Techniques for Simulation of Realistic Infrastructure Wireless Network Traffic

Caleb Phillips, Douglas Sicker, Dirk Grunwald
Department of Computer Science
University of Colorado
Boulder, Colorado, USA
Email: {caleb.phillips,sicker,grunwald}@colorado.edu

Suresh Singh
Department of Computer Science
Portland State University
Portland, Oregon, USA
Email: singh@cs.pdx.edu

***Abstract*—In the design of wireless networking protocols and systems, simulation has become the primary form of initial validation and performance evaluation. Hence, ensuring the realism of simulators and simulation methods is fundamental for simulated results to be interpretable. In this paper, we provide a simulation framework for infrastructure wireless network traffic that allows researchers to use publicly available captured traces as a primary or background traffic source. We investigate the question of trace classification as a necessary task for these traces to be useful and apply our framework to a well-known power-saving application, showing that the use of real traffic provides substantially different results as compared to traffic generated from an application-specific fitted model or contrived source. Additionally, we show how trace classification provides unique insights into application behavior in both typical and extreme scenarios.**

I. Introduction

The validation and performance analysis of networking systems, both wired and wireless, is typically done in simulation or with testbeds. In both cases, the question of realism is of primary importance for results to be interpretable. In [1], Venkatesh et al. approach the question of how background traffic generation affects simulation and testbed results. They find that realistic traffic generation (particularly background traffic) has not been a priority in many recent publications, and attempt to quantify the harm this can cause by studying three applications. Their results show that traffic realism can have a substantial effect on simulation accuracy, but that this effect is application specific. Indeed, some applications' performance is minimally dependent on the underlying traffic pattern. However for many applications the underlying traffic does effect performance, sometimes substantially, and in these cases care must be paid to its realism.

In this work, we take the view that the simplest solution is likely to be correct in many cases, and that the best way to simulate realistic traffic is to use real traffic to begin with. The corpus of publicly available traces [2] has been steadily growing over time, and has recently reached a point of sufficient diversity to justify this approach. However, there are still barriers for the researcher. Particularly, it is not clear which traces to pick from the large selection. Even if the goal is to generate traffic using a custom parameterized model, as is a popular approach in the literature, the choice of which traces to use in model generation is still complex.

To approach this problem, we suggest several novel classification metrics for traces and propose a repeated-measures approach. We also take the stance that users are individuals and that grouping them arbitrarily may result in traffic models that have assimilated interesting attributes [3], [4]. Hence, these classifiers are aimed at user traces and allow the researcher to group these traces by attributes that are of particular interest for a given application. We show how this framework can be applied to a well-known powersaving application and attempt to quantify the ill-effects of using simpler traffic generators in its analysis.

Our work here is primarily concerned with modeling traffic in a specific type of network: an 802.11x wireless network operating in infrastructure mode. We focus on this type of network for two reasons: (1) there is a relatively large collection of publicly available traces for this type of network and (2) due to their massive popularity, infrastructure-mode wireless networks are of continued interest to the research community. However, it should be noted that our framework and classifiers could just as easily be applied to any other type of network traffic generation task.[1]

In the next section we will give an overview of related work. Section III will describe our proposed system for traffic generation and section IV proposes our classification metrics. In section V, we will apply the framework to the Bounded Slowdown powersaving application, present results, and then in VI we will conclude.

II. Related Work

The networking research literature contains no shortage of papers discussing the realism of simulation. In our community, seminal papers in this area are arguably [5] (for networking simulation in general) and [6] (for wireless networks in particular). Only recently, however, have researchers attempted to quantify the effects of traffic realism on simulator and testbed results [1].

Principally, we see two threads among work that considers traffic generation. The first thread is application-specific

[1]All of our source code and data can be found at http://systems.cs.colorado.edu

