frames are independently intra-coded, the I-frame size is used to determine the texture complexity of the entire GoP. In the temporal domain, the ratio of size of the first P-frame to the size of the I-frame in the same GoP is used to indicate the temporal correlation, as explained below. The P1 frame is both intra- and inter-coded with sole reference to the I-frame. The P1 frame size is determined by $P1 = P1t \times \Delta$, where Δ denotes the motion vectors from frame I to frame P1 in the same GoP, and P1t is the texture information contained in frame P1, which is approximately the same as the texture information in the I-frame of the same GoP. Hence, the ratio between the first P-frame size and the I-frame size in the same GoP, φ, indicates the correlation in the temporal domain $\varphi = P1/I = \Delta$. The larger Δ, the higher the video motion speed, and the lower the correlation in the temporal domain.

Combining the three states in the spatial domain with the three states in the temporal domain gives nine states for each GoP, as shown in Fig. 1. For example, state LM denotes a GoP with low correlation in the spatial domain and medium correlation in the temporal domain. Note that any two states are connected. The transition probability between two states is jointly determined by the temporal and spatial transition probabilities. Since the temporal and spatial processes are independent, the transition probability from state LM to state MM, for example, is Pr{LM → MM} = Prs{L → M} × Prt{M → M}, where Prs{L → M} and Prt{M → M} denote the spatial transition probabilities from the L ("Low") state to the M ("Medium") state, and the temporal transition probability from the M ("Medium") state to the M ("Medium") state, respectively. Each Prt and Prs is obtained by counting the number of transitions between two assigned states in the trace.

b. Frame-level Markov chain

Since the GoP-level model cannot capture the burstiness of the traffic arrival rate within the GoP, the model should be extended to consider different types of frames inside each GoP.

The time step for the frame-level Markov model equals the duration of a video frame. For MPEG video, each state in the GoP-level model corresponds to the 12-step Markov chain at the frame-level, as shown in Fig. 1.

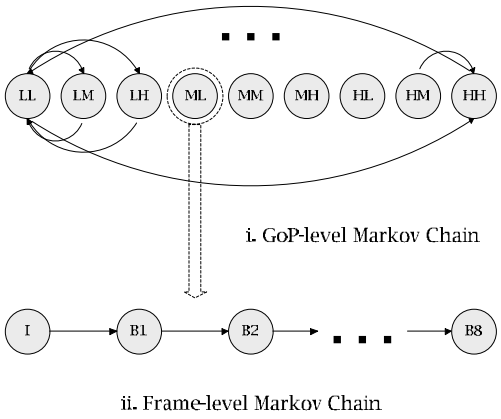

Fig. 1. The IPTV Traffic Model.

This represents 12 frames in the GoP. The state transition probabilities within the GoP are deterministic. Each state corresponds to a different frame type with a different traffic arrival rate. With a limited number of states, a large number of frames with a wide range of frame sizes will belong to the same state. Thus, a critical issue is how to associate the frame sizes with the states.

The size of an I-frame is determined by the spatial domain correlation only. Therefore, the I-frame sizes in states XL, XM, and XH are the same for X = L,M,H in the spatial domain. A simple method to determine the frame size of each state is to average the size of all I- frames belonging to that state:

$$\overline{I^X} = \sum_{I_j \in \{\text{state XL,XM,XH}\}} I_j \Big/ N^X, \qquad (1)$$

where N^X is the total number of I-frames in states XL, XM, and XH, and I_j is the I-frame size in the j-th GoP. I^X in (1) is the average I-frame size in states XL, XM, and XH, which can be used to represent the I-frame size in the corresponding states. Similarly, the P1 frame size can be determined for each state using averages.

For the remaining B- and P-frames, we first examine the intra-GoP correlation. The correlation coefficient of the two sequences i and j is

$$R(i,j) = \frac{CoV(i,j)}{\sqrt{CoV(i,i)CoV(j,j)}},$$

where CoV denotes covariance. The R value between the B/P frames and P1 in the same GoP is examined. The value of R is less dependent on a video content, a compression scheme, etc.

