2. Foundations and Research Field

The concept of *workload* is one of the most fundamental aspects in performance evaluation of modern computer and communication systems. While a precise definition of a system's workload is elusive, a commonly accepted definition considers the amount of requests offered to a system by its users during some specific period of time [Fer72, Fer84]. For instance, if the System Under Test (SUT) is a stand-alone Web server, then its workload consists of all queries submitted to the server during an observation interval. A similar definition of a workload can be found in [MAD04]. A system's workload is defined as a set of all inputs that the system receives from its environment. In the context of workload for computer networks, the system is usually represented by the communication system serving the requests with their corresponding resource demands. It becomes apparent, that the workload induced in the communication system is mainly characterized by the requests offered to it. So, for the purpose of workload modelling, the requests, which may be created by the environment and handed over to the system, along with their resource demands have to be described precisely.

A solid understanding of how users typically behave when interacting with an application and the workload patterns derived from such behaviour helps the researcher to identify not only users' needs but also application features that are more useful and attractive as well as possible system vulnerabilities. Moreover, the performance optimization, tuning, and management of a system with many clients and complex server and network infrastructures, which is typical of many popular Internet applications like, e.g., YouTube or online social networks, depend heavily on an accurate knowledge of the workload typically experienced by such a system.

Therefore, the *measurement*, *characterization*, *modelling*, and *generation* of real workloads are the key steps driving the design of new cost-effective network applications and services as well as the optimization of existing ones [AlA11].

A. Kolesnikov, *Load Modelling and Generation in IP-based Networks*, DOI 10.1007/978-3-658-19102-3_2

Workload characterization: consists of identifying the basic components that compose the target workload, which depend both on the nature of the target application and on the purpose of the characterization.

Workload modelling: consists of building a representation that mimics the real workload under study, based on the identified components.

Workload measurement: is a key step to all tasks in performance engineering and relates to gathering representative datasets to support the characterization task, helping the researcher to obtain workload parameters and establish a link between the real workload and its model.

Workload generation: consists of the injection of requests, flows, or packets into a network in a controlled manner. At this point, synthetic (or artificial) workload and traffic generators can be used to create synthetic loads and inject synthetic traffic at different interfaces in real or simulated networks according to a workload or traffic model specified by the experimenter.

Understanding of the important characteristics of offered workloads can help to improve the design and construction of efficient network systems and mechanisms. Realistic workload models can further support cost-effective capacity planning, network and service dimensioning and management decisions, and support performance analysis and optimization studies. Furthermore, workload and traffic models are very often indispensable for generating synthetic workloads and traffic for experimental purposes during load testing in the networking research community and industry. We discuss this issue in the next section in more detail.

2.1. Workload Modelling and Specification Techniques

A large number of workload and traffic models for different sources of load and traffic has been proposed in the networking research community. The studies may differ, among others, in:

The type of workload or traffic sources: the characterization, modelling and simulation of network workloads can be related to different (types of) network applications or services, e.g.:

- Web workloads have been studied, e.g., in [BMS11, BMS14] in terms of different content and service complexity metrics. Characteristics of the

resulting Web traffic have been investigated, e.g., in [IhP11] in order to improve the service response time and to evaluate the effectiveness of caching and intermediary systems. The author in [Cha10] used active measurements to obtain a set of different characteristics of Web workloads and traffic in order to assess the efficiency of client side caching for modern Web sites. The study [SAMFU12] discovered potential pitfalls in the analyses and modelling of Web/HTTP traffic, such as the non-consideration of persistent connections or pipelined requests, and the mismatches between the values reported in the request headers from the actual content type and data volume being transmitted. The authors in [CaM10] analysed the Web traffic intensity and its temporal variability using the Web server logs.

- Voice traffic from Voice over Internet Protocol (VoIP) applications using different types of voice codecs has been analysed in a series of studies [MSS05, PEA05, HGB06, MBM09, HHCW10]. The large-scale study in [BMPR10] presents results of VoIP traffic measurements and analyses at a backbone link of a commercial ISP in Italy.
- Video traffic from live video streaming applications has been studied, e.g., in [BMW05] using the UDP protocol for the delivery of video packets, or in [BBM10] for the delivery of real-time video streams using the TCP protocol. A large-scale study of video streaming applications in operational networks presented, e.g., in [EGRSS11] may help to understand such important video streaming characteristics as the use of the adaptive bit rate streaming protocols, achieved streaming rates, and details of the user behaviour (e.g., the content popularity or number of cancelled video sessions). The results of the study may be used, e.g., in order to identify potentials for object caching during the delivery of the video content. Further, a number of models for MPEG-like encoded VBR video traffic sources considering different types of frames in the video stream have been proposed [Ros95, SRS03].
- The live (multicast) TV component of the Internet Protocol Television (IPTV) service has been studied, e.g., in [CRCM08, QGL09, GJR11] in respect to the user access patterns, channel popularity or switching dynamics of the users in such a system. Corresponding analyses of the Video on Demand (VoD) service component (where the user access patterns have a direct impact on the performance of the VoD servers) have been presented in [GJCG13]. An extensive empirical analysis of access patterns and user behaviour in a large centralized VoD system at China Telecom has been conducted in an earlier work [YZZZ06]. Further,

results of statistical analysis and modelling of VoD and VoIP workload characteristics in a nationwide commercial IP network in Korea have been presented in [CSK11].

- Analyses of SMTP and POP3 email traffic have been conducted, e.g., in [OhC05, AcP12]. Further, models to study the evolution of email networks in 3G mobile network scenarios have been presented in [SKR07].

The purpose of the study: one of the major objectives of workload characterisation and modelling studies is the identification and analysis of basic components that compose the target workload. Understanding of the key workload features can significantly contribute to the design and development of efficient network applications, services, and systems. For example, analyses of the video content popularity [YZZZ06, CKR09] can help to improve the corresponding techniques for the popularity prediction and to support the design of more effective caching and content delivery strategies [HLR07, QGL09, EGT11].

Realistic workload and traffic models can be used in performance analysis and optimization studies and support capacity planning and dimensioning decisions for different network applications and services. For example, traffic models for HTTP, FTP, near real-time video streaming, VoIP, gaming, and live video streaming sources have been used in the specifications of standards proposals for different network technologies, e.g., for the performance evaluation of the cdma2000 systems [3GPP2] or the multi-hop relay system in the IEEE 802.16 broadband wireless access systems [IEEE802.16].

The studies may also focus on the investigation of temporal variations in the network workload and analyses of the distinct hourly, daily and weekly patterns which may be present in the corresponding traffic [SPT07]. On the one hand, when the workload is analysed over a period of great variability and treated as a static snapshot, the analysis will reflect an "average" behaviour which might not accurately describe the workload experienced by the network at any time interval. On the other hand, a sound workload characterization should be performed over time periods of approximate stability, to avoid introducing spurious effects due to the aggregation of multiple workloads [AlA11]. For example, Gaussian traffic models have been used in [Has06] to bound the probability of overload on network links and other network resources, which only take the stationary distribution of the traffic rate into account.

Further, the workload studies may be concerned with the evolution of network applications and services, which results in permanently changing workload patterns. A thorough understanding of the dynamic workload properties can be exploited in order to optimize the system performance. For example, understanding of the evolution of the user behaviour and workload offered to the YouTube and similar Web 2.0 video sharing services is crucial to evaluate the data rate requirements and scalability of the YouTube (and similar Web 2.0) video sharing sites [AbS10]. Furthermore, the analyses of the distribution of the popular video files suggest that proxy caching of the popular YouTube videos can reduce the network traffic and increase the scalability of the YouTube Web site.

The workload modelling studies can also be focused on the identification of qualitative patterns (also called invariants) that may hold across different workloads of the same target application or different applications of the same type (e.g., file transfer, Web traffic, video streaming, etc.) and may provide a valuable and accurate insight into the application design, optimization, and management [FGV06].

Finally, workload and traffic models can be used for the generation of network workloads and the corresponding traffic in network simulations [CCG04, LAJ07] or real network testbeds [BPGP12, BDP12].

The origin of the analysed workload or traffic: for example, workload modelling studies may follow a source-based approach and concentrate on the characterization of traffic generated by different (types of) applications and services running on the single hosts in the network (e.g., [DPRPV08, ViV09]). Or the studies may also consider the aggregated traffic as it appears, e.g., on backbone or high-speed access links, and analyse the effects of superimposition of multiple synthetic traffic sources (e.g., the temporal variability [Has06, LBFE09] and dependency of aggregated traffic characteristics [SPT07]).

The modelling methodology: for example, the class of the underlying stochastic model used in the development of the concrete workload or traffic model (see below).

The challenging task of workload characterization and modelling is further exacerbated by the problem of limited availability of real representative workloads for the analyses due to privacy restrictions imposed, e.g., by service providers or governmental law authorities. For this reason, measurement-based studies may very often rely on data sampling, thus raising the issue of

a possible sampling bias and its implications for accurate workload characterization [MMV05].

Furthermore, workload modelling studies may require large-scale real-world datasets, which may be collected also for different (types of) applications, across different periods of time (cf. the large-scale studies of user access patterns for live (multicast) IPTV [GJR11] or VoD applications [GJCG13]). In such large-scale scenarios, the use of (partially available) local user access patterns would have inherent limitations in the face of the country-wide or global nature of the considered applications or services. Once again, availability of traces with real measurement data is often a strongly restricting factor, so that the researchers may be inclined to rely on simulation models to conduct their design and development efforts [CSK11].

Workload characterization and modelling studies can employ a variety of different modelling techniques, ranging from conventional inferential statistics, to more sophisticated methods using Markov models, Markov-modulated processes [Kin90], or arrival curves [KüT06], in addition to clustering, principal-component analysis, and other data mining techniques [HMS00, Jai91]. Furthermore, there exist a series of very complex stochastic processes aimed mainly at modelling of temporal dependencies in traffic characteristics at different time scales (e.g., Fractional Auto-Regressive Integrated Moving Average (FARIMA) [SSLL09], Fractional Sum-Difference (FSD), or Finite Brownian Motion (FBM), cf. [GrS05]).

Generally, the workload characterization and modelling is a very broad research field with a plethora of studies with their respective specific modelling purposes and objectives. However, not every workload model developed in such studies can be directly used for the generation of realistic synthetic network load or traffic for experimental purposes. At this point, a representation of the workload or traffic (model) which can be executed in a workload or traffic generator is of particular importance. Therefore, we recall that one of the major objectives of this thesis is the elaboration of a generally applicable method for the sufficiently formal, precise, and complete description of workload models which would allow one to generate the corresponding real workloads or traffic as a sequence of requests at different service interfaces in networks. The development of workload models for concrete types of applications or services in networks is, however, not in the primary focus of this thesis.

