Analyzing TCP Flow Arrivals

D. Rossi, L. Muscariello, M. Mellia
Dipartimento di Elettronica
Politecnico di Torino

Abstract

In this paper we study the TCP flow arrival process, starting from the aggregated measurement at the TCP flow level taken from our campus network. After introducing the tools used to collect and process TCP flow level statistics, we analyze the statistical properties of the TCP flow inter-arrival process.

We create different traffic aggregates by splitting the original trace, such that i) each traffic aggregate has, bytewise, the same amount of traffic, and ii) it is constituted by all the TCP flows with the same source/destination IP addresses (i.e., belonging to the same traffic relation). In addition, the splitting algorithm packs the largest traffic relations in the first traffic aggregates; therefore, subsequently generated aggregates are constituted by an increasing number of smaller traffic relations. This induces a divisions of TCP-elephants and TCP-mice into different traffic aggregates.

The statistical characteristics of each aggregates are presented, showing that the TCP flow arrival process exhibits long range dependencies which tends to vanish on traffic aggregates composed by many traffic relations made of TCP-mice mainly.

I. Introduction

Since the pioneering work of Danzig [1], [2], [3], and Paxons [4], [5] the interest in data collection, measurement and analysis to characterize either the network or the users behavior increased steadily, also because it was clear from the very beginning that “measuring” the Internet was not an easy job. The lack of simple, yet satisfactory model like the traditional Erlang teletraffic theory for the circuit-switched networks, still pose this research field as a central topic of the current research community. Moreover, the well known Long Range Dependency (LRD) behavior shown by the Internet traffic makes traffic measuring and modeling even more interesting. Indeed, after the two seminal papers [5], [6], in which authors showed that traffic traces captured on both LANs and WANs exhibit LRD properties, many works focused on studying the behavior of data traffic in packet networks. This, with the intent of both trying to find a physical explanation of the properties displayed by the traffic, and to find accurate stochastic processes that can be used for the traffic description in analytical models.

Considering the design of the Internet, it is possible to devise three different layers at which study Internet traffic: Application, Transport and Network, to which user sessions, TCP or UDP flows, and IP packets respectively correspond.

Indeed, a simple “click” on a web link, causes the generation of a request at the application level (i.e., an HTTP request), which is translated into many transport level connections (TCP flows); each connection, then, generates a sequence of data messages that are transported by the network (IP packets). In this paper, we concentrate our attention to the flow level, and to the TCP flow level in particular, given that the majority of the traffic is today transported using the TCP protocol. The motivation behind this choice is that while it was shown (see for example [5], [7]) that the arrival processes of both packets and flows exhibit LRD properties, a lot of researchers concentrated their attention to the packet level, while the flow level traffic characteristics are relatively less studied. Moreover, even if the packet level is of great interest to support router design, e.g., for buffer dimensioning, the study of the TCP flow level is becoming more and more important, since the flow arrival process is of direct role in the dimensioning processes of web

This work was supported by the Italian Ministry for Education and Scientific Research through the TANGO Project.

servers and proxies. Another strong motivation to support the study of flow arrival process is represented by the increasing diffusion of network apparatuses which operates at the flow level, e.g. Network Address Translators or Load Balancer; indeed, their design and scalability mainly depend on the number of flows they have to keep track to perform packet manipulations between incoming and outgoing data.

Going back at the packet level, the prevailing justification to the LRD presence at this level is supposed to be the heavy tailed distribution of files size [8]: the presence of long-lived flows, called in the literature "elephants", induces correlation to the packet level traffic, even if the majority of the traffic is build by short-lived flows, or "mice". The question we try to answer in this paper is weather the presence of mice and elephants has an influence to the LRD characteristics at the flow level as well. To face this topic, we collected several days of live traffic from our campus network at the Politecnico di Torino, which consist of more than 7000 hosts, the majority of which are clients. Instead of considering the packet level trace, we performed a live collection of data directly at the TCP flow level, using `Tstat` [9], [10], a tool able to keep track of single TCP flows by looking at both the data and acknowledgment segments. The flow level trace was then post-processed by `DiaNa` [11], a novel tool which allowed to easily derive several simple as well as very complex measurement indexes in a very efficient way. Both tools are under development and made available to the research community as open source.

To gauge the impact of elephants and mice on the TCP flow arrival process, we follow an approach similar to [12], [13], that creates a number of artificial scenarios, deriving each of them from the original trace into a number of sub-traces, mimicking the splitting/aggregation process that traffic aggregates experience following different paths inside the network. We then study the statistical properties of the flow arrival process of different sub-traces, showing that the LRD tends to vanish on traffic aggregates composed mostly of TCP-mice.

The rest of the paper is organized as follows. Sec. II provides a survey of related works and results; problem definition and input data analysis is the object of Sec. III, in which the adopted algorithm to derive traffic aggregates will be briefly described highlighting its features. The result of the analysis on traffic aggregates is presented in Sec. IV, and conclusions will be drawn in Sec. V. Finally, in appendix Sec. we briefly describe the novel software tool `DiaNa`.

II. Related Works

In the last years a lot of effort has been spent to the traffic analysis issue in the Internet, at both IP and TCP level. It is well known IP arrival process is characterized by long-range dependence and how important is to consider this property for network planning, buffer dimensioning in primis. Long-range dependence study in the network domain has been pioneered by many works as [6], [5], [8], [14], all agreeing on the possible causes of LRD of IP traffic. Indeed, they identified heavy-tailed file size distribution (and, consequently, TCP flows size) as the main cause of long-term correlation at IP level, that keeps true for WAN and LAN traffic.

IP traffic is also characterized by more complicated scaling phenomena, which will not be our issues, that have been well discussed in [13], where authors consider also small-scales phenomena, which have less direct impacts on engineering problems. Authors in [13] used an interesting manipulation of data at TCP level, which allowed to study the relation between the IP and TCP scaling behavior both at small and large scales. The gained results also confirmed that IP traffic LRD properties are partly inherited as consequence of TCP level properties (e.g., the distribution of the connection duration and flow size), while others scaling properties seem to depend on packets arrivals within flows.

The statistical analysis of real measured traffic, due to the significant amount of collected data and research efforts, gave new impulses to traffic modeling as well. Here, we briefly summarize the different approaches followed in the last years, where a number of attempts were made to develop models for LRD data traffic at packet level mainly. Among the different approaches, Fractional Brownian Motion

(FBM) received a lot of attentions thanks to pioneering work [15]; while providing good approximations, the FBM models, however, are not able to capture both the short and long term correlation structure of network traffic.

Therefore, the poor FBM scaling properties drove many research efforts toward Multifractal models, whose attractive is due to their scale-invariance properties. [16], [17] mainly focus on the physical explanation of network dynamics, showing multifractal proprieties for the first time. Indeed, also other works such as [18], [19] suggest multifractal models as possibly being the best fit to measured data. For example, in [18], authors tried to explain the scaling phenomena, from small to large scales, from direct inference of the network dynamics. In particular, they identify in the heavy-tailed nature of the TCP flows number per Web session the cause of LRD effects clearly visible at large scales in the TCP traffic. This effect, analogous to an $M/G/\infty$ effect with infinite variance service time distribution, was pointed out in [19] and already known in literature. Finally, [20], [21] have been more interested on measure-based traffic modeling issue, identifying a multifractal class of processes able to describe the multiscaling properties of TCP/IP traffic, from small to large scales; however, the real impact of the for the small traffic scales properties still remain questionable.

All these traffic characterization works deviate considerably from classical Markovian models which continue to be widely used for performance evaluation purposes with good results [22], [23], [24], [25]; in all the above works for example, the Markov Modulated Poisson Process (MMPP), is considered as one of the best Markov process that emulates traffic behavior. However, in [24], [25] authors also point out that MMPP do not bring long-term correlation; authors therefore, define the local Hurst parameter using an approximated LRD definition valid on a limited range of time scales.

Finally, another approach to model Internet traffic involves the emulation of the real hierarchical nature of network dynamics, e.g., considering users sessions, TCP flows and IP packets; e.g., in [26], each of the model's components was fitted to empirical ddp, such as the distribution of both TCP flows and web pages size, and the arrival distribution of page and flows.

All of these different approaches reach similar conclusion using different techniques. The common point of view has always been to take into account the real traffic behavior, in order to be able to either i) use more reasonable tools for network planning or ii) explain the links between causes and effects of network traffic phenomena. The analysis brought network community to awareness on the real traffic which has to be taken into account for a correct performance evaluation of the systems.

In this paper, we are interested on the large scale mainly as explained in the following section, where we will describe a new criterion to classify traffic starting from the TCP level and not considering the packet level at all. We define rules to aggregate TCP flows and define an high level entity with good engineering properties to split homogeneous traffic. Studying then the different traffic aggregates, we try to gain an insight on TCP flow interarrival behavior.

III. Problem Definition

In this section, we first introduce the different aggregation level and the notation that will be used in the remaining of the paper. We then describe the measuring setup and the input trace characteristics that are relevant to understand the significance of the presented results. Finally, we describe the splitting/aggregation criterion that will be used to derive different traffic aggregates, showing briefly its properties and its technical aspects.

A. Preliminary Definitions

When performing trace analysis, it is possible to focus the attention on different level of aggregations. Considering the Internet architecture, it is common to devise three different levels, i.e., IP-packet, TCP/UDP flow, and user session level. While the definition of the first two layers is commonly accepted, the user session one is more fuzzy, as it derives from the user behavior. In this paper we therefore decided to follow a different approach in the definition of aggregates. In particular, to mimic the splitting/aggregation process that data experience following different paths in the network, we define four level of aggregation, sketched in Fig. 1: IP packets, TCP flows, Traffic Relations (TR), and Traffic Aggregates (TA). Being interested into the Flow arrival process, we will neglect the packet level, and also the UDP traffic, because of its connectionless nature, and because it consists of a small portion of the current Internet traffic. Let

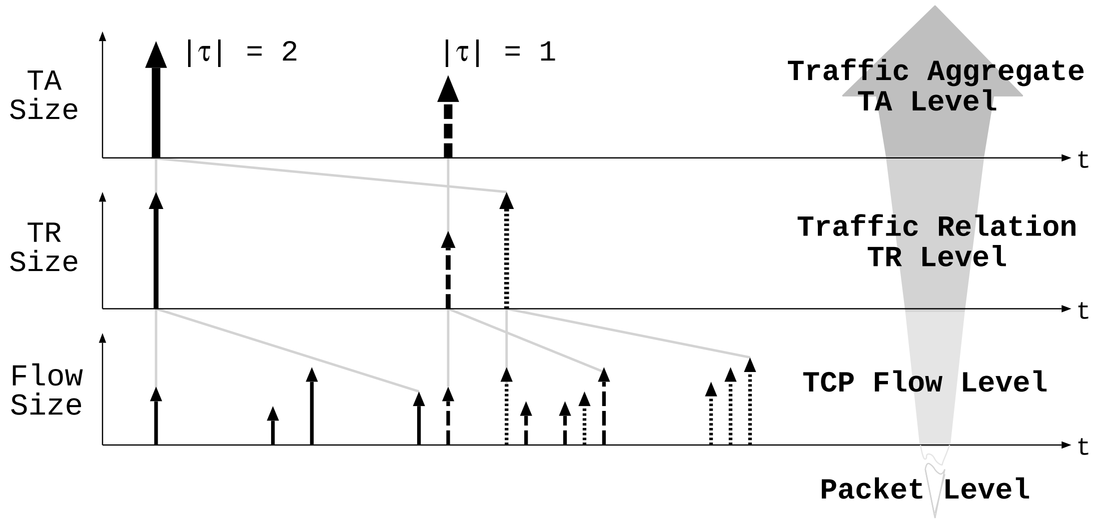

Fig. 1. Aggregation Level From Packet Level to TA Level

us first introduce the formal definition of the different aggregation levels considered in this paper, as well as the related notation:

- **TCP Flow Level**
  A single TCP connection[1] is constituted, as usual, by several packets exchanged between the same client c (i.e., the host that performed the *active* open) and the same server s (i.e., the host that performed the *passive* open), having besides the same (source,destination) TCP ports pair. We will consider only successfully opened TCP flows –i.e., whose three-way-handshake was successful– and we will consider the flow arrival time as the observation time of the first client SYN segment. We denote with $F_i(c,s)$ the bytewise size of the i-th TCP Flow tracked during the observation interval; the flow size considers the amount of bytes flowing from the server d toward the client s, which is usually the most relevant part of the connection.
- **Traffic Relation Level**
  A Traffic Relation (TR) $\mathcal{T}(c,s)$ aggregates all the $|\mathcal{T}(c,s)|$ TCP flows, 'having c as source and s as destination; we indicate its size, expressed in bytes, with $T_R(c,s) = \Sigma_{i=1}^{|\mathcal{T}(c,s)|} F_i(c,s)$. The intuition behind this aggregation criterion is that all the packets within TCP flows belonging to the same TR usually follow the same path(s) in the network, yielding then the same statistical properties on links along the path(s).
- **Traffic Aggregate Level**
  Considering that several TRs can cross the same links along their paths, we define a higher level of aggregation, which we call Traffic Aggregate (TA). The J-th TA is stated by τ_J and its bytewise size is $T_A(J) = \Sigma_{(c,s)\in\{\tau_J\}} T_R(c,s)$; the number of traffic relations belonging to a traffic aggregate is indicated with $|\tau_J|$.