Typically, there is a high correlation between frames in the same GoP. Therefore, the remaining P/B frame sizes in each state are generated based on the P1 frame size, using the linear equations:

$$\overline{F_T^K} = \propto_T^K \overline{P_1^K}, \text{ for } T \in \{B_1, B_2, \ldots, B_8, P_2, P_3\} \quad (2)$$

where $\overline{F_T^K}$ denotes the average size for each frame type and is calculated similar to (1), and K indicates the three states in the temporal domain. Different from the approach in and where the average P- and B-frame sizes in the GoP are used to represent all P/B frames in the same GoP, we differentiate between the P- and B-frame sizes in different locations. This is because HD video has much larger frame sizes than the video data investigated previously. Even though the values of $\propto_T^K$ for different types of P/B frames may be close, after multiplication in (2) with a large variety of I-frame sizes, the discrepancy will be significant.

3. Load Balancing Model with Round-Robin DNS

One of most common implementations of load balancing using Domain Name System (DNS) is the Berkeley Internet Name Domain (BIND). This allows address records (A records) to be duplicated for a specific host, with different IP addresses. The name server then alternatively rotates addresses for any one name that has multiple A records, and is known as a DNS round robin.

Typically, when a client computer wants to initiate a connection to a certain service, it makes a request to resolve the service provider name. This request will be processed by the DNS server which – in the case of Round Robin DNS – will act as the load distribution controller. Instead of returning a fixed address, the DNS server returns an address of a pool of available server addresses. The DNS server goes through all the addresses in the pool one by one, and distributes the requests evenly among all the available service providers.

Figure 2 shows a simplified model of the Round Robin load 'distribution' mechanism.

In order to create a model for this mechanism, we'll have to make some assumptions:

1. Routing of packets from source to destination throughout the network is taking place immediately.

2. The size of DNS requests and the size of DNS responses are fixed for the whole scenario.

3. All queues have infinite sizes. Practically, DNS requests and responses are relatively tiny when compared to the amount of RAM.

4. All traffic has the same priority in queues, no weighted queuing is applied.

The object is to determine how many requests can be responded before request-timeouts start occurring.

4. Model Formulation

How long will it take for a DNS query to be complete?

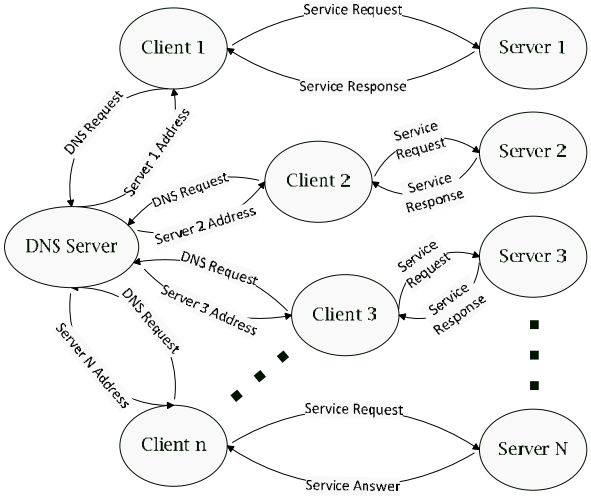

Fig. 2. Round-robin DNS Mechanism.

Assuming that the time required to receive and process a request is higher than the time required to send a response, the total time required for a query to finish is the total time required to receive and process all the preceding requests in the server queue, plus the time it takes to send a response back to the client. This formula should be suitable according to the previous assumption:

$$T = (N+1) \cdot \left(P + \frac{S_Q}{BW_1 * C_1} \right) + \frac{S_R}{BW_2 \cdot C_2}$$

where T is the time (in milliseconds) necessary for a DNS query to finish, N is the number of DNS requests preceding the request in question in the server queue, P is the time (in milliseconds) necessary for the server to process a certain query, S_Q is the size (in Kilobytes) of the DNS request, BW_1 is the link bandwidth (in Mbps) from the clients to the DNS server, BW_2 is the link bandwidth (in Mbps) from the DNS server to the clients, C_1 is the congestion of the link from the clients to the DNS server (a value between 0 and 1), C_2 is the congestion of the link from the DNS server to the clients (a value between 0 and 1).