In the following sections we present some selected well-known methods proposed for modelling and specification of workloads in the networking research community. Some of these methods may allow to reflect only one specific characteristic (or dimension) of workload, e.g., packet inter-arrival

time or packet size. For example, the models based on the class of univariate Markov processes [Kin90] or arrival curves [KüT06] can, in general, support only one such dimension. This fact may represent a significant challenge for modelling (in respect to the complexity of the resulting model) because real network workloads do very often possess a number of different (and possibly also dependent) characteristics.

2.1.1. Selected Workload Modelling Techniques

In this section we first present the Markov Modulated Poisson Process (MMPP) [Hef80], Poisson Pareto Burst Process (PPBP) [ZNA03], and Batch Markovian Arrival Process (BMAP) [Luc91] model classes which can be used for modelling of network workloads in particular in order to describe the burstiness in the observed traffic. Next, we describe the Hidden Markov Model (HMM) [Rab89] model class which, among others, provides a means to build workload models capable to jointly take into account the first order statistics as well as temporal dynamics and correlation of different network traffic characteristics such as the inter packet time and packet size. Finally, we shortly address the techniques for modelling advanced properties of network workloads such as self-similarity and long range dependence.

Markov Modulated Poisson Process (MMPP)

Markov Modulated Poisson Process (MMPP) is a generalisation of the Poisson process where the job arrival rate may change over time. The use of MMPP for the modelling of network traffic has been first proposed in [Hef80]. An m-state MMPP can be viewed as m independent Poisson processes where λ_i is the arrival rate of the i^{th} process. An underlying continuous-time m-state Markov chain determines which of m arrival processes is active, i.e., the one in accordance with which arrivals are generated. After the i^{th} arrival process is activated, it remains active for an exponentially distributed amount of time with mean σ_i^{-1}. At the end of the active period, the j^{th} process is chosen as the next active process with probability $p_{i,j}$ where $\sum_j p_{i,j} = 1$ and $p_{i,i} = 0$. So, an m-state MMPP can be characterized by the parameters λ_i, σ_i^{-1}, and $p_{i,j}$, where $i, j = 1, 2, \ldots, m$.

The effectiveness of MMPP as a traffic model and, in particular, its ability to capture the burstiness in the traffic, have been evaluated, e.g., in the context of resource provisioning in Web applications [RCW12]. An important issue is the estimation of parameters of an MMPP such that the job arrivals generated using this MMPP have statistical properties that are

similar to those derived from a real trace. Therefore, several algorithms have been proposed in the literature to fit an MMPP to the observed data [HeL86, DeM93, BaF07].

Poisson Pareto Burst Process (PPBP)

The use of Poisson Pareto Burst Process (PPBP) for the modelling of aggregated traffic as it appears, e.g., on backbone or high-speed access links or in the Internet, has been first proposed in [ZNA03]. PPBP allows to specify multiple overlapping bursts whose lengths follow a heavy-tailed (Pareto) distribution. The authors in [ZNA03] presented methods to map the parameters of the PPBP to the set of measurable network traffic characteristics, described a technique for fitting the PPBP to a given traffic trace, and showed the ability of PPBP to accurately predict the queueing performance of a sample trace of aggregated Internet traffic.

Batch Markovian Arrival Process (BMAP)

The (continuous-time) Batch Markovian Arrival Process (BMAP) was proposed by Lucantoni [Luc91] as a generalization of the (simple) Markovian arrival process (introduced, e.g., in [LMN90]) by allowing more than one arrival at a time. The use of the BMAP model class for the development of analytically tractable models for aggregate IP traffic focusing on the burstiness and self-similarity properties is presented, e.g., in [KLL03]. The use of a discrete time version of the BMAP process model (called dBMAP) to characterize the long-range dependence present in traffic traces of aggregate link traffic has been proposed, e.g, in [SPV04]. The proposed model jointly characterizes the packet arrival process and the packet size distribution of IP traffic. In particular, packet arrivals occur according to a discrete-time Markov modulated Poisson process (called dMMPP) and, each arrival is further characterized by a batch whose size has a general distribution that may depend on the phase of the dMMPP describing the packet arrival process. The authors developed a parameter fitting procedure that is capable of achieving accurate replication of queuing behaviour for IP traffic exhibiting long-range dependence.

Hidden Markov Models (HMM)

Hidden Markov Models (HMM) have been used, e.g., in [SaV01, WWT02] for modelling the states of packet channels using the corresponding loss probabilities and end-to-end delay distributions. Further, a specific HMM has

been proposed in [DPRPV08] and used for the packet-level characterization of the network traffic in terms of the Inter Packet Time (IPT) and Packet Size (PS) stochastic processes. The authors in [DPRPV08] followed a source-based approach, i.e. sessions of traffic generated by different network applications and services running on single hosts have been analysed separately (and the proposed models do not focus on aggregated traffic which can be observed, e.g., on backbone or high-speed access links). A Hidden Markov Model can be defined as a probabilistic function of a (hidden) Markov chain and is composed of the following two variables:

- The hidden-state variable x_n, whose temporal evolution follows a Markov-chain behaviour. The state at discrete time n is represented by $x_n \in \{s_1, \ldots, s_N\}$ where N is the number of states.
- The observable variable y_n, that stochastically depends on the hidden state. The observable at discrete time n is represented by $y_n \in \{o_1, \ldots, o_M\}$ where M is the number of observables.

The authors of [DPRPV08] adopted a specific HMM with the discrete random state variable x_n introduced to account for memory and correlation phenomena between IPT and PS, which are assumed to be statistically independent given the state. The observable variable is a continuous bi-dimensional vector $y_n = [d_n, b_n]^T$ where d_n and b_n describe the IPT and the PS for the nth packet, respectively, and are specified by means of conditionally independent (given the state) Gamma distributions. The proposed model allows to capture important joint dynamics (in terms of both marginal distributions and auto- and cross-covariances) of IPT and PS and remains still analytically tractable. Further, the model capabilities of learning, generation, and prediction have been evaluated and concrete realistic packet-level models have been constructed from the automated analysis of empirical traffic traces. The approach has been applied to traffic traces of various application-layer protocols and services, e.g., Simple Mail Transfer Protocol (SMTP), HTTP, an online network game, and an instant messaging application. The obtained models have been validated by comparing the synthetically generated sequences of IPT-PS pairs with the corresponding values from the original traces. The experimental investigation conducted by the authors revealed that the proposed HMM-based models can provide acceptable results with a moderate number of states ($N = 5$ for SMTP and HTTP models, or $N = 4$ for online gaming and instant messaging models).

Modelling of Advanced Characteristics of Network Workloads

Investigation of the effects of advanced traffic properties such as traffic variability (i.e. fluctuation of traffic characteristics as a function of time) has been a subject of a large number of studies in the networking research community (cf. [WTSW95, CrB96, SPT07, CMCS08, FCS08, LBFE09, SSLL09], to name a few). High variability in traffic may have, under certain conditions, a significant impact on the network performance [LTWW94, ENW96] and its understanding can help to improve the efficiency of different network techniques such as traffic-control mechanisms and QoS schemes [SPT07, LBFE09].

One of the reasons for the high variability in traffic could be the long-range dependence (LRD) property of the traffic process (if such property can be observed in the concrete traffic trace). In general, a (weakly) stationary discrete-time real-valued stochastic process $X = \{X_n, n = 0, 1, 2, \dots\}$, with mean $\mu = E[X_n]$ and variance $\sigma^2 = E[(X_n - \mu)^2] < \infty$, is long-range-dependent if $\sum_{m=1}^{\infty} r(m) = \infty$, where $r(m)$ measures the correlation between samples of X separated by m units of time. If $\sum_{m=1}^{\infty} r(m) < \infty$, then X is said to exhibit short-range dependence (SRD).

Several possible causes of correlation and LRD in the aggregated IP traffic have been identified, such as the inherent structure and interactions of protocol layers [MiG98] or the superimposition of traffic sources with heavy-tailed distributions of the transfer durations [CrB96, WTSW95], the latter being sufficient for the generation of self-similar traffic. Self-similar processes are often used to build models of traffic which possess the LRD property.

Self-similarity in the context of network traffic refers to the scaling of variability (i.e. burstiness) in traffic. A time series $X = \{X_t, t = 1, 2, \dots\}$ is said to be exactly second-order self-similar if $X_t \stackrel{d}{=} m^{-H} \sum_{i=m(t-1)+1}^{mt} X_i$ for $H \in \mathbb{R}, 1/2 < H < 1$ and $\forall m > 0$ where $\stackrel{d}{=}$ means equality in distribution and m is the time lag [CrB96]. The parameter H (which is called the Hurst parameter) measures the degree of self-similarity for the random processes used for modelling network traffic and represents, basically, a measure of the speed of decay of the tail of the autocorrelation function. The definition suggests a simple test for self-similarity in network traffic, called the variance-time plot. In such a test the variance of $\sum_{i=m(t-1)+1}^{mt} X_i$ is plotted against m on log-log axes, where the X_i are measurements of traffic in bytes or packets per time unit. Linear behaviour of the plot with the slope greater than $-1/2$ suggests non-trivial self-similarity in the random process used for

traffic modelling. For further details on the self-similar processes we refer the reader, e.g., to [GrS05].

We should emphasize, that in this thesis we follow rather the source-based approach to workload modelling. According to such approach, the workload model aims to characterize and analyse separately several sessions of traffic generated by different applications or services on single network hosts and does not focus on the aggregate link traffic (as it may appear, e.g., on backbone or high-speed access links or in the Internet). The possible effects of the superimposition of multiple synthetic traffic sources, e.g., the presence of self-similarity or long range dependence in the aggregated synthetic traffic, is outside of the scope of this thesis.

2.1.2. Selected Workload Specification Techniques

In the following we will introduce the user behaviour graphs, finite state machines, and timed transition automata as a possible means for the specification of user behaviour models.

User Behaviour Graphs

The user behaviour graphs have been proposed by Ferrari [Fer84] in order to describe workloads offered to an interactive communication system whose performance can be analysed by a product-form closed queueing network model satisfying the conditions of the BCMP theorem [BASK75, Kin90]. The basic component of the workload were the possible types of user commands or interactions of the terminal users with the system. The offered workload has been described as a set of partially overlapping sequences of commands issued by terminal users.

In order to describe the behaviour of each of the m interactive terminal users of the system a probabilistic user behaviour graph has been introduced in [Fer84]. Each node in the graph represents an interactive command type, with the exception of node 0, which is the "dormant node". Users who are not using the system reside in node 0. When a terminal session starts, the state of the terminal user becomes 1 (the "login node"). During the session, different commands are executed by the user and the corresponding nodes of the graph are visited following the arcs of the graph. At the beginning of each terminal time period, a terminal user chooses the next command based on the probabilities from the user behaviour graph. The R different possible command types modelled in the request nodes in the graph are modelled as R different classes of customers in the queueing network.

It should be noted, that the interactive workload to be modelled and the workload model to be constructed have been assumed to be stationary in [Fer84]. This means that the desired workload model is not intended to reproduce any particular dynamic variations in workload characteristics and aims at reproducing the approximately similar time-invariant distributions of characteristics of the original workload.

