IP Traffic Generator Based on Hidden Markov Models

Hasan Redžović, *Junior Member, IEEE,* Aleksandra Smiljanić, *Member, IEEE,* and Milan Bjelica, *Member, IEEE*

Abstract - **The constant development of communication networks imposes big challenges when providing necessary performance and keeping costs low. Successful network and device design rely on testing tools which are capable of simulating real network conditions. Normally, these tools utilize various commonly used IP traffic models in order to generate IP traffic. However, majority of those IP traffic models are useful for specific traffic scenarios and fail to adequately simulate wide range of different traffic patterns. In this paper, we present new IP traffic generator based on hidden Markov models (HMM). The important feature of hidden Markov models is their ability to automatically adjust parameters using Baum-Welch algorithm, which allows us to record real IP traffic and dynamically create accurate representation of that traffic. We develop application which utilizes hidden Markov models that is capable of generating a large number of different IP traffic patterns based on measurements of live traffic.**

Index Terms— **Traffic Generation, Traffic Models, Hidden Markov Models.**

I. Introduction

The communication networks are constantly increasing in order to cope with demands for lager network infrastructure and new services. This growth of communication networks requires efficient tools for analyzing and evaluation of their performance. Quality of service needs to be of adequate level to provide customer satisfaction and cost dictates investment's profitability. The network design must balance between quality of service and cost by optimizing performance to match real network conditions. Since traffic models simulate those network conditions, they are the core component of network performance evaluation and they need to be very accurate. Different traffic models can be used to generate IP traffic with the characteristics suited for specific networks and devices. Some of the most widely used traffic models are:

- Poisson distribution model is one of the oldest traffic models and it was widely used in traffic scenarios that comprise large number of independent traffic streams [1].
- Pareto's [2] and Weibull's [3] heavy-tailed distributions are often used to describe self-similar network traffic [4]. Heavy-tailed distributions have the property of a long-range dependence.
- In ON-OFF model each input alternates between active and idle periods with geometrically distributed durations. This model can be used for modelling of both system behavior and IP traffic scaling features [5].
- More complex models which are designed for specific traffic types [6, 7].

These traffic models are not designed capture the characteristics of arbitrary types of traffic loads. In the majority of cases, the types of network and traffic determine the choice of the traffic model used for analysis. If the traffic model is not chosen correctly, the characteristics of actual traffic will not be properly represented and that would lead to under-estimation or over-estimation of network performance. Traffic models should have a manageable number of parameters, and parameter estimation should be simple.

In this paper, we propose different traffic modeling strategy based on hidden Markov models (HMMs) [8]. The HMMs are normally used in speech recognition [9]. Speech has temporal structure and can be encoded as a sequence of spectral vectors spanning the audio frequency range. Each sequence of vectors usually represents one word. For any input sequence, HMM will produce occurrence probability of that sequence. The speech recognition is done by creating a large library of HMMs where parameters of each HMM are adjusted to maximize probability of specific word. In other words, each HMM represents one word. Then, input sequence is put through library of HMMs and each of them calculates the likelihood of its corresponding word. The word with the highest likelihood is the result of estimation.

We used HMM as a statistical IP traffic model, where input sequences are based of different traffic characteristics, similarly as in [8]. We implemented application hmm_ip_gen which is capable to record traffic in live network, parse results, create input sequences, train HMMs and generate output IP traffic. We, then, examined the performance of our generator in the case of gaming and streaming applications.

The paper is organized as follows. The principles of HMMs and Baum-Welch algorithm [10] are presented in Section 2. Sections 3 describes hmm_ip_gen application and how HMM is utilized for IP traffic simulation. Section 4 describes testing environment. In Section 5, we analyze results of conducted tests. Finally, Section 6 concludes the paper.

Hasan Redžović is with the Innovation Center of School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: hasanetf@live.com).

Aleksandra Smiljanić is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: aleksandra@etf.rs).

Milan Bjelica is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: milan@etf.rs).

