

GenSyn - a Java-based Generator of Synthetic Internet Traffic Linking User Behaviour Models to Real Network Protocols

Poul E. Heegaard
Telenor R&D, N-7004 Trondheim, Norway¹
phone: +47 7354 3845,
fax: + 47 7354 3700,
e-mail: Poul.Heegaard@telenor.com

Abstract

For the purpose of Quality of Service testing of new applications and network mechanisms in the Internet, a generator of controllable, scalable, synthetic but realistic IP traffic is required. This paper describes GenSyn - a generator of synthetic Internet traffic implemented in Java.

GenSyn has defined a flexible and scalable modelling framework. The stochastic user behaviour is described by state diagrams. The model is scalable because it allows a composition of users in each state, instead of creating a new instance of the process for every user. The stochastic user behaviour model controls the creation of TCP connections and UDP streams through interface modules that links the GenSyn process to the underlying Internet protocol stack on the workstation. This means that on transitions between specific states in the stochastic model, an interface process will be initiated and IP packets are sent and received through the network.

In this paper the flexible and scalable modelling framework of GenSyn is described, and two model examples are shown, an elastic TCP traffic stream (Web client) and a UDP traffic stream (video server). The main experiences so far is that the modelling framework is very flexible and that adding new interface modules for new model types is a straight forward task within the defined framework. The reason is mainly that GenSyn is implemented in Java. Furthermore, Java makes GenSyn easily portable which has been demonstrated by running GenSyn on top of Unix, Linux, and NT-platforms. However, the Java Virtual Machines limits the scalability because the time granularity is imprecise when the time between events is in order of millisecond. This will, for instance, may introduce jitter in real-time traffic streams like VoIP.

Keywords: traffic generator, user behaviour, real Internet traffic, TCP, UDP, IP, workload benchmark, web, MPEG, Java, QoS testing

1. Introduction

For the purpose of Quality of Service testing of new applications and network mechanisms in the Internet, a generator of controllable, re-producible, scalable, synthetic but realistic traffic is required. The generator will typically produce traffic in a controlled testbed environment where there are few real users and a corresponding low traffic load. Realistic, controllable, and re-producible traffic load is of great importance in order to test the Quality of Service of new applications, i.e. interactive video. In addition, to increase the understanding and get experience with the configuration of new QoS network mechanisms, the network must be offered a high load of heterogeneous traffic mixture.

Different source modelling approaches can be considered to describe a typical Internet source:

1. *Trace* - replay of a recorded stream of IP packets, i.e. a stream of IP packets obtained from measurements is replayed and offered to the test network (e.g. replay of a tcpdump-file). If the recorded stream contains traffic from elastic sources (e.g. TCP connections) the replay will not be representative unless the congestion situation in the network is exactly the same. This will rarely be the case in a network with a mixture of traffic streams.
2. *Aggregate generator* ("black box") - generation of IP packets according to a parametric stochastic process. If a recorded aggregate of packets from elastic sources is used to determine the parameters of the model, the same problem as described in *Trace* above will still be present.
3. *User behaviour model* ("white box") - generation of IP packets from physically based source models. More detailed measurements than for the two other approaches are required. However, the parameters in the model reflect the user behaviour and hence it is straight forward to change the model if e.g. the number of sources will be changed.