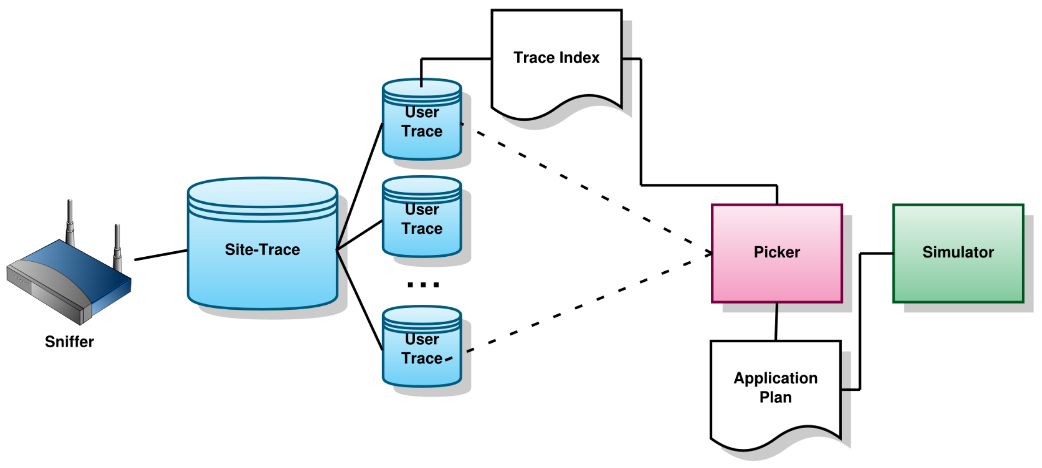


Fig. 1: Our traffic generation framework.

traffic models and generation schemes. These are plentiful and well implemented in modern network simulators. For analyzing the effect of a particular type of application traffic in some system, they are invaluable. Because it accounts for a large percentage of Internet traffic, there are many models of empirical Hyper-Text Transfer Protocol (HTTP) traffic [7], [8], [9]. In addition to straight-forward HTTP models, there are also web-workload oriented generators with specific goals such as server performance testing [10]. Models for other types of application-specific traffic include [11] which generates streaming workloads, [12] which models variable bit-rate video traffic, and [13] which models real-audio traffic.

The second thread is a more modern movement towards application-agnostic traffic generators. These range from being conservative to quite ambitious, but they almost universally operate by (1) reading in captured traces (2) distilling an empirical statistical model for some or all of the traffic and (3) generating traffic based on these models. It is these models that are most closely related to our work and therefore justify a bit more discussion.

In [14], Sommers and Barford propose the HARPOON traffic generation system. HARPOON uses 8 fitted variables to generate Internet Protocol (IP) flow level traffic which is statistically similar to the input traffic. Although it achieves the goal of concisely modeling large-scale temporal behavior, it fails to model small-scale behavior, which makes it of limited use to some applications. Venkatesh et al. present SWING in [15], which operates similarly to HARPOON, but addresses the limitations. Indeed, SWING can model both large and small-scale behavior and is even responsive to present network conditions. SWING, however, does not provide the user any guidance about which traces to use or how to use them, which we solve through classification and repeated measures. SWING is fundamentally different from our work as it is trace-oriented, while our work is user-trace-oriented, which we claim allows for more flexibility in the axis of user individuality. Also, there currently does not exist a functional implementation of SWING for researchers to make use of, without substantial modification or reimplementation, and is not being actively maintained by its author [16].

Finally, the work of Weigle et al. in [17] with the Tmix generator most closely mirror our system. They use real traces as input, which are reduced to the bare minimum features (times and sizes) and then replayed for the experiment. While this base functionality is similar to our approach, we make two improvements: (1) we use a much larger corpus of data from real-world deployments to validate our system and (2) by introducing user-trace classification, we can guide the selection of traces which both simplifies the job of the researcher and allows for the investigation of performance in pathological (e.g., highly loaded) configurations.

III. Traffic Generation Framework

The general outline of our framework is diagrammed in figure 1. Starting with a pcap-format trace, such as is generated by the popular tool `tcpdump`, we first identify and split on good clients, omitting trace noise as described in [18]. Each packet in each user trace is reduced to the bare minimum information: (1) frame timestamp encoded as the relative time since the previous frame and (2) the size in bytes. The resulting trace is then divided into upstream and downstream traces. We then create an index of all traces available, which contains some descriptive statistics (such as the trace duration in seconds), as well as classifiers discussed in the next section.

When the user is ready to generate a stochastic simulator application, the “picker” is invoked with a configuration file specifying needs and constraints. For instance, a configuration file may ask for eight active nodes (one Access Point (AP) and seven stations) with a minimum trace-length of one hour and conforming to some classifier constraints. The picker reads in the index, shuffles it, and then attempts to satisfy the needs, rejecting traces that do not fulfill the constraints. In some cases, the configuration is over-constrained, and it will fail. The only solution in this case is to acquire more traces, or to reduce the constraints. On successful completion, however, a random application plan is produced with the requirements requested. Because we have chosen to use the QualNet 4.5 network simulator [19], this is a QualNet application file, but could easily be adapted to any simulator software.