The previous formula doesn't consider the congestion caused by the DNS queries placed. It should be taken into account as follows:

$$C' = \frac{N_T \cdot S}{BW} + C$$

where N_T is the total number of requests/responses in the DNS server queue, S is the size of the request/response, and BW is the bandwidth of the link.

Now applying the congestion expression in the previous formula, we get:

$$T = (N+1)\cdot\left[P + \frac{S_Q}{BW_1\cdot\left(\frac{N\cdot S_Q}{BW_1}+C_1\right)}\right] + \frac{S_R}{BW_2\cdot C_2}$$

$$T = (N+1)\cdot\left[P + \frac{S_Q}{\left(N\cdot S_Q + BW_1\cdot C_1\right)}\right] + \frac{S_R}{BW_2\cdot C_2} \quad (1)$$

Formula (1) can calculate the time necessary for a DNS query to be finished, but it doesn't consider the congestion that might be caused by the DNS responses made by the server. In this formula, we assume that the time required to receive and process a DNS request is longer than the time to send a response back to the client.

In case the time required to send a DNS response is longer than the time required to receive and process the request, then the total time required for a query to finish will be the time required to send the DNS responses, plus the time required to receive and process the first request. Thus, a suitable formula to calculate the total time for a query to finish would be:

$$T = P + \frac{S_Q}{BW_1\cdot C_1} + \frac{(N+1)S_R}{BW_2\cdot\left(\frac{N\cdot S_R}{BW_2}+C_2\right)} \quad (2)$$

where T is the time (in milliseconds) necessary for a DNS query to finish, N is the number of DNS requests preceding the request in question in the server queue, P is the time (in milliseconds) necessary for the server to process a certain query, S_Q is the size (in Kilobytes) of the DNS request, BW_1 is the link bandwidth (in Mbps) from the clients to the DNS server, BW_2 is the link bandwidth (in Mbps) from the DNS server to the clients, C_1 is the congestion of the link from the clients to the DNS server (a value between 0 and 1), C_2 is the congestion of the link from the DNS server to the clients (a value between 0 and 1).

From formulas (1) and (2), we can determine the maximum number of DNS queries that can be handled before queries start timing out.

When the time required to receive and process a request by the server is higher than the time required for a response to be sent back to the client, the maximum number of client requests possible before DNS queries start timing out would be:

$$N_{total} = -\frac{C_2\cdot S_Q + BW_1\cdot C_1\cdot S_R}{S_R\cdot\left(S_Q + BW_1\cdot C_1 + BW_1\cdot C_1\cdot P - BW_1\cdot C_1\cdot T_{timeout}\right)} + \\ + \frac{BW_1\cdot C_1\cdot C_2\cdot P - BW_1\cdot C_1\cdot C_2\cdot T_{timeout}}{S_R\cdot\left(S_Q + BW_1\cdot C_1 + BW_1\cdot C_1\cdot P - BW_1\cdot C_1\cdot T_{timeout}\right)}$$

Now back to formula (2), we can draw a graph displaying the time required to complete a query under different circumstances. For our purposes, we fix the values of $S_Q = S_R = 512\,\text{bytes}$, $P = 0.04\,\text{sec}$, which are the average values for a typical DNS query. We assume $BW_1 = BW_2$ and apply for 100Mbps and 1Gbps. By substituting these values in the formula above, we obtain:

$$T = \frac{512\cdot N + 512}{12500000\cdot C_2 + 512\cdot N} + \frac{16}{390625\cdot C_1} + \frac{1}{25}\ (100\text{Mbps})$$

and,

$$T = \frac{512\cdot N + 512}{125000000\cdot C_2 + 512\cdot N} + \frac{8}{1953125\cdot C_1} + \frac{1}{25}\ (1\text{Gbps})$$

In the graphs, we use the same value range ($0 \sim 1$) for both C_1 and C_2 and we refer to it as C. We use the value range of ($0 \sim 10000$) for N.

Comparing graphs 1a and 1b, it is obvious that the usage of a link with higher bandwidth will decrease the query time by about 47% at worst case scenario – where congestion is at a peak and the number of queries in the server queue is 10000.

