# UniLoG: A Unified Load Generation Tool

Andrey Kolesnikov

Department of Computer Science, University of Hamburg,
Vogt-Kölln-Str. 30, 22527 Hamburg
kolesnikov@informatik.uni-hamburg.de

## 1 Motivation

Utilities for generating artificial (synthetic) loads are very important for analyses of performance and behavior of networks and their offered services. Load generators implemented by the industry[1] are mainly dedicated hardware components with very high performance and stringent precision requirements. In research and academia, mainly software based load generators are commonly used because of the expected higher flexibility in operation and maintenance (e.g. due to easy deployment of constituent load generating modules in the network, code customizations for a specific research purpose, etc.) while components of real operating systems and protocol stacks can be used to guarantee realistic load generation at lower costs. However, many existing tools are dedicated to a specific modeling study (e.g., `Guernica` [1] along with its specific Dweb model for Web traffic, or `Harpoon` [2] modeling IP traffic flows) or are focusing on generating traffic at some specific interface in a network (e.g., `ITG` [3] or `Brute` [4] were designed to generate traffic on UDP and TCP service interfaces). The proposed solutions quite often do not provide an adequate flexibility, e.g. in case the underlying model is to be modified or a completely new model is to be used. Therefore, the unified load generator `UniLoG` is presented in this paper, which combines the specification and generation of network loads in one single coherent approach. The basic principle underlying the design and elaboration of `UniLoG` is to start with a formal description of an abstract load model by means of a finite user behavior automaton (UBA, introduced in Sec. 2) and thereafter to use interface-specific adapters to map the abstract requests to the concrete requests as they are "understood" by the service providing component at the real interface in question. An overview of the distributed `UniLoG` architecture is given in Sec. 3 and a concrete example of its practical use in QoS studies for video streaming is demonstrated in Sec. 4.

## 2 Load Specification Method

In our terms, load $L = L(E, S, IF, T)$ is defined as a *sequence of requests* offered by an environment $E$ (which consists of virtual, i.e. modeled, *service users*) to a service system $S$ (e.g., TCP service) at a well-defined interface $IF$

[1] e.g., Agilent N2X, IXIA Test Applications, LANforge FIRE.

J.B. Schmitt (Ed.): MMB & DFT 2012, LNCS 7201, pp. 253–257, 2012.

(e.g., TCP service interface) during the time interval $T$. Each abstract request in this sequence is a tuple $(t_i, r_i)$, whereby $t_i \in T$ denotes the arrival time of the abstract request $r_i$ at $IF$ and, $t_i \in \mathbb{R}$, $i = 1, 2, \ldots, n$, $n \in \mathbb{N}$, $t_i \leq t_j$ for $i < j$. The user of `UniLoG` (which may be, e.g., a single researcher, experimenter, test engineer or a whole quality assurance team) can choose the interface $IF$ for load modeling dependent on the objectives of the specific study to be carried out. For example, the TCP service interface is chosen in the use case presented shortly in Sec. 4 in order to analyze a set of QoS metrics for RTSP video streaming over TCP under different TCP background loads. Further, the behaviour of one or many service users at $IF$ is described by means of the corresponding UBA using the following three kinds of different user states to specify the possible sequences of abstract requests as induced by virtual service users: 1) *Request-* or *R-states* model the generation of requests of exactly one abstract request type (e.g., `TCP_connect` or `TCP_send`), 2) *System-* or *S-states* model the waiting for certain kinds of system events (e.g., receipt of a TCP socket error or blocking status message), and 3) *Delay-* or *D-states* are used to model the delays between subsequent requests or system events (e.g., *interarrival times* between `TCP_send` requests). The values of request attributes (e.g., `dest_addr` for modeling the IP address(es) of TCP receiver(s) or `data_len` for the payload length of TCP requests) in R-states and delays in D-states can be specified by means of different statistical distributions, traces (e.g., PCAP files preprocessed by `tshark` using its different dissectors and filters to extract the request attributes needed) or as a constant in very simple cases. The transitions between user states can be specified by means of transition probabilities (yet assuming a uniform distribution) or by means of "guards" implemented recently (i.e. conditions on variables from the global memory of the UBA which can evaluate either to `true` or to `false`). Finally, the execution of the parameterized UBA (PUBA) model results in a specific trajectory of the underlying stochastic process and a concrete sequence of abstract requests is generated which is conform to the specified load $L$.