1. The initial work of the GenSyn development was carried out as an employee at SINTEF Telecom & Informatics.

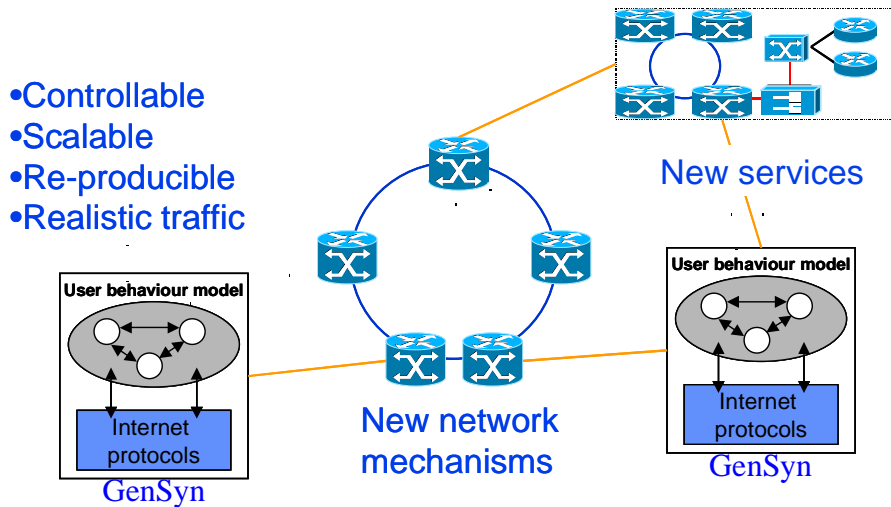


Figure 1 Load generation in an Internet test environment

For generation of traffic in Internet testbeds both user behaviour models, see e.g. [MTW99], [ChLi99], [Vic98], [Ake99], and bulk-transfers, e.g. tcp [MTW99], are being used. GenSyn combines the user behaviour approach with bulk-transfer of data via communication streams. Application of stochastic user behaviour models described by state diagrams introduces flexibility, scalability, physically interpretable model parameters. This part of the modelling framework is similar to ideas used in a former ATM traffic generator, called ATM100 or STG¹ [HMM93]. However, instead of developing a specialized hardware instrument, GenSyn applies modern Web- and Java-technology and exploits the Internet protocol suite (TCP/IP) that is already available. This means that the generator is only a software process that imitates the user behaviour and dynamically controls the creation and deletion of one or more links (threads) to physical HTTP- and TCP-connections. The generator is not only a simulator, it generates real IP packets that flow through a real (test) Internet.

In this paper the modelling framework of GenSyn is described. The general framework is given in Section 2, while Section 3 contains two modelling examples, a Web client and an MPEG coded video stream. Section 4 discusses implementation issues and limitations in the current version of GenSyn. Finally, the paper is closed in Section 5 with some general remarks and a list of ongoing and further work.

1. Synthesised Traffic Generator - an instrument for generation of ATM cells.

2. Modelling framework

2.1 Multi-level stochastic behaviour

The model described using the GenSyn framework will attempt to reproduce the inner workings of the physically source. This includes many stochastic processes, both human, environmental, and communication equipment. An example of the variety of activity levels in a source is illustrated in Figure 2. The example is originally from ATM, but this general multi-level behaviour is independent of the communication technology. Consider for instance a telephony user. The user is present (at the office) {end-user present level}, he is making a phone call {connection level}, during the call he is speaking and listening {dialogue level}, when he is speaking he sends voice and take short breaks {burst level}, when voice is sent it is wrapped in packets {packet level}, each packet is segmented in cells {cell level}. The last two levels are technology dependent. The typical time constants involved on various levels are indicated in the table in Figure 2. Hence, the aggregated packet or cell stream that can be observed on the transport or cell level will have a communication pattern that is generated as a result of many interacting stochastic processes with different time constants. In [HeHo95] and [WTSW97] this multilevel superposition of stochastic processes with different time constants is considered to be an explanation of the observed self-similar behaviour observed in aggregated traffic streams (packet or cell level) on the Internet [LTWW94].

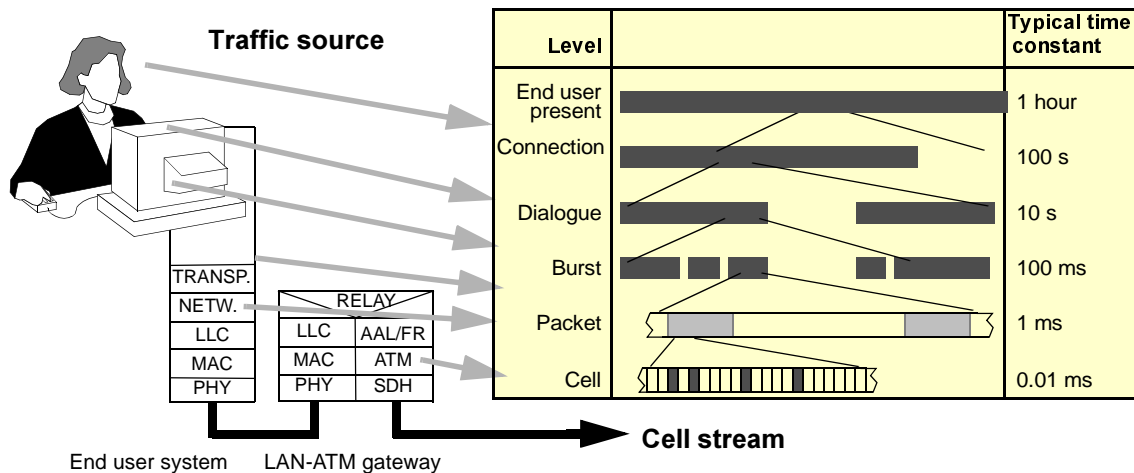


Figure 2 Illustration of different activity levels in a source.

The aggregated stream will be a heterogeneous mixture of traffic from various sources where each source will be influenced by both user behaviour, equipment and protocols, see Figure 2.

User behaviour:

- the end user behaviour - setup/disconnect a session, application mixture (web browsing, software downloads, chat, games, email, streaming (video, audio))
- user-network interaction (slow variation) - interest/impatience, takes a break and returns later due to congestion, cost, etc.
- variation in information stream - variable video coding (MPEG).

Equipment and protocols:

- end user equipment constraints - access capacity, processor capacity, disk, video coding processing
- communication system constraints - buffer space, transmission capacity, router capacity
- network mechanisms - routing strategies, priorities (diff-serv), weighted fair queuing, resource reservations (RSVP, intserv)
- protocols, e.g. TCP congestion control and avoidance.