The application plan specifies a unique trace to be replayed for each participating node. The AP node is used as the head node in the network and is the destination or source for all traffic. In this way, a user-trace may make a HTTP request, which will be sent from the client to the AP. When the HTTP response would have returned, the AP will generate traffic back to the client. Because the timings are taken verbatim from real traces, this includes any delay which would have been involved in this exchange.

Since each application plan is stochastically generated, a repeated-measures approach follows easily. The researcher can repeatedly run the picker to generate different plans and then aggregate the results at the point where the sample size is large enough relative to the variance on the variable of interest. All in all, this is a very straight-forward system, which is our principle goal in designing it.

However, simplicity does have its own costs - simulating traffic by replaying traces has some limitations which are worth mentioning:

- Replayed traces cannot respond to network conditions. So, for instance, if your application concerns congestion control, trace replay is probably not appropriate. This concern is raised by [5], and to our knowledge, SWING [15] is the only empirical traffic generator to address it.
- Replayed traces exactly model the network they were captured on - warts and all. Hence, if the network was highly congested, queues and other systems may skew traffic patterns. It should be noted, however, that this level of congestion is rare in practice [20] and can be easily identified and excluded (if desired) using our classifiers.
- Timings are exactly as they were in the original trace. This has the effect of double-counting the time required to transmit the packets on the LAN itself. This occurs because the time spent traversing the LAN is implicitly included in the interarrival times of the user trace. Hence, any additional delay inserted by the simulator will be redundant. We assume this to be negligible for most (comparatively high latency) exchanges. For instance, in the powersaving application we study below, decisions are made on the order of hundreds of milliseconds compared to the tens of milliseconds lost due to double-counting. If needed, a small patch to the simulator software (QualNet in our case) could also solve this issue by not introducing any local transfer delay, and instead assuming this is accounted for in the trace.

If, for one of these reasons, trace-replay is inappropriate for a given task, then the simulation component in figure 1 can be easily replaced with a parameterized simulator, such as SWING [15] while continuing to use the rest of the framework to guide its models[2].

[2]In future work, we intend to explore this sort of hybrid solution in greater depth.

Direction	μ	σ	Neg. Log-Likelihood
Downstream	-4.5086	2.1368	-3.1589e+06
Upstream	-4.2452	2.5450	-2.9732e+06

TABLE I: Lognormal Fit Parameters for Interarrival Times

IV. Classification Systems

In this section, we will describe the user-trace classifiers available to the user. Because we have investigated the usefulness of these metrics with respect to a particular dataset, we will begin by listing the sources of data we used.

A. Data

For our dataset, we use 294 unique publicly available [2] user-traces, which were recorded using passive vicinity sniffing techniques. The three trace sets used are:

- *PDX/VWave2006*: A collection of traces collected at public hotspots around Portland, Oregon including a cafeteria, library, two coffee shops, and a public square. The initial characterization of these traces is in [20].
- *UW/SigComm2004*: A subset of the traces collected by University of Washington researchers at SigComm 2004. Some characterization of these traces is in [21].
- *Microsoft/OSDI2006*: A subset of traces collected by Microsoft researchers at OSDI 2006. Specifically we used the traces from sniffers S4 and S5 concerning two APs.