[1]In this paper we use the term "flow" and "connection" interchangeably.

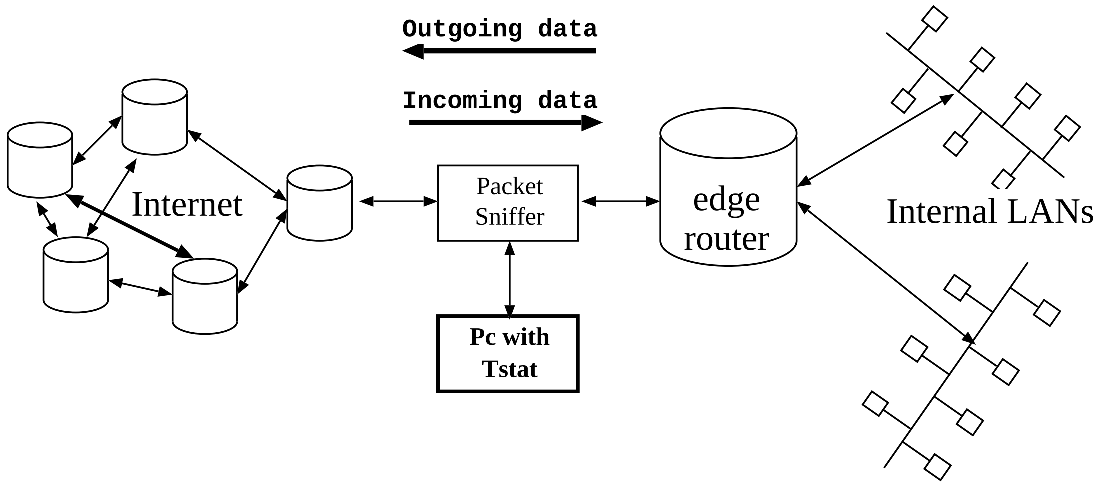

Fig. 2. The Measure Setup

In the following, rather than using the bytewise size of flows, TRs and TAs, we will will refer to their *weight*, that is, their size expressed in bytes normalized over the total amount of bytes observed in the whole trace $W = \Sigma_i \Sigma_{(c,s)} F_i(c,s)$; thus

$\hat{w}_i(c,s) = F_i(c,s)/W$	TCP Flow Level
$\hat{w}_{cs} = T_R(c,s)/W$	Traffic Relation Level
$\hat{w}_J = T_A(J)/W$	Traffic Aggregate Level

B. Input Data

The analysis was conduced over different traces collected in several days over our Institution ISP link during October 2002. Our campus network is built upon a large 100 Mbps Ethernet LAN, which collects traffic from more than 7,000 hosts. The LAN is then connected to the Internet by a single router, whose WAN link has a capacity of 28 Mbps[2].

We used `Tstat` to perform the live analysis of the incoming and outgoing packets sniffed on the router WAN link, as schemed in Fig. 2, obtaining several statistics at both packet and flow levels. Moreover, `Tstat` dumped a flow-level trace, which has then been split into several traces, each of which refers to a period of time equal to an entire day of real traffic. Besides, since our campus network is mainly populated by clients, we consider in this analysis only the flows originated by clients internal to our Institution LAN. Each trace has been then separately analyzed, preliminary eliminating the non-stationary time-interval (i.e., night/day effect) and considering then a busy-period from 8:00 to 18:00. Given that the qualitative results observed on several traces do not change, in this paper we present results derived from a single working day trace, whose properties are briefly reported in Tab. I. During the 10 hours of observation, 2,380 clients contacted about 36,000 different servers, generating more than 172,000 TRs, for a total of more then 2.19 million TCP flows, or 71.76 million packets, or nearly 80 GBytes of data.

In the considered mixture of traffic, TCP protocol represents the 94% of the total packets, which allows us to neglect the influence of the other protocols (e.g., UDP); considering the application services, TCP connections are mainly constituted by HTTP flows, representing the 86% of the total services and more than half of the totally exchanged bytes. More in detail, there are 10664 different identified TCP server ports, 99 of which are well-known: these account for 95% of the flows and for the 57% of the traffic volume.

[2]The data-link level is based on an AAL-5 ATM virtual circuit at 34 Mbps (OC1).

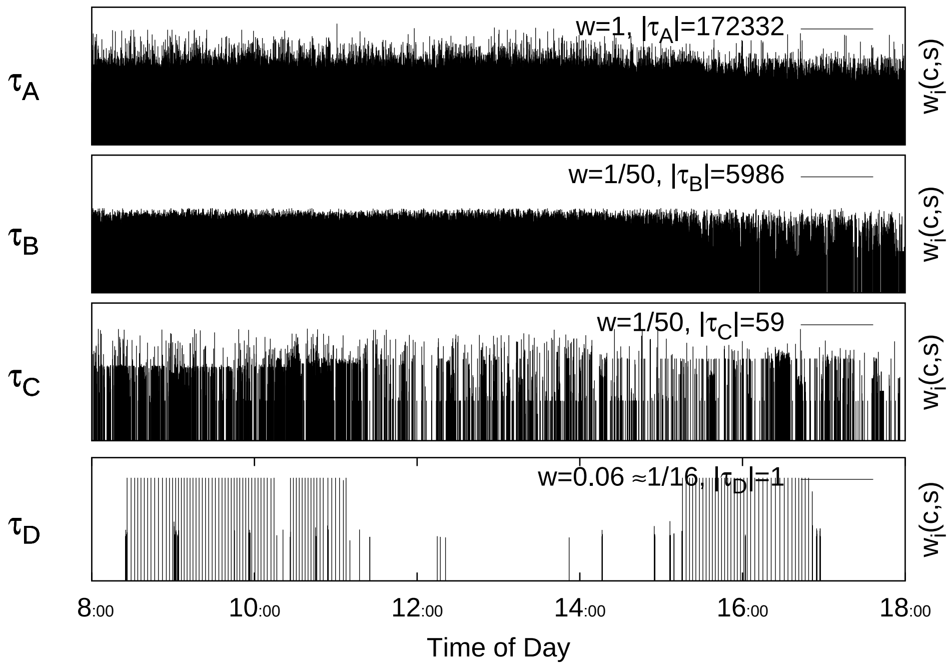

Fig. 3. Flow Size and Arrival Times for Different TAs

Considering the different traffic aggregation levels previously defined, Fig. 3 shows examples of the flow arrival time sequence. Each vertical line represents a single TCP flow, which started at the corresponding time instant of the x-axis, and whose weight $\hat{w}_i(c, s)$ is reported on the y-axis. The upper plot shows trace τ_A, whose $\hat{w}_A = 1$, and represents the largest possible TA, built considering all connections among all the possible source-destination pairs. Trace sub-portions τ_B and τ_C, while being constituted by a rather different number of TRs ($|\tau_B| = 5986$ and $|\tau_C| = 59$), have indeed the same weight $\hat{w}_B = \hat{w}_C = 1/50$. Observing Fig. 3, it can be gathered that τ_B aggregates a larger number of flows than τ_C; furthermore, weight of τ_B flows is smaller (i.e., TCP flows tend to be “mice”), while τ_C is build by a much smaller number of heavier (i.e., “elephants”) TCP flows. This intuition will be confirmed by the data analysis presented in Sec. IV. Finally, TCP flows shown in τ_D constitute a unique traffic relation; this TR is built by a small number of TCP flows, whose weight is very large, so that they amount to 1/16 of the total traffic.

To give the reader more details on the statistical properties of the different TRs, Fig. 4 shows the distribution of $\hat{w}_{cs}$ for all the TRs (using a lin/log plot). It can be noticed that, except for the largest and smallest TRs, the distribution can be approximated by a linear function, i.e., the amount of bytes exchanged by considering different client/server couples follows an exponential distribution. More interesting is instead the distribution of the TCP-flows number per traffic relation, shown in Fig. 5 using a log/log plot. Indeed, the almost linear shape of the ddp shows that it exhibits a heavy tail, which could be at the basis of the LRD properties in the TCP-flow arrival process. The parallel with the heavy-tailed ddp of the flow-size, which induces LRD properties to the packet level, is straightforward.

TABLE I
TRACE INFORMATIONS

Internal Clients	2,380	Flows Number	$2.19 \cdot 10^6$
External Servers	35,988	Packets Number	$71.76 \cdot 10^6$
Traffic Relations	172,574	Total Trace Size	79.9 GB