2.2 Linking stochastic user behaviour models to real communication streams

GenSyn models the user behaviour in a state based source model, while the communication systems are not modelled. The equipment constraints and protocol behaviour are automatically included through the linking of the stochastic processes to the built-in protocol stack on the workstation. This means that no incorrect assumptions about the protocols or network mechanisms will be made, it is for instance

not necessary to know the details about the MPEG coding or the TCP slow start mechanisms.

In general it can be said that the modelling framework combines the better of two worlds, it gets the flexibility and scalability of state diagram description with composition of users combined with the accuracy of protocol and network behaviour by using the actual protocol instead of a model.

2.2.1 Division of state space

Figure 3 shows a principal sketch of the modelling framework to illustrate the linking of a composite state space description and the protocol stack. This efficient combination is accomplished by dividing the modelling state space into:

- *stochastic state space*, Ω_S , where the stochastic user behaviour is described by a state machine where each state can contain a collection (composition) of many users¹, and
- *communication state space*, Ω_C , which is a “placeholder” for the users that are waiting for response from the communication system.

All transitions from the stochastic to the communication state space, $\Omega_S \rightarrow \Omega_C$, will instantiate an *interface module* that creates a communication stream to and/or from the workstation. A communication stream is either a TCP connection or a UDP datagram stream. The user that instantiated the communication stream will be placed in a *communication state* and stay there until the communication is completed. Hence, a transition from the

1. The state model has to be (semi-)Markovian, i.e. each state sojourn time must be negative exponentially distributed, in order to make the composition of users in each state.

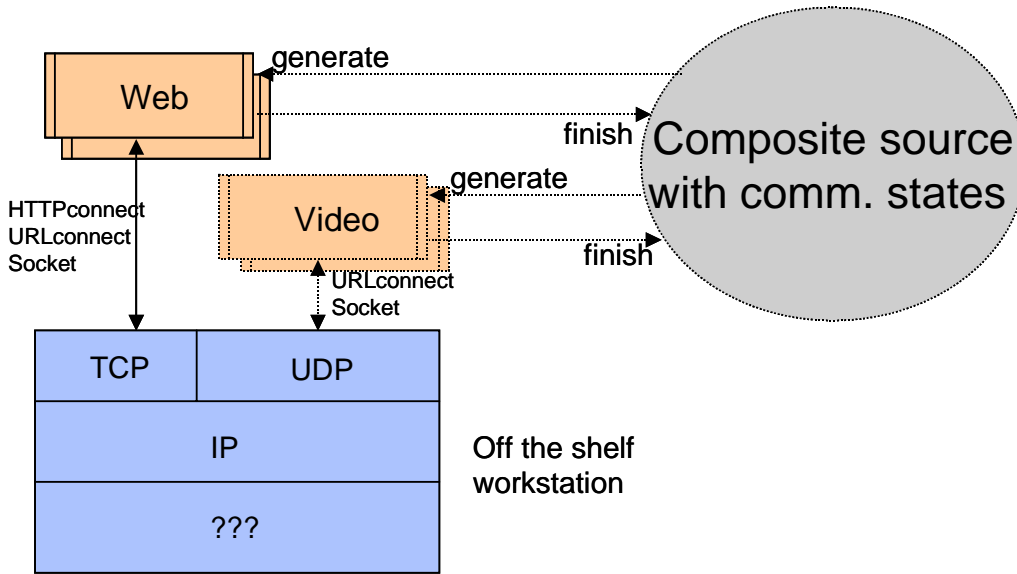


Figure 3 The interface modules are linking the stochastic behaviour to built-in protocol stack.

communication state space back to the stochastic state space, $\Omega_C \rightarrow \Omega_S$, is initiated on completion of communication and the corresponding pointer to the interface module is removed.

The number of states will not change during the evolution of the generator process, only the number of users in each state will change. All states need an attribute for the current number of users in each state, and the communication states need an additional attribute for the storage of information about the user (or process) identities that awaits a communication stream to complete. Hence, only a modest increase in the generator process is observed as the number of users increases. This solution is chosen to optimise the scalability against the accuracy in the protocol modelling.

2.2.2 Notation

The following notation will be used in this paper:

Ω - global state space, $\Omega = \Omega_S \cup \Omega_C$

Ω_S - stochastic state space

Ω_C - communication state space

m - state vector, $m = \{m_i\}_{i \in \Omega}$

m_i - number of users in state i

I_i - identity vector of users with an open communication stream in state $i \in \Omega_C$

$\theta_{i,j}$ - state transition rate from state i to state j , $i \in \Omega_S$

θ_i - total transition rate in state i , $i \in \Omega_S$, $\theta_i = \sum_{j \in \Omega} \theta_{i,j}$

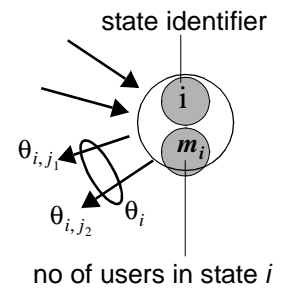
$E(T_i)$ - expected sojourn time $E(T_i) = 1/(\theta_i m_i)$, $i \in \Omega_S$

In Figure 4, the notation is shown in an example demonstrating the division of state space and the links to the interface modules.