The representation of the workload model by means of user behaviour graphs has been used in [Fer84] in order to analyse the problem of reducing the number of command types that appear in the workload model while preserving their relative frequencies of occurrence. The author has shown, that the workload model resulting from a simple aggregation of command types in command classes (which may ignore the existing sequential dependencies among different command types) may be – under certain conditions such as steady-state assumption, product-form queueing model – sufficient to generate workloads with similar characteristics as in the case when the sequential dependencies are considered.

User behaviour graphs have been used, e.g., by Calzarossa [CMT90] or later by Menascé [Men03, MeV00] as customer behaviour model graphs (CBMGs) and customer visit models (CVMs) for the modelling of workloads induced by Web e-business applications.

Finite State Machines

Finite state machines provide a simple and straightforward technique for the specification of workload models because they allow one to directly represent the waiting of the entities for certain events (inputs), and their reaction to them (outputs) including the transition to a successor state.

A finite state machine[1] can be defined (cf. [Kön12]) as a quintuple (S, I, O, T, s_0), where

- S is a finite, non-empty set of user states ($|S| < \infty$),
- I a finite, non-empty set of inputs,
- O a finite, non empty set of outputs,
- $T \subseteq S \times (I \cup \{\tau\}) \times O \times S$ a state transition function, and
- $s_0 \in S$ is the initial state of the automaton.

[1] The terms finite state machine and automaton are used synonymously in the following.

A transition $t \in T$ is defined by the quadruple (s, i, o, s') whereby $s \in S$ denotes the current state, $i \in I \cup \{\tau\}$ an input (event), $o \in O$ the associated output, and $s' \in S$ the successor state. Note that $\tau \notin I$ is a special event which designates an empty input and can be used for modelling spontaneous transitions to describe internal events. The execution of the transitions takes place simultaneously.

It should be noted that finite state machines allow to describe only the functional control flow, e.g., the sequences of requests submitted by the service users at a service interface. However, the automaton may become too complex (in terms of the number of required states) in case when changes in the data structures (e.g., the modifications of values of request attributes) or timing aspects are to be represented. Therefore, different extensions of the basic automaton concept have been proposed in order to be able to adequately represent the data flow and the timing aspects.

The concept of finite state machines has been used as a foundation for the user behaviour automata introduced by Wolfinger in [WoK90]. In Sec. 3.2 of this thesis, we present a generalization of the basic concept of user behaviour automata and propose a set of extensions which are required for the adequate representation of data flows and timing aspects during the specification of network workload and user behaviour models.

Timed Transition Automata

The concept of timed transition automata has been used, e.g., in [MJS08] for the specification of possible user interactions in structured interface environments like, e.g., Web applications and Web services. The domain of the action types available to a user in such an environment has been described by a state transition diagram extended with time constraints and each possible type of action is represented by a state transition label in the automaton.

A timed transition automaton (TTA) can be defined (cf. [AlD94]) as a tuple $(\Sigma, S, s_0, C, E, F)$ where

- Σ is a finite alphabet,
- S is a finite set of states,
- s_0 is an initial state,
- C is a finite set of clocks,
- $F \subseteq S$ is a set of final acceptance states,

- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ defines the transition table for the automaton. Each transition $e \in E$ is a quintuple $e = (s, s', a, \Lambda, \delta)$ representing a transition from state s into state s' on input symbol a which can occur at a certain time τ when clock constraint δ is verified by the current values of clocks. The transition also resets to 0 the clocks from the subset $\Lambda \subseteq C$ of clocks.

A TTA is able to recognize timed words, i.e. a finite sequence of pairs $[(a_0, \tau_0), \ldots, (a_k, \tau_k)]$ made by symbols $a_i \in \Sigma^*$ over a given alphabet Σ and time values $\tau_i \in \mathbb{R}$ for $i \in [0, k]$ with $\tau_i \leq \tau_{i+1}$ with $i \in [0, k-1]$. The pairs in the sequence can be seen as a sequence of log records, describing user actions or events annotated with the time in which they occurred.

The TTA has been used in [MJS08] to specify the timing constraints for the user actions to be executed only when some certain time conditions are met (e.g., submitting a reply from a server search engine within a given interval of time). A domain automaton can then be defined in order to represent the legal sequences of user actions which can occur in the system.

2.2. State-of-the-Art in Workload Generation

Generation of realistic synthetic workload and traffic is very often required for experimental activities in networking research. The corresponding workload and traffic generators can be implemented as hardware or software platforms or include both hardware and software components.

Commercial hardware platforms are typically able to reach a high degree of performance and precision and are usually provided with detailed datasheets containing certified specifications of the supported precision and performance characteristics (e.g., packet and data rate). Therefore, due to their reliability, the hardware-based platforms can be indispensable, e.g., for performance, capacity, and stress tests of different network hardware appliances and devices (such as switches, routers, firewalls, IDPSs, etc.).

For example, Spirent AX/4000 [AX4000] is a large-scale feature-rich high-performance hardware traffic generator with a modular, multi-port architecture capable of testing access, mobile backhaul, routing, multicast, switching, MPLS and other technologies in Asynchronous Transfer Mode (ATM), IP, Frame Relay and Ethernet networks at speeds up to 10 Gbps. The platform provides a set of different conformance test suites for a number of protocols and a set of corresponding traffic models. Hardware appliances (such as IXIA Optixia series [IXIA], Agilent/HP 1735A LAN Protocol test modules, or Napatech [Napatech] devices) can also perform a trace-replay,

i.e., inject traffic from a trace captured on real network links. However, as the corresponding models and the trace replay capability are implemented in hardware, introducing new features is rather difficult and the approach may not provide a sufficient flexibility for selected test scenarios. In particular, hardware-based traffic generators can hardly be deployed on a large number of nodes (mainly due to economical reasons), which may in some way limit their applicability in tests to be performed with complex workloads in large-scale networks or testbed scenarios in order to be representative of the reality.

In contrast to a limited flexibility of hardware-based generators, software-based generators typically allow a very easy configuration of the traffic stream to be generated (often with graphical interfaces) and can be rapidly modified and extended for a specific research purpose. Such, new features, statistical models, support of additional protocol stacks, new operating systems and hardware platforms can be added and the tools can be more easily deployed onto a large number of network nodes in order to reproduce distributed scenarios. Moreover, when executed on top of real operating systems and network protocol stacks, the software-based generators may allow to perform more realistic experiments and to test concrete implementations of different protocol mechanisms [BDP10].

However, the software-based platforms inherently rely on the used hardware (which may be intentionally chosen to be commodity or Commercial off-the-shelf (COTS) hardware for economical reasons), the adopted operating system (which may provide explicit real-time extensions), and the software configuration of the host(s) used for traffic generation. For this reason, the accuracy, precision and performance characteristics of the traffic generation process can strongly vary among different software-based generators.

In the following, we will concentrate on the software-based workload and traffic generators. Generally, software-based workload generators can be classified according to the *modelling methodology* which has been applied in the underlying workload model and implemented in the generator, while the particular differentiating factors may be, e.g.:

Abstraction level: is determined by the types of objects and entities considered in the underlying workload model. For example, in case of *application-level* model of Web traffic, the HTTP request/response pairs exchanged between the Web client and the Web server may be such entities. On the *flow-level*, the traffic can be described by means of flows (identified by the IP address and port number of the sender and the receiver and

the number of the transport protocol to be used, e.g. TCP, UDP, or Stream Control Transmission Protocol (SCTP)) with the specified number of packets, bytes, and duration. On the *packet-level*, the traffic may consist of packets characterized by means of stochastic variables for the distribution of the packet inter-departure times and packet sizes.

Generation method: the two major approaches to generate synthetic network workload are *trace-based* and *analytical model-based* methods which can be applied at different abstraction levels (see above). Because of the well-known strengths and weaknesses of these two approaches, workload generators can combine both techniques in order to achieve a higher degree of flexibility.

Recall that trace replay provides a simple, straightforward technique to inject traffic with almost arbitrary application payload pattern and may be very useful in situations when traffic to be generated is not responsive to changing network conditions. However, the experiments may be limited to the concrete available traces and their characteristics. Further, relevant traffic traces may be subject to privacy restrictions imposed by service providers and are hardly available for the purpose of testing (while storing such traces may be officially forbidden).

As opposed to the traced-based techniques, stochastic models may provide the required flexibility. However, the decisions which relevant properties of the real workload are to be reproduced and the correctness and validity of the workload model for the specific scenario must be proven in order to produce sufficiently realistic workloads.

Open-loop versus closed-loop generation: this feature characterizes the ability of the workload generator (and the underlying workload model) to appropriately respond to changing network conditions as emphasized in [FlP01]. In the open-loop mode the generator operates independently from the observations of the network conditions that must be performed during the workload generation. In the closed-loop mode the tool is able to change its behaviour during the workload generation according to these observations and to appropriately modify the characteristics of the traffic to be generated (e.g., to adjust the parameters of the statistical distributions of the inter-departure times and sizes of packets, or to change the content of the packet payloads).

Application field: can the traffic generator be used in the network simulation (or emulation) environments and/or in the real network testbeds? Is the

generated workload more appropriate to analyse the characteristics of the network or the characteristics of the used applications (e.g., a Web application server)?

Further, the software-based workload and traffic generators may also differ in their *architectural features*, e.g.:

Target service interface: a target service interface (or a set of service interfaces) at which the workload generator is able to inject the generated requests or traffic. Strictly considered, the target service interface is not to be confused with the abstraction level of the underlying workload model used in the generator. For example, Web traffic can be generated according to an application-level model of HTTP traffic sources and injected at the application layer HTTP service interface into the network (such as in `Surge` [BaC98]). Alternatively, Web traffic can be generated according to a model which incorporates a set of application-level, flow-level, and packet-level characteristics of the traffic induced by the HTTP sources and, thereafter, be injected at the network layer (such as in `Harpoon` [SoB04] or `LiTGen` [RRB07a, RRB07b]) or transport layer (e.g., `Swing` [ViV06, ViV09]) service interface.

Software and hardware co-design: does the generator architecture make use of dedicated hardware components like, e.g., the Intel IXP2400 Network Processor (NP) [IXP2400] as it is the case in `BRUNO` [APF08a, APF08b] or `Pktgen` [BBCR06].

User-space versus kernel-space: the generator architecture may consist of only user-space modules (e.g., `MGEN` [MGEN] or `D-ITG` [AEPV05]), only kernel-space modules (e.g., `KUTE` [ZKA05]) or include both user-space and kernel-space modules (as it is the case with the generator proposed in [BPGP12]).

Distributed workload generation: does the generator provide only centralized workload generation function or is it able to produce and inject workloads from geographically distributed hosts (e.g., `D-ITG`, or `LoadStorm` [loadstorm])?

Scalability on multi-core platforms: does the generator architecture make use of parallelism and is it able to appropriately exploit the multi-core processor architectures and multi-queue Network Interface Cards (NICs) [BPGP12]?

