Trace based application layer modeling in ns-3

Prakash Agrawal and Mythili Vutukuru
Department of Computer Science and Engineering, Indian Institute of Technology, Bombay
{*prakashagr, mythili* }*@cse.iitb.ac.in*

***Abstract*—Ns-3 is a widely used as a the network simulator of choice by researchers. It contains many well tested and high quality models of network protocols. However, the application layer models of ns-3 are very simplistic, and do not capture all aspects of real life applications. As a result, there is often a huge gap between the results of real experiments and the corresponding simulations. This problem is particularly exacerbated for wireless simulations, where many networking phenomena like wireless channel contention crucially depend on the application traffic characteristics. One way to bridge the gap between experiments and simulations is to incorporate knowledge from network traces into simulations. To this end, our work builds a trace-based application layer simulator in ns-3. Given a network trace collected from a user, our TraceReplay application layer model automatically generates traffic that is faithful to the real application in the ns-3 simulator. TraceReplay infers and replays only application layer delays like user think times, lets the simulator control the lower layer phenomena. TraceReplay extracts application layer characteristics from a single trace, and replays this information across many users in simulation, by using suitable randomization. Our model is also generic enough to replay any application layer protocol. Validation of our simulation model shows that simulation results obtained using TraceReplay are significantly different from those using other models, and are closer to experimental observations.**

1. Introduction

Ns-3 is a widely used open-source discrete-event network simulator. It contains a large number of high-quality network models for all layers of the network stack, using which it is very easy to build various network scenarios and simulate them. While ns-3 provides a large number of models for the transport and lower layers, it provides only a limited choice at the application layer. The most commonly used application layer models in ns-3 are BulkSend, which generates uni-directional bulk TCP traffic, and OnOff, which generates CBR traffic with on and off times. None of these models correspond exactly to any application (e.g., HTTP). Furthermore, these models assume that there are no parallel TCP connections between any pair of nodes, and that there are no variable user-side or application-layer delays.

Real life applications, however, are much more complex. For example, when downloading a file from a webpage, there are often several parallel connections between the web client and server. In addition to the file being downloaded, HTTP requests and responses are also generated for a large number of other objects on the webpage. Using BulkSend to model this complex traffic seems very simplistic, and can fail to account for several interesting networking phenomena. For example, previous research [6] has found that the small amount of upload traffic generated while downloading files from a webpage over WiFi can significantly increase wireless contention, leading to much longer download times than using just one TCP connection for download. This divergence between simulation and experimental results makes it hard to run realistic network simulations, and casts a doubt over the applicability of research evaluated only via simulations. However, simulations are vital to quickly testing out new ideas, and it is therefore critical to strive for more realistic network simulations.

In addition to being far removed from reality, the application layer models of ns-3 also require manual setting of suitable parameters. For example, a researcher wishing to model a new application (e.g., an IoT smart meter) must manually select a suitable model (e.g., OnOff) and must set its parameters (e.g., on time and off time) manually.

The key insight of our work is that a network trace (pcap) collected during an experiment contains significant information about application layer traffic, and finding a way to *replay* network traces within a simulator can enable generation of realistic application layer traffic with minimal manual effort. To this end, we have built and tested a new application layer model called TraceReplay in ns-3. To use TraceReplay, a user collects a pcap trace on a single machine while the application is running. The user then sets up an ns-3 simulation with the TraceReplay application layer model, providing the network trace as input. Note that a single network trace can be used to generate traffic at multiple nodes in a simulation, because TraceReplay can be made to introduce some randomization in the trace.

To correctly generate application layer traffic, TraceReplay collects information about the various transport connections between nodes, the amount of data sent and received on each connection, packet sizes, and inter-packet delays. It is very important to note that TraceReplay carefully identifies and replays *only* the application layer delays (e.g., user think times) from the trace. Other inter-packet delays, such as transport or network layer delays, are dictated by the conditions within the simulator, and will not be (and, should not be) replayed from a trace. Therefore, the network conditions in the simulator can be very different from the

network conditions when the trace was collected. The logic of TraceReplay is agnostic to the actual application layer protocol being used, so that it can replay a trace with *any* application layer protocol based on TCP (e.g., HTTP or SSH), enabling easy modeling of several applications. TraceReplay faithfully models multiple parallel connections between hosts (as with parallel connections in HTTP). It uses heuristics to identify and preserve the causal dependencies (e.g., bytes corresponding to a HTTP request must be sent before the bytes of the response) between data transfers within one connection and across connections.