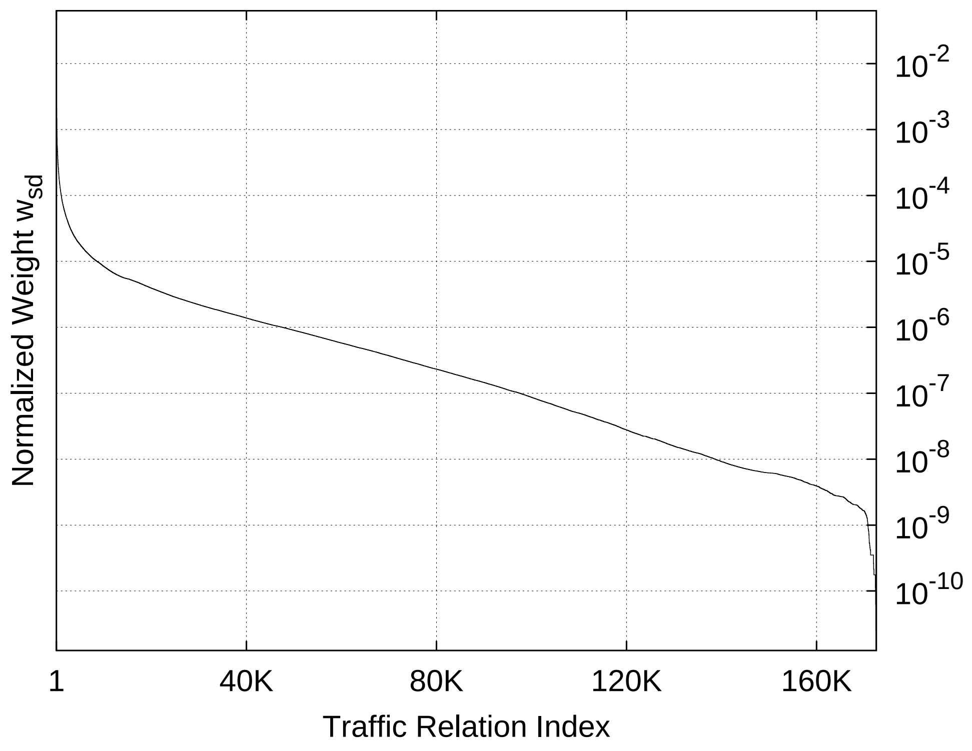

Fig. 4. TR Size Distribution

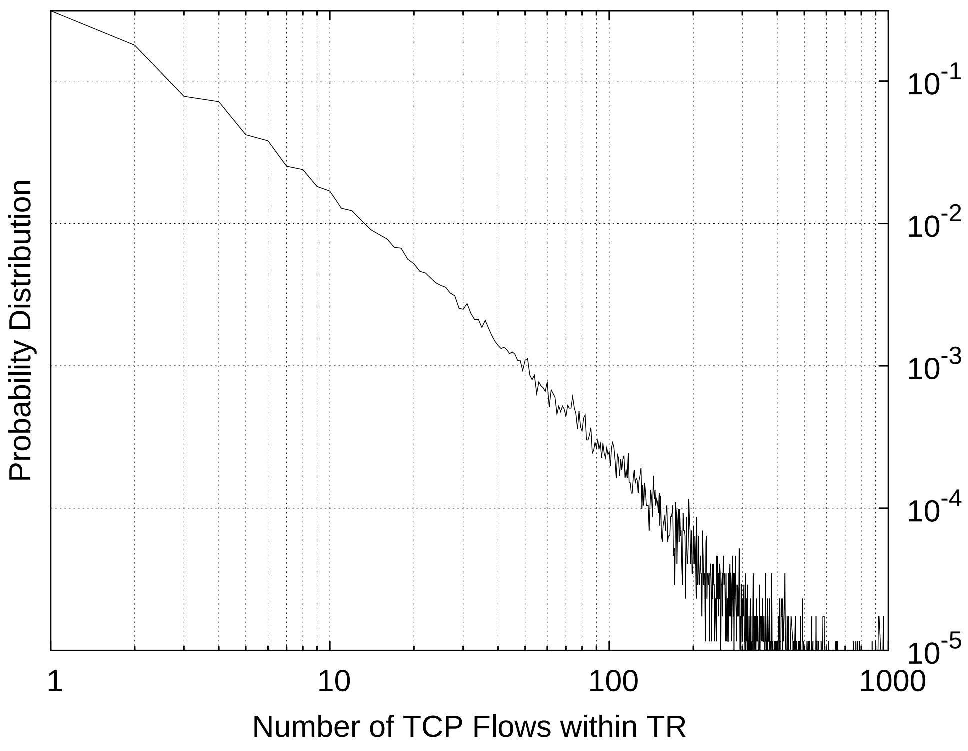

Fig. 5. TR Flow Number Distribution

C. Properties of the Aggregation Criterion

We designed the aggregation criterion in order to satisfy some properties that help the analysis and interpretation of the results.

The key-point is that the original trace is split into K different TA, such that i) each TA has, bytewise, the same amount of traffic, i.e., the K-th portion of the total traffic and ii) each TA aggregates together one or more TRs. Indeed, we considered TR aggregation a natural choice, since it preserve the characteristics of packet within TCP flows following the same network path, having therefore similar properties.

Being possible to find more than one solution to the previous problem, the splitting algorithm we implemented packs the largest TRs in the first TAs; besides, in virtue of the bytewise traffic constraint, subsequently generated aggregates are constituted by an increasing number of smaller TRs. Therefore the TAs, while composed by several TRs related to heterogeneous network paths, are ordered by the number $|\tau_J|$ of TRs constituting them.

To formalize the problem, and to introduce the notation that will be used also to present the results, let us define:

- Class K: the number of TA in which we split the trace;
- Slot J: a specific TA of class K, namely $\tau_J(K)$, $J \in [1, K]$;
- Weight $\hat{w}_J(K)$: the weight of slot J of class K;
- Target Weight $\hat{w}(K) = 1/K$: the ideal portion of the traffic that should be present in each TA at class K.

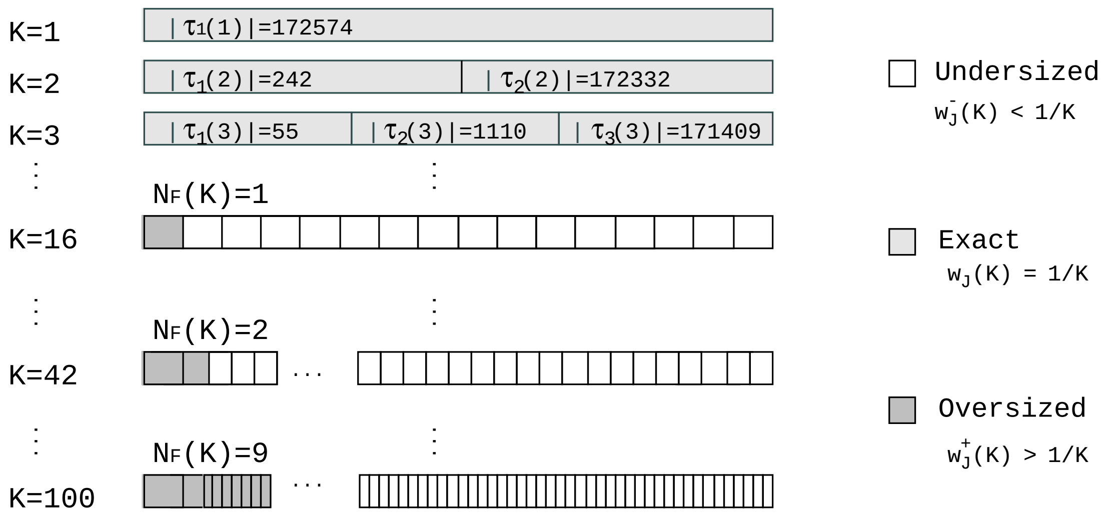

Fig. 6. Trace Partitioning: Algorithmic Behavior

Fig. 6 sketches the splitting procedure. When considering class $K = 1$ we have a single TA of weight $\hat{w}(K) = 1$, derived by aggregating all the TRs, which corresponds to the original trace. Considering $K = 2$, we have two TAs, namely $\tau_1(2)$ and $\tau_2(2)$; the former is build by 242 TRs, which account to $\hat{w}(K) = 1/2$ of relative traffic, while the latter contains all the remaining TRs. This procedure can be repeated for increasing values of K, until the weight of a single traffic relation becomes larger than the target weight. Being impossible to split a single TR into smaller aggregates, we are forced to consider TAs having a weight $\hat{w}^+_J(K) > \hat{w}(K)$.

The weight $\hat{w}(K)$ has therefore to be interpreted as an *ideal* target, in the sense that it is possible that one or more TRs will have a weight larger than $\hat{w}(K)$, as the number of slots grows. In such cases, there will be a $N(K)$ number of *fixed* slots, i.e., TAs constituted by a single TR, of weight $\hat{w}^+_i(K) > 1/K$; the remaining weight will be distributed over the $K - N(K)$ non-fixed slots; therefore the definition of

$\hat{w}_J(K)$ is:

$$\hat{w}_J(K) = \begin{cases} \hat{w}_J^+(K) > \dfrac{1}{K}, & {\scriptstyle 0<J\leq N_F(K)} \\ \dfrac{1-\sum_{i=1}^{N_F(K)} \hat{w}_i^+(K)}{K-N_F(K)} < \dfrac{1}{K}, & {\scriptstyle N_F(K)<J\leq K} \end{cases}$$

In the dataset considered in this paper, for example, the TR τ_D shown in Fig. 3 is the largest of the whole trace, having $\hat{w}_D = 0.062 > 1/16$. Therefore, from class $K = 16$ on, the slot J=1 will be always occupied by this aggregate, i.e., $\tau_1(K) = \tau_D, \forall K \geq 16$, as evidenced in Fig. 6.

D. Trace Partitioning Model and Algorithm

More formally, the problem can be conduced to a well known optimization problem $P||C_{max}$ of job scheduling over identical parallel machines [32], which is known to be strongly NP-hard. Traffi c relations TR are the jobs that have to be scheduled on a fi xed number K of machines (i.e., TA) minimizing the maximum completion time (i.e., the TA weight).

The previously introduced ideal target $\hat{w}_K = 1/K$ is the optimum solution in the case of *preemptive* scheduling. Since we preserve the TR identities, preemption is not allowed; however, it is straightforward that minimizing the maximum deviation of the completion time from $w(K)$ is equivalent to the objective function that minimize the maximum completion time.

We defi ne $\underline{R}$ as the $1 \times |\tau_1(1)|$ vector of the jobs length (i.e., TR weights $\hat{w}_i(c,s)$) and $\underline{A}$ as the $1 \times K$ vector of the machine completion times (i.e., TA weights $\hat{w}_J$). Stating with $\underline{M}$ the mapping matrix (i.e., $\underline{M}_{ij} = 1$ means that the i-th job is assigned to the j-th machine), and stating with $\underline{M}_i$ its i-th column, we have:

$$\begin{array}{l} min\ p \\ p \geq z_i, \quad \forall i \in [1,K] \subset \mathbb{N} \\ \text{s.t.} \begin{cases} \sum_j \underline{M}_{ij} = 1, & \forall i \\ z_i \geq \underline{M}_i \cdot \underline{R} - A_i, & \forall i \\ z_i \geq A_i - \underline{M}_i \cdot \underline{R}, & \forall i \end{cases} \end{array}$$

The greedy adopted solution, which has the advantage over, e.g., an LPT[32] solution of the clustering properties earlier discussed, implies the preliminary bytewise sorting of the traffi c relations, and three simple rules:

- allow a machine load to exceed $1/K$ if the machine has no previously scheduled job;
- keep scheduling the biggest unscheduled job into the same machine while the load is still below $1/K$;
- remove the scheduled job from the unscheduled jobs list as soon as job has been scheduled.

A representation of the algorithm output applied to the TA creation problem is provided in Fig. 7; the x-axis represents the number of TR within TA and the y-axis represents the weight of each generated TA, i.e, $\hat{w}_J(K)$. For the ease of the read, results for classes $K \in \{10, 50, 100\}$ are highlighted, whereas neither classes $K < 3$ nor fi xed slots are reported in the picture. Observing the fi gure, and recalling the distribution of the TR weights plotted in Fig. 4, we notice that, given a class K, the number $|\tau_J|$ of TRs inside the J-th slot increases as J increases. For example, the last slot (the one on the rightmost part of the plot) exhibits always a number of TRs larger than 100,000. On the contrary, looking at the fi rst classes, we observe that the number of TRs tends to decrease for increasing K, showing the “packing” enforced by the selected algorithm.

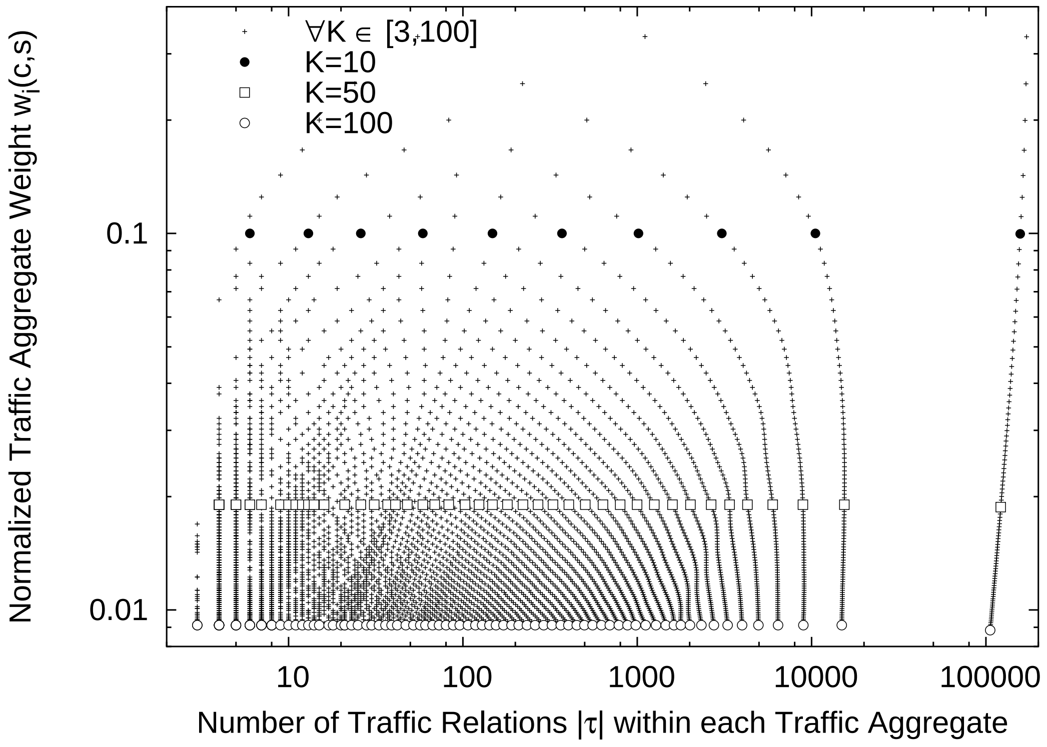

Fig. 7. Trace Partitioning: Samples for Different Aggregated Classes K

IV. Results

In this section we investigate the most interesting properties of the artificially built traffic aggregates; the analysis will be conduced in terms of, mainly, aggregated size and TCP flow interarrival times, coupling their study with the knowledge of the underlaying layers – i.e., traffic relation and flow levels.