Performance characteristics: The *performance* of a traffic generator can be characterized, e.g., by the maximum achievable packet and data rate for a given packet length. Further, according to the definitions in Paredes-Farrera et al. [PFFG06], the term of *precision* is related to the quality and stability of the system, while the term of *accuracy* is related to measurements of the similarity among the created values with the true ones. Therefore, when one is interested in timeliness of generated packets, the precision can be referred, e.g., as the standard deviation of generation times, while the accuracy can be described by the average errors between the actual and specified generation times.

For example, the experimenter may be interested in the ability of a particular traffic generator to saturate the capacity of a 1 Gigabit or a 10 Gigabit Ethernet link (also with the smallest possible 64 byte long Ethernet packets).

Conditions and availability: is the workload generator available as a commercial tool or is it freely available, may be also as an open source tool?

The following list of workload and traffic generators is not pretended to be complete. In contrast, we tried to choose the most representative examples of traffic generators for networking research in order to be able to demonstrate the different possible approaches followed by various solutions. Related surveys of the traffic generation tools available for networking research can also be found, e.g., in [AEPV05], [BDP10], or [BDP12].

At this point we should emphasize, that the workload modelling and generation approach proposed in this thesis is strongly oriented on the target service interface for workload generation. This target service interface for load generation is to be chosen by the experimenter strongly according to the objectives of the particular workload study being carried out. For this reason, we decided to arrange the list of existing traffic generators according to the target service interface, at which the generated traffic is injected. We start with the generation of Web traffic at the application level HTTP service interfaces.

2.2.1. Web Workload and Traffic Generation

Software-based Web workload and traffic generators are based either on traces reflecting real Web user sessions or on workload models that are designed and implemented to generate HTTP requests. Floyd and Paxson demonstrated in their study [FlP01] how difficult it is to generate representative Web requests,

especially when some particular characteristics in a dynamic Web site should be modelled, and how these characteristics impact on the behaviour of the Web clients.

One of the first studies trying to identify the common characteristics in Web server workloads is the work done by Arlitt and Williamson [ArW97], which used logs of Web server accesses at six different sites (three from university environments, two from scientific research organizations, and one from a commercial Internet service provider). The observed workload characteristics were used to identify the possible strategies for the design of a caching system to improve Web server performance.

Web workload generation at the application-level (emulated Web clients, real Web servers)

Barford and Crovella [BaC98] applied a number of observations of Web server usage to create a realistic Web workload generation tool, called `SURGE` (Scalable URL Reference Generator) which mimics a set of real users accessing a server and generates Uniform Resource Locator (URL) references matching empirical measurements of request and server file size distribution, relative file popularity, embedded file references, temporal locality of reference, and idle periods of individual users. The relevance of these Web workload characteristics as well as their concrete values were identified based on single (non-recurring) measurements, so that later revisiting done by Williams et al. [WAW05] was required due to emerging Web technologies and a nearly 30-fold increase in overall traffic volume in 2005.

The study [ACC02] proposes different benchmarks (partly based on the TPC-W benchmark [TPC-W] which has been declared obsolete in 2005) to be used for online book-store applications, auction sites, and bulletin boards with dynamic Web content. The benchmarks use a real Web server infrastructure (consisting of a Web, application, and a database server) and specify a predefined set of Web pages and database items which can be requested by the experimenter from the client side. Along with the real Web server application objects, the authors provided a freely available workload generator tool (a Web client emulator) to drive a dynamic content Web server with various workloads specified in the benchmarks. Following the TPC-W specifications, the workload generated by the client emulator consists of a specified number of concurrent clients and their interactions with the SUT. Each emulated client opens a session (which is a persistent connection) with the SUT and repeatedly makes a request, parses the server's response to the request, and, after emulating the specified amount of time

("thinking time") of a real client, follows a (hyper-)link embedded in the response. The tool uses a simple state machine with a transition probability matrix to determine the next link (contained in the server response) to be followed in the automaton. A state in the transition matrix corresponds to a particular interaction of the SUT and the Web page while a transition corresponds to clicking on a link in the page. Different system utilization statistics can be collected on the machines belonging to the SUT including, e.g., the throughput and response time statistics, and utilization of Central Processing Unit (CPU), memory, network and disk for the duration of the experiment. The tool has been used for different research studies on dynamic Web content generation, clustering, caching, and Web application server design.

`GUERNICA` [OSPG09] is a Web workload generator with the ability to precisely generate the dynamic workload of Web 2.0 by implementing the `Dweb` model introduced in [OSPG05]. The underlying model makes use of the customer behaviour model graphs proposed by Menascé et al. [MeV00] and is based on the following three main concepts: 1) *navigation*, which defines the behaviour of a single dynamic Web user interacting with the Web server(s) and is specified as a sequence of URLs for HTTP requests where each visited URL depends on the previously visited one, 2) *workload test*, consisting of the set of navigations launched during the simulation process which can be executed concurrently, and 3) *workload distribution*, which refers to a set of workload tests that are concurrently executed by one or more generators in different nodes or in different machines when simulating the Web client's behaviour.

`GUERNICA` consists of three main components implemented as Web applications using the Web services technology: the workload generators, the performance evaluator, and the performance tests planner. These components allow to carry out the workload test process consisting of the following four steps: 1) defining the client behaviour by using the *navigation* concept, 2) defining the workload of the target site by using the *workload test* and the *workload distribution* concepts, 3) executing the workload tests gathering performance statistics, and 4) analysing the performance of the target site on the basis of the obtained statistics. Further, in order to obtain concrete sequences of users' navigations, an external Mozilla plug-in has been integrated in `GUERNICA` to capture the URL requests from the users in the Mozilla browser.

The following two tools are examples of commercial Web workload generators performing an application-level trace replay of Web/HTTP traffic recorded from real browser user sessions (e.g., in the HTTP archive format

[Odv15]).

LoadStorm [loadstorm] is a cloud-based platform for load testing of Web applications and (mobile) Web services. The tool is provided with the large supplemental set (a "cloud") of dedicated load generation machines and allows to perform the generation of Web traffic also from geographically very strongly distributed Web clients (e.g., hosts located in the USA, Ireland, Singapore, and Tokyo can participate in the same load experiment, provided there are dedicated LoadStorm cloud hosts available in these regions). In order to mimic the behaviour of real users, LoadStorm relies on the recordings of user interactions which can be captured using the developer tools of the browser and stored, e.g., in the HTTP archive (HAR) format (cf. [Odv15]). The recording contains every request made by the browser (including Hypertext Markup Language (HTML), Cascading Style Sheet (CSS), images, Javascript, and Asynchronous JavaScript and XML (AJAX)) and can be customized for each individual virtual user to be emulated using the advanced user interface. For example, the experimenter can specify customized test data and think times between subsequent requests, time-outs for different types of object requests, user names and passwords, custom query strings, and provide different application security identifiers (CSRF tokens, SessionIDs, hidden input fields, etc.). The tool includes a scenario builder to specify different actions of virtual users like open a new page, click a specific link, click a random link, or submit a form. Finally, LoadStorm provides reporting of key performance characteristics (such as the number of active users, throughput, requests rate, response time, error rate, etc.) and in-depth request error analysis during the test (of errors captured, e.g., from response status codes, request time-outs, and server connection problems).

WAPT Pro is a tool for workload, stress, and performance testing for Web applications provided by the SoftLogica Inc. http://www.loadtestingtool.com. The procedure of performing the load tests is similar to LoadStorm, i.e. the experimenter constructs the test by navigating through the Web site in the browser to record a user session. Each session is recorded to a virtual user profile as a sequence of HTTP requests. WAPT provides an extended framework for editing the properties of every particular request in the profile (e.g., request headers, page elements, and other options) and can then replay different profiles with a specified number of virtual users (also considering the specifications how the number of virtual users changes during the test). Furthermore, it includes capabilities to perform testing from different geographical locations using a number of load agents. The tool automatically generates cookies and session variables for correct user sessions, supports testing secure HTTPS web sites with different types of

user authentication and client certificates, and provides detailed reports on different performance characteristics and errors after the test completion.

Web workload generation at the application-level (emulated Web clients and Web servers)

In the context of a comprehensive study [SCK03] a set of models has been derived from an analysis of the content from six representative news and e-commerce sites. The models capture the characteristics of dynamic Web content both in terms of independent parameters (such as the number of objects, distribution of the object sizes and object freshness times) as well as derived parameters (such as content reusability across time and linked documents). The authors proposed a Java-based dynamic content emulator (`DYCE`), which emulates a Web server that serves dynamic Web content. The emulator uses the proposed models to generate parameterizable server-side include-based dynamic content and serve requests for the whole documents or separate objects being requested (e.g., from an idealized Web cache simulator provided by the authors for the validation of `DYCE`). Further, it uses delay models from previous research in order to replicate the appropriate delays induced by the dynamic content generation [ICDD00]. In comparison, e.g., to `SURGE` [BaC98] which has been designed to model client access patterns to static Web pages, `DYCE` focuses on the complementary goal of emulating the behaviour of the Web server, both in terms of its workload properties and the nature of the dynamic content itself.

`ParaSynTG` [KRL08] is a synthetic trace generator for source-level representation of Web traffic with different characteristics such as document size and type, popularity (in terms of frequency of reference), temporal locality, and the fraction of dynamic requests and of requests been requested only once ("one-timers"). `ParaSynTG` is able to consider the dependency between the size of the documents and their frequency of reference as well as between the type of documents and their size. The tool has been designed for the generation of synthetic Web workload traces only (which can be used, e.g., in simulation experiments) and, in the opposite to the design objectives of our Web workload generator `UniLoG`, provides no facilities to generate and to inject real HTTP requests into the network.

Generation of Web-like traffic at the packet-level or flow-level

The Web traffic generators presented above have been designed with the primary goal to generate the workload for a Web service or a Web ap-

plication (which are hosted at a Web server or a number of Web servers and may involve additional application and database servers). Therefore, these generators attempt to include more application details and follow a "page-based" approach, i.e. they explicitly consider the Web page structure, the location of page components on the server(s), the human actions of thinking and page selection to control the creation of new HTTP requests, etc. Such page-based methods can also be used when the researcher's aim is to generate a "Web-like" traffic at the transport layer interface (e.g., at the TCP service interface) as it has been done e.g. in [BaC98] or in [LAJ07].

For example, the Web traffic generator used in [LAJ07] in order to study the effects of Active Queue Management (AQM) and Explicit Congestion Notification (ECN) techniques on Web performance consisted of a program to emulate client-side user actions (the "browser") and a server-side program to respond to client generated requests (the "server"). The client and the server communicate by means of the TCP socket interface using the socket operations `connect()`, `send()`, and `recv()`. For each request, the client generates a message of random size sampled from the request size distribution and sends this message over the network to an instance of the server program. The message specifies the number of bytes the server has to return as a response (which is determined according to the distribution of response sizes separately for top-level or embedded request). The server generates a message of the specified size and transmits it back to the browser. Despite a relatively comprehensive model for the HTTP source used by the client-side "browser", the resulting test traffic remains to be an HTTP-like TCP traffic, because the tool does not set the HTTP request headers in the generated messages. Such HTTP-like traffic may be sufficient in [LAJ07] for the performance evaluation of AQM and ECN techniques (which are both QoS mechanisms employed at the network layer) but it will be not suitable for the performance evaluation of, e.g, Web proxies and caches at the application layer.