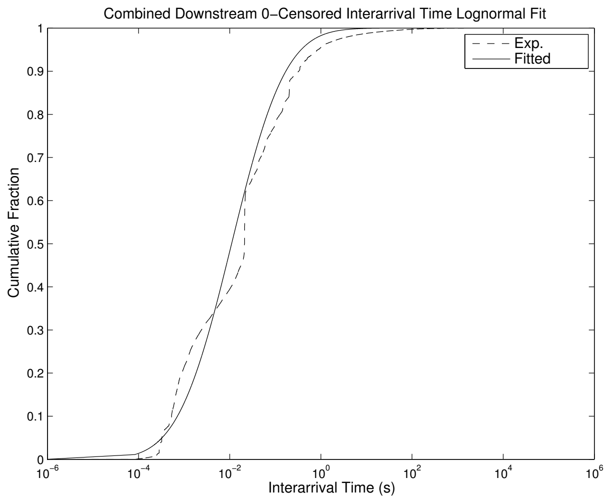
B. Lognormal Interarrival Fits

Our first classifier is based on the observation that the zero-censored packet interarrival times for many of our traces, and for all upstream and downstream data combined across users, is well-fitted by a Lognormal distribution[3]. Figure 2 shows an empirical CDF for the interarrival times of all traces as compared to a lognormal fit and table I gives the fitted parameters. Aside from being an interesting result in its own, this motivates the observation that we can describe the packet arrival process for users in terms of their adherence or deviation from this distribution. Hence, our first classifier is a lognormal fit of the interarrival times in each user-trace. The parameters of this fit can be compared to the global expectations in table I and used to categorize users as being abnormally idle, busy, or bursty.

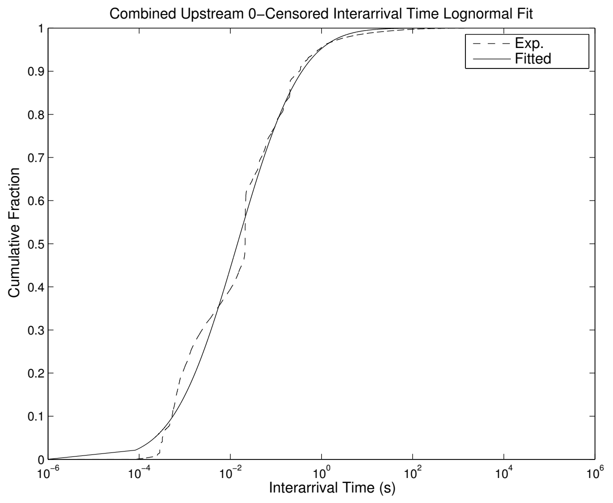
C. Stochastic Similarity Clustering

Our second classifier is based on the stochastic similarity metric introduced by Nechyba et al. in [22], and first applied to trace classification in our work in [18]. This similarity metric provides a convenient number between 0 and 1 which describes the extent to which two observation sequences are similar. Each user trace is converted to a 1-second bucketed 4-state activity trace of the type proposed in [18]. We then perform a factorial similarity analysis of all the user-activity

[3]Unless specified otherwise, all distributional fitting was performed with the Matlab statistics toolbox.



(a) Downstream



(b) Upstream

Fig. 2: Lognormal Interarrival-time Fit CDFs

traces, finding the similarity for each unique pair of user-activity traces. With our data, this results in 42,777 unique combinations of the 294 users, once self-pairs are removed, which takes a substantial amount of time to calculate. We then use the complement of the resulting similarity values as the distances between points in a clustering algorithm. This is demonstrated in the following equation, where d_{ij} is the distance between users i and j and O_i is the observation sequence of user i.

$$d_{ij} = 1 - \sigma(O_i, O_j) = \sqrt{\frac{P_{ji}P_{ij}}{P_{ii}P_{jj}}} \tag{1}$$

The similarity function σ operates on the probabilities, P_{ij}, which are the length-scaled probabilities of the observations O_i given a Hidden Markov Model for O_j learned using Baum-Welch Expectation-Maximization. The details of this are outside of the scope of the present work, but are explained further in [22] and [18].

Because initial investigation shows that the similarity values are mostly bimodal (i.e., many users are very similar and many users are very different), we choose to cluster into three groups - one for each extreme and one for the remaining moderates. The trace index is then augmented with the cluster group for each user and this can be used to select users who are similar or different.

D. Packet Allan Deviation

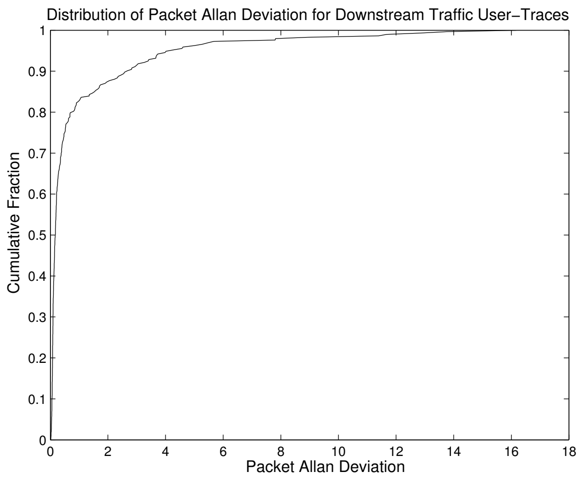
Because traffic burstiness can have a large impact on application performance [1], we decided to include a dedicated metric for this aspect. Allan deviation was introduced in [23] and was used to describe the burstiness of traffic in the RoofNet network in [24]. It differs from normal standard deviation, because it quantifies the variance between successive τ-period samples in an observation sequence $\{x_1, x_2, ..., x_n\}$:

$$\sigma_\tau = \sqrt{\frac{\sum_{i=1}^{n-1}(x_{i+1} - x_i)^2}{2(n-1)}} \tag{2}$$

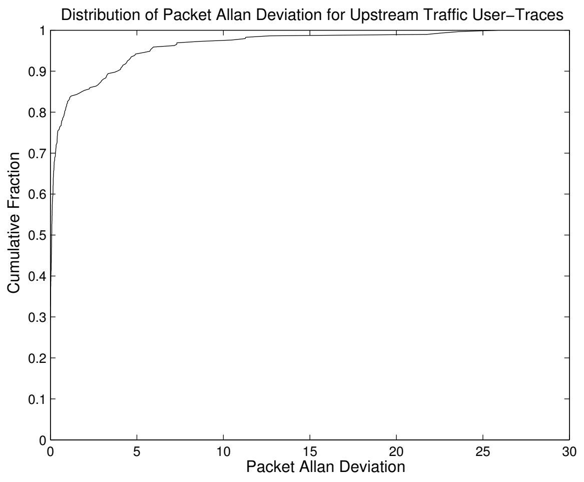
We calculate the Allan variance of the packet process using the Matlab function `avar` by Alaa Makdissi with the sampling period $\tau = 1s$. In other words, for each user trace, we determine the number of packets sent (in the case of the upstream trace) or received (in the case of the downstream trace) in each 1-second interval, and use this as input to equation 2. The resulting values are appended to the user-trace index as our final classifier. Figure 3 shows the distribution of this metric in the user-traces. We can see that packet Allan deviation follows a heavy-tailed (Pareto-like) distribution in the real traces with most users providing relatively calm traffic, and that on average downstream traffic is more bursty than upstream traffic with a mean of 1.16 versus 0.90 respectively. However, despite being more calm on average, the upstream traffic features more outliers with abnormally bursty traffic, four of whom achieve a packet Allan deviation of more than 20.

V. The Case of Bounded Slowdown

In the stock 802.11 power-saving scheme (PSM-Static), the AP is responsible for buffering packets for sleeping stations. In each beacon (approximately every 100ms), the AP sends out a traffic indication map (TIM) which lists which stations have buffered traffic. The station is responsible for listening for these TIMs and when it is ready to receive the buffered data it prompts the AP to start transmission. The station may sleep through one or more beacons depending on the listen interval communicated during association. When the sleeping station has traffic to transmit it can send it at any time. This scheme is static because stations sleep a fixed amount regardless of network conditions.



(a) Downstream



(b) Upstream

Fig. 3: Distribution of Packet Allan Deviation for User-Traces

ID	Type	Description
bursty	Trace Replay	Users in the upper 10-percentile for Allan variance
calm	Trace Replay	Users in the lower 10-percentile for Allan variance
dis	Trace Replay	A uniform selection of the most dissimilar users (i.e., across similarity clusters)
norm	Trace Replay	Uniform random selection of user traces
cbr	Contrived	Constant bitrate randomly chosen between 1 and 40 packets/sec as used in [25]
http	Singe App	HTTP traffic [8] and a maximum think time of 1s as used in [26]

TABLE II: Traffic sources used in analysis of Bounded Slowdown

Bounded slowdown is an alternative power-saving scheme proposed by Krashinsky and Balakrishnan in [26]. As compared to the stock 802.11 power-saving scheme, which is agnostic to user-behavior, Bounded Slowdown (BSD) is tuned for HTTP-like request-response traffic and user think times. At the core, BSD is not complex, it uses a sort of geometric backoff, and makes a point to stay awake for a brief period following any transmission. Specifically, immediately after a transmission it stays awake for b/p seconds where b is the beacon interval of the AP (usually in the area of 100ms) and p is the BSD percent, which is chosen by the implementor. After this brief awake period, BSD attempts to sleep for $t_{sleep} = t_{elapsed} * p$ seconds, where $t_{elapsed}$ is the time elapsed since the last packet transmission. If the device is still idle after this period (i.e., it has awoken naturally), it recalculates t_{sleep} (i.e., the backoff) and goes to sleep again, such that $t_{sleep} <= t_{max}$ with $t_{max} = 0.9s$. Although this is simple, the authors show that it works much better than PSM-Static. They perform this validation by using a single-application packet generator built into NS-2 [27] and based on the work of Mah in [8].