There are several advantages of using TraceReplay. In addition to the convenience of not having to worry about picking a suitable model and parameters, TraceReplay also leads to simulation results that are closer to reality. TraceReplay is particularly useful in conducting wireless simulations, because traffic characteristics greatly influence wireless channel contention and other phenomena, making it crucial to get the application traffic model right.

We have built and integrated the TraceReplay application layer model with the latest release of the ns-3 source code. We validated our model by verifying that the traffic generated by the model in simulations closely matches the actual traffic generated in experiments. We have presented results for two application layer protocols—HTTP and SSH—but our model has no protocol-specific code, and hence generalizes to any application layer protocol. Furthermore, we show that simulations with TraceReplay produce results that are closer to real life that those using other application layer models.

2. Related Work

There has been prior work on generating realistic HTTP traffic in ns-3. Web traffic generator [5], 3GPP FTP traffic simulator [1], transactional traffic generator [4] and other similar work use various HTTP parameters (e.g., file size distribution, think time), obtained either from traces or from user input, to generate synthetic traffic in the simulator. Our work differs from these papers in several ways. In prior work, the parameters once learnt or set stay constant throughout the simulation. Further, none of these models support parallel TCP connections between HTTP clients and servers. Finally, the approaches in these papers cannot be easily generalized to other applications.

The idea of using network traces to aid simulations and emulations itself is not new. For example, T-RATE [3] presents a model for using network traces to simulate better rate adaptations algorithms at the MAC layer of ns-3. Mahimahi [7] presents a framework for recording HTTP traffic and later replaying it in experiments (i.e., not in a simulator) under different network conditions. Our work was inspired by these and similar efforts.

Tmix [8] is probably the work closest to ours. Tmix replays packet traces in the ns-2 network simulator, much like TraceReplay. However, the significant difference between Tmix and our work is that TraceReplay carefully identifies are preserves causal dependencies and ordering of packets across multiple connections, while Tmix replays traffic in each TCP connection independently.

3. Existing application layer models in ns-3

The BulkSend application in ns-3 models a single unidirectional TCP/UDP flow between source and sink nodes in the simulation. A BulkSend source tries to send the specified amount of data as fast as possible, using MSS-sized packets, without any application layer delays. BulkSend is a good choice to model long-running TCP flows, large file downloads over FTP, and so on. The OnOff application layer model is similar to BulkSend, with the difference that the traffic generation has an "on" time and an "off" time. Data is generated at a specified rate during on periods, and no data is generated during off periods. Finally, ns-3 also has a UDP trace replay application layer model that can be used to replay MPEG4 stream trace files.

There are many shortcomings of the existing application layer models, due to which they may not be able to produce realistic simulation results. We discuss some of them below.

Unidirectional flow of traffic. In all the application layer models discussed above, the flow of traffic is always from the server to the client. However, most real life applications have bi-directional data flows (e.g., HTTP request and response), that are often causally dependent on each other, making it hard to model such applications with two unidirectional flows. For example, even if one were to use two BulkSend applications, one in each direction, between a simulated HTTP client and server, there is no way to guarantee that the bytes corresponding to the HTTP request in one direction will be sent before the bytes of the HTTP response in the other direction.

Fixed packet sizes and inter-packet gaps. BulkSend always sends MSS-sized packets, while real life applications display a wide variety of packet sizes. Further, one can only introduce a fixed application-layer delay between packets using the "off time" knob of the OnOff application, while real applications often exhibit highly variable inter-packet gaps at different stages of the application processing.

No support for parallel connections. Finally, none of the application models support multiple parallel transfers between two nodes, a feature that is a hallmark of modern application layer protocols like HTTP.

4. TraceReplay Application

We now describe the design of our TraceReplay application layer model. To use TraceReplay, a user must run the desired application at a single host, and collect a trace (pcap) file. This trace can then be replayed at multiple nodes in ns-3 using randomization. When a trace is replayed at a certain simulated node, the traffic sent and received by the simulated node will be similar to that of the host on which the trace was collected.