## 3 Overview of the UniLoG Architecture

To provide a high degree of flexibility in generating traffic mixes of different structure and intensity for various scenarios, a distributed architecture has been designed for `UniLoG` (cf. Fig. 1). To create a load required to measure, e.g. the throughput of a link or to test a web application running on a powerful server the resources of a single workstation can be insufficient. Likewise, in order to analyse the performance of different routing mechanisms or transport protocols, a kind of geographical distribution of traffic sources (and sinks) may be required. `UniLoG` resolves this problem by using several load generating nodes in each of which a special service called `UniLoG` load generation agent (or simple load agent) is installed. Load agents are responsible for the generation of traffic loads from virtual users and can be controlled by the experimenter from one central point in the network (management station, cf. Fig. 1) by means of different commands (e.g. `loadPUBAModel()`, `start()`, `startAt()`, `stop()`, etc.) encapsulated into HTTP messages for transport (a simplistic HTTP server with SSL/TLS support is included in each load

agent for this reason). Further, each load agent contains a generator component (generator of abstract requests, GAR), which is responsible for the execution of the PUBA model and generation of initially abstract load requests according to the specifications therein. The actual generation and injection of real requests at the chosen service interface (e.g. TCP service interface) is accomplished by means of the correspondent specific adapter (e.g. `UniLoG.TCP` adapter). Among others, values of parameters needed to generate real TCP requests (e.g., source and destination port number(s)) can be supplied by the adapter in case they were omitted by the experimenter in the PUBA. Adapters for UDP, TCP and IP service interfaces are already available and a new adapter for generating realistic Web traffic loads at HTTP service interface (using a real, remotely controlled Web browser) has been developed recently.

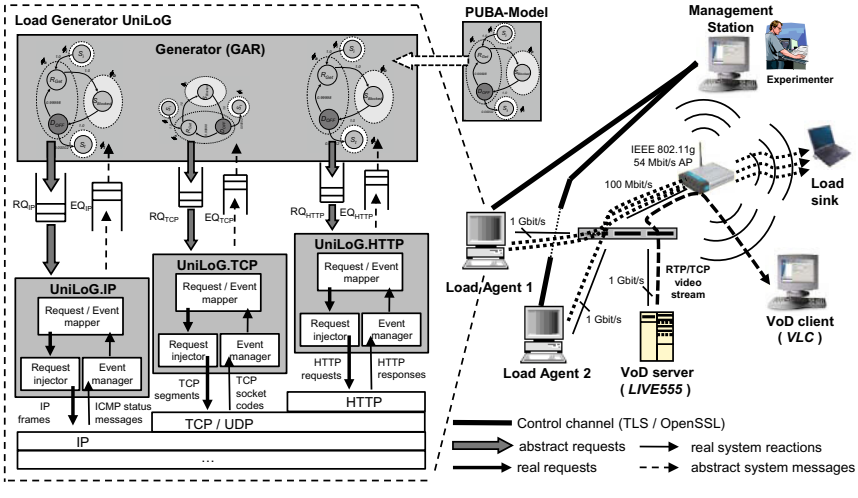


**Fig. 1.** Distributed architecture of the `UniLoG` tool

The performance of the adapters is characterized mainly by the maximum packet and data rate of the generated streams (i.e. UDP, TCP or IP streams) achievable by the correspondent adapter and the (mean) difference between the specified and actual packet injection times. For example in a 1 Gbit/s Ethernet LAN, `UniLoG.IP` adapter can generate IP streams with a data rate up to 725 Mbit/s while spending ca. 18 $\mu s$ on each IP request for UBA execution and request preparation overhead. Compared to earlier work [5], significant progress is achieved in the functionality of `UniLoG` by integration of passive measurement tools like `tshark` in order to obtain required parameters for the UBA models or to estimate characteristics of traffic loads observed and/or generated during the experiments. Moreover, the performance of each single load agent is increased using prototypical implementation of TCP and UDP adapters on the basis of a real-time operating system (RTOS-32).