2.2.3 Stochastic state model - the behaviour of a single source

The user behaviour is described by a finite state continuous time Markov process. A general state has the following attributes:

- a state identifier, i ,
- number of users m_i in state i ,
- a state sojourn time distribution (negative exponential distribution),
- list of transition rates, $\theta_{i,j}$, and probabilities, $p_{i,j} = \theta_{i,j}/\theta_i$
- list of neighbour states, i.e. states that can be reached in one transition from state i .



A state model is semi-Markovian if all states have state sojourn times that are negative exponential distributed, or if neither of the states in the model have more than one non-exponential sojourn time. In case of a state with non-exponential time distribution, an approximation by a phase-type distribution is feasible by substituting this state with a combination of states that have negative exponential time distributions. This means it is possible to model state

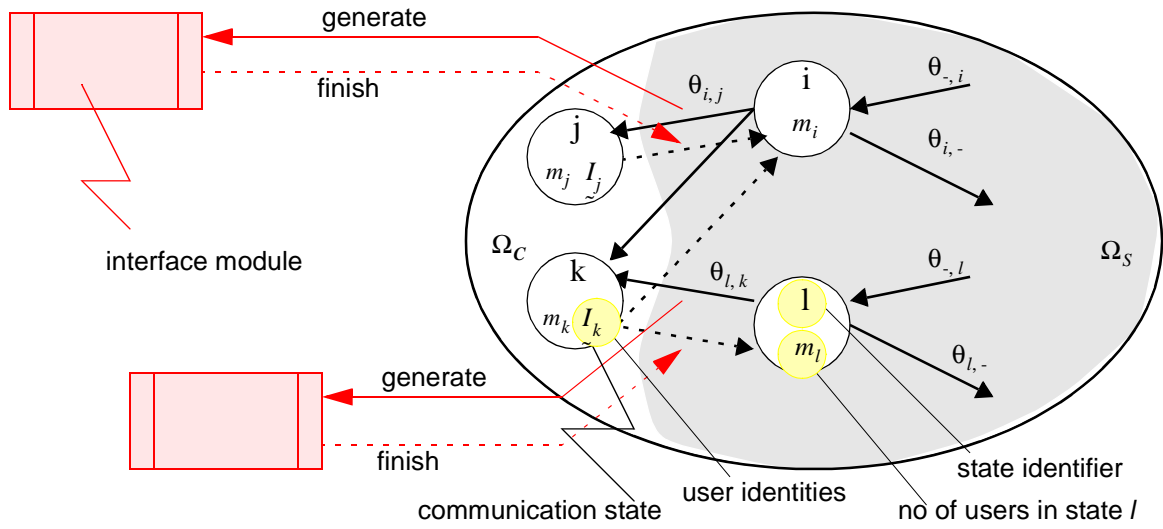


Figure 4 The modelling framework: division of state space

sojourn times that follows a hyper-exponential, hypo-exponential, or Coxian distribution. All these distributions is a combination of states with negative exponentially distributed state sojourn times.

When all state sojourn times are exponential, the procedure described in Figure 5 can be applied to determine the next stochastic event. Observe, however, if a communication stream is completed while waiting for the next scheduled stochastic event to occur, a state change caused by this completion, see Figure 6.

1. Sample the time T to next event in Ω_S , the expected value is $E(T) = 1 / (\sum_{j \in \Omega_S} \theta_j m_j)$
2. Wait T
3. Sample which state $i \in \Omega_S$ where the next event took place, the probability is $\theta_i m_i / \sum_{j \in \Omega_S} \theta_j m_j$
4. Sample the next state j from state i , the probability is $p_{i,j} = \theta_{i,j} / \theta_i$
5. Move a user from state i to state j by updating the $m_i + 1$ and $m_j - 1$.

Figure 5 Update the state vector when the next event is a stochastic event in Ω_S .

2.2.4 Communication State model - link to the network

The communication states are the “placeholders” for all users that currently have an open communication stream. The states sojourn times, T_i , $i \in \Omega_C$, for all users in the

communication states are fully determined by the behaviour of the underlying communication system, i.e. the performance of the end user equipment, protocols, network mechanisms. A user will be “locked” in the communication state as long as the communication stream is open, and immediately be removed when the stream is closed. Hence, for user x the transition from state i in Ω_C to state j in Ω_S is considered to be a *conditional transition*, i.e.

$$\theta_{i,j}(x) = \begin{cases} \infty & \text{comm. stream opened by } x \text{ is closed} \\ 0 & \text{comm. stream opened by } x \text{ is open} \end{cases} \quad (1)$$

where state $i \in \Omega_C$ and state $j \in \Omega_S$.