The question we want to ask in this section, is to what extent may the performance results presented in [26] have been biased by the fact that they used single-application traffic generated by a fitted distribution, which serves to ask to what extent does traffic realism effect application layer results in this application. To this end, we implemented the BSD algorithm in the QualNet simulator [19]. QualNet comes with an HTTP traffic generator based on the same work in [8], making a comparison straightforward.

We approached the validation with the attributes of our framework in mind - particularly the use of classifiers to vary user traffic models and repeated measures. We performed a factorial experimental design, trying each of three power-saving schemes: None, PSM-Static, and BSD with various workloads which are summarized in table II. Each simulation run is one hour long, and contains 8 nodes total - one AP and seven clients, uniform randomly placed in a 400x400m square area. We ran each combination with 10 different randomly generated application workloads and each of these with 10 different seeds, resulting in 100 runs with each combination of power-saving scheme and traffic generator. The BSD parameter p was set to 50% in all runs as it appeared to be the winning value in the original analysis of the tradeoff between delay and power saving.

The results of these simulations are in figures 4 and 5. We are looking at two application-specific metrics of performance here. The first, in figure 4, is the cumulative distribution function (CDF) of the percentage of time clients slept in all runs for a given configuration. The top row shows the

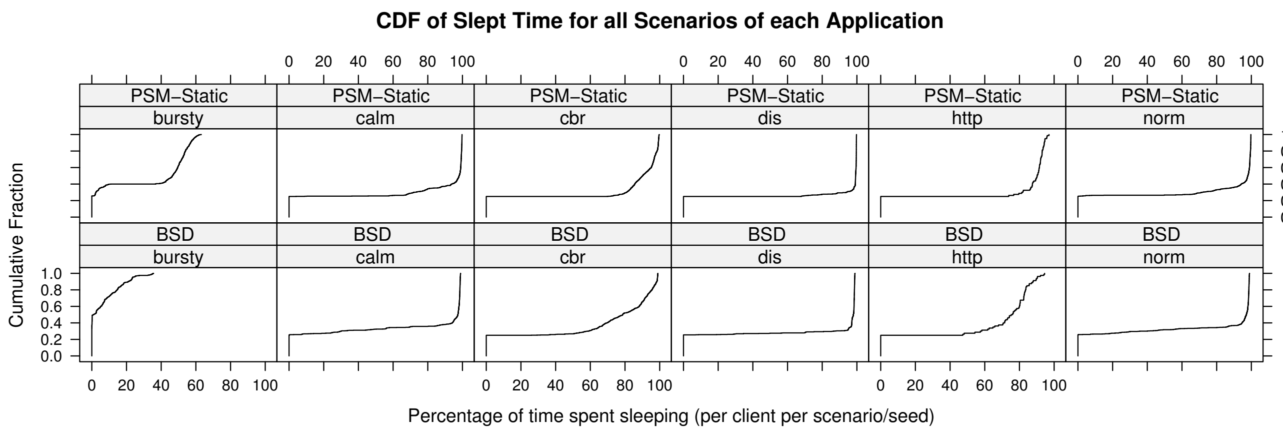


Fig. 4: CDFs of the percentage of time slept in each simulation scenario for all users and all runs