II. Hidden Markov Model and Baum-Welch Algorithm

The canonical probabilistic models for temporal or sequential data is called Markov model. In Markov model, it is assumed that future states only depend on the current states and not on the events that occurred before it. A hidden Markov model (HMM) is a Markov model with underlying states that are not observable (hidden). The hidden states can only be observed through another set of stochastic processes that produce the sequence of observed symbols. In Markov models, the state is directly visible to the observer, and, therefore, the state transition probabilities are the only defined parameters. In a hidden Markov model, the state is not directly visible, instead, we can see only output symbol (also named observation). Each state has a probability distribution over the possible output observations. Therefore, the sequence of observations generated by an HMM gives some information about the sequence of states.

In HMM, the states are denoted as $Q = \{q_1, q_2,..., q_N\}$, where N is the number of states in the model. State transition matrix is defined as:

$$A = \{a_{ij}\}_{N\times N}, a_{ij} = P(\mathrm{q}_j^{(t+1)} \mid \mathrm{q}_i^{(t)}), \qquad t = 1,2,...,T-1 \quad (1)$$

We will estimate states based on certain observations which are correlated with the states. Let us denote observations as $V = \{v_1, v_2,..., v_M\}$ where M is the number of observation values. The observation matrix correlates observations with states

$$B = \{b_{jk}\}_{N\times M}, b_{jk} = P(v_k^{(t)} \mid \mathrm{q}_j^t), \qquad t = 1,2,...,T \quad (2)$$

An initial state distribution is necessary to run the HMM model:

$$\pi = \{\pi_i\}_{1\times N}, \pi_i = P(\mathrm{q}_i^{(\mathrm{t})}), \qquad \mathrm{t} = 1 \quad (3)$$

The initial state distribution defines the likelihood to be in each state at the beginning of observation. The matrices π, A and B completely define HMM and they are usually written as $\lambda = (A, B, \pi)$. Observation sequence in time is denoted by:

$$O = (o_1, o_2 ... o_T) \quad (4)$$

Using HMM with defined parameters λ, it is possible to determine the most likely state sequence that will produce the observation sequence (4). In other word, by observing the sequence (4), we can determine the most likely sequence of states. There are three fundamental problems that can be solved using HMMs:

- Problem 1: given a sequence of observation O and $\lambda = (A, B, \pi)$, find $P(O|\lambda)$;
- Problem 2: given a sequence of observation O and $\lambda = (A, B, \pi)$, find optimal state sequence;
- Problem 3: given a sequence of observation O and dimensions N and M, find the model $\lambda = (A, B, \pi)$ that maximizes probability of O.

Below, we are going to briefly describe efficient algorithms for solving mentioned problems.

A. *Solution to Problem 1*

The direct computation of $P(O|\lambda)$ is very hard and in many cases not possible, because it requires $2TN^T$ multiplications. The recursive forward algorithm (also called *α-pass*) is more efficient way to find $P(O|\lambda)$. Let $X = (x_0, x_1, ..., x_{T-1})$ be the state sequence. For $t = 0, 1,..., T - 1$ and $i = 0, 1,..., N - 1$, parameters $\alpha_t(i)$ are defined:

$$\alpha_t(i) = P(o_0, o_1, ..., o_t, \mathrm{x}_t = \mathrm{q}_i \mid \lambda) \quad (5)$$

The parameter $\alpha_t(i)$ is the probability of the partial observation sequence up to time t, where the underlying Markov process is in state q_i at time t. First, initial values are computed:

$$\alpha_0(i) = \pi_i b_{iO_0}, \qquad i = 0,1,...,N-1 \quad (6)$$

For $t = 1, 2,..., T - 1$ and $i = 0, 1,..., N - 1$, $\alpha_t(i)$ are recursively computed:

$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) \cdot a_{ji}\right] b_{iO_t} \quad (7)$$

Then, conditional probability $P(O|\lambda)$ is computed:

$$P(O \mid \lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i) \quad (8)$$

The forward algorithm requires N^2T multiplications which is much easier to calculate in comparison to $2N^TT$ multiplications required for direct approach.