`PackMime-HTTP` [CCG04] is a tool for generating realistic synthetic Web traffic in network simulations using the source-level models for aggregated HTTP traffic proposed in [CCG04] and implemented as the corresponding objects in the ns-2 network simulator. Aggregated HTTP traffic (as it appears, e.g., on backbone or high-speed access links) is described as a collection of independent TCP connections, each characterized by a set of source variables: arrival time of the connection, round-trip time for the client and for the server, number of request/response exchanges, time gaps between exchanges, sizes of individual requests and responses, and server delays.

The authors argued, that such a "connection-based" approach for modelling of HTTP traffic is able to capture relationships and significant dependencies in the collection of the source variables which were not considered in the existing page-based models. Further, the approach is more likely to scale to modelling the traffic generated by other application classes and different application traffic mixes (provided that the applications use TCP for transport, e.g., file transfer, Internet video streaming, instant messaging, peer-to-peer file sharing)[2]. Therefore, the authors recommend to use their connection-based approach for the generation of synthetic Web traffic carried by network links, routers, and protocol stacks (i.e., in the "traffic for the network" scenarios).

In `PackMime-HTTP` a lot of effort has been spent by the authors on the ability to consider different network and protocol characteristics (such as the round-trip times for the client and the server, link capacities and error rates, and the dynamic TCP interactions between Web clients and servers). As a consequence, not only the Web clients and Web servers but also the other components of the network under study have to be modelled. For example, the authors in [CCG04] had to provide the interaction of the proposed `PackMime-HTTP` model with the TCP layer objects in the ns-2 network simulator. Therefore, the application field of the traffic generators following the approach of `PackMime-HTTP` in [CCG04] or `Swing` in [ViV09] can be assumed to be rather restricted to scenarios, which are similar to those covered by the network simulation experiments.

Finally, `Harpoon` [SoB04] and `LiTGen` [RRB07a, RRB07b] are open-loop generators of aggregated network traffic at the flow level which are able to generate traffic also from HTTP sources. We will describe these solutions later in this section, as they are, strictly considered, generators of IP traffic at network layer service interfaces.

2.2.2. Traffic Generation at Transport Layer Service Interfaces

MGEN

The Multi-Generator (`MGEN`) is an open source software developed by the Naval Research Laboratory (NRL) PROTocol Engineering Advanced Net-

[2] The connection-based approach proposed in [CCG04] has been later used in [ViV09] in order to generate realistic and responsive network traffic consisting of mixes from different application classes. The corresponding traffic generator `Swing` developed in [ViV09] will be presented later in this section (because it is, strictly considered, a generator of TCP traffic streams).

working (PROTEAN) Research Group which provides (in its current version 5.0) the ability to generate, receive, and log real-time traffic patterns of unicast and/or multicast UDP and TCP applications in order to perform IP network performance tests and measurements [MGEN]. The tool suite currently runs on various Unix-based (including MacOS X) and Win32 platforms, is implemented in user-space, and can also be used in network simulation environments like `ns-2` and `Opnet`. Traffic generated by `MGEN` consists of a series of sequence-numbered messages with different sizes and inter-departure times determined according to a traffic pattern specified by the experimenter. Currently, `MGEN` supports the pattern types `PERIODIC`, `POISSON`, `BURST`, `JITTER`, and `CLONE` (the latter allows to extract the message sizes and/or inter-message times from a trace file in the binary tcpdump format). Further, script files are used in order to control the generated loading patterns over the course of time. Finally, `MGEN` log data can be used to calculate performance statistics, e.g., on throughput, packet loss rates, and communication delay. However, the performance of `MGEN` is reported to be rather low [DBP07, BDP10]. Such, the maximum achievable packet rate for small packets of 64 byte length remains below 80.000 pps. Further, the accuracy of the message inter-departure times may be violated when the *precise* option is disabled (which activates polling, if needed, to precisely schedule the message inter-departure times).

RUDE/CRUDE

`RUDE/CRUDE` (Real-time UDP Data Emitter / Collector for RUDE) is a small and flexible user-space generator of UDP traffic which can be received and logged using the corresponding collector module [RUDE]. The development of `RUDE` was motivated mainly by the accuracy limitations in the `MGEN` traffic generator due to the used low-resolution system timers in the Linux kernel on PC-platforms (the *precise* option was not available in `MGEN` at that time). Therefore, the operation and configuration of `RUDE` are very similar to `MGEN`. The tool can generate and measure only UDP traffic, is provided with a non-extensible script language to control the generated traffic patterns over time, and is not suitable to work at high packet rates, especially with small frame lengths (cf. [BGPS05, BDP10]). The `RUDE` project seems to be not longer supported (since the last release 0.70 in 2002, cf. [RUDE]).

ITG / D-ITG

The Internet Traffic Generator (ITG) has been introduced in [APV04, AEPV04] with the aim to generate (network, transport, and application layer) traffic at packet level and accurately replicate appropriate stochastic processes for both inter departure time (IDT) and packet size (PS) random variables. For this reason, ITG supported a set of different statistical distributions (e.g., exponential, uniform, constant, Pareto, Cauchy, normal, etc.) and has been first used to generate synthetic UDP and TCP traffic according to the source-level models for different application-level traffic sources, e.g., Telnet, SMTP, Network News Transfer Protocol (NNTP), FTP, HTTP, Domain Name System (DNS), VoIP, Video, etc.

In [AEPV05] a distributed platform for traffic generation (called D-ITG) has been developed on the basis of ITG in order to increase the number of application and traffic scenarios and improve the scalability and performance of the original centralized ITG traffic generator. D-ITG provides facilities for measurement of different traffic parameters at the packet level (like delay, jitter, packet loss and throughput). In the first variant of the proposed architecture, a log server is in charge of recording the information transmitted by senders and receivers and the required communication is based either on TCP or UDP. In the second variant senders and receivers make use of a Message Passing Interface (MPI) library to implement a control channel.

As of [AEPV05], D-ITG was provided with a set of classical packet-level models for different traffic sources, e.g.:

- TELNET, NNTP, SMTP, and FTP traffic sources [DJCME92, Pax94, LFJ97],
- WWW traffic [CrB96, ArW97, LFJ97],
- VoIP traffic [LFJ97, Cisco2], and
- MPEG encoded video streams [GaW94, KrH95, Ros95, LFJ97].

However, the aforementioned models could probably be seen as rather out-dated already at the time of introduction of D-ITG. For this reason, a Hidden Markov Model (HMM) for Internet traffic sources at packet level has been proposed in [DPRPV08], which allows to jointly analyse Inter Packet Time (IPT) and Packet Size (PS) stochastic processes. The model is able to capture the behaviour of marginal distributions, mutual dependencies, and temporal structures of the traffic generated by a heterogeneous set of sources and can be used for traffic generation in D-ITG. According to

the source-based approach, the model does not focus on the aggregate link traffic but aims at replication of separate traffic sessions originating from single hosts and related to specific application-level protocols. The proposed approach has been applied to various real traffic traces in order to obtain concrete source models of packet-level traffic generated by SMTP, HTTP, a network game ("Age of Mythology"), and an instant messaging application (MSN Messenger).

According to the experimental results presented in [BDP10], `D-ITG` offers a quite moderate performance and accuracy. Such, the tool can achieve a maximum packet rate of ca. 140.000 pps, meaning that it is not able to saturate the capacity of a Gigabit Ethernet link with the smallest possible (64 byte) Ethernet packets (because the corresponding data rate achievable with such packets in `D-ITG` cannot exceed 650 Mbit/s).

The authors of `D-ITG` frequently emphasized that they aim to simulate (i.e., to reproduce a traffic profile according to the stochastic models of IDT and PS), and not to emulate the traffic (which is defined by the authors as a reproduction of traffic resulting from a specific protocol, e.g., reproduction of HTTP messages without using a browser). So, every time when the authors speak about the generation of, e.g, VoIP traffic, they actually mean a reproduction of a "VoIP-like" traffic induced at the transport service interface (e.g., UDP in this case) meaning that only the UDP header fields (with the randomly chosen payload buffer) and no other VoIP specific payload fields are set in the generated traffic. In the consequence, the traffic generated by `ITG` using its analytical modelling function can be used in the performance experiments on the network or transport layer (e.g., to evaluate performance of an IP router) but not in the experiments at the application layer (e.g., to evaluate performance of a VoIP gateway) because the generated "VoIP-like" traffic will not be recognized as real VoIP traffic by a real VoIP/SIP gateway. The same would be valid for the traffic from other network applications and services at the application layer.

For these reasons, the functionality of `D-ITG` has been extended in a new version [BDP12] in order to be able to combine the already existing analytical models with the trace-based techniques. As of [BDP12], the tool is now able of replicating Packet Capture (PCAP) traces which allows to support arbitrary payload patterns (i.e., also from application layer traffic sources). Further, the authors mentioned the possibility to improve the performance characteristics of the generator by means of using novel socket families (e.g., `PF_RING DNA`, cf. [ntop1] and later in this section) in the traffic transmitter component.

Network throughput measurement tools

`iPerf` [iPerf3] and `Netperf` [netperf] are typical examples of tools developed with the aim to be used primarily as network throughput measurement and benchmark tools. `iPerf` was originally developed by NLANR/DAST and the current version `iPerf3` of the tool provides facilities for active measurements of the maximum achievable throughput in IP networks. It supports tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and Internet Protocol Version 6 (IPv6)). For each test it reports the throughput, packet loss, delay jitter, and other parameters (like, e.g., observed buffer sizes). `Netperf` provides tests for both unidirectional throughput, and end-to-end latency for TCP and UDP traffic via BSD Sockets as well as SCTP traffic for both IPv4 and IPv6.

Benchmark tools like `iPerf3` and `Netperf` usually generate as much traffic as possible to measure the network performance. Therefore, strictly considered, they are no traffic generators because they cannot generate specific traffic profiles specified by the experimenter, e.g., in terms of inter-departure times and sizes of packets.

2.2.3. Traffic Generation at Network Layer Service Interfaces

Harpoon

`Harpoon` is a tool developed by Sommers and Barford in [SoB04] for generating representative packet traffic at the IP flow level. A flow is defined as a series of IP packets between a given pair of tuples (IP address, port number) using a specific transport protocol (e.g., TCP or UDP). The tool can be used in a router or emulation testbed environment and generates TCP and UDP packet flows that have the same byte, packet, temporal (in terms of the inter-arrival times of connections) and spatial (in terms of the IP address ranges for the sender and receiver) characteristics as measured at routers in live environments. `Harpoon` is distinguished from other tools that generate statistically representative traffic in that it can self-configure by automatically extracting parameters for its hierarchical traffic model from standard Netflow [Cisco1] logs or packet traces.

