Modelling Internet Workloads for IEEE 802.16

S. Forconi, G. Iazeolla, P. S. Kritzinger and P. Pileggi
Technical Report CS08-04-00

Data Network Architectures Group
Computer Science Department University of Cape Town
Private Bag, Rondebosch 7700, South Africa
Email: {ppleggi,psk@cs.uct.ac.za}

Abstract The IEEE 802.16-2004 standard contains the wireless MAN (WMAN) air interface specification. WMAN has become a major part of the emerging broadband wireless access technology particularly since it accounts for differentiated traffic classes (TCs). Differentiated traffic is immediately associated with Quality of Service (QoS) and this then becomes the objective of many WMAN studies. Since it is unrealistic to experiment with a real WMAN, the obvious alternative is to model its performance. Our objective is thus to develop a WMAN Base Station (BS) and Subscriber Station (SS) simulation model operating in the point-to-multipoint (PMP) architecture mode. As important as the simulation itself is a model representing the load or traffic. In this document we report on an IEEE 802.16 synthetic Workload Model (WLM) with a WLM Generator (WLG) and associated generators (TGs) that represent and generate internet traffic. Underlying the WLM is a Markov Modulated Arrival Process (MMBP) to combine the various WLGs.

I. Introduction

In Figure 1 from [22], represents a typical structure for a WMAN designed to provide broadband wireless access over a metropolitan area. The WMAN standard defines a base station (BS) for serving multiple subscriber station (SSs). Within a SS (e.g., building, house, small campus, etc.), a large number of end users with different broadband access requirements can be present. An SS sends wireless traffic to a single BS using either Time Division Duplexing (TDD) or Frequency DDivision Duplexing (FDD). Users can access the network with conventional office networks such as, Ethernet (IEEE Standard 802.3) or wireless LANs (IEEE Standard 802.11) for data and video, or convention phone lines for voice. Two types of architecture are supported by the IEEE 802.16 standard.

1) A Point to Multipoint (PMP) architecture where a BS connects to an internet gateway and each SS connects only to the BS and not to one another.
2) A Point to Point Mesh (PTP) architecture where each SS connects not only to the BS but also to each other but not necessary every other SS. Therefore, certain SSs may also perform some of the functions of the BS in the Mesh architecture.

Our research work in this paper only focus on the PMP architecture of the WMAN standard using TDD.

IEEE 802.16 defines four types of traffic classes (TCs) to represent internet traffic, typically HTML, VoIP, Video Streaming, P2P and FTP in wireless networks:

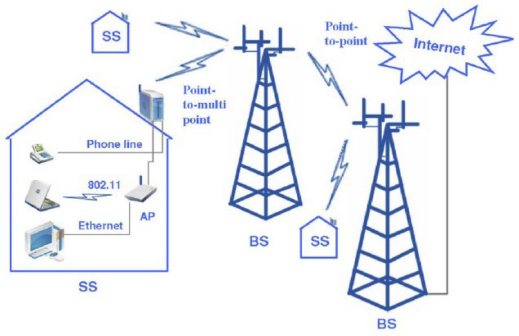

Fig. 1. A typical WMAN structure from [22]

1) Unsolicited Grant Service (UGS) is designed to support real-time, with strict delay requirements. These are applications that generate fixed-size data packets on a periodic basis, such as T1/E1 and Voice over IP.
2) Real-time Polling Service (rtPS) is designed to support real-time applications with less stringent delay requirements that generate variable size data packets on a periodic basis, such as moving pictures experts group (MPEG) streaming video.
3) Non-Real-Time Polling Service (nrtPS) is designed to support delay-tolerant, with minimum rate requirement data streams that with variable-sized data packets, such as FTP.
4) Best Effort (BE) is designed to support data streams for which no minimum transmission rate is required and therefore may be handled on a space-available basis, such as HTTP.
5) Extended real-time variable rate (ertPS), which was added in 802.16e-2005 (or Mobile-WiMAX), that supports real-time applications where the applications require guaranteed data rate and delay. This service is for applications that would typically, in 802.16-2004, subscribe to the *rtPS* service even though they may behave similarly to *UGS* traffic at times, such as VoIP with silence suppression.

Before describing the WMAX synthetic Workload Model, the topic of this report, we first describe a WMAN BS and SS simulation model operating in PMP mode. A model of any system, such as that in Figure 2, can be described by the framework shown in Figure 3. The system is composed of an actual workload and an actual machine. The actual workload is the traffic load that the actual machine, the WMAN, is

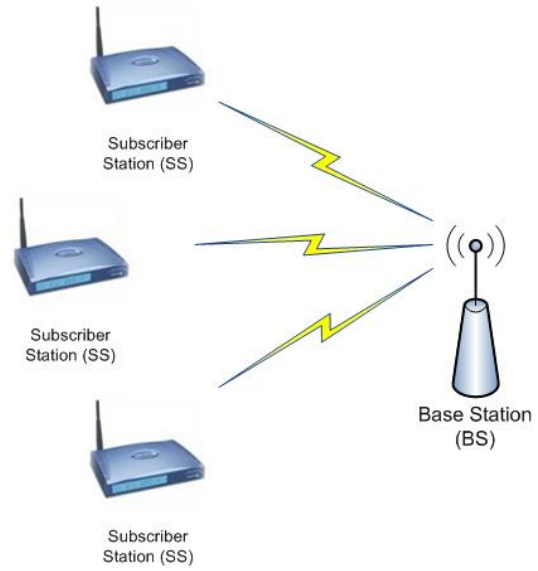

Fig. 2. The WMAN PMP architecture of the model

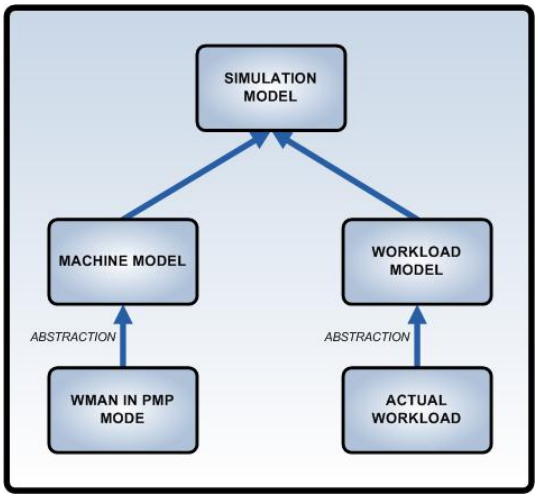

Fig. 3. Modelling framework

subjected to. The actual machine are the hardware and software components of the system. A model of the system, that is, an abstraction of the System to be modelled, therefore consists of an abstraction of both the actual workload, called the workload model (WLM), and the actual machine, call the machine model (MM).

The workload model can be an analytic abstraction of the actual workload, and the machine model an analytic, simulation or test network abstraction of the actual machine. In this document we develop an IEEE 802.16 synthetic workload model along the lines of the proposals by Lourens [21] where he defined a mathematical structure and its set of parameters, that is used to generate a representative traffic load for some system.

The WLM accommodates various traffic type generators by allowing the independent development of different packet-level traffic models (TMs) and associated traffic generators (TGs), merging these stochastically using a Markov Modulated Bernoulli Process (MMBP).

The WLM generates traffic for various different types of TCs, such as web traffic (HTML), Voice over IP (VoIP), streaming video (MPEG), etc. Each traffic class generation process is modelled independently by a suitable mathematical workload model, which we term the traffic model (TM). The model that represents the dynamic generation process, switching between the various TMs, i.e. the WLM, is a Markov Modulated Bernoulli Process (MMBP).

In order to develop the WLM, we followed the following steps: We

1) Isolated those traffic types which constitute, say, 95% of internet traffic, namely VoIP, Real TimeVideo Streaming, HTML and P2P.
2) Investigated models in terms of inter-arrival time and packet size distribution (eg Poisson, Exponential, Weibull, etc.) found in the literature for each traffic type.
3) Designed and implemented the MMBP WLM generator in Java.
4) Designed and implemented the TM generator for each traffic type we identified.