To use a certain trace in ns-3, the user creates a TraceReplay application and provides it a path to the pcap, a

parameter for randomization, and the addresses for the two endpoints of the application. The TraceReplay application generates traffic between two simulated network nodes—the node that initiates the first packet in the trace is called the client, and the other end point is the server. Note that the collected trace file may have traffic from the local host to several remote servers—all of these remote servers are abstracted as one simulated server during replay.

Our implementation of the TraceReplay application layer model adds three classes and about 1500 lines of code to the ns-3 codebase. The `TraceReplayApplication` class is responsible for parsing the input trace file, and scheduling appropriate data transfers between the `TraceClient` and `TraceServer` classes.

What information does our model replay? TraceReplay collects information about all the TCP connections between the client and server, and starts the connections at the time specified in the trace. On every connection, it generates packets of appropriate sizes as found in the trace, and transmits them one after the other, as fast as the simulated network allows. Note that TraceReplay ensures that there is enough space in the transmit buffer of the simulated socket to send the packet of the specified size, and defers transmission in case of insufficient buffer space.

TraceReplay can be used to replay any application layer protocol that runs atop TCP. Our evaluation replays primarily HTTP and SSH traffic, but nothing in the model limits the usage to these two protocols. TraceReplay currently does not support UDP-based application layer protocols.

While TraceReplay preserves packet sizes found in the trace, it does not strive the maintain the inter-packet gaps seen in the trace, *except* for the gap is inferred to be an application layer delay. Section 4.1 describes how application-layer delays are inferred by our model. This selective replaying of only application layer delays ensures that other network delays are governed by the network conditions in the simulator, which can be very different from the conditions when the trace was collected.

In addition to identifying and replaying application layer delays, TraceReplay also tries to maintain the causal ordering of packets within a connection and across connections. For example, the HTTP response bytes from the server must only be sent after the corresponding HTTP request bytes arrive on the same connection or on a different connection. Inferring such constraints from the trace and respecting them will ensure that timing of packet transmissions in the simulation closely matches that in the trace. Section 4.2 describes how we infer and enforce such ordering constraints.

4.1. Calculating application layer delays

We define an application layer delay as that caused due to application layer semantics or user think time, and not due to delays at the transport, network, or lower layers. We replay two kinds of inter-packet delays in TraceReplay.

Delays between HTTP requests. Once a HTTP client makes a GET request and fetches a page, the user may take time to read the page before clicking on another link and generating another GET request on the same TCP connection. Therefore, we replay any inter-packet gap between the first packet of a HTTP request and the packet immediately preceding it on the same TCP connection. We note that not all such delays correspond to user think times (e.g., multiple GET requests can be generated by a single click); however, we replay all such delays conservatively.

Delays greater than a threshold. While the above heuristic can be used to identify user think times for HTTP, the approach does not generalize to other applications. Therefore, we need another heuristic to identify application layer delays across applications. We hypothesize that application layer delays (e.g., delay between packets in an SSH flow caused due to the user typing the next command) are typically in the oder of seconds, while network delays are typically much shorter, even for longer network RTTs. To verify this hypothesis, we collected inter-packet delays for SSH traffic, for varying network RTTs. We varied the network RTT in the experiment using `tc` and `netem`. Figure 1 shows the inter-packet gaps during the SSH session, with varying RTTs. We manually looked over the traces and verified that all delays over 1 second always corresponded to user pausing to type a command. Therefore, a simple threshold on inter-packet gaps can almost always identify user think times from other network delays.