In the communication states a relation, e.g. a process identity, to all opened communication streams must exist. When a communication stream is opened and a new user enters state i , $i \in \Omega_C$, the process identity of the interface module related to this state transition need to be stored. For this purpose an identity vector I_i is added as a new attribute to the communication states in addition to the list in Section 2.2.3. When a communication stream is closed, e.g. when a file is downloaded, an instantaneous state transition will occur, and a user leaves this communication state ($m_i - 1$). Observe that the number of user in state i equals the number of elements in the identity vector in state i , $m_i = |I_i|$. The identity vector, I_i , is implemented as a list of pointers (process identities) to open communication streams.

In Figure 6 the addition to Figure 5 is given to handle both stochastic events and events triggered upon completion of a communication stream.

1. Sample the time T to next event in Ω_C as described in step 1 of Figure 5.
2. If $(\exists k \in \Omega_C) \wedge (T_k < T)$, i.e. a communication stream is closed before the time T elapses, then replace step 3 in Figure 5 with the following:
3. The next event takes place in state k in Ω_C where k is where the first stream closed i.e. $k = \{i | i \in \Omega_C \wedge (T_i = \min_{j \in \Omega_C}(T_j))\}$

Step 4 and 5 is the same as in Figure 5 except that the $p_{i,j}$ are given as parameters because they can not be derived from the conditional rate $\theta_{i,j}(x)$ which is now either ∞ or 0.

Figure 6 Update the state vector when the next event is an event in Ω_C .

The state sojourn time in a communication state may depend on the current congestion situation in the underlying network. In the case of downloading web pages, the congestion, the size and location of the requested page will contribute to the sojourn time. Furthermore, for web- and ftp-downloads an *impatience factor* is defined for each communication state. This enables the definition of an upper limit on the time spent in the communication state. The impatience factor is formulated as an condition of $\theta_{i,j}(x)$ in (1), and its randomness is defined in the stochastic part of the source model.

3. Model template examples

In this section, two examples are given to demonstrate the applicability and the flexibility of the modelling framework. In Section 3.1, a web client model is described that opens and closes TCP connections to real web servers. The second model is a video server described in Section 3.2. The video server is sending UDP datagrams according to MPEG coded video streams. In both models, measurements from the literature are applied to determine the parameters of the model.

3.1 Web client

Web traffic is the dominant elastic traffic in the Internet today. Web traffic can be modelled by web client that downloads web pages via HTTP connections from real web servers anywhere on the Internet. The behaviour of the user

client is described by state diagrams, while the interface module imitates a simplified web browser.

The model in this section is only an example of how web traffic can be modelled. The framework of GenSyn is not limited to this model, but can easily describe other state oriented models, change the model parameters, or change the parsing of a web page.

User behaviour in a web session

The overall model of the web client describes the user behaviour in a *web session* by only 3 states, see Figure 7 below. A web-session, as defined in [Vic98], is a sequence of packets with less than 30 minutes between two consecutive web page requests. The following states are defined:

- *Idle* - the user is in between web sessions.
- *Read* - the user reads the downloaded web page and considers what to do next, download another one, or close the web session?
- *Download* - the user opens a connection to a url-address, randomly selected from a list of addresses, and downloads this page, parse through it, and download all corresponding image files and applets.

The *Idle* and *Read* states are stochastic states. The state sojourn times of these two states are sampled from a probability density distribution. Each user that starts to download a web page enters the *Download* communication state. A pointer to the interface module that handles the communication stream is added to the identity vector of *Download*, I_{download} . When the web page, and all its content (text, images, applets), is downloaded, the interface module closes the connection and remove itself from I_{download} and the user returns to the *Read* state. The modelling framework enables the (random) setting of an upper limit of the download time, the impatience factor. This factor allows the connection to be closed before the entire web page is downloaded.

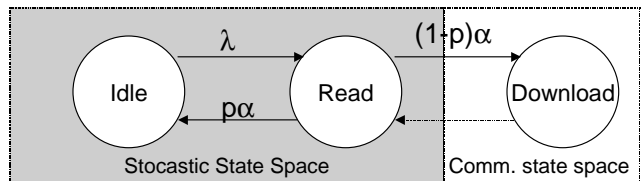


Figure 7 The overall model of an Web source

The parameters of the state model are extracted from the work described in [Vic98] where an aggregated stream of HTTP connections was separated and broken into web sessions.

- *Time between web-sessions* - The Idle state uses the web session separator criterion as the expected sojourn time $T_{\text{Idle}} \sim \text{neg.exp}(\lambda)$; $1/\lambda = 30 \cdot 60 = 1800$ [sec.].