The flow-level traffic generation is abstracted into a series of application-independent file transfers that use either TCP or UDP protocols for transport. `Harpoon` uses a hierarchical two-level flow-based traffic model which consists of sessions comprising a series of connections separated by durations drawn

from the inter-connection time distribution. Source and destination IP address selection is weighted to match the frequency distribution of the original flow data. The number of active sessions determines the overall average load offered by `Harpoon`. A heavy-tailed empirical file size distribution and an ON/OFF transfer model can generate self-similar packet-level behaviour. In summary, the model used in this tool is made up of a combination of five distributional models for TCP sessions: file size, inter-connection time, source and destination IP address ranges, and number of active sessions. Each of this distributions can be specified manually or extracted from packet traces or Netflow data collected at a live router.

It is important, that the approach taken in `Harpoon` uses source-level traffic descriptions that do not make assumptions about the transport layer, rather than packet-level descriptions based on prior network state embedded in low-level timings [FlP01].

Swing

`Swing` is a closed-loop, network responsive traffic generator for network emulation test-beds developed by Vishwanath and Vahdat in [ViV06, ViV09]. The tool uses a rather comprehensive structural model for the traffic observed at a single point in the real network and automatically extracts distributions for different characteristics of user, application, and network behaviour in order to generate synthetic traffic at a single target link modelled as a dumb-bell in a network emulation environment `ModelNet` [VYW02]. In particular, the proposed structural model consists of four levels: 1) Users, characterized by the client IP address, the number of requests, and the think time between individual requests, 2) Sessions, characterized by the number of parallel connections and the time between the start of connections, 3) Connections, characterized by the destination or server IP address, the number of request-response exchanges per connection, the size of the request and the corresponding response, think time between exchanges on a connection, type of the transport protocol (TCP or UDP), and packet size and packet arrival distributions for individual responses, 4) Network characteristics including link capacities, loss rates, and latencies (delays) for paths connecting each host in the original trace to the target link.

The authors claim that their main contributions are 1) the ability to both extract wide-area network conditions from an existing packet trace and to replay these network conditions with sufficient fidelity to reproduce essential characteristics of the original trace, and 2) the understanding of the requirements for matching the burstiness of the packet arrival process of an

original trace (e.g., well-known Auckland, MAWI, and CAIDA traces) at a variety of time scales, ranging from fine-grained (e.g., 1 ms) to coarse-grained (e.g., multiple minutes). `Swing` aims at matching burstiness in terms of 1) both number of bytes and number of packets, 2) both directions (arriving and departing) of a network interface, 3) a variety of individual applications within a trace (e.g., HTTP, peer-to-peer file sharing, SNMP, NNTP, etc.), and 4) original traces at a range of speeds and taken from a variety of locations.

The modelling methodology proposed with `Swing` has also some known limitations. First, the application behaviour is modelled based on the information extracted from the publicly available packet traces (Auckland, MAWI, CAIDA) which contain only network and transport layer headers. Second, the accuracy of the tool is limited by the accuracy of the used traces and the model parameters extracted for user, application, and network behaviour. Further, the focus is on generating traffic for the single network link modelled as a dumb-bell in a network emulation environment. So, the distribution of requests and responses among particular clients and servers in the original trace is not modelled.

LiTGen

`LiTGen` is an easy to use and tune open-loop traffic generator developed by Rolland, Ridoux, and Baynat in [RRB07a, RRB07b] that statistically models IP traffic on a per user and application basis. From a packet level capture originating in the operational wireless access network of Sprint Labs, and taking the example of Web (in [RRB07a]) and P2P and mail wireless traffic (in [RRB07b]), the authors show that their hierarchical traffic model is sufficient to reproduce accurately the traffic burstiness and scaling properties at small and large time scales. `LiTGen` relies on a hierarchical description of traffic entities, which are represented by one or several uncorrelated random variables either related to a time (duration or inter-arrival time) or a size metric. For example, the model used for Web traffic in [RRB07a] consists of four levels with the following corresponding entities: 1) Session level, characterized by the number of downloaded pages and the inter-session durations, 2) Page level, characterized by the page size (defined as the number of objects involved in a page) and the corresponding page reading duration, 3) Object level, with the objects inter-arrival times within a page and the number of packets in an object, and 4) Packet level, characterized by the inter-arrival times between packets in an object. Selected entities can

be removed from the model for simplicity, if needed (as it has been done, e.g., with the page level in [RRB07b]).

The authors emphasize that the proposed model is intentionally kept simple since the client / server interactions are not modelled and the network or protocol characteristics (like, e.g., round-trip times, link capacities, TCP dynamics) are not considered. So, the model does not rely on a complex emulator (that would reproduce the link layer or TCP dynamics) and allows fast computation when being executed on a commodity hardware (while, e.g., `Swing` relies on a third-party network emulator `ModelNet` requiring high computing resources). Similar to the methodology in [ViV06, ViV09], the authors used second-order analysis (wavelet-based methods) in order to identify the dependencies across the random variables composing the underlying traffic model and to prove the ability of `LiTGen` to reproduce accurately the captured traffic and its properties over a wide range of time scales. The analysis showed that an introduction of a simple dependency between the object sizes and the distribution of the packet inter-arrival times can succeed in reproducing the traffic correlation structure accurately. The authors claimed, therefore, that under certain conditions it can be possible to reproduce the second order traffic characteristics without introducing more complex non-renewal processes and considering network or protocol peculiarities in the `LiTGen` model while leading to a much simpler traffic generator than, e.g., `Swing`.

However, in order to use `LiTGen` in an operational network, one must characterize the dependency of the packets inter-arrival times distribution on the objects sizes. The authors propose to model this relation analytically, by finding suitable distributions for different object sizes or by involving simple (e.g., Markovian) TCP and/or network models as an input of the traffic generator. And this is exactly the same modelling effort made in `Swing` in order to provide for realistic and responsive traffic generation in network emulation environments. So, in the general case, the critique on the open-loop traffic generators stressed in [FlP01] can be applied also to `LiTGen`.

2.2.4. Traffic Generation at Data Link Layer Service Interfaces

KUTE

`KUTE` (a Kernel-based UDP Traffic Engine) is a generator of UDP traffic which is designed to achieve high performance over Gigabit Ethernet [KUTE,

ZKA05]. It is based on two Linux 2.6 kernel modules (the sender and the receiver) that operate directly on the network device driver bypassing the Linux kernel networking subsystem. The KUTE sender generates packets for a specified duration, computes the inter-packet gaps based on the specified sending rate (in packets per second), and uses polling of the CPU cycle counter in order to wait for the sending time of packets. The following parameters can be specified: source and destination IP address, source and destination ports, packet rate, packet length, duration of the flow, packet payload, Time To Live (TTL), Type of Service (ToS), and whether the UDP checksum and IP identification field should be used. The sender can create up to four different flows concurrently. The flows may have different packet rates, but must have the same duration. The KUTE receiver creates a packet inter-arrival histogram that can be accessed via the Linux proc file system. Furthermore, when the module is unloaded, it outputs the necessary information to compute the mean and standard deviation of the distribution into the kernel log file.

It should be noted that KUTE is strongly restricted to the generation of UDP packets and their injection as Ethernet frames (at the data link layer) with specified inter-departure times. The tool achieves a maximum packet rate of 740.000 pps (for packet length of 64 byte and infinitesimal inter-arrival times) and is able to saturate the capacity of the Gigabit Ethernet link with packets of 256 byte length [BGPS05]. However, KUTE does not provide any further traffic modelling support and cannot compute the traffic statistics directly because the Linux kernel does not provide floating point arithmetic. The sender can not be controlled from user-space while it is running. Since the architecture of the tool is strictly related to the architecture of the kernel, it lacks of extensibility and cannot take advantage of the support of kernel-space extensible interfaces.

BRUTE

The Browny and RobUst Traffic Engine (BRUTE) presented in [BGPS05] is a user-space application running on Linux 2.4-6, that is able to accurately generate customizable IPv4 and IPv6 Ethernet traffic flows with very high data rates. BRUTE uses a script language in order to control the generated traffic pattern over time and can be further extended by means of additional traffic patterns (T-modules) implemented in C language. A parser is responsible for reading the user commands from the script files and storing them in an internal database. The traffic engine examines the database entries and instantiates the corresponding traffic handlers (micro-engines) defined

in the T-modules. The micro-engines are sequentially executed in order to generate the specified traffic. BRUTE is provided at [BRUTE] with a set of predefined traffic patterns (T-modules): CBR (constant bit rate), CIDT (constant inter-departure time), POISSON (exponential inter-departure time), PAB (Poisson Arrival of Burst), CBR-EXP/OFF-EXP (VoIP), RTCP SR (send-report message to measure RTT), and TRIMODAL (trimodal Ethernet frame size distribution).

BRUTE is reported to be able to achieve a maximum packet rate of 650.000 pps for packets of 64 byte length (which corresponds to a data rate of approximately 400 Mbit/s) [BGPS05]. So, its performance is comparable to the performance of KUTE (which is a kernel-based solution) while providing a high level of accuracy and precision of the packet inter-arrival times. Further, as reported in [APF08a], BRUTE can achieve higher values of throughput (up to 1.090.000 pps) with 64 byte packets only in intermittent bursts.

BRUNO

A possible solution to improve the performance and accuracy of the traffic generation process may be the use of flexible hardware platforms and cooperative software/hardware design. For example, the Intel IXP2400 Network Processor (NP) is a multi-core processor dedicated to packet processing [IXP2400] which has been used in traffic generators presented, e.g., in [BBCR06] (Pktgen) and [APF08a, APF08b] (BRUNO).

BRUNO (BRUte on Network prOcessor) available at [BRUNO] is based on a modified version of BRUTE which has been designed to run on the PC that hosts the Network Processor card and is responsible for the computing of the packet lengths and departure times according to the specified traffic model [APF08a]. The host PC writes the computed data into the memory shared with the packet processing units of the Network Processor (so-called micro-engines) which are responsible for the generation of real packets and sending them with the proper timeliness.

In this way, BRUNO retains the high flexibility of BRUTE while improving its performance characteristics in terms of the achievable packet and data rate. Furthermore, a feedback mechanism and a time correction scheme have been introduced in BRUNO in order to improve the system precision and accuracy in reproduction of packet departure times determined according to the traffic model. The Traffic Generator micro-engines report in a feedback ring the actual packet departure times, which are then used by the Load Balancer micro-engine for an adaptive time modification. The experiments

carried out in [APF08a] prove the effectiveness of this approach in reducing the mean inter-departure time error. Further, results of experimental tests have shown the ability of `BRUNO` to generate 64 byte packets with a short term packet rate (calculated as a mean over intervals of 0.10 s) of up to 1.488.000 pps (which means that the tool can saturate the capacity of a Gigabit Ethernet link already with the 64 byte packets).