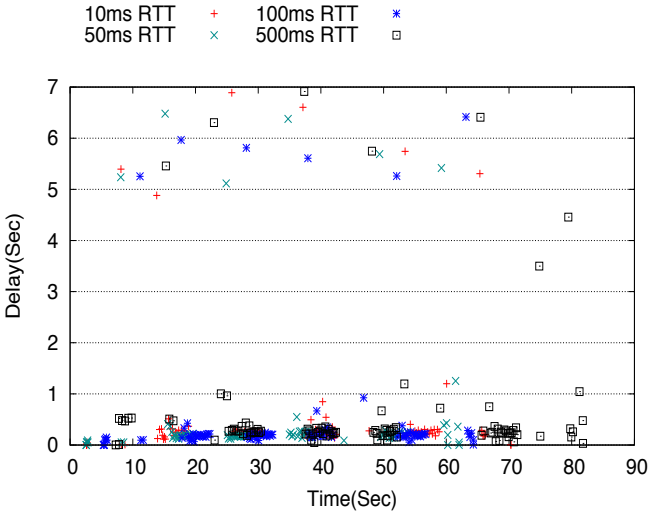

Figure 1. Time difference between two packets

Our TraceReplay model replays delays that match one of the above two heuristics, and does not replay any other packet delay. We also do not replay packet delays of retransmitted packets, because retransmissions could signal a network issue.

When replaying the same network trace at multiple nodes in simulation, we use the randomization parameter provided to TraceReplay to randomize application layer delays. Given randomization parameter R, TraceReplay adds a random value $r \in (-R, R)$ to application layer delays, and connection start times, in order to avoid synchronization between multiple nodes.

4.2. Ordering packets between client and server

When replaying packets in a single TCP connection, TraceReplay does not replay each of the two directions (client to server, and server to client) independently, because the bytes from the server (e.g., HTTP response) could be causally dependent on the bytes from the client (e.g., HTTP request). Therefore, TraceReplay maintains the property that if a packet X in one direction of the TCP flow occurred after another packet Y in the other direction in the original experimental trace, then it will be replayed in the same order in the simulation as well. Therefore, while the exact inter-packet gaps in simulation may differ from that in the trace, the relative ordering of packets across both directions of the connection will be preserved.

Causality can exist not just between packets of one connection, but also across multiple TCP connections. For example, a HTTP client may send a GET request for an image file on a TCP connection only after receiving the HTTP response specifying that image's URL on another connection. TraceReplay tries to preserve causality across connections as follows. Whenever an application layer delay is detected between two packets of a certain TCP connection (using the heuristics of Section 4.1), TraceReplay checks if any other connections exist between the same source and destination pairs in the trace. If other parallel connections exist, TraceReplay compares the number of packets sent on each of those connections in the trace before the current point of time with the actual number of packets sent on those connections in the simulation. If any of those connections is found to have sent fewer packets in simulation than in the experimental trace, TraceReplay delays the transmission of the current packet to a point where all parallel connections have made sufficient progress. These constraints of causality introduce an additional delay when replaying packets, and serve to pace packet transmissions in a realistic fashion, over other applications layer models like BulkSend that send data as soon as the network allows.

5. Validation and Observations

We now validate our simulation model, to verify both its correctness and usefulness, in the following two ways: (i) we replay an experimental trace collected at a single client in a single client simulation, and verify that TraceReplay is able to correctly replay the trace; and (ii) we run simulations with large number of nodes using TraceReplay and compare our results to similar ones using other application layer models. We show that `TraceReplay` produces significantly different and more realistic simulation results, especially in simulations with wireless bottlenecks.

To collect pcap traces, we run a variety of applications on a laptop connected to a WiFi access point, and collect traces of individual applications and activities. We then replay these traces in ns-3 using a topology as shown in Figure 2. Our simulation setup consists of one or more WiFi clients connected to a remote server via an AP. All WiFi clients replay the pcap collected in the experiment (with suitable randomization). The server node in simulation replays the traffic from all remote destinations in the collected pcap. Our experiment and simulations all run on 802.11g.

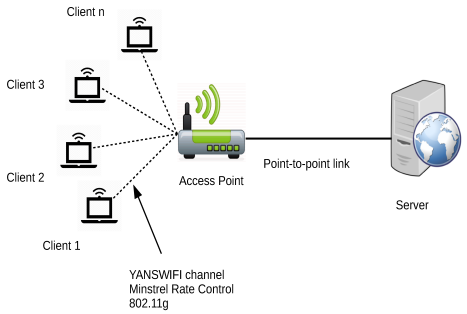

Figure 2. Test setup

5.1. HTTP download from local server

Our first experiment consists of a user downloading video lectures from a course website hosted on a local web server in our university campus. The user opens his web browser, authenticates himself, browses the website for content, and then downloads a 7 MB lecture video. Other applications like email are running in the background, and user also browses other sites for short periods of time while the download is ongoing. A network trace was collected during this entire activity of the user. This trace represents typical traffic that WiFi networks see in classrooms. The network trace consisted of mainly HTTP traffic, with a total of 56 TCP connections to various remote hosts.