B. *Solution to Problem 2*

Backward algorithm or *β-pass* is used for solving problem 2. The backward algorithm is almost analogous to forward algorithm, but, in this case, calculation starts at the end and goes toward the beginning of the observation sequence. For $t = 0, 1,..., T - 1$ and $i = 0, 1,..., N - 1$, we define parameters:

$$\beta_t(i) = P(o_{t+1}, o_{t+2} ... \, o_{T-1}, x_t = q_i \mid \lambda) \quad (9)$$

Then, the calculation process is roughly the same as in the forward algorithm. First, initial values are defined:

$$\beta_{T-1}(i) = 1 \qquad i = 0,1,...,N-1 \quad (10)$$

For $t = T$-2, T-3,…, 0 and $i = 0, 1 … N - 1$, parameters $\beta_t(i)$ are recursively computed:

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_{jO_{t+1}} \beta_{t+1}(j) \quad (11)$$

Using forward and backward algorithms, we can calculate parameters $\gamma_t(i)$ which will be used to find the most likely state q_i at time t. For t = 0, 1,..., T - 1 and i = 0, 1,..., N – 1:

$$\gamma_t(i) = P(x_t = q_i \mid O, \lambda) \tag{12}$$

Parameter $\alpha_t(i)$ measures the state probability up to time t and $\beta_t(i)$ measures the state probability after time t, and $\gamma_t(i)$ can be calculated as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} \tag{13}$$

It can be shown that the most likely state at time t is the state q_i for which $\gamma_t(i)$ is maximal.

C. Solution to Problem 3 (Baum-Welch algorithm)

The values of $\lambda = (A, B, \pi)$ can be adjusted to maximize probability of some specified observation sequence. First, "di-gamma" parameters need to be defined. For t = 0, 1,..., T-2 and $i, j \in \{0, 1,..., N\text{-}1\}$:

$$\gamma_t(i, j) = P(x_t = q_i, x_{t+1} = q_j \mid O, \lambda) \tag{14}$$

Parameter $\gamma_t(i, j)$ is the probability of being in state q_i at time t and transiting to state q_j at time t + 1. The di-gammas can be written in terms of α, β, A and B as:

$$\gamma_t(i, j) = \frac{\alpha_t(i)\alpha_{ij} b_{jO_{t+1}} \beta_{t+1}(j)}{P(O \mid \lambda)} \tag{15}$$

One can re-estimate $\lambda = (A, B, \pi)$ using di-gamma and parameter $\gamma_t(i)$ defined in (13). Re-estimated π_i values:

$$\pi_i = \gamma_0(i),\ i = 0, 1,\ldots, N - 1 \tag{16}$$

Re-estimated a_{ij} values for $i, j \in \{0, 1,..., N\text{-}1\}$:

$$a_{ij} = \frac{\sum_{t=0}^{T-2} \gamma_t(i, j)}{\sum_{t=0}^{T-2} y_t(i)} \tag{17}$$

Re-estimated b_{jk} values for $j \in \{0, 1,..., N\text{-}1\}$ and $k \in \{0, 1,..., M\text{-}1\}$:

$$b_{jk} = \frac{\sum_{\substack{t\in\{0,1,\ldots,T-1\} \\ O_t = k}} y_t(j)}{\sum_{t=0}^{T-1} y_t(j)} \tag{18}$$

Using re-estimation through iterations, model $\lambda = (A, B, \pi)$ is optimized to produce used training sequence with the highest probability. The described solution of problem 3 is also called Baum-Welch algorithm.

In our case, we used HMMs and Baum-Welch algorithm to generate IP traffic. This generation process has three basic steps: (*i*) The training observation sequences are created from the recorded IP traffic; (*ii*) The HMMs are trained using Baum-Welch algorithm and measured observation sequences; (*iii*) The trained HMMs then generate output sequences which are used to synthesize IP traffic based on the observed traffic.