To help the presentation and discussion of the measurement results, we first propose a visual representation, alternative to the one of Fig. 7, of the dataset obtained by applying the splitting algorithm. Indeed, the aggregation process induces a non-linear sampling of both the number $|\tau_J(K)|$ of TRs within each TA and the TA weight $\hat{w}_J(K)$, complicating the interpretation of the results. However, the same qualitative information can be immediately gathered if we plot data as a function of the class K and slot J indexes, using besides different gray-scale intensities to represent the measured quantity we are interested in.

As a first example of the new representation, Fig. 8 depicts the number $|\tau_J(K)|$ of the traffic relations mapped into each traffic aggregate $\tau_J(K)$. Looking at the plot and choosing a particular class K, every point of the vertical line represents therefore the number of TRs within each of the K possible TA – obtained by partitioning the trace into the K TAs having approximatively a $1/K$ weight. Given the partitioning algorithm used, it is straigth-forward to understand the reason why the higher is the slot considered, the larger is the number of TRs within the same TA, as the gray gradient clearly shows. To better appreciate this partitioning effect, contour lines are shown for $|\tau_J(K)| \in \{1, 10, 100, 1000, 10000\}$ as reference values; it can be gathered that the bottom white-colored zone of the plot is constituted by *fixed* slots (i.e., $|\tau| = 1$), whereas the slot $J = K$ has always $|\tau_K(K)| > 100,000, \forall K$, as already observed from Fig. 7. This further confirms the validity of our simple heuristic in providing the previously described aggregation properties.

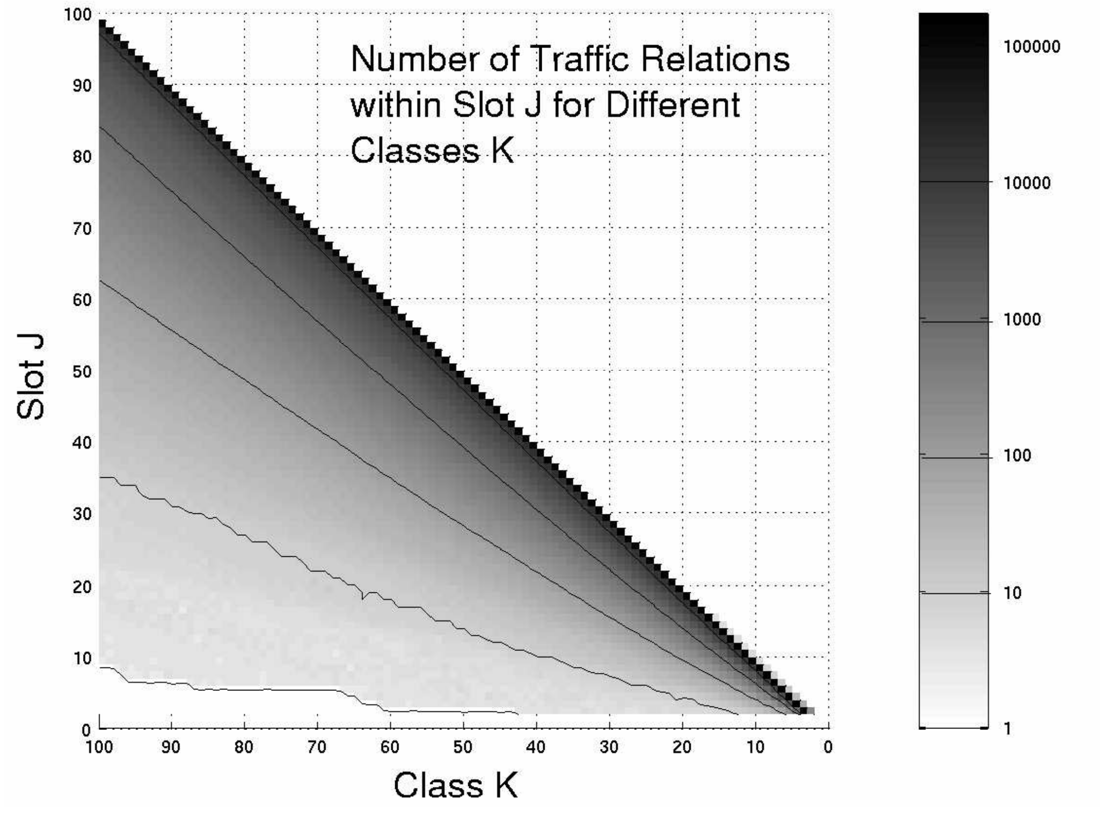

Fig. 8. Number of Traffic Relations $|\tau_J(K)|$ within each Traffic Aggregate

A. *Traffic Aggregate Bytewise Properties*

Having showed that the number of TR within the generated TA spread smoothly over a very wide range for any TA weight, we now investigate if these effects are reflected also by the number of TCP flows withing each TA, $\Sigma_{(c,s)\in\tau_J(K)}|\mathcal{T}(c,s)|$, shown in Fig. 9. Quite surprisingly, we observe that also the number of TCP connections within TA shows an almost similar spreading behavior: the larger number of TRs within a TA, the larger number of TCP flows within the same TA. Indeed, the smoothed upper part of the plot (i.e., roughly, slots $J > K/2$) is represent by TAs with a large number of TCP flows, larger than about 1,000; instead, TAs composed by few TRs contain a much smaller number of TCP connections: that is, the number of TCP flows keeps relatively small, while not as regular as in the previous case. Indeed, it must be pointed out that there are exception to this trend, as shown by the "darker" diagonal lines – e.g., those ending in slot $J = 25$ or $J = 54$, considering class $K = 100$. Probably, within these TAs there is one (or possibly more) TR which is built by a large number of short TCP flows.

Coupling this result with the bytewise constraint imposed on TAs within the same class, we can state that the bottom region is constituted by a small number of flows accounting for the same traffic volume generated by a huge number of lighter flows. This intuition is also confirmed by Fig. 10, which shows the mean TCP flow size in the different aggregates; the higher is the slot J considered, the larger are both the TRs and TCP flows number, the shorter are the TCP flows.

While this might be surprising at first, the intuition behind this clustering is that the largest TRs are built by heavier TCP-connection than the smaller TRs, i.e., TCP-elephants play a big role also in defining the TR weight. Therefore the splitting algorithm, packing together larger TRs, tends also to pack together TCP-elephants.

The TCP flow size variance, not shown to avoid cluttering the figures, further confirms the clustering of

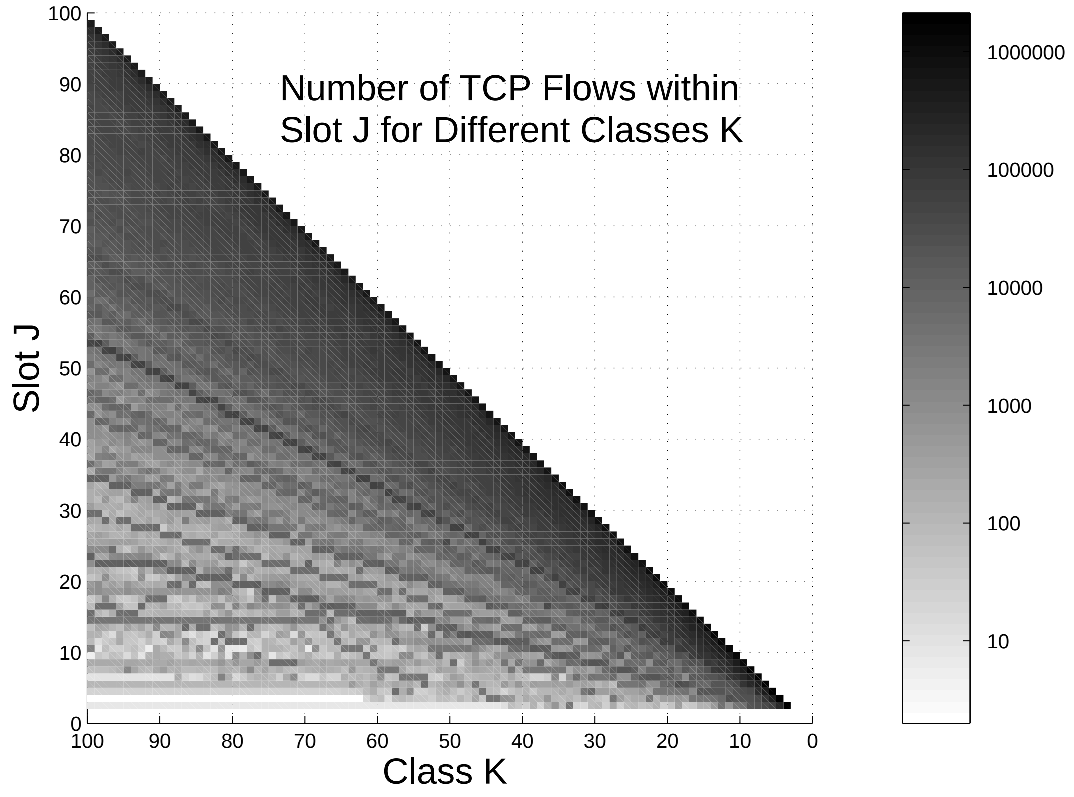

Fig. 9. Number of TCP Flows within each Traffic Aggregate

TCP elephant flows in the bottom side of the plot. To better highlight this trend, we adopted a threshold criterion: the mean values of the TCP flow size are compared against fixed quantization thresholds, which allows to better appreciate the mean flow size distribution in TAs. The results for threshold values set to $100, 250, 500, 1000$ kB are shown in Fig. 11, where higher threshold values correspond to darker colors. The resulting plot underlines that the mean flow size grows toward TA constituted by a smaller number of larger TR, which are in their turn, constituted by a small number of TCP-elephants. Similarly, the higher-order slots are mostly constituted by mice.

However, the gained result do not exclude the presence of TCP-mice in the bottom TAs, nor the presence of TCP-elephants in the top TAs; therefore, to further ensure that the clustering property of TCP-elephants in the first slots holds, we investigated how the TCP flow size distributions in the different aggregates vary as a function of the TA number K. We thus plot in Fig. 12 the empirical flow size distribution for each TAs, showing only the results of class $K = 3$ for the ease of the read. As expected, i) TCP-elephants are evidently concentrates in lower slots, as shown by the heavier tail of the distribution; ii) the TCP-elephants presence decreases as the slot increases, i.e., when moving toward higher slots. The complementary cumulative distribution function $P\{X > x\}$, shown in the inset of Fig. 12, clearly confirms this trend.

This finally confirms that the adopted aggregation criterion induces a division of TCP-elephants and TCP-mice into different TAs: TRs containing the highest number of TCP-elephants tends to be packed in the first TA. Therefore in the following, we will use for convenience the terms TA-mice and TA-elephants to indicate TA constituted (mostly) by TCP-mice and TCP-elephants flows respectively.