We now setup a simulation with a single WiFi client connected to an AP, and replay the trace on the client and the server. The bandwidth and delay of the point-to-point link from the AP to the server are set to 100 Mbps and 5 ms respectively, because the actual server in the experiment was connected over the LAN to the AP. We now compare the traffic generated in the simulation by TraceReplay to the traffic seen in the actual trace, to validate the correctness of TraceReplay. Figures 3 and 4 show the upload and download traffic (aggregated over 0.1 sec intervals) generated by the client in the experiment and with TraceReplay. We find that the traffic generated in simulation matches very closely to that in the experimental trace. Further, this realistic traffic was generated in simulation with very little effort, i.e., without the user having to set any parameters for any models.

Does a realistic replay of traffic translate to significantly different simulation results? To answer this question, we run simulations comparing TraceReplay with the BulkSend application layer model in ns-3, with varying number of clients. For the TraceReplay simulations, we replay the single trace collected above at all clients, with a suitable randomization parameter. For simulations with BulkSend, we generate traffic with two BulkSend connections, one for the upload and the other for download. The parameters of BulkSend (like total bytes to send) are derived from the experiment trace, in order to enable a fair comparison. Figure 5

shows the total time taken to fetch the video lecture from the course server in simulation, for both when simulation traffic was generated with TraceReplay and with BulkSend, as we vary the number of clients from 10 to 50. In all cases, we find that TraceReplay leads to higher download times than BulkSend. The download times generated from TraceReplay are found to be closer to what the course instructor observed (for comparable class sizes in real life), and also match results on TCP performance in large classrooms in our previous research [6].

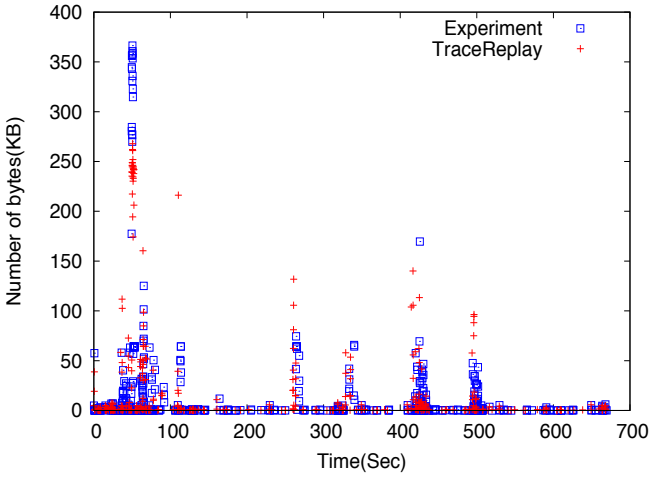

Figure 3. Download traffic when fetching content over HTTP from local server.

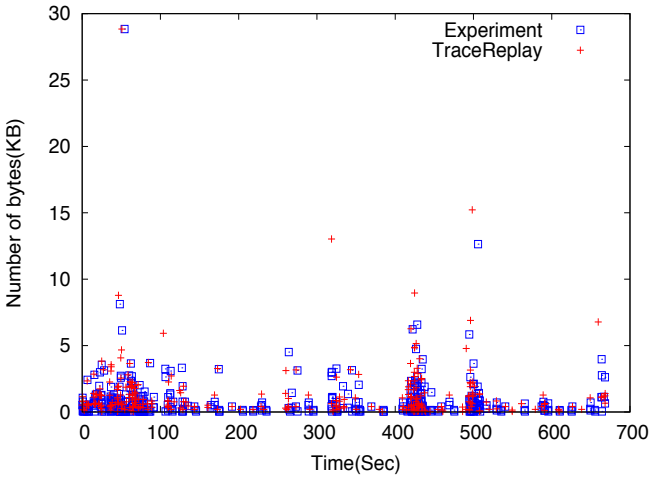

Figure 4. Upload traffic when fetching content over HTTP from local server.