In the remainder of the report, in Section II we present the MMBP workload model (MMBP WLM) necessary to realize our WLM and the interface with the design and implementation respectively. Section III briefly explains the TG that we have chosen, emphasizing the packet level model used to realize the WLM. Finally, Section IV shows, for each internet traffic modeled by the Traffic Model (TM), the corresponding design and implementation.

II. Workload model (WLM)

Workload in the internet is normally generated by an individual or user in the first case. The user has a certain behaviour which, in the case of interactive video for instance, generates IP traffic of a very different nature and QoS requirement than, say a P2P user. Should the user behaviour be part of the WLM or nor? In the case of P2P it makes no sense, in the case of say an internet browser, it does, as we shall see.

A. MMBP arrival process

The WLM uses a Markov Modulated Bernoulli (arrival) Process (MMBP) shown in Figure 4. This figure describes the general scenario of some system actor requesting the next packet information from the WLM. Each state has a different TM, depending on the traffic type and a corresponding traffic generator (TG). The information required are the key traffic features identified. These are the distribution of the

1) inter-arrival time (IAT), the time between successive arrivals, and
2) IP packet size.

An implementation of the MMBP requires that we know

1) the IAT and size distributions and parameters for the corresponding traffic type, and
2) the transition probability matrix.

Given measured traffic, there are well-known techniques as, for instance, described by Lindemann[5] to determine the state vector π and the transition probability matrix P in the discrete time MMBP case. Measured traffic datasets are available on the

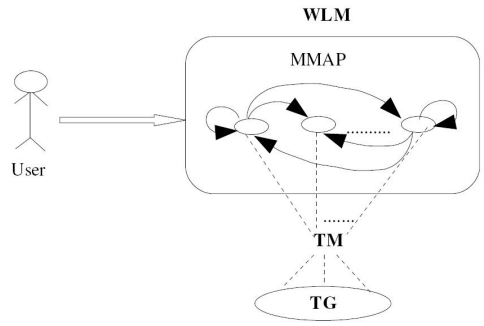

Fig. 4. Scenario of a system actor using the WLM

internet, for example from the *UNC/FORTH Archive of Wireless Traces, Models, and Tools*[20] and *the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD)*[13].

These traces, however, seldom differentiate between traffic types for the simple reason that, for P2P traffic various port numbers can be used and one can therefore not classify the traffic on port number, the only information in an IP packet to do so. There are other techniques, such as clustering discussed by Symington [19] for differentiating between traffic profiles, but not reliably by traffic class.

For this reasons described above, we did not use traffic measurements for the WLM reported here, although it is designed such that if such data were to become available, these can be used. We used state vector π where the value of each state $\pi_i, \quad i = 1, 4$, representing internet browsing, VoIP, P2P and interactive video, respectively was indeed from internet traffic measurements [15], but P was derived from $\sum_{i=1}^{4} \pi_i = 1$, $\sum_{j=1}^{4} \pi_{ij} = 1 \forall i$ and solving $\pi P = \pi$ symbolically for the remaining parameters. The code is listed in the Appendix B.

B. WLM Design

In designing the WLG and the WLM interface with the user, the Machine Model (MM, refer Figure 3), we consider three design stages, as shown in Figure 5. In the first stage, we define the MMBP in terms of its components (nouns) and capabilities (verbs). The MMBP entity is then, in Stage 2, translated into the WLG entity where nouns are translated into entity attributes and verbs into behaviours. The MMBP entity is also extended to include the behaviour specific to workload generation. An interface between the WLM and generic TM is a component of the WLM necessary for the ensuring the WLM is modular and extensible. In Stage 3, the generalised traffic Model behaviour is identified and related to the WLG behaviour in order to define the WLM interface. The various TGs need only inherit from the traffic Model entity and overwrite the desired methods to be included in the WLG. In order for the WLM generator to be general, it must be

- modular, accommodating easy and effective TM integration, and
- extensible, in that it allows the integration of multiple traffic models and the related TG.

In designing the WLG and the WLM interface, we consider three design stages, as shown in Figure 5. First, in Stage 1, we define the MMBP in terms of its components (nouns) and

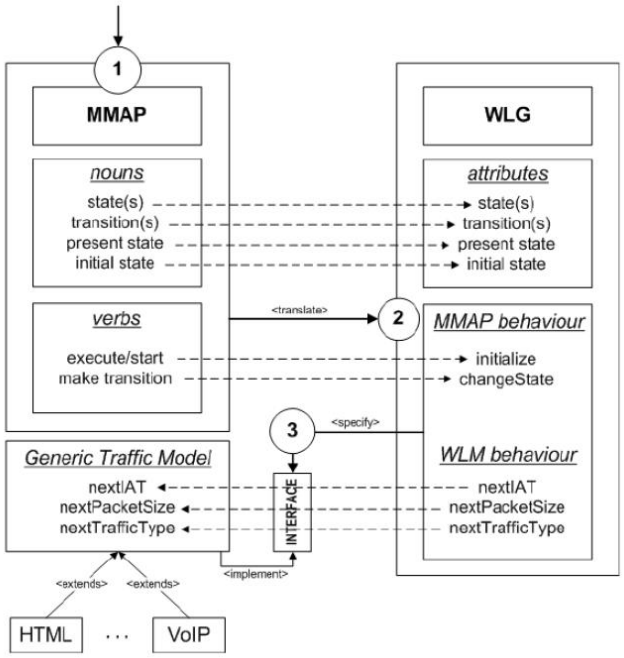

Fig. 5. Overview of the WLM design showing the translation from MMBP to WLM generator, extending the WLM Generator and showing the Traffic Model interface design

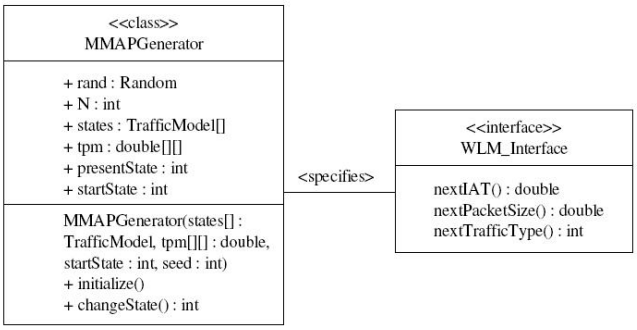

Fig. 6. MMBP WLM Generator and WLM interface class diagram

capabilities (verbs). The MMBP entity is then, in Stage 2, translated into the WLG entity where nouns are translated into entity attributes and verbs into behaviours. The MMBP entity is also extended to include the behaviour specific to workload generation. An interface between the WLM and generic TM is a component of the WLM necessary for the ensuring the WLM is modular and extensible. In Stage 3, the generalised traffic model behaviour is identified and related to the WLG behaviour in order to define the WLM interface. The various TGs need only inherit from the traffic Model entity and overwrite the desired methods to be included in the WLG.

C. Implementation

The Figure 6 show the class diagram of the MMBP WLM Generator and WLM interface. The code is given in Appendix B

III. Traffic Models and Generators for Internet Traffic

The internet traffic types, and their respective TM and TG, we have chosen are the following:

1) Web browsing using the HTML protocol.
2) VoIP using UDP.
3) Real Time Video Streaming
4) Peer to Peer (P2P) traffic. In particular, for the P2P applications,
   - BitTorrent and
   - Gnutella

   are predominant.

In the following sections we briefly describe each type with regard to packet inter arrival time (IAT), packet size and the corresponding traffic generators.

A. HTML

Many studies such as those by Lourens [21] or Frost [10] of internet traffic exist. Walter Lourens [21], following Staehle *et al* [9] derived a workload model of traffic generated by an individual browsing the web. Lourens measured internet traffic over a wire-line network and fitted various distributions to the measured data. In particular, the inter-arrival time (IAT) and packet size distributions in both up-link (UL) and down-link (DL) directions.