III. HMM IMPLEMENTATION

We created traffic generator application named hmm_ip_gen which utilizes HMMs described in previous section. HMMs are based on two important IP traffic features for network performance:

- Packet length – Ethernet packet can be between 64 B and 1500 B long. Packet size determines maximal number of packet that can be sand through the link. Shorter packets require more processing power of networking devices.
- Packet interarrival time – Packets are commonly appearing in bursts, i.e. after the first packet there is a large probability to receive the second packet. Also, depending on the IP traffic type (video streaming, games, VoIP, etc), the statistics of interarrival times between packet bursts can vary. Packet interarrival time distribution affects the number of packet received by network devices and more realistically depict the actual network conditions.

Application hmm_ip_gen is developed in C programing language and it is split in three phases (Fig. 1). In the first phase, hmm_ip_gen is recording real IP traffic on dedicated interface which is connected with live network. The IP traffic is recorded using pcap library and it is parsed into two group of training sequences: packet length sequences and interarrival time sequences.

In the second phase, packet length and interarrival time sequences are used as training observation sequences for HMMs. Before training process starts, HMMs are initialized by setting values for matrices A, B and π. If some of the IP traffic characteristics are known, it is recommended to provide reasonable approximations for the matrix values. Otherwise, hmm_ip_gen will set initially $a_{ij} \approx 1/N$, $b_{jk} \approx 1/M$, $\pi_i \approx 1/N$. Custom initial matrix values can increase HMM accuracy. It is important that the values of each matrix are not absolutely uniform, because the training process will fail. The HMM training procedure has the following steps:

- Packet length sequence or interarrival time sequence are used as training observation sequence O (4);
- Based on training observation sequence and initial values for matrices A, B and π, the parameters $\alpha_t(i)$ and $\beta_t(i)$ are calculated using (7) and (11), respectively;
- The probability $P(O| \lambda)$ is computed using (8);
- When $\alpha_t(i)$, $\beta_t(i)$ and $P(O|\lambda)$ are calculated, the parametars $\gamma_t(i)$ and $\gamma_t(i, j)$ can be computed using (13) and (16), respectively;
- Then, the parametes $\gamma_t(i)$ and $\gamma_t(i,j)$ are used to reestimate values of the matrices π, A and B using (16), (17) and (18), respectively;
- The new probability $P(O|\lambda)$ is computed using (8) and

compared to the old probability $P(O|\lambda)$. If new probability is higher, then, the new cycle of the HMM traning procedure begins, where new re-estimated values of matrices π, A and B are used. Otherwise, new re-estimated values are rejected and the HMM training procedure is finished.

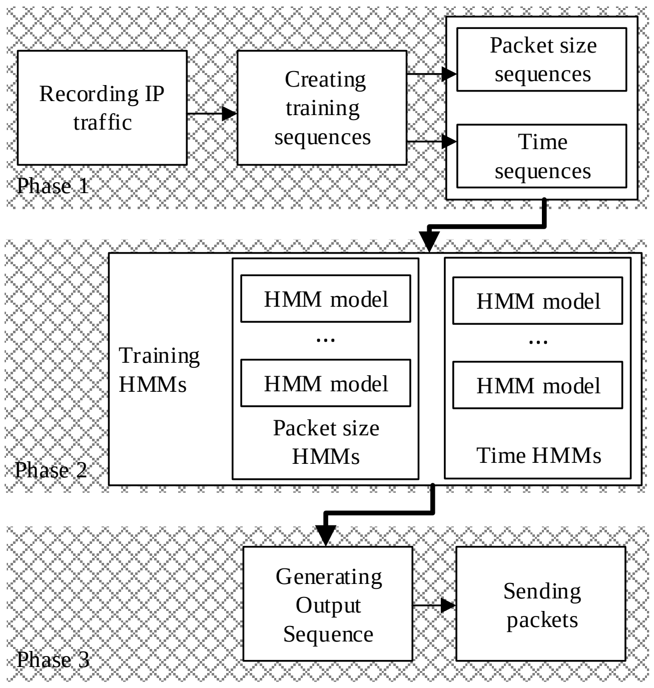

Fig. 1. Traffic generator application hmm_ip_gen architecture.