Why does TraceReplay lead to higher download times as compared to BulkSend? One reason is that TraceReplay accurately captures user think times and other application layer delays. It also delays sending packets to preserve causal ordering of packets, much like real applications. The most important reason, however, is that the traffic generated by TraceReplay introduces significantly higher contention on the wireless channel as compared to the traffic from BulkSend. With BulkSend, the small amount of upload

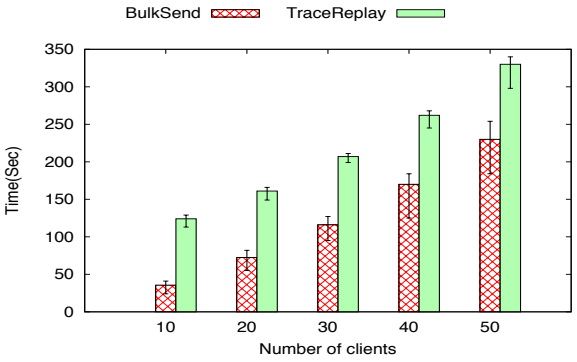

Figure 5. Total download time (min, max, avg) when fetching content over HTTP from local server, for TraceReplay and BulkSend

traffic proceeds independently from the large download, and lasts for a shorter duration, unlike in the real trace. As a result, for most part of the simulation, the contention on the wireless channel was very low. With TraceReplay, however, the upload traffic overlaps significantly with the download (much like in the actual experiment), and adds a constant "chattiness" on the wireless channel. This increases contention, leading to more wireless collisions and losses, resulting in higher download times.

5.2. HTTP download from remote server

Next, we perform another HTTP download experiment, but this time, from a remote server outside our campus. We downloaded a 26MB ns-3 tarball from the ns-3 website, and collected a pcap. Unlike in the previous experiment, the wired network from the AP to the server (not the wireless network) was the bottleneck. As a result, we set a suitable value of bandwidth and delay on the point-to-point link between the AP and server in our experiments to reflect this situation. We replayed the pcap at a single client in simulation and found that the pattern of traffic generated closely matches the trace. We skip those results due to lack of space.

Next, we run simulations with varying number of clients, using both TraceReplay and BulkSend as application layer models, and measure the time taken to complete the download in simulation. Figure 6 shows the download time of the file for varying number of clients, for the two application layer models. In this scenario, we find that BulkSend and TraceReplay produce similar download times. Even though TraceReplay carefully replays application layer delays, and recreates the right mix of upload and download traffic contention on the wireless link, the download time is this scenario is largely determined by the rate of the wired network bottleneck link, and not by other factors. As a result, the features of TraceReplay make no significant difference to the end result. This experiment is informative on when TraceReplay is useful and when it is not.

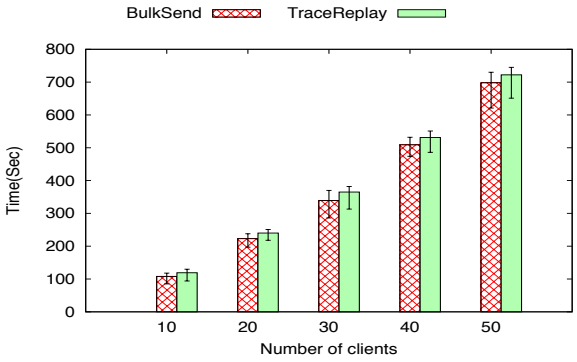

Figure 6. Total download time (min, max, avg) when fetching content over HTTP from remote server, for TraceReplay and BulkSend

5.3. SSH traffic

Finally, we validate our claim that TraceReplay can be used to replay any application layer protocol. We collect a network trace with a user running commands over an SSH session to a remote server. We then replay this trace in simulation. In the absence of TraceReplay, as SSH session would have been modeled using the OnOff model in ns-3, as it is a low rate application, with the OnOff model parameters being set manually. We run simulations of a single client running SSH, using both TraceReplay and OnOff, with suitable parameter setting for OnOff. We find that the inter-packet delays produced by TraceReplay match the real trace more closely, as TraceReplay can generate variable inter-packet gaps, while OnOff can only model fixed inter-packet gaps. Figure 7 shows the CDF of the difference in inter-packet gaps in the experiments, for TraceReplay and OnOff simulations. We can see that while the difference in inter-packet gaps is always close to 0 for OnOff, the CDF of TraceReplay closely matches that of the experiment. Next, we run simulations with varying number of SSH clients. Figure 8 shows the average jitter observed by the SSH clients as a function of number of clients in the wireless network, for both TraceReplay and OnOff. We find that clients suffer higher jitter on an average with TraceReplay, due to its ability to accurately model variable packet gaps caused by user think times and wireless contention.