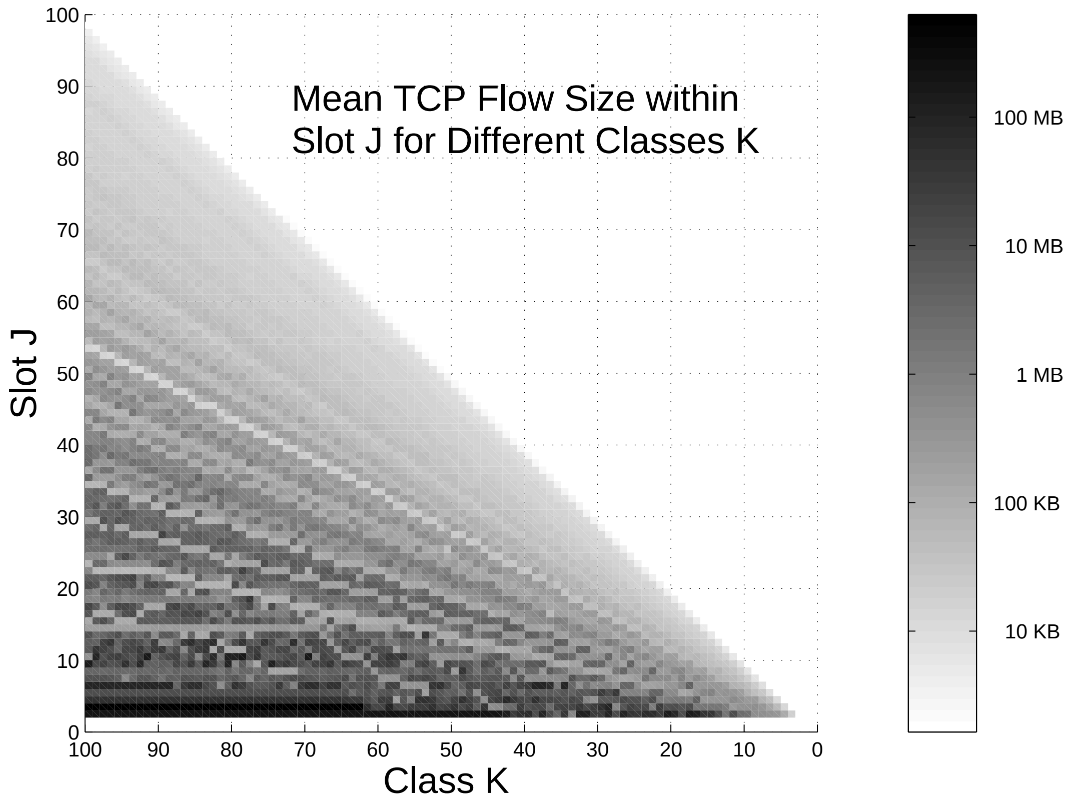

Fig. 10. Mean Size of TCP Flows within each Traffi c Aggregate

B. Inspecting TCP Interarrival Time Properties within TAs

The result gained in the previous section has clearly important consequences when studying the TCP fbw arrival process of different aggregates.

Let us fi rst consider the mean interarrival time of TCP fbws within each TA, shown in Fig. 14. Intuitively, TA-mice have a large number of both TR and TCP fbws, and therefore TCP fbw mean interarrival time is fairly small (less than 100ms). This is no longer true for TA-elephants, since a smaller number of fbws has to carry the same amount of data over the same temporal window. Therefore, the mean interarrival time is much larger (up to hours).

We recognize, however, that a possible problem might arise, affecting the statistical quality of the results: the high interarrival time may be due to non-stationarity in the TA-elephants traffi c, where TCP fbws may be separated by long silence gap. This effect becomes more visible as long as the class index K and so the aggregation level $|\tau_J(K)|$ decreases. We will try to underline whether the presented results are affected by this problem in the remaining part of the analysis.

The previous assertion is also confi rmed observing Fig. 14, which shows, within each TA, the plot of TCP fbw interarrival time variance. It can be noticed that the TCP fbw interarrival time variance is several orders of magnitude smaller considering TA-mice with respect to the TA-elephants.

Let us now consider the Hurst parameter $h(J,K)$ measured considering the interarrival time of TCP fbws within TAs. For each TA, we performed the calculation of $h(J,K)$ using the wavelet-based approach developed in [27]. We adopted the tools developed there and usually referred to as the *AV* estimator. Other approaches can be pursued to analyze traffi c traces, but the wavelet framework has emerged as one of the best estimators, as it offers a very versatile environment, as well as fast and effi cient algorithms.

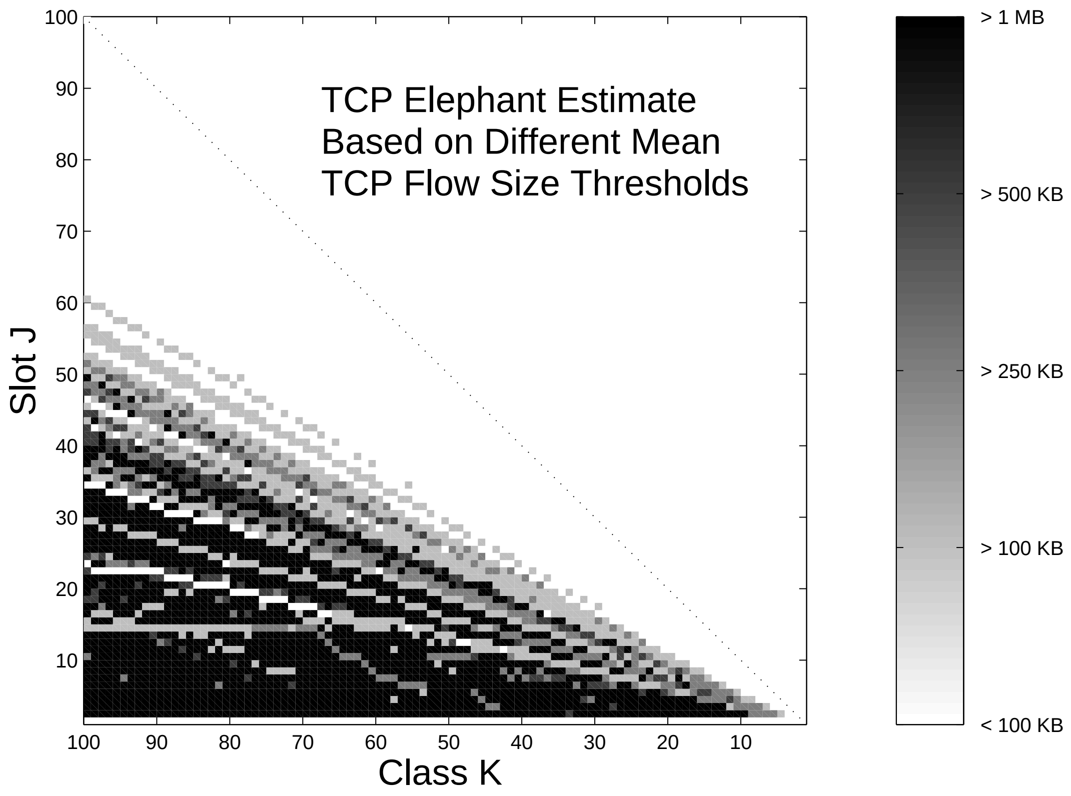

Fig. 11. Elephant TCP Flows within each Traffic Aggregate

The results are shown in Fig. 15, which clearly shows that the Hurst parameter tends to decrease for increasing slot, i.e., the TA-mice show $h(J, K)$ smaller than TA-elephants. Recalling the problem of the stationarity of the dataset obtained by the splitting algorithm, not all the $h(J, K)$ are significant; in particular, the one whose confidence interval is too large or the series is not stationary enough to give correct estimations were discarded, and not reported in the plot. Still, the increase in the Hurst parameter is visible for TA-elephants.

To better show this property, Fig. 16 presents detailed plots of $h(J, K)$ for $K \in \{10, 50, 100\}$ (on the top-left, top-right, bottom-left plots respectively) and for all the slots J of each class. It can be observed that the Hurst parameter always tends to decrease when considering the TA-mice slots, while it becomes unreliable for TA with few TRs, as testified by the larger confidence intervals. This is particularly visible when considering the $K = 100$ class. In the bottom-right plot, finally, we report a detail of the decaying feature of the $h(J, K)$ value for large J. In the x-axis, the J/K value is used, so that to allow a direct comparison among the three different classes. Notice that the last class, the one composed by many, small TRs which are aggregation of small TCP flows, always exhibits the same Hurst parameter.

Therefore, the most important observation, trustable due to good confidence interval, is that we are authorized to say that TA-mice behavior is driven by TCP-mice. Similarly, TA-elephants are driven by TCP-elephants. Moreover, the interarrival process dynamic in TA-elephants and TA-mice are completely different in nature because light TRs tend to contain a relatively small amount of data carried over many small TCP flows, which do not clearly exhibit LRD properties. On the contrary, TCP-elephants seem to introduce a more clear LRD effects in the interarrival time of flows within TA-elephants.

A possible justification of this effect might reside in the different behavior the users have when generating connections: indeed, when considering that TCP-mice are typically of Web browsing, the

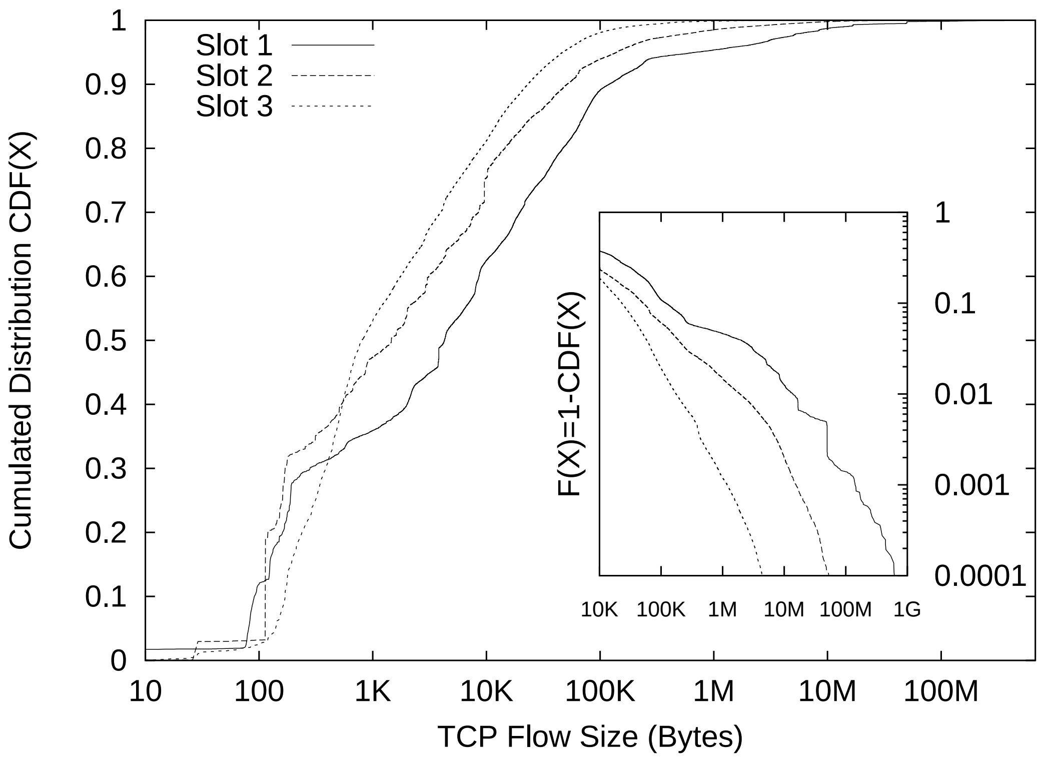

Fig. 12. TCP Flows Size Distribution of TAs (Class $K = 100$)

correlation generated by Web sessions tends to vanish when a large number of (small) TRs are aggregated together. On the other side, the TCP-elephants, which are rare but not negligible, seem to be generated with a higher degree of correlation so that i) TRs are larger, ii) when aggregating them, the number of users is still small.

For example, consider the different behavior of a user which is downloading large files, e.g., MP3s; one can suppose that the user starts downloading the first file, then –immediately after having finished the download– he starts downloading a second one, and so on until, e.g., the entire LP has been downloaded. The effect on the TCP flow arrival time is similar to a ON-OFF source behavior, whose ON period is heavy-tailed and vaguely equal to the file download period, which turns out to follow an heavy tailed distribution; besides, we do not care about the OFF period. This is clearly one of the possible causes of LRD properties at the flow level: see, for an example, the τ_D aggregate in Fig. 3.

Besides, consider again the heavy-tailed distribution of the TCP flow number within TRs, shown earlier in Fig. 5. If we further consider TRs as a superset of web sessions, we gather the same result stated in [19], that is, the $M/G/\infty$ effect with infinite variance service time distribution.