In the third phase, output sequences (packet length and interarrival time sequences) are generated using trained HMMs. The sequence generation process is based on the algorithm for selecting an element with the probability of its occurrence. Probabilities of the elements are stored in an array. The algorithm is presented by the pseudo code given in Fig. 2.

```
initialize and set **index** to zero;
initialize and set **cumulative probability** to zero;
create random uniform **number** in the range [0,1];

while **cumulative probability** is less than **number**:
    add next given probability of an array to
    **cumulative probability**;
    increment **index** by 1;
return **index**;
```

Fig. 2. The algorithm for selecting element.

Using the described algorithm, the output sequence is generated through the following steps:

- A state is selected from matrix π which represents initial hidden state with its calculated probability;
- An output observation is selected from matrix B which represents initial output observation with its probability for a given state;
- Then, the next hidden state is selected from matrix A, and, the next output observation is selected from matrix B;
- The previous step is repeated until the entire sequence is generated.

Output sequences define packets lengths and interarrival times between packet transmissions. The Ethernet packets were sent using Linux raw socket.

IV. Testing Environment and Experimental Evaluation

The testing environment comprises two machines M1 and M2, which are connected as shown in Fig. 3. Machine M1 is using hmm_ip_gen to record IP traffic from the network, train HMMs and generate IP traffic towards machine M2. The generated IP traffic should have approximately the same packet length and interarrival time distribution as recorded IP traffic form the network. We used three points in our testing environment to measure packet size and time distribution. These measurement points are labeled in Fig. 3 as A, B and C. The packet length and interarrival time distribution were measured at point A for the recorded IP traffic, at point B for output sequences (generated by HMMs) and at point C for the IP traffic recorded on machine M2. The pcap library was used on machines M1 and M2 for measuring IP traffic at points A and C respectively. Generated IP traffic was measured at point B within hmm_ip_gen application, before it was sent through raw socket towards machine M2. Tests are conducted for two types of IP traffic: video streaming and online multiplayer games.

In the first case, machine M1 was receiving video stream form the network, while hmm_ip_gen was recording IP traffic. And in the second case, machine M1 was having game session for simple browser online multiplayer game agar.io, while hmm_ip_gen was recording IP traffic.

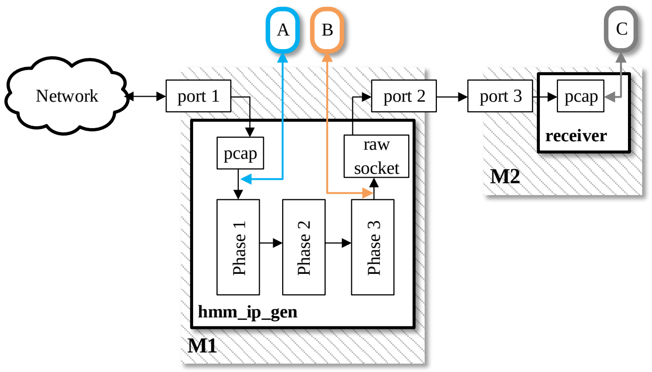

Fig. 3 Testing environment.

The packet length distribution of video streaming in all three points are shown on Fig. 4. The results clearly demonstrate that the packet length distribution in all three points are approximately the same. This means that the machine M2 (point C) is receiving the IP traffic generated by machine M1 which has approximately the same statistics as the traffic on the network (point A).

Also, Fig. 4 shows two dominant packet lengths: 25 % of packets are short (around 64 Bytes) and 20 % of packets are long (around 1500 Bytes). The long packets come from video streaming and TCP packets which carry video data. The short packets come from many different sources: TCP ACK (Acknowledgement), routing protocols, DHCP, ARP, etc.