Tools for packet-trace replay

Packet trace replay can be performed in a flexible manner in software at different network interfaces and a series of corresponding tools have been proposed in the network research community. The most prominent example is probably `Tcpreplay`, which has originally been designed for a classic packet-level trace replay of TCP traffic in order to inject malicious traffic patterns into IDPSs. In the meantime, `Tcpreplay` has strongly evolved and has obtained capabilities to replay traffic patterns to Web servers. Currently, `Tcpreplay` is a suite of free (GPLv3 licensed) open source utilities for UNIX and Win32 operating systems for editing and replaying previously captured network traffic in `libpcap` format (PCAP) with the aim to test a variety of network devices (such as switches, routers, firewalls, and IDPSs). In particular, `Tcpreplay` suite includes a set of PCAP file editors and network playback utilities, e.g.:

tcpprep: is a multi-pass PCAP file pre-processor that allows the researcher to classify the captured packets as originating from the client or server and split them into different output files to be used by `tcprewrite` and `tcpreplay`.

tcprewrite: is a PCAP file editor which allows to modify and rewrite the Ethernet, IP, and TCP/UDP packet headers.

tcpreplay: is the tool to replay PCAP files at arbitrary speeds onto the network with an option to replay with random IP addresses. `tcpreplay` supports both single and dual NIC modes for testing both sniffing and in-line devices.

tcpreplay-edit: extends the `tcpreplay` tool by a large set of functions to modify the packets on the fly during the replay.

tcpliveplay: provides the replay function for TCP traffic stored in a PCAP file with the possibility to adapt the rate to the responses of a concrete

remote TCP server. The utility can be used to conduct tests at the application layer.

As of the current version 4.0, `Tcpreplay` has been enhanced to support the corresponding functions for testing and tuning IP Flow/NetFlow hardware. The accuracy and the performance of the playback tool has been significantly improved by introducing support for the modified `netmap` device drivers for 10 Gigabit Ethernet NICs [netmap].

Similar tools have been proposed for high performance packet replay, e.g., `TCPopera` [HoW06] and `TCPivo` [FGB03]. `TCPopera` tries to accomplish two primary goals: (1) replaying TCP connections in a stateful manner, and (2) supporting traffic models for trace manipulation. To achieve these goals, `TCPopera` emulates a TCP protocol stack and replays trace records interactively in terms of TCP connection-level and IP flow-level parameters. The second tool, `TCPivo`, employs novel mechanisms for managing trace files and accurate low-overhead timers in order to achieve high throughput and accuracy. In addition, through the use of low-latency kernel patches and priority scheduling, `TCPivo` can be made highly resilient to background system load. Using these mechanisms, the tool is able to support packet replay and achieve sufficient packet and data rate, e.g., for the OC-3 links. Both tools `TCPopera` and `TCPivo` have been used in test environments for Intrusion Detection and Prevention Systems (IDPSs). However, these projects seem not to be supported any more for quite a long period of time.

High-performance traffic generation for 10 Gigabit Ethernet

With the rapidly increasing capacities of the links deployed on production networks (e.g., 10 Gigabit Ethernet links are becoming common) the high-performance traffic generation becomes very important. A possible direction of research to improve the performance of traffic generation is the use of parallelism, which is increasingly provided by modern commodity hardware[3]. One can expect that traffic generators can efficiently generate packets on multi-core systems if they are able to properly exploit such architectures. Further, the design of currently available 10 Gigabit Ethernet NICs (such as those based on the Intel 82599 controller) is already logically partitioned into several independent receive and transmit (RX/TX) hardware queues, so that multiple cores can therefore receive and transmit packets in parallel. From

[3]For example, the recently released processor family Intel Xeon Processors E7 v2 can have up to 15 cores providing up to 30 logical processors to the applications by means of the hyper-threading technology.

the operating system point of view, it is possible to simultaneously poll and send packets per queue thus maximizing the overall throughput. Therefore, it is important that the operating system makes these queues available to applications and does not force the multi-threaded applications to serialize their operations when all threads need to access the same Ethernet device [RDC11].

An other crucial factor for the performance of a traffic generator on multi-core systems is a capacity of the socket which is used for sending packets towards the NIC device driver. Most of the software-based generators presented in this section use either the `PF_PACKET` socket family (on Linux distributions) or `AF_INET` socket family (on Windows systems). However, these socket families have been designed in a single-core architecture and show a number of strong limitations and bottlenecks when being used on multi-core systems (cf. [BPGP12]):

- The `PF_PACKET` socket does not allow to select a specific hardware queue for transmission when used on top of multi-queue NICs. This results in thread serialisation when multiple threads send packets on the same device, no matter if they share the same socket or not.
- It is based on a per-packet `send()` system call which represents a remarkable overhead. The system calls versions provided for batch transmission (e.g. `sendmmsg()` on Linux and a version of `send()` on Windows) are hardly useful for a traffic generator, because the inter-departure times of packets in the batch cannot be specified in these calls.
- The packet payload must be transferred into the kernel, which induces a higher overhead than a normal `memcpy` operation performed to a memory-mapped region in the user-space.
- Further, packets are not immediately directed to the NIC device driver but pass through a series of mechanisms (like, e.g., registered packet filters, traffic control modules, etc.) which induce additional overhead. A single socket cannot be used exclusively for packet transmission, so that a severe performance penalty may result in a multi-core scenario when several sockets are used for parallel transmission.

A series of different solutions have been proposed to improve the efficiency of Linux networking subsystem in respect to the above-mentioned bottlenecks in the `PF_PACKET` socket. For example, `netmap` [netmap] integrates in the same interface a number of heavily modified device drivers mapping the

NIC transmit and receive buffers directly into the user space. A version of this driver has been integrated into the new `PF_RING DNA` framework [ntop1] and allows to saturate the capacity of a 10 Gigabit Ethernet link with the smallest possible (64 byte long) packets both in generation and in transmission, when simple test programs are used, e.g., for packet generation. However, even when the bottlenecks in the packet transmission are removed by using such a properly modified driver, a non multi-core aware design of the packet generation application itself may strongly limit the performance of the overall system (as it has been reported, e.g., in [ntop2] for the `Ostinato` packet traffic generator available at [Sri16] used in combination with `PF_RING DNA`).

Based on the research in [APF08a, APF08b], a modular architecture of an Ethernet traffic generator using the integrated co-design of both kernel-space and user-space components is presented in [BPGP12]. A set of *traffic engines* (which are completely user-space threads) is responsible for the generation of a global ordered stream of packets according to a set of independent traffic models and dispatching the generated packets across a set of *packet transmitters*. A set of parallel packet transmitters is in charge of actually sending the generated packets to the NIC (using polling in order to precisely meet the specified inter-departure times). Each transmitter is implemented using a novel socket type `PF_DIRECT` proposed by the authors (see below) and an active context implemented as a kernel-space thread, which can be assigned to a specific hardware queue on the NIC.

In order to avoid the above-mentioned limitations of the Linux networking subsystem, the authors in [BPGP12] designed a novel socket type `PF_DIRECT` which consists of 1) a memory-mapped single-producer-single-consumer queue for payload and meta-data (which avoids the overhead of a system call to copy data into the kernel-space and provides a wait-free mechanism for data sharing), 2) a pool of pre-allocated socket buffers (which allows to keep using the mandatory `sk_buff` socket structures in order to work with non-modified NIC device drivers), and 3) a direct interface to a hardware queue (in order to avoid the overhead induced by the optional traffic control or packet filter modules).

The results of experimental tests[4] revealed that the proposed traffic generator is able to saturate the capacity of a 10 Gigabit Ethernet link

[4]The experiments have been conducted on a machine using 6 cores with an Intel X5650 Xeon CPU (2.66 GHz clock, 12 MB cache), 12 GB DDR3 RAM, and an Intel E10G42BT NIC with the 82599 controller on board, running Linux with the 3.0.1 kernel and the *ixgbe* 3.4.24 NICs driver. The hyperthreading has been enabled, so that the experiments have been carried out on 12 virtual cores.

with 128 byte long packets and can achieve a packet rate of 13.000.000 pps with minimum size (64 byte) packets, which is very close to the theoretical maximum of 14.880.000 pps for the packet rate achievable on a 10 Gigabit Ethernet link.

The authors in [BPGP12] claim that the proposed modular architecture allows to transparently use parallelism for generating traffic according to arbitrary traffic models, which must conform to a simple interface and can be added by the user through a factory pattern implemented in C++. However, the presented experimental tests have been conducted with the simplistic models for CBR and Poisson Ethernet traffic only. Further, the closed-loop traffic generation required, e.g, to emulate TCP traffic is not supported by the tool in general.

A flexible high-speed packet generator `MoonGen` has been recently proposed for the generation of Ethernet packet traffic on 10 Gigabit Ethernet links [EGRWC15]. It can saturate the capacity of a 10 Gigabit Ethernet link with minimum-sized packets while using only a single CPU core by running on top of the packet processing framework Data Plane Development Kit (DPDK) [DPDK] and commodity hardware. The authors note, that `MoonGen` utilizes several hardware features of commodity NICs that have not been used in Ethernet packet generators previously. In particular, it uses hardware time-stamping capabilities of the NIC in order to perform measurements of latency with sub-microsecond precision and accuracy. Furthermore, the authors proposed a novel method to control the inter-packet gaps (between the Ethernet packets) in software in order to mitigate the timing issues arising with the software-based packet generator.

From the point of view of the author of this thesis, one can legitimately question how realistic the resulting traffic will be in respect of the variety of existing network applications which are currently using and probably will continue to use the original sockets (and not their versions optimized for performance in the proposed manner) on Linux and Windows platforms. We should note, that a significant drawback of the approach followed in [BPGP12, EGRWC15] is that it allows the traffic generating application to take direct control of the network hardware removing all software layers and making the application extremely vulnerable in case of a crash or malicious attack. This can hardly be allowed for regular network applications and services in particular due to the reasons of network security and stability. Therefore, the architecture proposed, e.g., in [BPGP12] is expected to remain a very specific solution for open-loop Ethernet traffic generation for performance tests in selected 10 Gigabit Ethernet scenarios.

Finally, we can conclude that the architecture of the workload generator to

be developed in this thesis should, wherever possible, make use of parallelism in the workload generation process in order to be able to exploit the potentials of current state-of-the-art multi-core system platforms equipped with multi-queue NICs used in combination with the appropriate multi-core aware device drivers and the corresponding network sockets software. We note, however, that the improvement of the networking subsystem of the underlying operating system, development of novel types of sockets or specialized NIC drivers (e.g., optimized for performance) is definitely outside of the scope of this thesis.

2.2.5. Workload Tests in Research and Industry

After the presentation of a comprehensive list of different workload and traffic generators in previous sections, we should address a remaining question how these tools can be actually used for workload tests in networking research experiments. Load testing can be generally defined as the simulation of multiple users using the observed service or application at the same time and working with it concurrently. According to this rather general definition the load testing may comprise several different types of testing. According to the testing objectives and procedure, the following types of testing can be identified.