The distribution for the HTML workload model that we have employed to realize the HTML TM Generator is reported in Table I and in Table II.

B. VoIP

Voice over Internet Protocol (VoIP) is a no an established technology. When modeling voice traffic, it is important to understand the speech process between two parties. There are two relevant events or state that are important to model voice traffic:

- talk-spurt and
- silence.

We can think of voice traffic generated by using a two state process. In other words a user alternates between a period of talk-spurt and a period of silence (listening). For simplicity, we indicate talk-spurt period with "ON period" and silence period with "OFF period" as illustrated in Figure 7.

Chuah [6] derived a workload model of traffic generated by Voice over IP. The ON period and OFF period are exponentially distributed with average duration with $\frac{1}{\alpha}$ and $\frac{1}{\beta}$, respectively. We set $\frac{1}{\alpha}$ and $\frac{1}{\beta}$ to be 1.004 seconds and 1.587 seconds, respectively Why? Where do these values come from? [16]?

Is important to understand that the exponential distribution is used to model the total "duration" of each period. In order to model the IAT and packet size processes, in the ON Period, fixed-size packets are generated at a constant interval since VoIP is Constant Bit Rate (CBR) process so the IAT is deterministic and constant. No packets are transmitted in the OFF Period. The size of the packets is again deterministic and depends on the corresponding voice codec.

Traditionally voice is Pulse Code Modulated (PCM). We use a PCM codec to generate Inter-Arrival Time and Packets Size

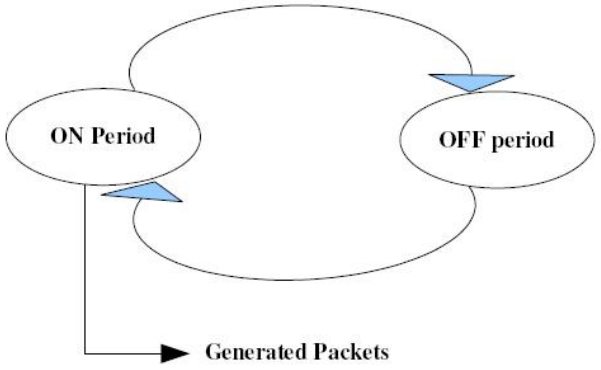

Fig. 7. Two-state process to modeling voice traffic

for our VoIP TM Generator. With a PCM codec the deterministic IAT is equal to 20 milliseconds [16]. Simultaneously the packet size is equal to 200 bytes [6] with 12 bytes for the RTP header, an 8 byte UDP header, a 20 bytes IP header and 160 bytes of data. The VoIP workload model that we employ for the VoIP TM Generator is reported in Table III.

C. Real Time Video Streaming

Real Time Video Streaming (rtVideo) applications include e-learning, video conferencing, Video-on-Demand etc. The main goal of streaming is that the data should arrive and play out continuously without interruption. However, this is constrained by fluctuations in network conditions. In order to minimize this, an adaptive streaming server keeps track of the network conditions and adapts the quality of the stream to minimize interruptions [8].

The process of creating rtVideo is to:

1) Capture the video content.
2) Digitize and edit the video file.
3) Encode the streaming video.
4) Deliver the streaming video.

Video traffic on the internet uses many types of Codec to encode the streaming video. There are two methods of encoding: Constant Bit Rate (CBR) and Variable Bit Rate (VBR). CBR is used by applications that require a fixed data rate that is continuously available during the connection lifetime and with a relatively tight upper bound on transfer delay. CBR is commonly used for uncompressed audio and video data. Also CBR encoding is designed to work optimally in a variety of streaming scenarios. It is possible to constrict the bit rate to guarantee consistent playback across a wide range of systems. The bit rate remains fairly constant and close to the target bit rate over the lifetime of the rtVideo stream.

VBR encoding is designed to work optimally in high bandwidth scenarios and is especially suited for encoding content that is a mixture of simple and complex data. The encoder allocates fewer bits to the simple parts of the content, leaving enough bits available to produce good quality for the more complex portions.

In [17] a rtVideo is modeled as a VBR characterized by a Pareto distribution. The size of a video-packet also follows a Pareto distribution. The distributions of the rtVideo workload

Model Parameter	Distribution	Parameters
Web Client Request IAT	Weibull	γ=0.371 α= 315778.506

TABLE I
HTML IAT WORKLOAD DISTRIBUTIONS

Model Parameter	Distribution	Parameters
Non-cached Web Client Response Size	Lognormal	ζ= 7.401 σ=1.405
Web Client Request Size	Lognormal	ζ= 5.883 σ=0.331

TABLE II
HTML PACKET SIZE WORKLOAD MODEL

	ON Period	OFF Period
Distribution	Exponential β=1sec	Exponential β=0.6sec
IAT	deterministic, 20ms	0
Packet Size	deterministic, 200 bytes	0

TABLE III
VOIP WORKLOAD MODEL ASSUMING PCM CODEC

Component	Distribution	Parameters
IAT	Truncated Pareto	α = 1.2, K = 2.5 ms, mean = 6 ms, maximum = 12.5 ms
Packet Size	Truncated Pareto	α = 1.2, K = 40 bytes, mean = 100 bytes, maximum = 250 bytes

TABLE IV
REAL TIME VIDEO STREAMING)(RTVIDEO) WORKLOAD MODEL

model that we have chosen are listed in Table IV derived from [17] and [23]:

D. Peer-to-Peer (P2P)

Peer-to-peer (P2P) is a communications architecture in which each party, or peer, has the same capabilities and either party can initiate a communication session. In [3] a P2P network is defined as a network of computers configured to allow particular files and folders to be shared with everyone or with selected users. All versions of Windows, Mac and Linux can function as a peer in a P2P network and allow their files to be shared. Files are shared directly between systems on the network without the need for a central server. In other words, each computer on a P2P network becomes a file server as well as a client. P2P describe applications in which users can use the Internet to exchange files with each other directly or through a mediating server. The only requirements for a computer to join a P2P network are an Internet connection and P2P software.

A major influence on the development of the internet, after the emergence of the World Wide Web, was the appearance of the P2P applications, such as Napster[12], Kazaa[7], Gnutella[1], BitTorrent[2], and so on and many measurement studies have been done such as, for example, by Saroiu *et al* [18].

An internet traffic study[14] by the German company, IPOQUE, determined that BitTorrent and Gnutella are the two dominant P2P applications. Hence we have chosen measurement of these two applications as representative as representative of P2P traffic on the internet.

In the following sections we discuss each of these applications and the packet level model we use in the WLM.

1) BitTorrent: BitTorrent, defined in [4], is a P2P application where users download from other users and do not use a centralized directory as in the original Napster service. BitTorrent also makes every downloading user an uploading user. It was released in the summer of 2001.

The term "BitTorrent" refers to the small metadata file the user receives upon clicking on a download link on a website (a file that ends in `.torrent`). Metadata here means that the file contains information about the data to be downloaded, not the data itself. Instead of downloading the entire file, BitTorrent breaks it into chunks and distributes these among several participating users. When a user downloads a "torrent", it also uploads it to another user. BitTorrent ensures every user participates in uploading. Figure 8 illustrates a BitTorrent application. The traffic distributions for the P2P BitTorrent workload model that we have employed for the BitTorrent TM generator are listed in Table V from [11]:

2) Gnutella: Gnutella[1] is a decentralized P2P application first released in 2000. Using a Gnutella client, a user can search, download and upload files from anywhere on the internet. The initial list of hosts is usually obtained from a transitory Web lookup. A peer broadcasts a Ping message to find the active hosts forming the initial network. Those hosts are neighbours and the actual query is broadcast to the neighbouring hosts. The neighbouring peer sends the query to its own neighbours again, even when it itself has the target file. All the messages are forwarded to neighbours within the limited number of hops defined by the Time to Live (TTL) in the message header. If the desired files were found, the servant sends back the matched result set to the neighbour through which it received the query. Figure 9 illustrates the Gnutella application.