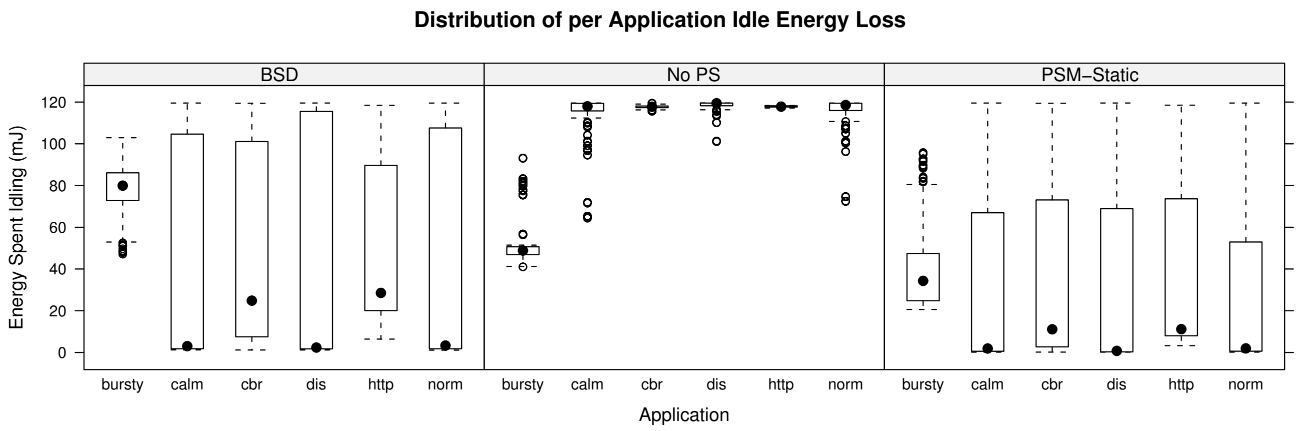


Fig. 5: Power-loss due to idling in each simulation scenario for all users and all runs

performance of PSM-Static and the bottom row shows the performance of BSD. Good performance in this metric is characterized by a CDF with a large proportion of users on the right side of the x-axis (e.g., BSD with *dis*similar traffic). Alternately, poor performing scenarios will have a greater proportion of users on the left side, indicating that they were able to sleep very little (e.g., BSD with bursty traffic). Although there are similarities, it is clear that the HTTP and CBR traffic generates results that differ substantially from all of the results based on real traces. Also, by using the classifiers we are able to see that most users are bimodally distributed between those who sleep very little of the time and those who sleep much of the time. This is especially visible in the pathological case of the bursty users, most of whom succeed in sleeping very little - a use scenario which is not explored in the original work at all due to the limitation of the traffic source.

Figure 5 plots another aspect of the application performance - power loss due to idling, which is a sort of inverse of the metric plotted in figure 4. In this figure, which is a box and whiskers plot, An ideal powersaving scheme would reduce the power spent idling to zero because it will be sleeping the entire time it is idle. This figure plots the extent to which each scenario failed to do this. We can see here that the replay based traffic tells a different story than the CBR and HTTP traffic. In terms of this metric, the real user traces actually spend less power idling on average, but also have results across the spectrum. Looking at this metric, the case for Bounded Slowdown over PSM-Static is not so clear. For instance, PSM-Static outperforms BSD almost universally in the case of primarily bursty users.

Finally, to understand why the application-specific traffic produces these substantially different results, we can apply our classification metrics to its traffic, and study how this differs from values derived from real traces. Firstly, we find that the zero-censored interarrival times of the HTTP traffic are not lognormal. Instead, they follow a Generalized Pareto distribution with the parameters summarized in table III. Next, we calculate the mean Allan variance for both the real traces chosen by our picker, and those generated by the HTTP traffic generator. The difference is quite large. For downstream traffic, the real traces show a mean Allan variation of 1.16 versus 5.62 for the HTTP traffic. For upstream traffic, the real traces have a mean Allan variation of 0.90 versus 3.32 for the HTTP

Direction	k	σ	θ	Log-Likelihood	k Std. Err.	σ Std. Err.
Downstream	0.6512	0.0045	0	10142.2	0.0091	0.0001
Upstream	2.8521	0.0087	0	54034.0	0.0329	0.0018

TABLE III: Generalized Pareto Fit Parameters for Interarrival Times from Model-Generated HTTP Traffic

traffic. Lastly, we apply the factorial similarity classifier to the model-generated HTTP traffic traces. The results show that all the traces are extremely similar with a minimum similarity of 0.99 for both upstream and downstream traffic. In summary, the HTTP traffic generator is creating user traces that are extremely similar, unrealistically bursty, and have abnormal interarrival processes. It appears that the application specific traffic fails to model user individuality and is not showing the whole picture, which serves to explain the application-layer results it is responsible for.