Performance testing: In this type of tests, the workload is increased gradually by adding more and more virtual users to the test while the performance parameters of the System Under Test (SUT), e.g., throughput, response time, error rate, etc., are monitored at any test phase.

Capacity testing: This type of test is concerned with one of the most common questions in load testing: how many concurrent users the service or application can handle while maintaining an acceptable response time and error rate? Virtual users are added gradually to the test, but in this case the values of the performance parameters are known in advance and the experimenter just needs to check that the expected target values are really achieved. Performance or capacity tests help to reveal potential bottlenecks in the observed service or application. For example, a Web application can consist of several modules used to process requests. If one of them has a technical limitation, it limits the performance of the whole system.

Stress testing: When the load goes beyond the capacity limit of the SUT, the observed service or application starts responding very slowly and can even produce errors. The main purposes of stress testing are: a) to find

the capacity limit, b) to check that when the capacity limit is reached, the system handles the stress situation correctly, i.e. it produces graceful overload notifications and does not crash, c) when the load is reduced back to regular level, the system returns to normal operation retaining the performance characteristics.

Volume testing: During a volume test, the experimenter tries to maximize the amount of processed data and/or the complexity of each transaction, operation, or request. For example, for testing the file upload facility of a Web application, the experimenter should use the largest files available. And, in order to test the application's search engine functions, he should try to produce the longest search results possible.

Endurance testing: This type of testing is used to check that the system can stand the load for a long time or a large number of transactions or requests. It usually reveals various types of resource allocation problems. If a small memory leak is present, this may not be evident on a quick test, but will influence the performance after a long time. For endurance testing it is recommended to use changing periodic load to provoke resource reallocation.

Regression testing: Integration of the load testing as a part of the regular development process by creating regression load tests and applying them to every new version of the application or service being designed.

Considering the above-mentioned types of load testing with their partially different testing goals, it will be a very interesting and challenging task to elaborate a unified method for load specification and generation which would allow to perform such different types of tests in one single coherent approach.

2.3. A Unified Approach to Workload Modelling and Generation in Computer Networks

The wide range of complex tasks to be fulfilled in a computer network forces one to structure the communication software and to use layered architectures for the horizontal layering of different network functions. Such architectures define the functionality of the particular layers, the interaction principles between them, and specify the concrete types of services provided by every layer at the corresponding service interface(s) [Kön12].

So, for the task of workload modelling and generation, it is very important to choose the target service interface, at which the workload is to be considered. The choice of the service interface, in turn, determines the possible types of requests submitted by the service user(s) (comprising the environment E) to the service providing components (representing the system S) at the target service interface (IF).

In this dissertation, we will use the definition of workload for computer and communications systems proposed by Wolfinger in [WoK90].

Definition (Workload). The workload $L = L(E, S, IF, T)$ denotes the total sequence of requests which is offered by an environment E to a service system S at a well-defined interface IF during the time interval T.

It should be emphasized, that the workload definition given above is strongly oriented on the target service interface IF for the modelling and generation of workload. Therefore, an appropriate approach for modelling and generation of workloads based on this interface-oriented definition must provide a means for the characterization of 1) the arrival process of requests at IF during T, and 2) the resource requirements and other relevant attributes of the individual requests. Such characterization must be as precise and as realistic as it is necessary for the specific research purpose and desired spectrum of use of the workload model. Furthermore, considering the discussion of various workload and traffic generators in networking research along with different possible types of load testing presented in Sec. 2.2, a unified approach for workload modelling and generation in this thesis should meet at least the following basic requirements:

Support for different levels of abstraction: it should be possible to specify workload models using different levels of abstraction, e.g., application-level, flow-level, or packet-level models, cf. Sec. 2.2).

Support for different levels of detail: it should be easily possible to refine or coarsen the specification of the relevant workload characteristics if required in a concrete modelling study. The choice of the appropriate level of detail refers to both inter-arrival times and attributes of requests.

Measurement-based workload modelling: directly use measurements regarding arrival process as well as types and attribute values of requests as observed in measurement studies tracing workload and traffic generation from real network applications and services. In particular, the approach should be able to combine both trace-based and analytical model-based techniques (cf. Sec. 2.2) in order to achieve a high degree of flexibility in the specification of different model entities.

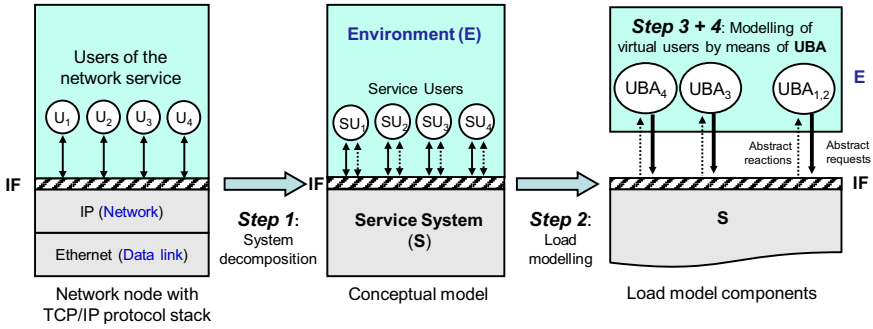

Figure 2.1.: Unified approach to workload modelling illustrated for the case of modelling at the IPv4 network service interface (own Fig.).

Support for different service interfaces: it should be possible to generate the workload as a sequence of requests at a concrete real service interface according to the specifications in the underlying workload model. For example, Web traffic can be generated as a sequence of HTTP requests and the corresponding HTTP responses at the (application layer) HTTP service interface according to an application-level model of Web traffic.

Consider the current system state: wherever required, the approach should provide support also for the "closed-loop" workload models which reflect the dependency of the workload generation process from the current network state and are, therefore, responsive to changing network conditions [FlP01]. We note that it can be necessary to consider the current state of the system S in the workload model in case when the internal behaviour of the service provider (affected, e.g., by the changing network conditions) significantly influences the interactions of the user with the system at the service interface.

A generalized approach for workload modelling and description has been introduced by Wolfinger in [WoK90]. According to the proposed approach, the procedure of constructing a workload model can be accomplished systematically based on the four main steps which are illustrated in Fig. 2.1 and are described in the following. The first of these steps is motivated by the fact that workload modelling necessarily requires a well-defined interface at which the workload is offered (cf. the definition of *workload*).

Step 1: Decomposition of system and environment. At the beginning, the modeller (which may be a single researcher, experimenter, test engineer or

a whole quality assurance team) has to decide where to place the boundary line between what he considers as system S on the one hand and environment E on the other hand. This decomposition directly provides the interface IF between S and E which can, e.g., consist of one particular local interface IF_l or correspond to the union of several (also geographically distributed) interfaces IF_1, IF_2, ..., IF_k in the network. Further, the modeller identifies the set of load generating users which are relevant for the given modelling task and, therefore, belong to the load generating environment E. These users can correspond, e.g., to human end users or to some load generating applications or system processes.

Step 2: Choice of abstraction level in modelling. At this step the modeller has to decide which requests (passed from E to S) and which reactions (produced by S and observable by E) have to be taken into account in the load model. For this reason, relevant users are observed and analysed in respect to the requests they are generating. Depending on the objectives of the current modelling task the modeller can decide, e.g., to include only the typical requests from users in the load model. These typical requests are further characterized by the unique request type (so that disjoint request classes are to be built by the modeller for this reason) and the set of associated type-specific attributes (with predefined domains for attribute values). The possible system reactions are to be handled in the same manner (in respect of reaction types and attributes).

Step 3: Analysis and description of possible interactions. At this step, the modeller is concerned with the specification of possible sequences of interactions between E and S at the chosen interface IF. This is quite similar to some service specification for a communication service, which also specifies the sequences of service primitives which are possible over time. In order to specify the interactions between the service users and the system S, real service users are mapped to virtual users which are represented by the corresponding components in the modelling domain. Finally, the behaviour of virtual users is described by means of an appropriate formal specification method (e.g., by means of UBAs introduced in Chapter 3). For the choice of an appropriate specification method it is very important that the model description allows its execution without lot of effort (e.g., by means of a corresponding load generator).

Step 4: Description of actual interactions between E and S. In order to describe the actual interactions between the environment E and the system S, the following two tasks have to be solved:

- For each virtual user $U \in E$ in the environment the sequence of requests $L(U, S, IF, T)$ which U generates during the time interval T and passes to S, has to be described. For a given interface IF a sequence of requests generated by the user U can be represented by a finite vector of (*time*, *request*)-tuples $L(U, S, IF, T) = ((t_1,r_1), (t_2,r_2), \ldots, (t_k, r_k))$ for some $k \in \mathbb{N}$. In each (*time*, *request*)-tuple (t_i, r_i) the value of $t_i \in T$ denotes the generation time of the request r_i at IF. The request generation times t_i are assumed to be real values ($t_i \in \mathbb{R}$) and their sequence (t_1, t_2, ..., t_k) characterizing the arrival process of requests in the described request sequence is assumed to be non-decreasing (thus, $t_i \leq t_j$ for any $i < j, i \in \mathbb{N}, j \in \mathbb{N}, 1 \leq i \leq k, 1 \leq j \leq k$).
- The total workload $L(E, S, IF, T)$ offered from the environment E to the system S is described by means of the *superposition* of the sequences of requests generated by the virtual users U being part of the environment E. The superposition of request sequences may be specified in different ways, depending on the interface IF chosen for workload modelling. For example, the requests from different users can be arranged according to a chosen service discipline (e.g., First-Come, First-Served (FCFS)) before they are handed over to the system S.

At the end of step 4, the experimenter should be able to generate the total workload L for an experiment of the given observation time interval T. To accomplish this task, each of the users U being part of the environment E can be replaced by an individual load generator creating the specified sequence of requests $L(U, S, IF, T)$.

We should emphasize, that the target service interface IF for workload modelling must be chosen by the experimenter strongly according to the objectives of the specific experiment or the particular study to be carried out. For example, the experimenter would choose the target interface presumably to the IP service interface in order to evaluate the performance of IP forwarding functions in a router. But the experimenter would rather decide to select the HTTP service interface as a target interface for workload modelling in order to estimate the mean response time of a Web server under different server loads. Further, in the case study presented in Chapter 11 of this thesis, values of different QoS metrics for RTSP video streaming in Wireless Local Area Networks (WLANs) have been obtained while the reliable TCP transport service has been used for the transmission of the RTP video frames. Therefore, it was straightforward to choose the TCP transport service interface in order to generate background traffic (represented by additional TCP traffic sources) in the experimental WLAN.

http://www.springer.com/978-3-658-19101-6

Load Modelling and Generation in IP-based Networks
A Unified Approach and Tool Support
Kolesnikov, A.
2017, XXI, 316 p. 58 illus., 11 illus. in color., Softcover
ISBN: 978-3-658-19101-6