The distributions for P2P Gnutella workload model that we have employed for the P2P Gnutella TM Generator are given in Table VI from [11].

Component	Distribution	Parameters
IAT	-	0
Packet size	Deterministic	128 bytes

TABLE V
P2P BITTORRENT WORKLOAD MODEL

Traffic Type	IAT distribution/parameter values	packet size distribution/parameters
HTML Downlink	Weibull, γ = 0.371 σ = 1.405	Lognormal ζ = 5.884 σ = 0.331
HTML Uplink	Weibull, γ = 0.371 σ = 1.405	Lognormal ζ = 7.401 σ = 1.405
VoIP ON Period	Deterministics, 20 milisecond	Deterministic, 200 bytes
VoIP OFF Period	None	None
rtVideo Streaming	Truncated Pareto α = 1.2, K = 2.5ms, Mean = 6 ms Maximum = 12.5 ms	Truncated Pareto, α = 1.2 K = 40 bytes, Mean = 100 Maximum = 250 bytes
P2P_BitTorrent	None	Deterministic, 128 bytes
P2P_Gnutella	None	Deterministic, 528 bytes

TABLE VII
TRAFFIC TYPES WORKLOAD MODELS

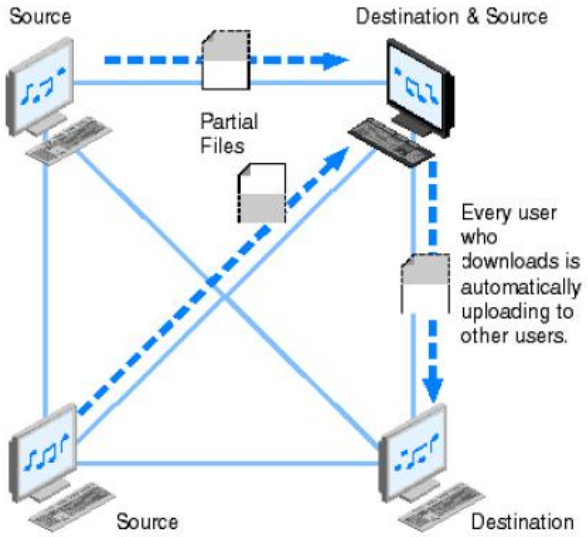

Fig. 8. The P2P BitTorrent application

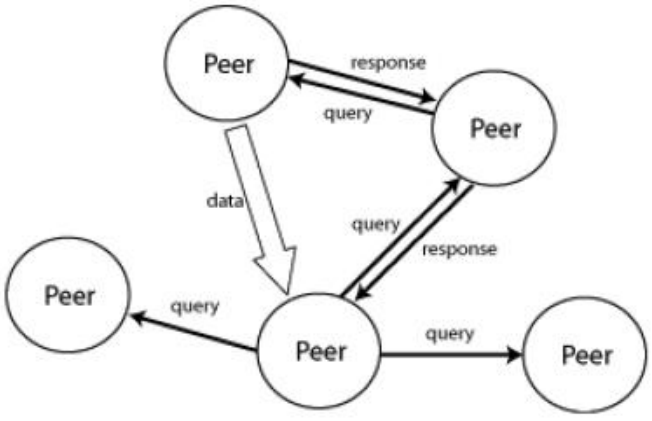

Fig. 9. The Gnutella application

E. Summary

Finally, Table VII lists the distributions and their corresponding parameters for the Packet Level Model of the traffic types, or TGs.

Component	Distribution	Parameters
IAT	None	0
Packet Size	Deterministic	fixed 528 bytes

TABLE VI
P2P GNUTELLA WORKLOAD MODEL

IV. TRAFFIC MODEL

The Traffic Model (cf. Figure 5) is that component of the WLM that presents the individual Traffic Generators to the MMBP. I am not sure *what* you wanted to say here Sonia.

In Section III we described the distributions for the IAT and size of packets for each traffic type chosen for our WLM. In the following sections we explain the design and implementation we have chosen.

A. HTML TG design and implementation

For the HTML TG we distinguish between Uplink and Downlink traffic since the traffic characteristics are different for each. In order to decide whether the TG is in the Uplink state or the Downlink state we generate a random variable that reflects the amount of average amount of traffic respectively in the uplink and the downlink.

Figure 10 show the HTML TM Generator Design, the distributions chosen and the parameter values we use for each distribution.You have to be consistent. In this figure you give the parameter values of the distributions. In what follows you do not. Even if the value is constant, give the value. The Figure 11 shows the class diagram of the HTML TM Generator. The code is listed in Appendix C.

B. VoIP design and implementation

As is commonly done, we assume the TG to be in either an OFF state (silent) or an ON state. A VoIP session begins with an ON period which (cf Section III) is modeled by an exponential distribution with mean $\alpha = 1$ second. During ON Period the IATs are deterministic and with a PCM codec equals

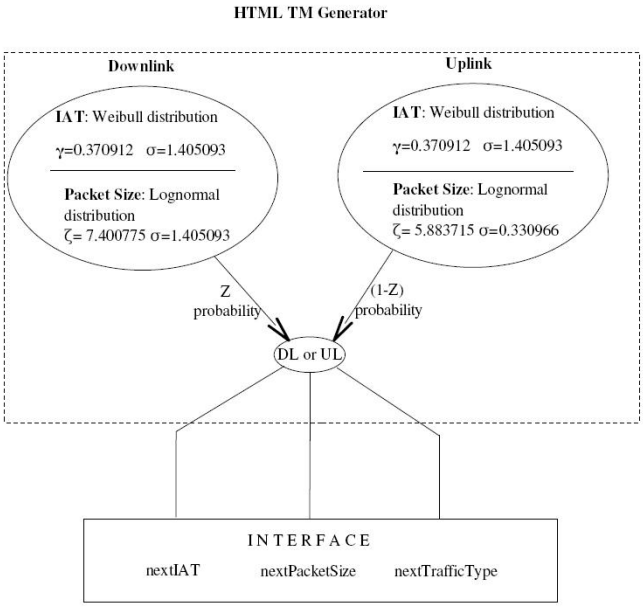

Fig. 10. HTML TM Generator

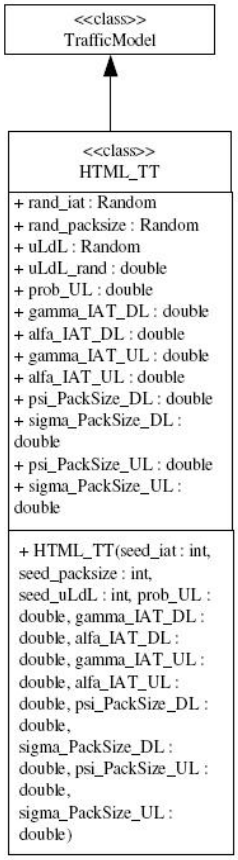

Fig. 11. HTML TG Class Diagram

20 milliseconds. The Packet Size is also constant and equal to 200 bytes with a PCM codec.

In the OFF Period we have the duration modeled by exponential distribution with mean 0.6 seconds. There is no traffic during this period.

The Figure 12 show how we have represented our VoIP TM Generator. Figure 13 illustrates the class diagram of the VoIP

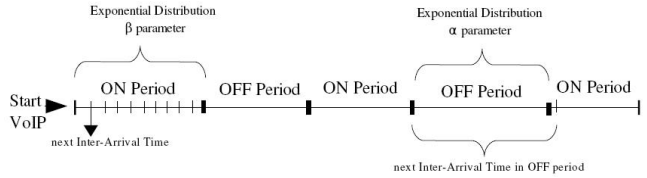

Fig. 12. VoIP TM Generator

TM Generator. The code is given in Appendix C.