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
λ	1/1800			
α	1/10	1/60	1/300	1/1000
a	0.6448	0.3133	0.02509	0.01681
p	0.02392	{using $p=1/(40.8+1)$ }		

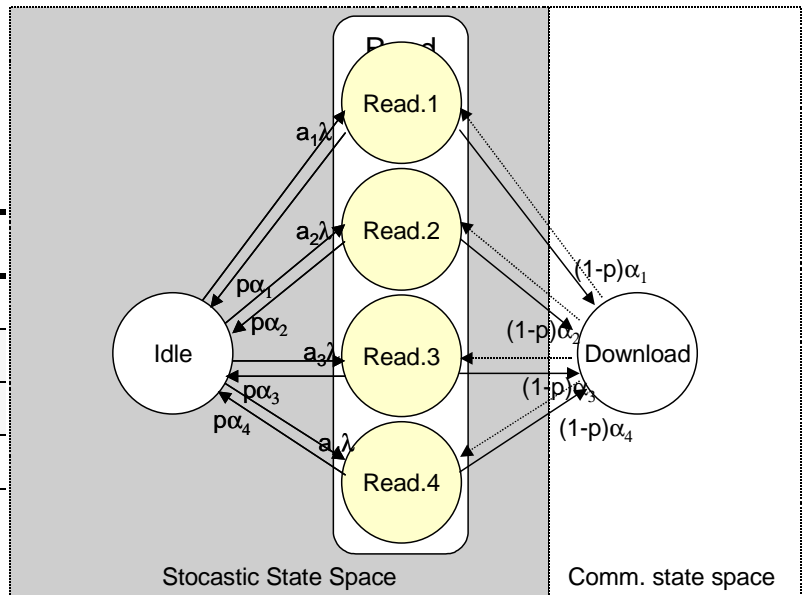


Figure 8 The complete model of an Web source with table of parameters.

- *Time between requests* - Within a web session the mean time between requests is $\bar{X}=42.8$ [sec.] with a coefficient of variation equal to $S/\bar{X}=2.9$. This means that the *Read* state in the overall model is not negative exponential distributed (in that case the $S/\bar{X} = 1$). This state is therefore substituted by a hyper-exponential distribution with 4 branches, each with different time constants. The parameters are determined to fit the truncated-Pareto model used in [Vic98].

Substituting the Read state with 4 sub-states to model the hyper exponential distribution, the complete model and the model parameters are given in Figure 8.

Interface module - a web browser

The interface module describes a Web-browser that downloads web pages from real web servers. The browser downloads the source file of the web page, parses this, and downloads the images and java applets identified on this page. These are also downloaded, in parallel with each other and in parallel with the download of the web source file.

The URL-addresses are randomly chosen from a list of 2500 addresses from all over the world. This list can easily be changed. For example, as an option, GenSyn offers to dynamically update the URL-list as the generator is running and new pages are visited. This is done by inclusion of some, or all, of the HREF addresses found when parsing through the source file of a web page.

3.2 Video streaming

In many QoS test scenarios it will be interesting to measure the sensitivity of changing the relative amount (e.g. in bytes) of TCP versus UDP traffic. Although some of the applications running on top of UDP will be adaptive, the UDP itself is not adaptive to network congestions. This is foreseen to be a problem for TCP connection when the relative amount of UDP traffic increases in a highly loaded network. Hence, to test these influences and interactions, some models of sources generating UDP traffic are required.

The UDP example included in this section is a model of a video server. The server sends a stream of UDP packets to a specific IP address, representing the fictitious client that requested the video. The server behaviour is described by state diagrams, while the interface module establishes a communication stream that sends UDP datagrams according to a trace file with MPEG coded video frames.

User behaviour of a video server

The user behaviour in this case is very simple, described by two states. The clients that are requesting video streams are not explicitly modelled, they are included in the server model as requests.

- *Idle* - the server has not received any video request. This is the only stochastic state in the model, and the time between requests is negative exponential distributed

with an expected value of 8 hours, i.e. each user client requests a video stream every 8 hour.

- *Send video* - the server sends a stream of video to an IP address randomly chosen from a list of IP addresses. The average duration of a video stream is 30 minutes.

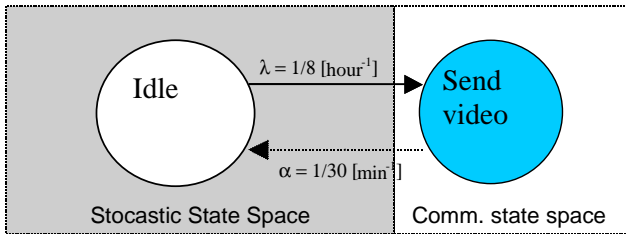


Figure 9 The model of a video server.

Recall that in each state there will be many users, one for each client requesting and viewing a video.

Interface module - sending variable number of UDP packets

The video streams are the MPEG-1 coded video sequences made available by Oliver Rose [Rose95]. Each video trace contains n video frames $X = \{X_1, X_2, \dots, X_n\}$ ($n = 40000$). A frame consists of a variable number of bits using MPEG coding and the most common *Group of Picture* pattern *IPBPBPBPBPBP*, see Figure 10.

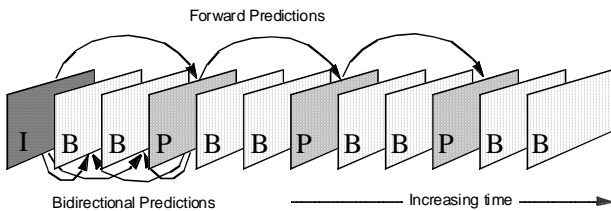


Figure 10 MPEG coded video: An example of a Group of Picture pattern.