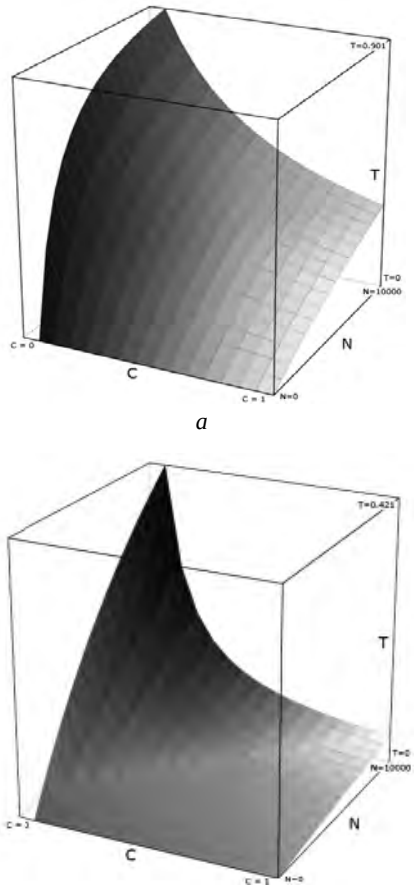

Fig. 3. The relation between the time required to complete a query, the number of queries in the queue and the congestion of the link using (a) a 100 Mbps link. (b) a 1 Gbps link.

These results make a very comforting prediction, as a downtime of 0.901 seconds is barely noticeable. But two serious issues that weren't addressed here are the DNS caching at the client-end, and the fact that in a case of a failure, the 'new' server that a certain client was switched to has no means to recognise where to start off.

The first issue could cause a downtime up to 5 minutes, unless the DNS cache was forced to refresh. The second issue causes disruption and the user has to navigate back to where the failed server left off. Both these problems can be solved by implementing a communication mechanism between the clients and the servers to update the statuses of servers dynamically. This feature is not available in the generic Round Robin DNS load balancing.

5. Conclusions

Calculations done and the results encountered show that a few steps can be made to improve the responsiveness of the load balancing mechanism. But regardless of the changes applied at the server end, there are still deficiencies that urgently require a new mechanism to address and solve them.

References

1. G. O'Driscoll, Next Generation IPTV Services and Technologies. – New York, USA: Wiley-Interscience. – 2008.

2. F. Wan, Lin Cai, T. Aaron, A Simple, Two-Level Markovian Traffic Model for IPTV Video Sources // In Proc. IEEE Global Telecommunications Conference. – New Orleans, USA. – 2008. – P. 1–5.

3. H. McDevitt, Load Sharing with DNS // http://ntrg.cs.tcd.ie/undergrad/4ba2.01/group8/DNS.html

4. Round Robin DNS Load Balancing // http://content.websitegear.com/article/load_balance_dns.htm

МОДЕЛІ ТА МЕХАНІЗМИ БАЛАНСУВАННЯ IPTV VOD ТРАФІКУ

Михайло Климаш, Яссер Хайалі, Орест Лаврів

Запропоновано моделі балансування трафіку IPTV VoD для оцінки існуючих механізмів балансування. Проведено аналіз кожного з цих механізмів за допомогою моделі IPTV трафіку.

Mykhailo Klymash – Ph.D., D.Sc., Professor, the Head of Telecommunications Department of the Institute of Telecommunications, Radioelectronics and Electronic Engineering, Lviv Polytechnic National University, Ukraine.

Area of interest: load balancing mechamisms in video information flows and their responseveness.

Yasser Hayali – M.Sc., the postgraduate of the Telecommunications Department of the Institute of Telecommunications, Radioelectronics and Electronic Engineering, Lviv Polytechnic National University, Ukraine.

Area of interest: load balancing mechamisms in video information flows and their responseveness.

Orest Lavriv – Ph.D., Assistant Professor of the Telecommunications Department of the Institute of Telecommunications, Radioelectronics and Electronic Engineering, Lviv Polytechnic National University, Ukraine.

Area of interest: load balancing mechamisms in video information flows and their responseveness.