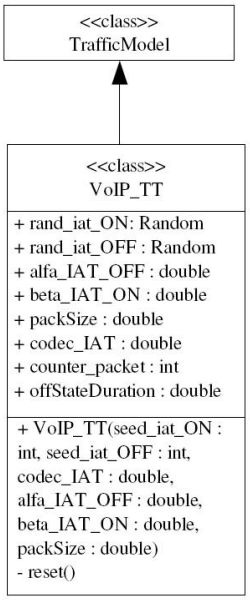

Fig. 13. VoIP TM Class Diagram

C. rtVideo design and implementation

The rtVideo TM Generator is characterized by a Pareto distribution and can incorporate any codec for the generation of video traffic. The IAT is measured in milliseconds and packet size is measured in bytes. Figure 14 illustrates the Real Time Video Streaming TM Generator Design. Figure 15 shows the class diagram of the HTML TM Generator. The code is given in Appendix C.

D. P2P BitTorrent and Gnutella design and implementation

P2P traffic constitutes by far the most traffic on the internet. So much so that some commentators have predicted that the internet will soon collapse under the volume of traffic. Different from normal HTML, data or video traffic, response times are virtually irrelevant and transactions can last days. On the scale of normal traffic the application IAT is huge and there is almost always bits of file to be transferred. In other words, the mean IAT is chosen to be zero while the packet size is fixed as shown in the figures.

Figure 16 show the P2P BitTorrent TM Generator Design. Figure 17 report the class diagram of the P2P BitTorrent TM

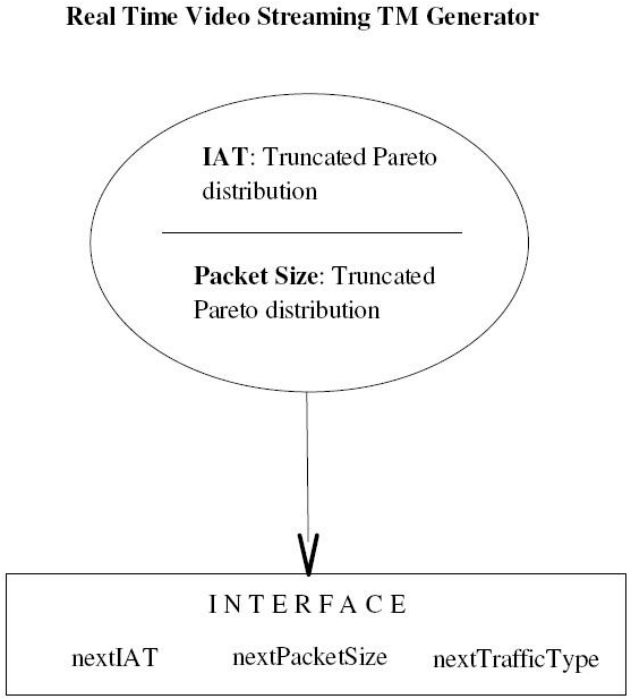

Fig. 14. Real Time Video Streaming TM Generator

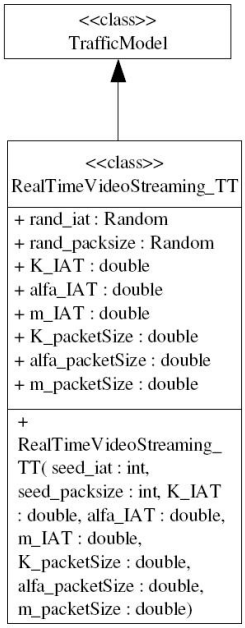

Fig. 15. Real Time Video Streaming TM Class Diagram

Generator. The code is given in Appendix C. Figure 19 shows the class diagram of the P2P Gnutella TM Generator. The code is listed in Appendix C.

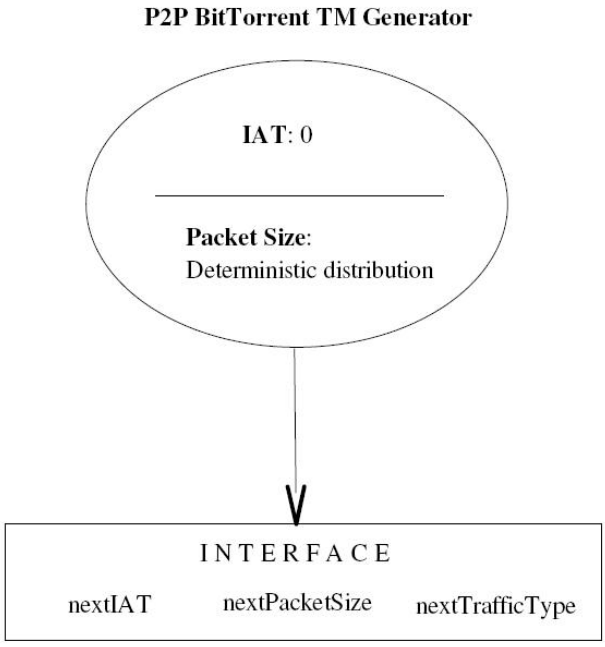

Fig. 16. P2P BitTorrent TM Generator

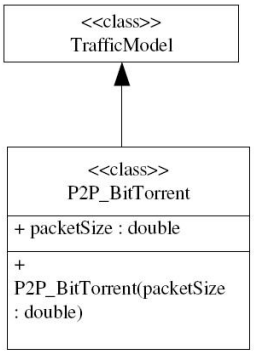

Fig. 17. P2P BitTorrent TM Class Diagram

REFERENCES

[1] http://ww2.cs.fsu.edu/ jungkkim/P2P.html, 2008.
[2] http://www.bittorrent.org, 2008.
[3] "encyclopedia2," http://encyclopedia2.thefreedictionary.com/, 2008.
[4] "encyclopedia2," http://encyclopedia2.thefreedictionary.com/, 2008.
[5] C. L. A. Klemm and M. Lohmann, "Modeling IP Traffic Using the Batch Markovian Arrival Process," *Performance Evaluation*, vol. 54, pp. 149–173, 2003.
[6] C.-N. Chuah, "A Scalable Framework for IP-Network Resource Provisioning Through Aggregation and Hierarchical Control," Ph.D. dissertation, University of California at Berkeley, 2001.
[7] E. Cohen, "Replication strategies in unstructured peer-to-peer networks," 2002, pp. 177–190.
[8] N. Cranley and M. Davis, "Performance evaluation of video streaming with background traffic over IEEE 802.11 WLAN networks," in *WMuNeP*, A. A. F. Loureiro and W. Zhuang, Eds. ACM, 2005, pp. 131–139.
[9] K. L. D. Staehle and P. Tran-Gia, "Source Traffic Modeling of Wireless Applications," University of Würzburg, Technical Report TR 261, 1999.
[10] V. Frost and B. Melamed, "Traffic Modeling for Telecommunications Networks," *IEEE Communications Magazine*, pp. 70–81, March 1994.
[11] M. Perényi, T. D. Dang, A. Gefferth, and S. Molnár, "Identification and analysis of peer-to-peer traffic," *JCM*, vol. 1, no. 7, pp. 36–46, 2006.
[12] S. P. Ratnasamy, S. P. Ratnasamy, and S. P. Ratnasamy, "A scalable content-addressable network," in *In Proceedings of ACM SIGCOMM*, 2001, pp. 161–172.
[13] J. Robinson and T. Randhawa, "Saturation throughput analysis of IEEE 802.11e enhanced distributed coordination function," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 5, pp. 917–928, 2004.
[14] H. Schulze and K. Mochalski, "Internet study 2007," http://www.ipoque.com/resources/internet-studies/internet-study-2007, 2007.