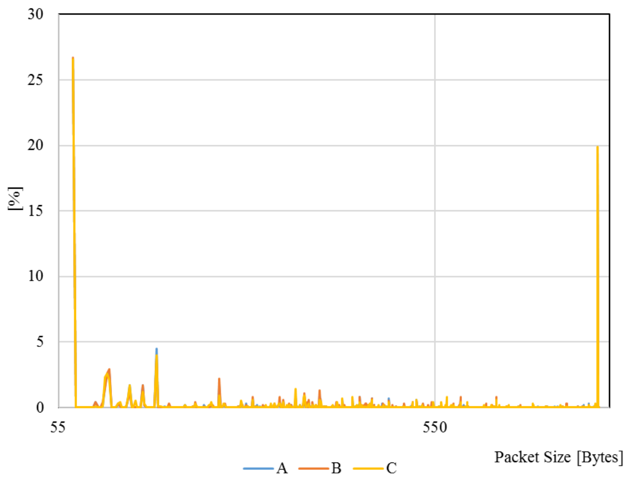

Fig. 4 Packet size distribution for video streaming.

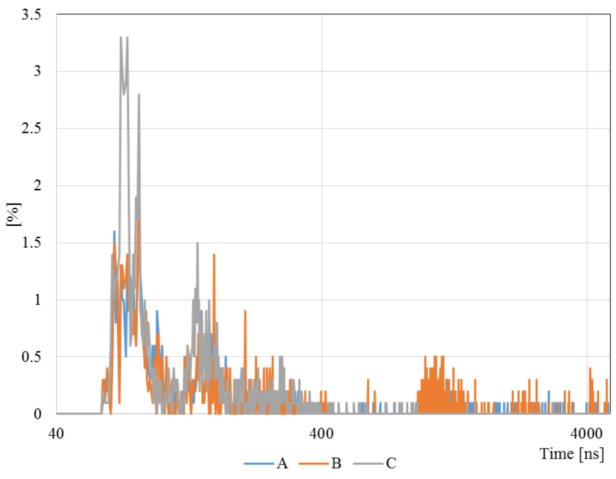

Fig. 5 Time distribution for video streaming.

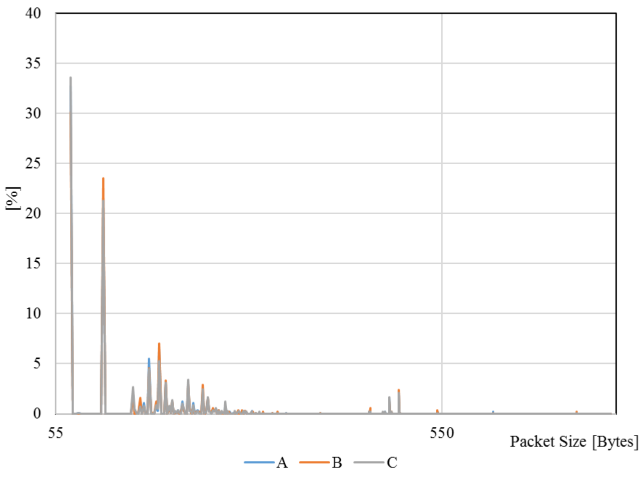

Fig. 6 Packet size distribution for online multiplayer game agar.io.

The interarrival time distribution for video streaming is shown in Fig. 5. Recorded network IP traffic (point A) and generated output sequences (point B) have very close interarrival time distributions. However, recorded IP traffic on machine M2 (point C) has time distribution which slightly differs when compared to point A and point B measurements. This difference in time distributions is due to the limitation of Linux socket as it is difficult to achieve precise transmission between two successive packets for short interarrival times (around 100 ns). Still, the overall interarrival time distribution at point C is quite similar to the measurements at points A and B. It is worth noting that accuracy of time distribution is mainly limited by software and hardware used to generate IP traffic and not by HMMs. Time distribution at point C can be improved by optimizing Linux socket or by choosing another more accurate method for sending packets.

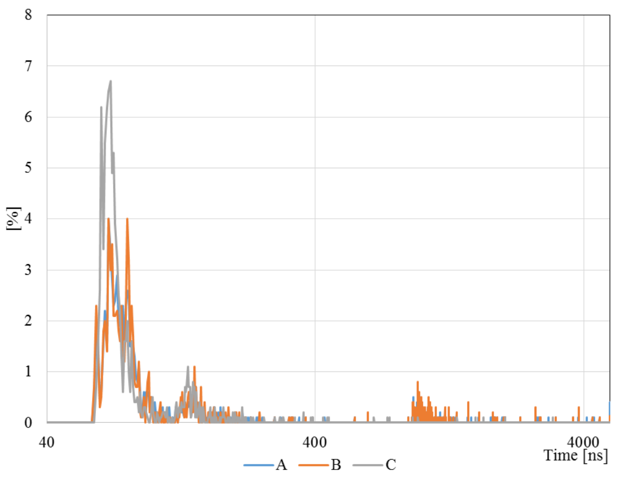

Fig. 7 Time distribution for online multiplayer game agar.io.