The X_i bits in video frame i are converted to Y_i UDP packets of 1024 bytes, i.e. $Y_i = \lceil X_i / (1024 \cdot 8) \rceil$. A new video frame is sent every 40 ms and the size of the video frame is given by the current position in the MPEG trace. All Y_i UDP packets in position i are sent back-to-back at maximum line speed, see Figure 11. A video is randomly, and uniformly, selected among the $K = 19$ different video streams that are available.

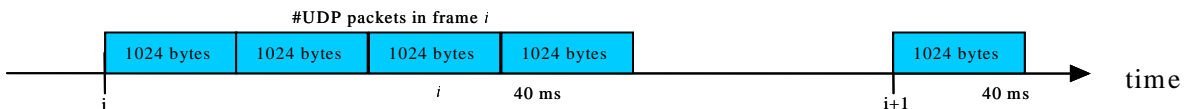


Figure 11 MPEG coded video: The streaming of UPD datagrams.

Although this model uses a set of MPEG-1 coded video sequences it is very simple to use MPEG-2 instead as long as the resulting input trace files have the same format as X . It is also easy to change the size of UDP packets, and the time between each video frame. This is very convenient when investigating the sensitivity and importance of e.g. video coding techniques and frame segmentation.

As an example of a trace generated by GenSyn, Figure 12 shows a section of a trace running the video model. This is a screen capture from the visualization monitor that is implemented in the generator. The trace shows kbit/s for the last measurement period (here with a length of 1 second) and the average kbit/s up to now (the smooth curve).

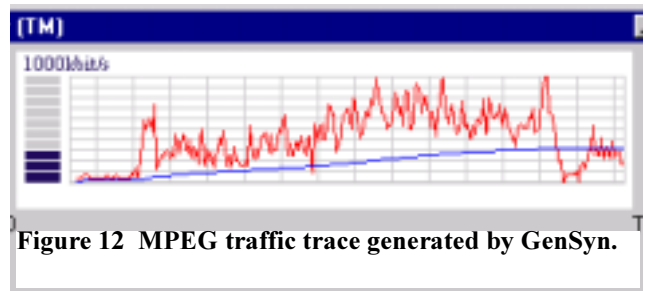


Figure 12 MPEG traffic trace generated by GenSyn.

4. Implementation issues

The design of GenSyn had the following overall requirements [HeLu99]:

- *portable* - run on Windows, Linux, Unix, and produce the same results
- *distributed* - run in parallel on several workstations - and be easy to distribute
- *scalable* - run many active users in parallel on a single workstation.

The two first requirements made Java an attractive choice as the implementation language. Java provides simple, high level, and well defined interfaces and methods for communication (via APIs) with the underlying protocol stack. This makes it fairly easy to create HTTP and TCP connections and to send streams of UDP packets.

The flip side of the coin is that Java limits the scalability of GenSyn due to two constraints:

- *Time scheduling and granularity*. The sleep function in Java is inaccurate for time granularity in milliseconds, and returns different results on various platforms. This will limit the number of users that can be defined in the model because many users means short time between

events in a composite model. The experience so far, running between 300 and 3000 web users on a single processor machine, indicates that the generator running on one workstation should limit the number of users to keep the expected time between stochastic events of at least 10 ms.

- *Threads and memory.* The number of parallel threads that can be run on one single workstation is limited by the available memory. Hence, the GenSyn should be executed on a dedicated workstation. This was expected to be the critical restriction because it introduces an upper limit to the number of simultaneous communication streams, and hence the number of users in the model.

5. Closing comments

The GenSyn is a Java process that generates IP traffic using a flexible, scalable, stochastic, modelling framework for describing the user behaviour of Internet sources. This stochastic behaviour model is linked to the underlying protocol stack and generates real packets into the network. This is a novel modelling approach that chooses the best of two worlds; flexibility and scalability of composite state models, and accuracy in the protocol behaviour by use of the underlying protocol stack instead of making a model of it.

5.1 Main experience from deployment of GenSyn

The main experience is that it is fairly easy to implement the modelling framework in Java. The modelling framework itself is flexible and prepared for modelling of many different Internet applications. Several models are now available described in the GenSyn framework, including web, ftp, VoIP, MPEG video, and leased line (each user generating a deterministic packet flow, both with respect to packet size and time between packets). The models developed can easily be changed with respect to many factors, including the number of users, state sojourn times, size of packets, time between packets, IP destination and source addresses, file and web page locations.

The scalability of the generator is limited by the time granularity and time scheduler of the Java Runtime Environment (JRE). The portability of the generator is limited by both the inconsistencies between different versions of Java APIs, and the differences in the time granularity and runtime schedulers of JRE running on to of different platforms and OS (NT, Linux, Unix). Java is not for real time applications.

In the current implementation some measurement functionality exists. This includes

- source and destination ports are added to the UDP packets generated to enable filtering packets by tcpdump.