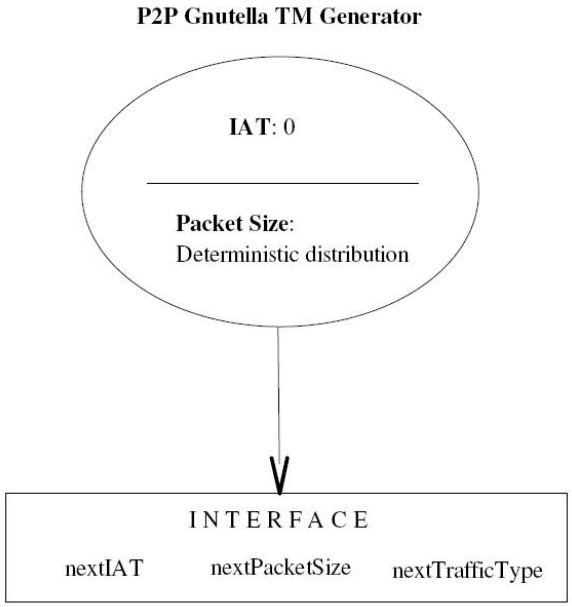

Fig. 18. P2P Gnutella TM Generator

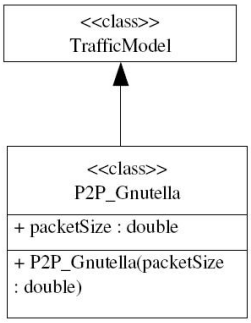

Fig. 19. P2P Gnutella TM Class Diagram

[15] ——, "The Impact of P2P File Sharing, Voice over IP, Skype, Joost, Instant Messaging, One-Click Hosting and Media Streaming such as YouTube on the Internet," http://www.ipoque.com/userfiles/file/internet_study_2007.pdf, 2007.
[16] J. Seger, "Modelling Approach for VoIP Traffic Aggregations for Transferring Tele-traffic Trunks in a QoS enabled IP-Backbone Environment," in *International Workshop on Inter-domain Performance and Simulation*, 2003, faculty for Electrical Engineering and Information Technology Department of Electronic Systems and Switching University of Dortmund.
[17] S. Shin and B.-H. Ryu, "Packet loss fair scheduling scheme for real-time traffic in OFDMA systems," vol. 26, no. 5, pp. 391–396, oct 2004.
[18] S. D. G. Stefan Saroiu, P. Krishna Gummadi, "A Measurement Study of Peer-to-Peer File Sharing Systems," Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA, 98195-2350, Technical Report UW-CSE-01-06-02, 2002.
[19] A. Symington, "A Hardware Testbed for Measuring IEEE 802.11g DCF Performance," Master's thesis, University of Cape Town, Department of Computer Science, December 2008.
[20] R. S. W. Willinger, M. S. Taqqu and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, p. 7186, 1997.
[21] L. O. Walters, "A Web Browsing Workload Model For Simulation," Master's thesis, University of Cape Town, May 2004.
[22] H. Wang, B. He, and D. P. Agrawal, "Above packet level admission control and bandwidth allocation for IEEE 802.16 wireless MAN," *Simulation Modelling Practice and Theory*, vol. 15, no. 4, pp. 366–382, April 2007.
[23] H. Xu, "Video streaming traffic model for 802.16m evaluation methodology document," November 2007, iEEE 802.16 Broadband Wireless Access Working Group.

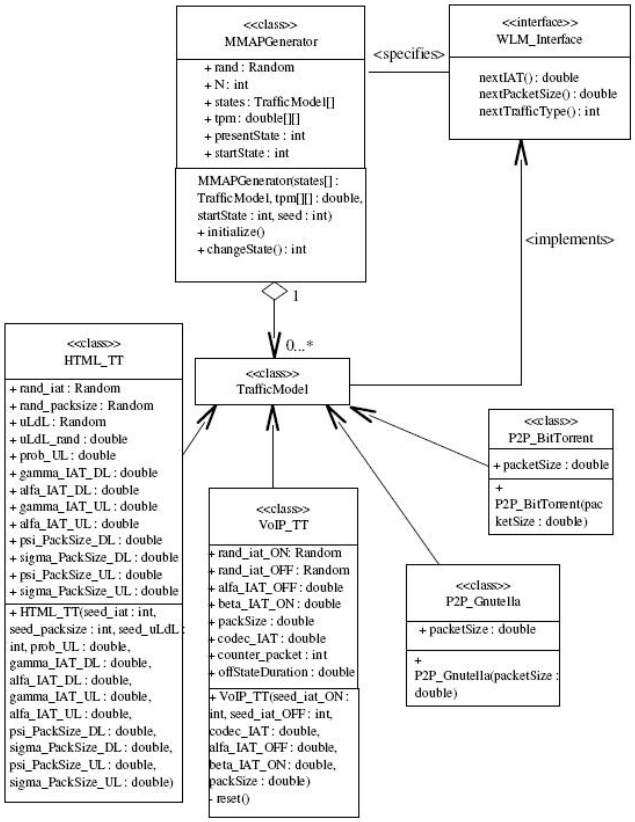

Fig. 20. WLM Class Diagram

APPENDIX A
WLM CLASS DIAGRAM

Figure 20 illustrates the WLM Class Diagram.

APPENDIX B
MMBP GENERATOR, WLM INTERFACE AND MMBP TEST SOURCE CODE

MMBPGenerator.java

```
import java.util.Random;

public class MMBPGenerator
{
    //attributes
    Random rand;
    int N;//array dimension
    TrafficModel states[];//array of traffic (types) model
    double tpm[][];
    int presentState;
    int startState;

    //constructor
    public MMBPGenerator(TrafficModel states[], double tpm[][], int startState, int seed)
    {
        this.states = states;
        this.tpm = tpm;
        this.startState = startState;
        this.presentState = this.startState;
        this.rand = new Random(seed);
        N = this.states.length;//set the array dimension
    }

    //this method set the current state with the initial state
    void initialize()
    {
        presentState = startState;
    }

    //this method calculate the next state where we arrive depending on the transition probability matrix
    int changeState()
    {
        double num_rand = rand.nextDouble();
        int nextState=0;
        double compound=0.0;

        for(int i=0; i<N; i++)
        {
            if( num_rand < (tpm[presentState][i] + compound) )
            {
                nextState = i;
                    break;
```

```
            }
            compound= compound + tpm[presentState][i];
        }
            presentState = nextState;
            return (presentState);
    }

    //This method permit to User(Simulator) to ask for generating the next IAT
    double nextIAT()
    {
        return states[presentState].nextIAT();
    }

    //This method permit to User(Simulator) to ask for generating the next Packet Size
    double nextPacketSize()
    {
        return states[presentState].nextPacketSize();
    }

    //This method permit to User(Simulator) to ask for generating the next Traffic Type
    int nextTrafficType()
    {
        return states[presentState].nextTrafficType();

    }


    //main method
    public static void main(String args[])
    {
        int randomSeed=5;
        TrafficModel states[] = {new TrafficModel(), new TrafficModel()};
        double tpm[][] = {{0.3, 0.7},{0.8, 0.2}};// transition probability matrix
        int startState = 0; //initial state
        MMBPGenerator prova = new MMBPGenerator(states, tpm, startState, randomSeed); //Object
        prova.initialize();

        int samples = 10000000;
        int value[] = {0, 0};
        for(int i=0; i<samples; i++)
        {
            int ris = prova.changeState();
            value[ris]++;
        }
        System.out.println("number of 0 = "+value[0]);
        System.out.println("number of 1 = "+value[1]);

        //calculate "pi"
        double pi1 = ((double)value[0]/samples);
        double pi2 = ((double)value[1]/samples);
        System.out.println("pi1 = "+pi1);
        System.out.println("pi2 = "+pi2);

        //check pi
        double check_pi = (pi1 + pi2);
        System.out.println("somma pi= "+check_pi);
    }
}
```