Finally, TAs tends to aggregate several TRs, separating short term correlated connections (TA-mice) from low-rate ON-OFF connections with infinite variance ON time (TA-elephants). This clearly recall to the well know phenomena described in [8]; that is, the superposition of ON-OFF sources (as well as "packet trains") exhibiting the Noah effect (infinite variance in the ON or OFF period) produces the so called "Joseph effect": the resulting aggregated process exhibits self-similarity and in particular LRD properties.

V. Conclusions

In this paper we have studied the TCP flow arrival process, starting from the aggregated measurement taken from our campus network; specifically, we performed a live collection of data directly at the TCP flow

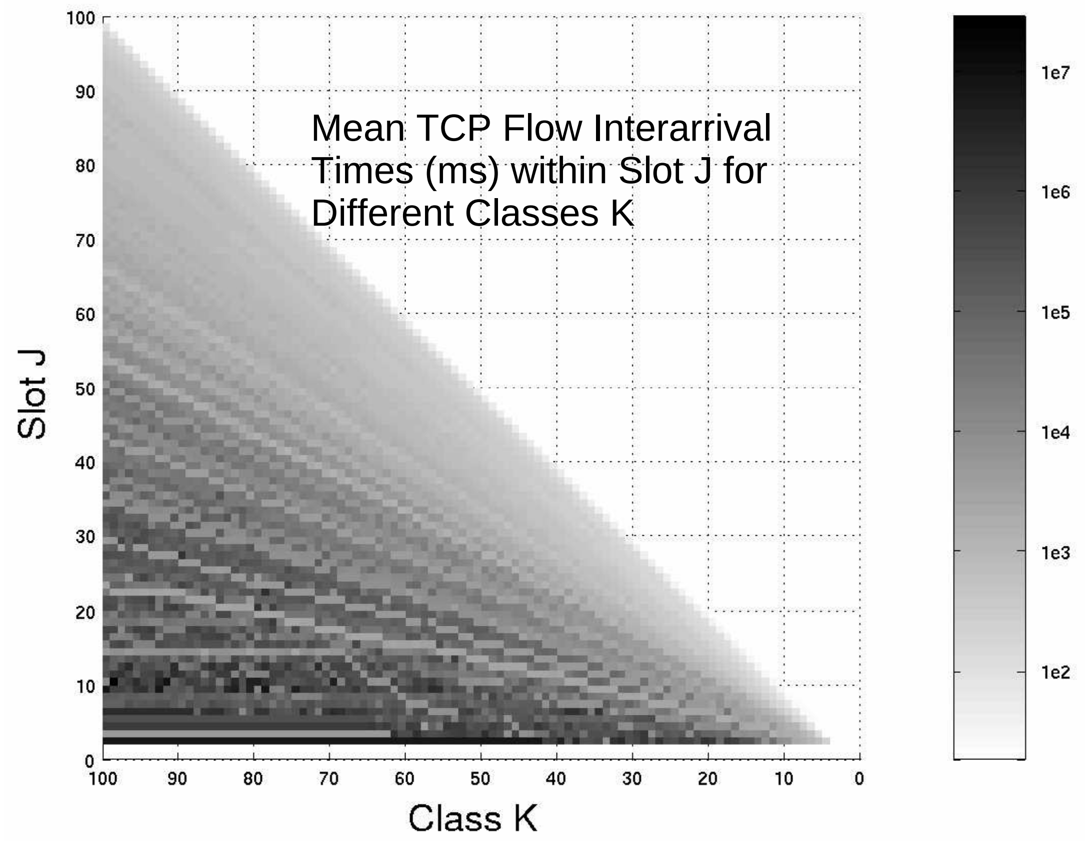

Fig. 13. Interarrival Time Mean of TCP Flows within TAs

level, neglecting therefore the underlaying IP-packet level. Trace were both obtained and post-processed through software tools developed by our research group, publicly available to the community as open sources.

We focused our attention beyond the TCP level, defi ning two layered high-level traffi c entities. At a fi rst level beyond TCP, we identifi ed traffi c relations, which are constituted by TCP fbws with the same path properties. At the highest level, we considered traffi c relation aggregates having homogeneous bytewise weight; most important, the followed approach enabled us to divide the whole traffi c into aggregates mainly made of either TCP-elephants or TCP-mice.

This permitted to gain some interesting insights on the TCP fbw arrival process. First, we have observed, as already known, that long range dependence at TCP level can be caused from the fact that the number of fbws within a traffi c aggregate is heavy-tailed. In addition, the traffi c aggregate properties allowed us to see that TCP-elephants aggregates behave like ON-OFF sources characterized by an heavy-tailed activity period. Besides, we were able to observe that LRD at TCP level vanishes for TCP-mice aggregates: this strongly suggests that even ON-OFF behavior is responsible of the LRD at TCP level.

APPENDIX

The analysis conduced in this paper was carried on with the use of `DiaNa`, which stands for *Data Inspector aka Nonsense Analyzer*, a software developed by our research group. The tool, being now consolidated after more than one year of intensive test and use, is made freely available [11] to the research community.

The presented framework does not intend to be a replacement for any existing analysis tool; nevertheless, despite the wide variety of already available software, there are cases where the need of a tools such as

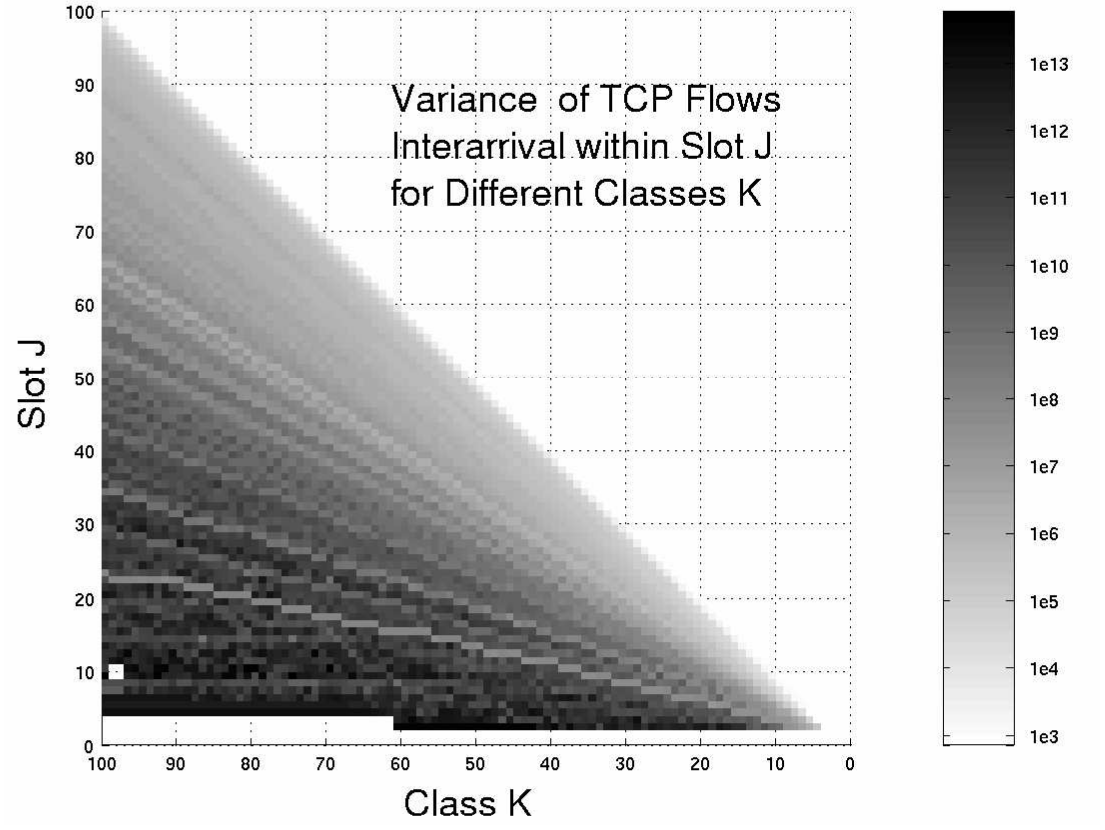

Fig. 14. Interarrival Time Variance of TCP Flows within TAs

`DiaNa` arises. Indeed, the software has been specially designed -even though its use is not restricted- to deal with huge amount of data, of which `Tstat`'s traces represent a natural example; for example, all the data manipulation of the presented work has been performed using `DiaNa`. More specifically, the software parses text files holding information structurally representable as matrices.

The `DiaNa` package is constituted by a set of shell tools and a graphical user interface (GUI), both entirely written in Perl [29]: in the following, we refer to the formers as `diana_tools` and to the entire package as `DiaNa`, covering their essential features. For further details, please refer to both the `DiaNa` Tutorial [28] and the `DiaNa` man-pages, included into the standard distribution.

A. The `DiaNa` *Architecture*

The relationship among the `DiaNa` GUI, the `diana_tools` and Perl is sketched in Fig. 17. From a top down point of view, the graphical Perl/Tk GUI acts as a centralized information collector and coordinator of the lower-level shell-tools; each tool performs an atomic task on the input data, offering the possibility of defining and exploiting custom file formats, upon which algorithmic expressions may be built. The *GUI* automatizes the interaction among the tools offering further an IDE to manage the `DiaNa` syntactical items; notably, among its additional features, there are:

- a *Perl shell*, supporting auto-completion and interactive history;
- a *Gnuplot[30] front-end*, coupling a drag-and-drop approach to a template mechanism and direct code editing;
- customizability through *plug-ins*, intended as Perl scripts to be run on-the-fly.

The *shell-tools* set covers a wide range of tasks, such as ranking the input data on a database-like manner, performing algebraic and textual operations and evaluating first and second order statistics and

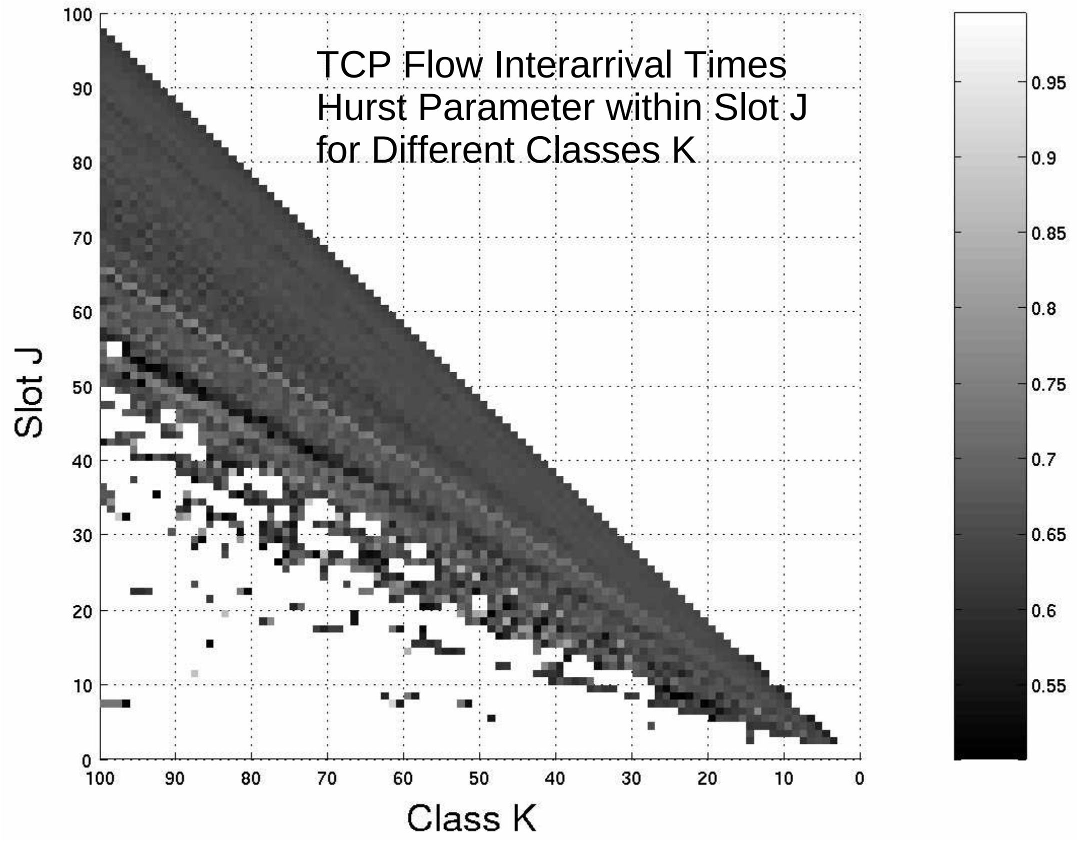

Fig. 15. Interarrival Time Hurst Parameter of TCP Flows within TAs

empiric probability distributions. The `DiaNa` *syntax* represents a small superset of the already full blown Perl one. Basically, *formats* allow to attribute labels and comments to identify different fi le fi elds; these labels can be used in *expressions* for spreadsheet-like computation; fi nally, *ranges* are a convenient way to defi ne partitions of the real numbers fi eld $\mathbb{R}$.