6. Conclusion

Our work described an application layer packet generation model for ns-3, that uses an experimental pcap trace file to generate realistic traffic for simulations. Our TraceReplay application layer model is easy to use, and can be used to generate traffic of a large variety of application layer protocols. TraceReplay uses heuristics to identify application layer delays and causal relationships between packet flows, and carefully replays these in simulation, while letting the simulator dictate the lower layer behavior. Simulations using TraceReplay show markedly different results from simulations using other ns-3 application layer models, and

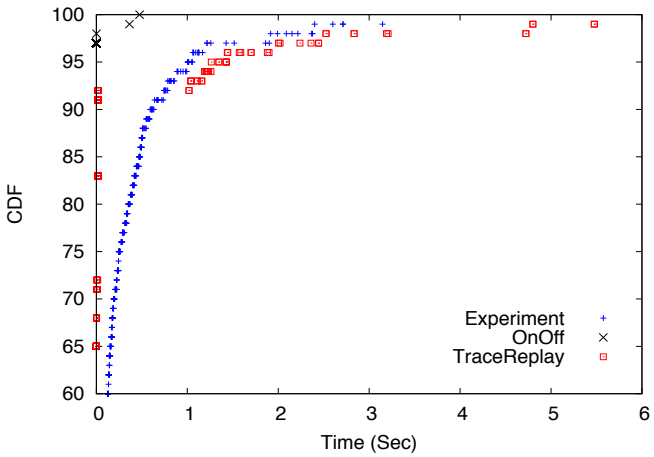

Figure 7. CDF of difference in inter-packet delay for a single SSH client.

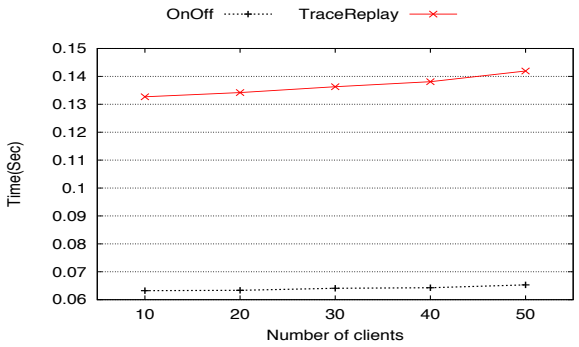

Figure 8. Average jitter with varying number of SSH clients.

are closer to experimental results. TraceReplay is particularly useful in simulating wireless networks, because traffic characteristics have a significant impact on wireless channel contention, and small improvements in traffic models can greatly enhance the realistic nature of simulation results.

References

[1] 3GPP FTP traffic models — http://www.ict-codiv.eu/private/docs/deliverables/d5.4.pdf, 2007.

[2] Ns-3 documentation — www.nsnam.org/ns-3-23/documentation/, 2015.

[3] Ali Abedi and Tim Brecht. T-RATE: A Framework for the Trace-Driven Evaluation of 801.11 Rate Adaptation Algorithms. In *MASCOTS*, Sept, 2014.

[4] Yufei Cheng, Egemen K. Cetinkaya, and James P.G. Sterbenz. Transactional Traffic Generator Implementation in ns-3. In *SimlTools*, 2013.

[5] Hyoung-Kee Choi and John O. Limb. A Behavioral Model of Web Traffic. In *ICNP*, Nov, 1999.

[6] Mukulika Maity, Bhaskaran Raman, and Mythili Vutukuru. TCP Download Performance in Dense WiFi Scenarios. In *COMSNETS*, January, 2015.

[7] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: accurate record-and-replay for http. In *USENIX ATC*, 2015.

[8] Michele C. Weigle, Prashanth Adurthi, Felix Hernandez-Campos, Kevin Jeffay, and F. Donelson Smith. Tmix: a tool for generating realistic TCP application workloads in ns-2. In *ACM CCR*, 2006.