- tracefile with records of the size of a web page or ftp file, or the length of a phone connection or a video stream
- summary report on the total amount of submitted and received data (in bytes and packets), and the number of unsuccessful attempts

In current version of GenSyn, no end-to-end measurements of real-time performance like delay, jitter (delay variation), loss, are done by GenSyn. These measurements are carried out by the use of trace software like tcpdump. GenSyn has also been used in combination with other embedded load generation and measurement equipment like SmartBits [SBit] where GenSyn provides the background load of controllable and realistic elastic load (tcp connections).

5.2 Ongoing and planned work

Currently, a lot of work is being done on GenSyn and more is planned in the near future. The key issues are:

Extend the model template library - in the current version of the generator there are interface modules to support the download of web pages and files through http, video streaming, and VoIP. This library is likely to be extended with a few more models, e.g. a model for generating errors by sending network management messages alerting the OSS.

Validation and verification - The correctness of GenSyn relative to the models defined is verified, see [HeLu99]. The traffic stream from the models defined in the GenSyn framework is yet not fully verified, i.e. the generated stream of packets are not compared with other measurements.

Network measurements - GenSyn is now being deployed in a fully equipped IP platform with DiffServ and MPLS functionality and several different applications. The measurements from these experiments will demonstrate the applicability with respect to generate realistic traffic, and serve as a verification of the GenSyn process. The objective of these experiments is to study the sensitivity in end-to-end performance of changing e.g. the tcp/udp mixture, increasing the total load, changing the traffic characteristics (burstiness), and changing the underlying mechanisms providing support for QoS differentiation. Test scenarios are defined in a testnet with only best-effort support. For comparison, the same scenarios are used in a testnet that is supporting QoS classification and differentiation.

Extend the built-in measurement functionality in the generator. Possible inclusion of end-to-end real-time measurements will be investigated.

Redesign of the GenSyn will be considered to improve the real time properties and to extend the measurement functionality. Furthermore, a scenario editor will be considered for controlling experiments and distributing the GenSyn processes on many machines and processors.

Improve the time granularity of the generator to allow more users and to avoid the jitter introduced to the real time sources.

Acknowledgements

I would like to thank Manyi Lu, SINTEF Telecom and Informatics (now Clustra AS), and Per Christian Bjelke, Telenor R&D for assistance on program design and for excellent Java programming to make GenSyn come alive. I would also like to thank my former employer SINTEF Telecom and Informatics for supporting phase one of the project.

References

- [Ake99] Å. Arvidsson. *On traffic models for TCP/IP*. 16th International Teletraffic congress, ITC'16, Edinburgh, UK, June 7-11 1999.
- [BaCr98] P. Badford and M. Crovella. *Generating representative Web workloads for network and server performance valuation*. In Proceedings of ACM SIGMETRICS'98.
- [ChLi99] H.-K. Choi, J.O. Limb. *A Behavioural Model of Web Traffic*. 7th International Conference on Network Protocols (ICNP'99), October 1999, Toronto, Canada. extended version: <http://users.ece.gatech.edu/~hkchoi/paper/model.pdf>
- [HeHo95] B.E. Helvik, L. Hofseth. *Self-similar traffic and multilevel source models*. In Ilkka Norros and Jorma Virtamo, editors, The 12th Nordic Teletraffic Seminar (NTS-12), pages 407-420, Espoo, Finland, 22 - 24 August 1995. VTT Information Technology.
- [HeLu99] P. E. Heegaard and M. Lu. GenSyn - Java based generator of synthetic Internet traffic. SINTEF Technical report STF40 A99078.
- [Helv95] B. E. Helvik. *Synthetic Load Generation for ATM Traffic Measurements*. Teletronikk, vol. 91, no. 2/3, pp. 174 - 194, October 1995.
- [HMM93] B. E. Helvik, Ole Melteig and Leif Morland: *The synthesized traffic generator; objectives, design and capabilities*. In Integrated Broadband Communication Networks and Services (IBCN&S). IFIP, Elsevier, Copenhagen, Denmark, April 20-23 1993. (STF40 A993077)
- [LTWW94] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. *On the self-similar nature of Ethernet traffic* (extended version) IEEE/ACM Transaction of Networking, pp 1-15, 1994.
- [MTW99] G.J. Miller, K. Thompson, and Rick Wilder. Performance Measurements on the vBNS. In proceedings from Interop '98, Las Vegas, NV, May 1998.
- [Rose95] O. Rose. *Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems*. Technical Report 101, University of Wurzburg, Institute of Computer Science, February 1995. The traces obtained from: <ftp-info.informatik.uni-wuerzburg.de/pub/MPEG>.
- [SBit] SmartBits info: <http://www.netcomsystems.com/>
- [Vic98] N. Vicari. *Models of WWW-Traffic: a Comparison of Pareto and Logarithmic Histogram Models*. University of Wurzburg. Institute of Computer Science. Research Report Series. Report No.: 198 March 1998
- [WTSW97] W. Willinger, M.S. Taqqu, R. Sherman, 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, Feb. 1 1997, pp 71-86.