## 4 QoS Measurements for RTSP Streaming under Load

Along with many other scenarios, e.g. those mentioned in Sec. 3, `UniLoG` can be effectively used in QoS studies for video streaming applications as shown in Fig. 1. Here, an H.264 coded HD video of ca. 10 min length is transmitted (with mean required IP throughput of 5.7 Mbit/s and peaks up to 18.7 Mbit/s) from the VoD server located in the LAN segment to the VoD client in WLAN using real-time streaming protocol (RTSP) for the streaming session and RTP over TCP for the real-time transport of video frames. In each streaming experiment, background load in WLAN is generated by `UniLoG` which consists of a fixed number of constituent TCP flows (from load agents to load sink) with the same required throughput[2] which is not increasing or decreasing during the experiment. In a series of experiments, the number of TCP flows is kept constant but their required throughput is increased from 0.5 to 16 Mbit/s.

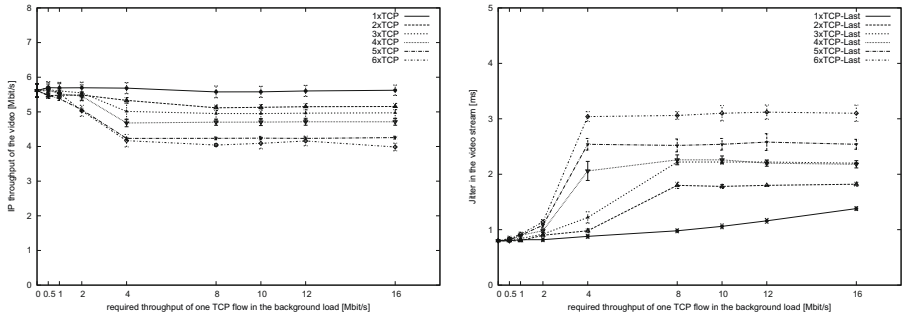


**Fig. 2.** IP throughput (left) and Jitter (right) of the RTSP/RTP video stream using TCP under different TCP background loads

Values of important quantitative QoS parameters like IP throughput, RTP jitter, packet loss, sequence errors and round-trip-time are obtained from the PCAP traces captured at the VoD client and server using the `tshark` tool combined with own Python scripts (cf. Fig. 2). The comparison of streaming characteristics obtained for the case of the stream delivery over the reliable TCP transport service with the results presented in [5] for the unreliable RTP/UDP transport may provide a very suitable guidance (e.g., for network operators) to configure reliable infrastructures for high-quality video streaming services.

## References

1. Pena-Ortiz, R., et al.: Dweb model: Representing Web 2.0 dynamism. Computer Communications 32(6), 1118–1128 (2009)

[2] Using TCP segments of constant length (1460 Byte), and constant interarrival times chosen to achieve required throughput of 0.5, 1, 2, . . . , 16 Mbit/s for one TCP flow.

2. Sommers, J., Barford, P.: Self-Configuring Network Traffic Generation. In: Proc. of IMC 2004, Taormina, Sicily, pp. 68–80 (2004)
3. Avallone, S., Pescape, A., Ventre, G.: Analysis and experimentation of Internet Traffic Generator. In: Proc. of New2an 2004, St. Petersburg, Russia, pp. 70–75 (2004)
4. Bonelli, N., et al.: BRUTE: A High Performance and Extensible Traffic Generator. In: Proc. of SPECTS 2005, Philadelphia, pp. 839–845 (2005)
5. Kolesnikov, A., Kulas, M.: Load Modeling and Generation for IP-Based Networks: A Unified Approach and Tool Support. In: Müller-Clostermann, B., Echtle, K., Rathgeb, E.P. (eds.) MMB&DFT 2010. LNCS, vol. 5987, pp. 91–106. Springer, Heidelberg (2010)