WLM_Interface.java

```
/*
 Thanks to this interface anyone can use the workload model and can obtain packet informations that are:
 Inter Arrival Time, Packet Size and Traffic Type.
*/

public interface WLM_Interface
{
    double nextIAT();//this method gives the Inter Arrival Times packets

    double nextPacketSize();//this method gives the packets size

    int nextTrafficType();//this method gives the traffic type (i.e. HTML, P2P, ectera)

}
```

PiCalculator.java

```
/*
 * This progam determines [pi] in [pi]P=[pi] for some hard-coded transition probability matrix P
 */

public class PiCalculator
{
    public static void main(String [] args)
    {
        // the transition probability matrix P
        double tpm[][] = {{0.3,0.7},{0.8,0.2}};

        // initial pi, i.e. pi_0
        double pi_0[] = {0.5,0.5};
        System.out.println("pi_" + 0 + " = { " + pi_0[0] + ", " + pi_0[1] + "}");
        double pi_next[] = {0.0,0.0};
        // previous is used so that next updates properly, without overriding values before used in calculations
        double pi_previous [] = {pi_0[0],pi_0[1]};

        // the number of iterations to observe convergence
        int iterations = 60;

        for (int i = 1; i <= iterations; i ++)
        {
            // calculate the next pi from the previous pi
            pi_next [0] = (pi_previous[0]*tpm[0][0] + pi_previous[1]*tpm[1][0]);
            pi_next [1] = (pi_previous[0]*tpm[0][1] + pi_previous[1]*tpm[1][1]);
            // output
            System.out.println("pi_" + i + " = { " + pi_next[0] + ", " + pi_next[1] + "}");
            // update the previous to be the currently calculated pi
            pi_previous[0] = pi_next [0];
            pi_previous [1] = pi_next [1];
        }
    }
}
```

APPENDIX C
INTERNET TRAFFIC TYPE SOURCE CODE

TrafficModel.java

```
/*This class implements the WLM_Interface to obtain effectively packet informations*/

public class TrafficModel implements WLM_Interface
{
    public double nextIAT()
    {
        return 0.0;
    }

    public double nextPacketSize()
    {
        return 0.0;
    }

    public int nextTrafficType()
    {
        return 0;
    }

}
```

HTML_TT.java

```
/*
 * This class implements the HTML Traffic Types
 *
 * */


import java.util.Random;

public class HTML_TT extends TrafficModel
{
    Random rand_iat;//Object Random for IAT
    Random rand_packsize;//Object Random for Packet Size
    Random uLdL;//Object Random for Uplink or Downlink
    double uLdL_rand;//random varible that determine if work in DL or UL
    double prob_UL;//variable to determine DL or UL
    //parameters
    double gamma_IAT_DL, alfa_IAT_DL, gamma_IAT_UL, alfa_IAT_UL;
    double psi_PackSize_DL, sigma_PackSize_DL, psi_PackSize_UL, sigma_PackSize_UL;

    //constructor define the parameter list
    public HTML_TT(int seed_iat, int seed_packsize, int seed_uLdL, double prob_UL, double gamma_IAT_DL,
                   double alfa_IAT_DL, double gamma_IAT_UL, double alfa_IAT_UL, double psi_PackSize_DL,
                   double sigma_PackSize_DL, double psi_PackSize_UL, double sigma_PackSize_UL)
    {
        rand_iat = new Random(seed_iat);//parameter into the brackets is the seed for IAT
        rand_packsize = new Random(seed_packsize);//parameter into the brackets is the seed for Packet Size
        uLdL = new Random(seed_uLdL);////parameter into the brackets is the seed for Uplink or Downlink
        this.prob_UL = prob_UL;
        this.gamma_IAT_DL = gamma_IAT_DL;//weibull's gamma parameter (shape)
        this.alfa_IAT_DL = alfa_IAT_DL;//weibull's alpha parameter (scale)
        this.gamma_IAT_UL = gamma_IAT_UL;//weibull's gamma parameter (shape)
        this.alfa_IAT_UL = alfa_IAT_UL;//weibull's alpha parameter (scale)
        this.psi_PackSize_DL = psi_PackSize_DL;//lognormal's psi parameter (mean)
        this.sigma_PackSize_DL = sigma_PackSize_DL;//lognormal's sigma parameter (standard deviation)
        this.psi_PackSize_UL = psi_PackSize_UL;//lognormal's psi parameter (mean)
        this.sigma_PackSize_UL = sigma_PackSize_UL;//lognormal's sigma parameter (standard deviation)
        uLdL_rand=0;
    }

    /*Method to obtain the next IAT.
     * Is important to call before this method and then the method nextPacketSize()
     * */
    public double nextIAT()
    {

        uLdL_rand = uLdL.nextDouble();

        if(uLdL_rand >= prob_UL ){//DL if uLdL_rand>=0.5 (Downlink)

            double U = rand_iat.nextDouble();//Uniform distribution

            //inverse of weibull distribution (DL: Web Client Request IAT)
            return alfa_IAT_DL * Math.pow(-1*Math.log(U), (1/gamma_IAT_DL));//return random variable
        }
        else{//UL (The same of DL because if I do 100 request (UL) to Internet, I receive 100 response (DL))

            double U = rand_iat.nextDouble();//Uniform distribution

            //inverse of weibull distribution (UL: Web Client Request IAT)
            return alfa_IAT_UL * Math.pow(-1*Math.log(U), (1/gamma_IAT_UL));//return random variable
        }
    }

    /*Method to obtain the next Packet Size.*/
       public double nextPacketSize()
      {
        if(uLdL_rand >= prob_UL ){//DL (Downlink)

            //Normal distribution
            double normal_dist = 0.0;
            double var_rand;
            for(int i=0; i<12; i++){
                var_rand = rand_packsize.nextDouble();//random variable
                normal_dist = normal_dist + var_rand;
                var_rand=0.0;
            }
            normal_dist = normal_dist - 6;

            //inverse of lognormal ditribution (DL: Non-cached Web Client Response Size)
            return psi_PackSize_DL * Math.exp(sigma_PackSize_DL * normal_dist);
       }
       else{//UL (Uplink)

            //Normal distribution
            double normal_dist = 0.0;
            double var_rand;
```

```
            for(int i=0; i<12; i++){
                var_rand = rand_packsize.nextDouble();//random variable
                normal_dist = normal_dist + var_rand;
                var_rand=0.0;
            }
            normal_dist = normal_dist - 6;

            //inverse of lognormal ditribution (UL: Web Client Request Size)
            return psi_PackSize_UL * Math.exp(sigma_PackSize_UL * normal_dist);
        }
    }

   /*Method to obtain the next Traffic Type.*/
    public int nextTrafficType()
    {
        return 0;//the value 0 identified the HTML Traffic Type
    }


    public static void main(String[] args)
    {
        HTML_TT gen = new HTML_TT(3, 3333, 7, 0.5, 0.370912, 315778.506, 0.370912, 315778.506, 7.400775, 1.405093, 5.883715, 0.330966);

        for(int i=0; i<10000; i++){
            gen.nextIAT();
            gen.nextPacketSize();
        }
    }

}
```