B. The `DiaNa` Syntax

Although the `DiaNa` syntax provides a fairly small increment to the Perl one, nevertheless it would be cumbersome to fully describe it here. Therefore, rather than covering all the details of the added items (i.e., format, ranges and expressions), we will just give a rough idea of their use by describing the interaction between *formats* and *expressions*.

1) Input Files and Formats: The `diana_tools` matrix abstraction of a generic input data fi le is sketched in Fig. 18. Input fi les may contain *comments* –i.e., lines beginning with a confi gurable prefi x– which can be discarded or processed; comment rows at the beginning of the input represent its *header*. Different *rows* are individuated by a new-line termination character, while an Input Field Separator (IFS) –which can be a character or a regular expression [31]– is responsible for *columns* partitioning. The presented software allows, as it has been said, to perform spreadsheet computation on huge amount of data. In such a case, it is however clear that not all data can be directly available in computer's physical memory; rather, the adopted strategy is to loop over data rows, doubly limiting the information available at any point of time in both the *future* and the *past*. That is, as soon as data *rows* have been read from input, they become available for processing and are possibly buffered; all the *columns* of the buffered rows represent the sub-portion of the input available to the running algorithm. This crucial point is enforced in Fig. 18, by representing the buffered rows with colors fading toward the past.

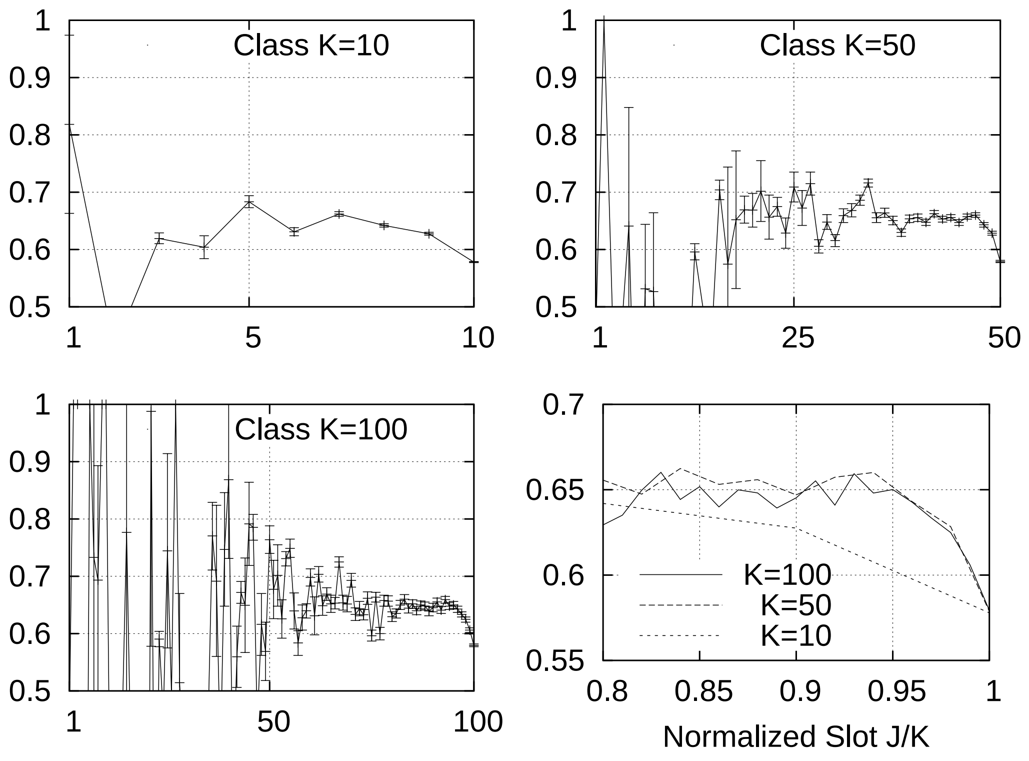

Fig. 16. Interarrival Time Hurst Parameter of TCP Flows within TAs

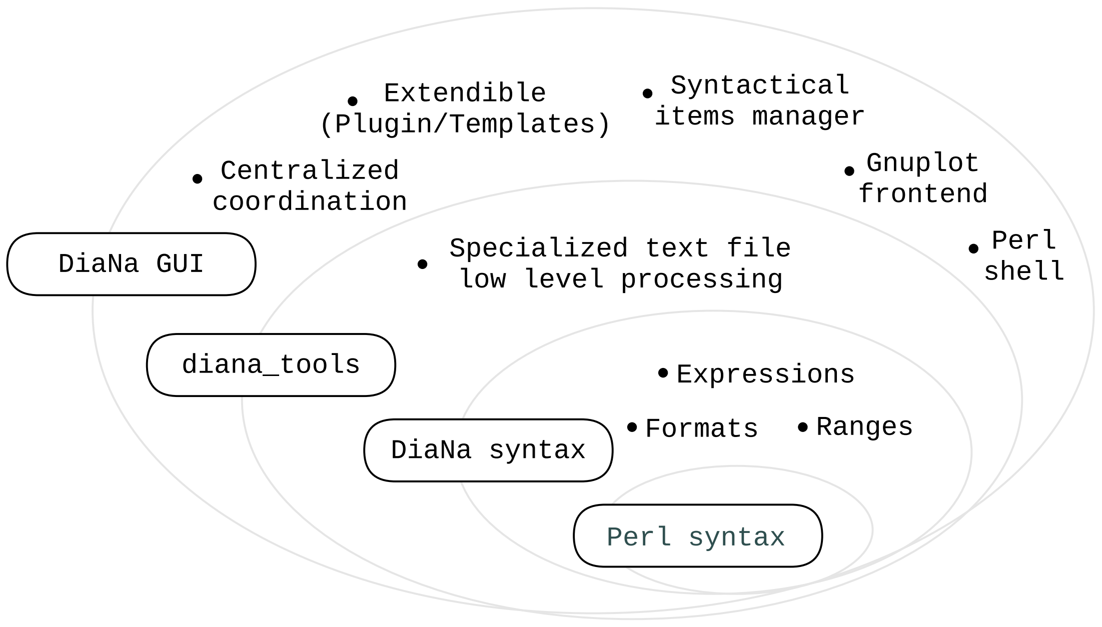

Fig. 17. Conceptual Layering of the `DiaNa` Framework

Columnar fi elds play a privileged role over fi le rows; therefore, a *format* allows to attribute a *(label,comment)* pair to each fi le column. Each format defi nition is usually stored in an external fi le; however, columnar *labels* may be directly stored in the input *header*.

2) Formats and Expressions: `DiaNa` expressions are the glue that ties input fi les to formats, allowing to quick reference matrix cells in Perl-coded algorithms and/or algebraic expressions. As will be outlined

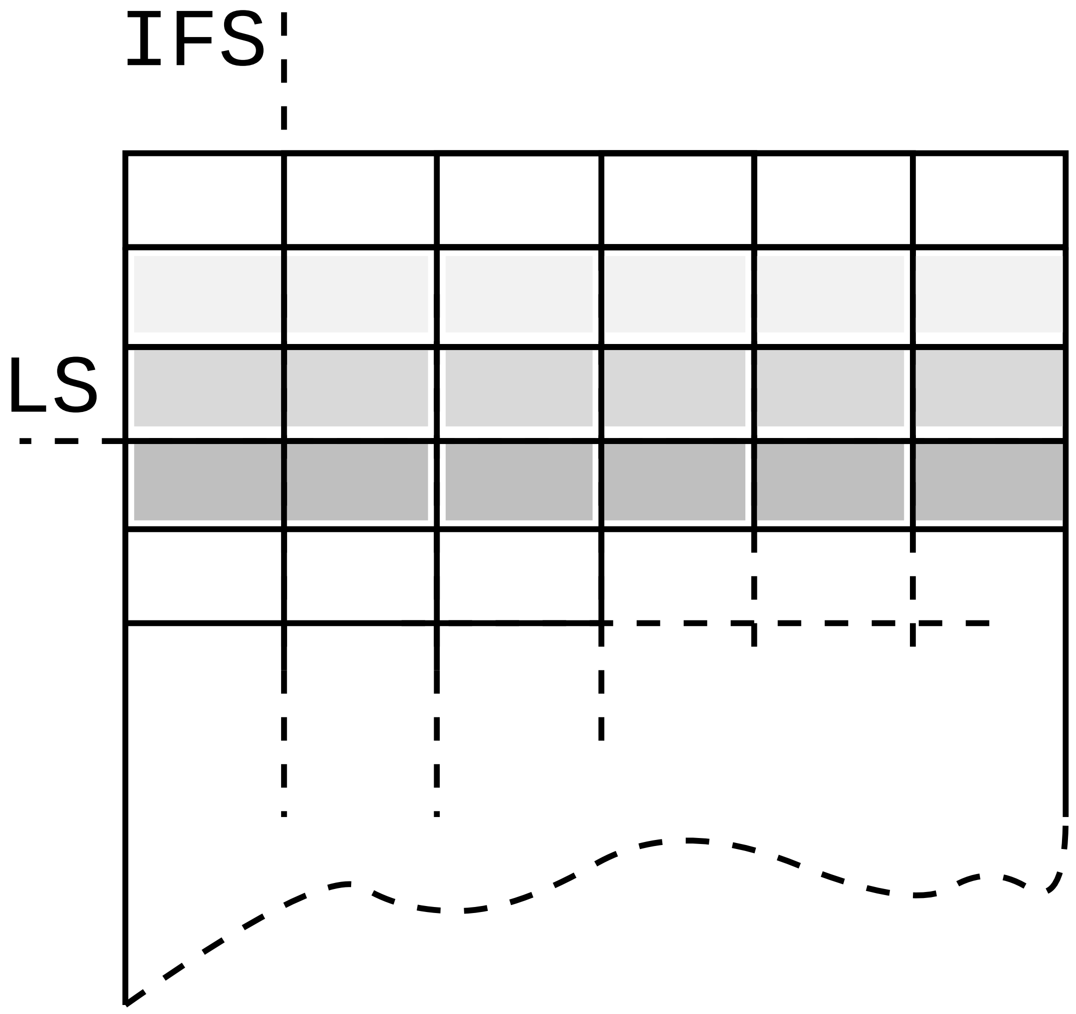

Fig. 18. Input Data Scheme from the `DiaNa` Perspective

later, input fi les can be read either in *serial* or in *parallel*; therefore, expressions are interpreted in a serial or parallel fashion accordingly to the input context: in this section, we will focus on the former kind[3].

Some input processing task may require only the knowledge of all the columnar fi eld values at that point, whereas this is not true in general; therefore, as input fi les are buffered along the loop over the input, expressions allow to profi t of this sort of *memory*. Moreover, since *formats* defi ne a 1-to-1 correspondence between fi le columns and labels, these latter can be used as in expressions' tokens. *Tokens* are strings beginning with the diesis `#` sign, which is interpreted by Perl as a comment delimiter character; that way, `DiaNa` tokens do not clashes with the under-laying Perl syntax. Serial tokens are mainly:

`#j` the j-th columnar fi eld of the current row
`#i:j` the j-th columnar fi eld of the the abs(i)-th previously read row
`#label` expands to the columnar fi eld, whose label is `label`, of the current row
`#i:label` expands to the columnar fi eld, whose label is `label`, of the abs(i)-th previously read row

Several tokens can then be combined together following the usual Perl syntax, including algebraic, textual and subroutine based manipulations. Indeed, `DiaNa` *expressions* are, at a lower level, fragments of Perl

[3]Although parallel expressions' syntax is formally identical to the serial one, nevertheless its interpretation is radically different.

code, which are pre-parsed once at startup, in order to speed up their evaluation during processing. Moreover, real memory requirements are automatically determined in both the *vertical deepness* (i.e., how many input rows have to be buffered) and the *horizontal extension* (i.e., which fields have to be buffered), so that `DiaNa` memory can be represented as a sparse matrix.