The packet length distribution of agar.io is shown in Fig. 6. Unlike in video streaming, the traffic of online multiplayer games does not include long packets. The agar.io server require frequent updates of all players in session for calculating players positions and other related parameters. Then, server sends update to each player and thus providing synchronization between players so they can interact with each other. This communication between server and player require IP packets with relatively small lengths. This IP traffic between server and machine M1 is shown in Fig. 6 has peak of 23% for packet lengths around 73 Bytes. In Fig. 6, packet length distribution at all three points is approximately the same as in the case of video streaming. The interarrival time distribution of agar.io is shown on Fig. 7 and it has similar characteristics as interarrival time distribution of video streaming.

V. Conclusion

We presented basic principles of HMM and how it can be used for implementing the IP traffic generator. Our traffic generator can learn the statistics of the network traffic, and later use created HMM models for network testing and performance evaluation. The presented results demonstrate the hmm_ip_gen capability to easily and very accurately mimic wide range of different IP traffic patterns. The hmm_ip_gen flexibility and ability to store large number of trained HMMs can be suitable for testing and evaluation of various networks and network devices. Application hmm_ip_gen can be improved in future by adding different

I/O methods for receiving and sending packets which will enhance the accuracy of the interarrival time distribution. Also, trained HMMs in hmm_ip_gen can be adjusted to not only generate output sequence but also to recognize different IP traffic patterns in the networks.

Acknowledgment

This work was supported by the Serbian Ministry of Science and Education (project TR-32022), and companies Telekom Srbija, and Informatika.

References

[1] T. Takine and K. Okazaki, "IP traffic modeling: most relevant time-scale and local Poisson property," in *International Conference on Informatics Research for Development of Knowledge Society Infrastructure*, Kyoto, 2004.

[2] J. Gordon, "Pareto process as a model of self-similar packet traffic," in *Global Telecommunications Conference (GLOBECOM)*, Singapore, 1995.

[3] M. A. Arfeen, K. Pawlikowski and D. McNickle, "The role of the Weibull distribution in Internet traffic modeling," in *International Teletraffic Congress*, Shanghai, 2013.

[4] L. Zhen, N. Nicolas and J.-V. Cesar, "Traffic model and performance evaluation of Web servers," Performance Evaluation, vol. 46, no. 2-3, pp. 77-100, 2001

[5] H. J. Chao, High Performance Switches and Routers, New Jersey: Wiley-IEEE Press, 2007.

[6] N. Nikaein, M. Laner, K. Zhou, P. Svoboda, D. Drajic, M. Popovic, S. Krco, "Simple Traffic Modeling Framework for Machine Type Communication," *International Symposium on Wireless Communication Systems*, Ilmenau, Germany, 2013.

[7] D. Drajic, S. Krco, I. Tomic, P. Svoboda, M. Popovic, N. Nikaein, N. Zeljkovic,"Traffic generation application for simulating online games and M2M applications via wireless networks," *Conference on Wireless On-demand Network Systems and Services*, Courmayeur, Italy, 2012.

[8] Z. Ghahramani, "An Introduction to Hidden Markov Models and Bayesian Networks," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 15, no. 1, pp. 9-42, 2001.

[9] M. Gales and S. Young, "The Application of Hidden Markov Models in Speech Recognition," *Foundations and Trends in Communications and Information Theory*, vol. 1, no. 3, p. 195–304, 2007.

[10] S. Tu, "Derivation of Baum-Welch Algorithm for Hidden Markov Models", https://people.eecs.berkeley.edu/~stephentu .