VoIP_TT.java

```
/*
 * This class implements the VoIP Traffic Types
 *
 * */

import java.util.Random;

public class VoIP_TT extends TrafficModel
{

    Random rand_iat_ON;//Object Random for IAT in ON period
    Random rand_iat_OFF;//Object Random for IAT in OFF period
    //parameter
    double alfa_IAT_OFF;
    double beta_IAT_ON;
    double packSize;
    double codec_IAT; //interarrival time with pcm codec in millisecond in this case
    int counter_packet;//counter of the packets sends during ON period
    double offStateDuration = 0.0;

    public VoIP_TT(int seed_iat_ON, int seed_iat_OFF, double codec_IAT, double alfa_IAT_OFF, double beta_IAT_ON, double packSize)
    {
        rand_iat_ON = new Random(seed_iat_ON);//parameter into the brackets is the seed for IAT in ON period
        rand_iat_OFF = new Random(seed_iat_OFF);//parameter into the brackets is the seed for IAT in OFF period

        this.alfa_IAT_OFF = alfa_IAT_OFF;
        this.beta_IAT_ON = beta_IAT_ON;
        this.packSize = packSize;
        this.codec_IAT = codec_IAT;

        //we assume that the comunication start with an ON period, so we call reset method
        this.reset();
    }


    private void reset()
    {
        double X_ON = rand_iat_ON.nextDouble();//random variable
        double tStateON = beta_IAT_ON*Math.exp(-1*(beta_IAT_ON*X_ON));//duration time in ON period modelled with exponential ditribution
        counter_packet = (int)(tStateON / codec_IAT);//number of packets sends during ON period
        System.out.println(counter_packet);
    }

    /*Method to obtain the next IAT.
     * Is important to call before this method and then the method nextPacketSize()
     * */
    public double nextIAT()
    {
        if(counter_packet > 0)//ON period
        {
            counter_packet--;//send one packet
            return codec_IAT;//the user have the next IAT
        }
        else//OFF period
        {
            System.out.println("In off state");
            double X_OFF = rand_iat_OFF.nextDouble();//random variable
            double tStateOFF = alfa_IAT_OFF*Math.exp(-1*(alfa_IAT_OFF*X_OFF));//duration time in OFF period modelled with exponential ditribution

            offStateDuration = tStateOFF;

            //calcute ON period because after duration of OFF period will be an ON period and after the first packet send in ON period we will have the next IAT
            this.reset();

            counter_packet--;//send one packet
            return (codec_IAT + offStateDuration);//the next IAT will be after the duration of the OFF period plus the first packet send in the ON period
        }
    }

    /*Method to obtain the next Packet Size.*/
    public double nextPacketSize()
    {
    %\bibliographystyle{IEEE}
\bibliography{myrefs}   return packSize;//fixed-size packet with PCM codec 200 bytes = 12 byte RTP header + 8 byte UDP header + 20 byte IP header + 160 byte data
    }

    /*Method to obtain the next Traffic Type.*/
    public int nextTrafficType()
    {
        return 1;//the value 1 identified the VoIP Traffic Type
    }
```

```
    public static void main(String[] args)
    {
        VoIP_TT gen = new VoIP_TT(30, 289, 0.020, 0.6, 1.00, 200.0);
        double time =0;
        for(int i = 0; i < 1000; i++)
        {
            System.out.println(time);
            double next = gen.nextIAT();
            time += next;
        }

    }
}
```

RealTimeVideoStreaming_TT.java

```
/*
 * This class implements the Real Time Video Streaming Traffic Types
 *
 * */

import java.util.Random;

public class RealTimeVideoStreaming_TT extends TrafficModel
{

    Random rand_iat;//Object Random for IAT
    Random rand_packsize;//Object Random for the Packet Size
    //parameter
    double K_IAT;
    double alfa_IAT;
    double m_IAT;
    double K_packetSize;
    double alfa_packetSize;
    double m_packetSize;


    public RealTimeVideoStreaming_TT(int seed_iat, int seed_packsize, double K_IAT, double alfa_IAT, double m_IAT, double K_packetSize, double alfa_packetSize, double m_packetSize)
    {
        rand_iat = new Random(seed_iat);//parameter into the brackets is the seed for IAT
        rand_packsize = new Random(seed_packsize);//parameter into the brackets is the seed for Packet Size
        this.K_IAT = K_IAT;
        this.alfa_IAT = alfa_IAT;
        this.m_IAT = m_IAT;
        this.K_packetSize = K_packetSize;
        this.alfa_packetSize = alfa_packetSize;
        this.m_packetSize = m_packetSize;
    }

    /*Method to obtain the next IAT.
     * Is important to call before this method and then the method nextPacketSize()
     * */
    public double nextIAT()
    {
        double U = rand_iat.nextDouble();//Uniform distribution

        //inverse of Truncated Pareto CDF
        System.out.println("IAT: "+ K_IAT / Math.pow((1 - (U * (1 - Math.pow(K_IAT/m_IAT, alfa_IAT)))), 1/alfa_IAT));
        return K_IAT / Math.pow((1 - (U * (1 - Math.pow(K_IAT/m_IAT, alfa_IAT)))), 1/alfa_IAT);
    }

    /*Method to obtain the next Packet Size.*/
    public double nextPacketSize()
    {
        double U = rand_iat.nextDouble();//Uniform distribution

        //inverse of Truncated Pareto CDF
        System.out.println("PS: "+ K_packetSize / Math.pow((1 - (U * (1 - Math.pow(K_IAT/m_packetSize, alfa_packetSize)))), 1/alfa_packetSize));
        return K_packetSize / Math.pow((1 - (U * (1 - Math.pow(K_IAT/m_packetSize, alfa_packetSize)))), 1/alfa_packetSize);
    }

    /*Method to obtain the next Traffic Type.*/
    public int nextTrafficType()
    {
        return 2;//the value 2 identified the Real Time Video Streaming Traffic Type
    }

    public static void main(String[] args)
    {
        RealTimeVideoStreaming_TT gen = new RealTimeVideoStreaming_TT(16, 5000, 0.0025, 1.2, 0.0125, 40, 1.2, 250);

        for(int i=0; i<10000; i++){
            gen.nextIAT();
            gen.nextPacketSize();
        }

    }

}
```

P2P_BitTorrent.java

```
/*
 * This class implements the BitTorrent Traffic Types belonging to P2P Traffic
 *
 * */

public class P2P_BitTorrent extends TrafficModel
{
    //parameter
    double packetSize;

    //constructor
    public P2P_BitTorrent(double packetSize)
    {
        this.packetSize = packetSize;
    }

    /*Method to obtain the next IAT.
     * Is important to call before this method and then the method nextPacketSize()
     * */
    public double nextIAT()
```

```
    {
        return 0.0;/* with P2P traffic we haven't a distribution for Inter Arrival Time at packet level,
                      for this reason we model IAT equal to zero that meaning that in
                      entry in a P2P's MMBP state is sure that there are IAT and we can assume that
                      are endless so we have decide to model this with zero value*/
    }

    /*Method to obtain the next Packet Size.*/
    public double nextPacketSize()
    {
        //Deterministic distribution
        System.out.println("PS: "+packetSize);
        return packetSize;
    }

    /*Method to obtain the next Traffic Type.*/
    public int nextTrafficType()
    {
        return 3;//the value 3 identified the BitTorrent Traffic Type
    }

    public static void main(String[] args)
    {
        P2P_BitTorrent gen = new P2P_BitTorrent(128);//128 byte is the size of BitTorrent packet
        gen.nextPacketSize();
    }
}
```

P2P_Gnutella.java

```
/*
 * This class implements the Gnutella Traffic Types belonging to P2P Traffic
 *
 * */

public class P2P_Gnutella extends TrafficModel
{
    //parameter
    double packetSize;

    //constructor
    public P2P_Gnutella(double packetSize)
    {
        this.packetSize = packetSize;
    }

    /*Method to obtain the next IAT.
     * Is important to call before this method and then the method nextPacketSize()
     * */
    public double nextIAT()
    {
        return 0.0;/* with P2P traffic we haven't a distribution for Inter Arrival Time at packet level,
                      for this reason we model IAT equal to zero that meaning that in
                      entry in a P2P's MMBP state is sure that there are IAT and we can assume that
                      are endless so we have decide to model this with zero value*/
    }

    /*Method to obtain the next Packet Size.*/
    public double nextPacketSize()
    {
        //Deterministic distribution
        System.out.println("PS: "+packetSize);
        return packetSize;
    }

    /*Method to obtain the next Traffic Type.*/
    public int nextTrafficType()
    {
        return 4;//the value 4 identified the Gnutella Traffic Type
    }

    public static void main(String[] args)
    {
        P2P_Gnutella gen = new P2P_Gnutella(528);//528 byte is the size of BitTorrent packet
        gen.nextPacketSize();
    }
}
```