VI. Conclusions

In this paper, we have presented a novel framework for traffic generation in networking simulators and testbeds. In order to narrow our range of study, we have looked at a specific type of wireless network for which there is great interest in the community. Compared to competing systems for traffic generation, our system comports to Occam's Razor, being as simple as possible both in implementation and in its use. We have suggested three classification metrics for user-traces and shown how they can be used to easily generate traffic for both representative and pathological scenarios of particular interest. While trace replay may not be appropriate for every application, we believe that it can be faithfully used to generate realistic background and foreground traffic in a great number of situations. In cases where it is inappropriate, the modular design of our framework allows more complex simulation methods to be used as well. Ultimately, our hope is that the availability of a simple and straight-forward method for traffic simulation will encourage researchers and system designers to validate their proposals with realistic traffic in addition to other application-specific traffic sources.

References

[1] K. V. Vishwanath and A. Vahdat, "Evaluating distributed systems: does background traffic matter?" in *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 227–240.

[2] "Crawdad," http://crawdad.org, January 2009.

[3] E. Rich, "Users are individuals: individualizing user models," *Int. J. Hum.-Comput. Stud.*, vol. 51, no. 2, pp. 323–338, 1999.

[4] F. J. Anscombe, "Graphs in statistical analysis," *The American Statistician*, vol. 27, no. 1, pp. 17–21, 1973. [Online]. Available: http://www.jstor.org/stable/2682899

[5] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 392–403, 2001.

[6] D. Kotz, C. Newport, and C. Elliot, "The mistaken axioms of wireless-network research," Dartmouth College of Computer Science, Tech. Rep., 2003.

[7] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 1998, pp. 151–160.

[8] B. A. Mah, "An empirical model of http network traffic," in *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*. Washington, DC, USA: IEEE Computer Society, 1997, p. 592.

[9] K.-C. Lan and J. Heidemann, "Rapid model parameterization from traffic measurements," *ACM Trans. Model. Comput. Simul.*, vol. 12, no. 3, pp. 201–229, 2002.

[10] Y. chung Cheng, U. Hlzle, N. Cardwell, S. Savage, and G. M. Voelker, "Monkey see, monkey do: A tool for tcp tracing and replaying," in *In USENIX Annual Technical Conference*, 2004.

[11] S. Jin and A. Bestavros, "Gismo: a generator of internet streaming media objects and workloads," *SIGMETRICS Perform. Eval. Rev.*, vol. 29, no. 3, pp. 2–10, 2001.

[12] M. W. Garrett and W. Willinger, "Analysis, modeling and generation of self-similar vbr video traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 269–280, 1994.

[13] A. Mena and J. Heidemann, "An empirical study of real audio traffic," *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 101–110 vol.1, 2000.

[14] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2004, pp. 68–81.

[15] K. V. Vishwanath and A. Vahdat, "Realistic and responsive network traffic generation," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2006, pp. 111–122.

[16] "Private coorespondance with k. vishwanath," March 2009.

[17] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix: a tool for generating realistic tcp application workloads in ns-2," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 3, pp. 65–76, 2006.

[18] C. Phillips and S. Singh, "An empirical activity model for wlan users," in *IEEE INFOCOM MiniSymposium*, 2008.

[19] "Qualnet," http://www.scalable-networks.com/, January 2009.

[20] C. Phillips and S. Singh, "Analysis of wlan traffic in the wild," in *IFIP Networking*, Networking.

[21] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan, "Measurement-based characterization of 802.11 in a hotspot setting," in *E-WIND '05: Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*. New York, NY, USA: ACM, 2005, pp. 5–10.

[22] M. C. Nechyba, S. Member, Y. Xu, and S. Member, "Stochastic similarity for validating human control strategy models," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 437–451, 1998.

[23] D. Allan, "Time and frequency (time-domain) characterization, estimation, and prediction of precision clocks and oscillators," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, vol. 34, no. 6, pp. 647–654, Nov 1987.

[24] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 121–132, 2004.

[25] M. Takai, J. Martin, R. Bagrodia, and A. Ren, "Directional virtual carrier sensing for directional antennas in mobile ad hoc networks," in *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*. New York, NY, USA: ACM, 2002, pp. 183–193.

[26] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access with bounded slowdown," *Wirel. Netw.*, vol. 11, no. 1-2, pp. 135–148, 2005.

[27] "ns-2," http://www.isi.edu/nsnam/ns/, January 2009.