C. The `diana_tools` Architecture

The `diana_tools` are a set of tools, entirely developed in Perl, providing a framework for performing different tasks on quasi-matrix data. Among the offered features, we may cite the i) discrimination of input based on a given condition, ii) input format conversion through algebraic expressions, iii) first and second order statistics collection, iv) empirical probability distribution –possibly conditioned– computation. Each tool is designed to perform a well defined task and is developed around a common core, sketched in Fig. 19. The `diana_tools` core provides a set of useful built-in peers to simplify the implementation of new and unsupported features. Starting from the left side of the scheme, we shall notice that tools

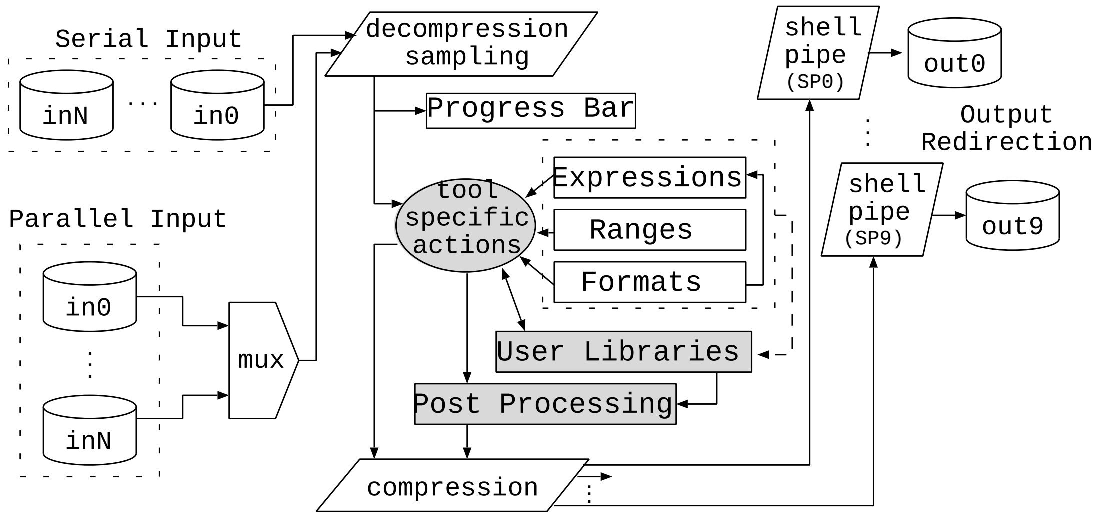

Fig. 19. Architecture of a Generic `diana_tool`

may work on either in a *parallel* or *serial* fashion over the input files: in the former case, several buffered rows for each separate file will be available for processing. Input files are allowed to be compressed with the most common compression schemes (i.e., `tar`, `gzip`, `bz2` as well as their combination), further providing optional and configurable uniform or Montecarlo sampling. Besides, input files may have a known format, in which case format labels can be used in expressions; format is possibly described in the input file header and file comments can be processed or discarded.

At each step in the loop over the input rows, a progress meter is optionally shown; data get read and buffered, becoming available for inline-processing: the specific action performed clearly depends on the tool purpose. For example, the most simple tool called `diana_loop`, does not perform any action but does rather offer the possibility of executing inline user-specific actions. Indeed, one of the reasons why Perl has been chosen among other candidates is that it does offer a very straightforward way of direct supporting any user-coded libraries: i.e., text script are simply sourced by the tool on demand. This alleviate the user from a wide serie of tasks, allowing him to concentrate on the pure algorithmic part (e.g. packet trace anonymization, trace-splitting into aggregates, . . .).

Another reason is that Perl supports a wide variety of structures such as *hashes* and *arrays*; moreover, these can be grown on demand without memory management needs, and nested combination of the above structures is allowed without restrictions. This can be very useful to post process the parsed data, after all the input have been processed, just before the tool execution ends; also, Perl structures can be stored and re-loaded as a Perl library for another instance of `diana_tools` processing (e.g., computing a mapping function of TRs to TAs using an hash).

Finally, up to ten output streams can be used simultaneously, each of which can be fed into a different shell pipe; this is clearly useful to parallelize algorithmic work on the same input, therefore reducing intensive-requirement tasks such as disk access. Indeed, there are a wide number of situations where input reading may be the most time consuming task of the whole process; parallelizing several tasks entails better performances, when, e.g., computing TRs and TAs inter-arrival times.

D. `DiaNa` and Perl

Although the presented architecture is undoubtedly appealing, it could be objected that the the features earlier described cannot be offered by high-level Perl scripts without a significant cost.

It should be said, however, the general framework does mainly acts as a parser, in the sense that it does produce the optimized Perl code that is going to be executed. By optimization we mean the avoidance of useless redundant tests and subroutine calls and expressions pre-parsing to get faster evaluation; e.g., the `diana_tools` core will not call the sampling routine nor will test for its need if sampling is not used, will not use more memory than needed and will include memory initialization and update in loops only if necessary, ...

Most of the `diana_tools` overhead is due to the startup phase, that is, when the `DiaNa` module and user-libraries are loaded: tasks such as file length estimation, option and expressions parsing, syntactical error checking and code generations are done in this step. Therefore, the initial overhead effect tends to vanish as the input files size grows. Moreover, critical tasks such as intensive computation routines can be written in C and then linked to be directly used as a Perl extension[4]; also, it is worth to mention that Perl programs can be compiled to a byte-code form, or even to lower level C or assembly language if needs be.

REFERENCES

[1] R. Caceres, P. Danzig, S. Jamin and D. Mitzel, *Characteristics of Wide-Area TCP/IP Conversations,* ACM SIGCOMM, 1991

[2] P. Danzig and S. Jamin, *`tcplib`: A library of TCP Internetwork Traffic Characteristics,* USC Technical report, 1991

[3] P. Danzig, S. Jamin, R. Caceres, D. Mitzel, and D. Mestrin, *An Empirical Workload Model for Driving Wide-Area TCP/IP Network Simulations,* Internetworking: Research and Experience, Vol.3, No.1, pp.1–26, 1992

[4] V. Paxons, *Empirically Derived Analytic Models of Wide-Area TCP Connections,* IEEE/ACM Transactions on Networking, Vol.2, pp.316–336, Aug. 1994

[5] V. Paxson and S. Floyd, *Wide-area Traffic: The Failure of Poisson Modeling,* IEEE/ACM Transactions on Networking, Vol.3, No.3, pp.226–244, Jun. 1995

[6] W.E. Leland, M.S. Taqqu, W. Willinger and V. Wilson, *On the Self-Similar Nature of Ethernet Traffic (Extended version),* IEEE/ACM Transaction on Networking, Vol.2, No.1, pp.1–15, Jan. 1994

[7] A. Feldmann, *Characteristics of TCP Connection Arrivals,* Park and Willinger (editors) *Self-Similar Network Traffic and Performance Evaluation*, Wiley-Interscience, 2000

[8] W. Willinger, M.S. Taqqu, R. Sherman and D.V. Wilson, *Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level,* IEEE/ACM Transaction on Networking, Vol.5, No.1, pp.71–86, Jan. 1997

[9] M. Mellia, A. Carpani and R. Lo Cigno, *Measuring IP and TCP behavior on Edge Nodes,* IEEE Globecom, Taipei (TW), Nov 2002

[10] `Tstat` home page `http://www.tlc-networks.polito.it/tstat`

[11] `DiaNa` home page `http://www.tlc-networks.polito.it/diana`

[12] A. Erramilli, O. Narayan and W. Willinger, *Experimental Queueing Analysis with Long-Range Dependent Packet Traffic,* IEEE/ACM Transactions on Networking, Vol.4, No.2, pp.209–223, 1996

[13] N. Hohn, D. Veitch and P. Abry, *, Does Fractal Scaling at the IP Level depend on TCP Flow Arrival Processes ?,* 2nd Internet Measurement Workshop, Marseille, Nov. 2002

[14] M.E. Crovella and A. Bestavros, *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes,* IEEE/ACM Transactions on Networking, Vol.5, No.6, pp.835–846, 1997

[4]Perl's XS interface allows to either dynamically load or statically link into Perl existing C libraries

[15] I. Norros, *On the Use of Fractional Brownian Motion in the Theory of Connectionless Networks,* IEEE Journal on Selected Areas in Communications, Vol.13, pp.953–962, Aug. 1995

[16] M. Taqqu and V. Teverosky, *Is Network Traffi c Self-Similar or Multifractal ?,* Fractals, Vol.5, No.1, pp.63–73, 1997

[17] R.H. Riedi and J. Lévy Véhel, *Multifractal Properties of TCP Traffi c: a Numerical Study,* IEEE Transactions on Networking, Nov. 1997 (Extended version appeared as INRIA research report 3129, Mar. 1997)

[18] A. Gilbert, W. Willinger and A. Feldmann, *Scaling Analysis of Conservative Cascades, with Applications to Network Traffi c,* IEEE Transactions on Information Theory, Vol.45, No.3, pp.971–992, Apr. 1999

[19] A. Feldmann, A. Gilbert, W. Willinger and T. Kurtz, *The Changing Nature of Network Traffi c: Scaling Phenomena,* Computer Communication Review 28, No.2, Ap. 1998

[20] D. Veitch, P. Abry, P. Flandrin and P. Chainais, *Infi nitely Divisible Cascade Analysis of Network Traffi c Data,* IEEE International Conference on Acoustics, Speech, and Sgnal Processing (ICASSP'00), 2000.

[21] S.Roux, D.Veitch, P.Abry, L.Huang, P.Flandrin and J.Micheel, *Statistical Scaling Analysis of TCP/IP Data,* IEEE International Conference on Acoustics, Speech, and Sgnal Processing (ICASSP'01), Special session: Network Inference and Traffi c Modeling, May 2001

[22] A. Horváth and M. Telek, *A Markovian Point Process Exhibiting Multifractal Behavior and its Application to Traffi c Modeling,* 4th International Conference on Matrix-Analytic Methods in Stochastic Models, Adelaide (Australia), Jul. 2002

[23] A.T. Andersen and B.F. Nielsen, *A Markovian Approach for Modeling Packet Traffi c with Long-Range Dependence* IEEE Journal on Selected Areas on Communication, Vol.16, No.5, pp.719–732, 1998.

[24] S. Robert and J.Y. Le Boudec, *New Models for Pseudo Self-Similar Traffi c,* Performance Evaluation, Vol.30, No.1-2, pp.57–68, 1997.

[25] S. Robert and J.Y. Le Boudec, *Can Self-Similar Traffi c be Modeled by Markovian Process ?,* International Zurich Seminar on Digital Communication, Feb. 1996

[26] A. Reyes Lecuona, E. González Parada, E. Casilari, J.C. Casasola and A. Díaz Estrella, *A Page-Oriented WWW Traffi c Model for Wireless System Simulations*, ITC16, Edinburgh, Jun. 1999

[27] P. Abry, P. Flandrin, M.S. Taqqu and D.Veitch, *Self-similarity and Long-Range Dependence Through the Wavelet Lens,* In *Long Range Dependence: Theory and Applications*, Doukhan, Oppenheim, 2000

[28] D. Rossi, *The `DiaNa` Tutorial*, available at `http://www1.tlc.polito.it/diana`

[29] L. Wall, *Larry Wall's* Very Own *Perl Page*, available at `http://www.wall.org/ larry/perl.html`

[30] *Gnuplot Central*, available at `http://www.gnuplot.info`

[31] J. Friedl, *Mastering Regular Expressions* O'Reilly, Jan. 1997

[32] S.C. Graves, A.H.G. Rinnooy Kan and P.H. Zipkin, *Logistic of Production and Inventory*, Nemhauser and Rinnoy Kan (editors), *Handbooks in Operation Research and Management Science*, Vol.4, North-Holland, 1993.