Transactional Traffic Generator Implementation in ns-3

Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz
Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045, USA
{yfcheng, ekc, jpgs}@ittc.ku.edu

ABSTRACT

Traffic generators have been essential for representing real-world traffic in network simulation studies. Furthermore, *Internet-like* traffic is a necessity when analysing the impact of different kinds of traffic on the network. In this paper, we present the implementation details of our HTTP traffic generator in the ns-3 network simulator. It is able to generate *Internet-like* as well as *user-defined* HTTP traffic. We further verify the correctness of the traffic distribution function generation module as well as the transaction handling mechanisms in this model. Based on different network characteristics from previous work, we are able to generate similar simulation results and carry out more detailed HTTP simulations.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: General, Model Development, Model Validation and Analysis; C.2.2 [**Computer-Communication Networks**]: Applications— *HTTP traffic model*

General Terms

Implementation, Analysis, Testing, Verification

Keywords

HTTP persistent, pipelined implementation, traffic distributions, ns-3 simulation, Internet traffic

1. INTRODUCTION

A transactional traffic generator is an essential part for network simulation of the Internet. It is responsible for injecting synthetic traffic into the simulation according to a model of how application users would behave in certain circumstances. As the major contributor of transactional traffic, Hypertext Transfer Protocol (HTTP) [4, 11] is a pervasive application protocol and consumes a significant share of application flow in the Internet. Web traffic has transformed from plain-text Web pages to large size pages with embedded objects. An HTTP traffic model is needed to accurately represent and simulate Web traffic with the sustaining influence of HTTP over the Web.

Simulation has become the backbone for network traffic research, since the simulation environment provides easily accessible resources to study new protocols and models. As an open-source simulator, ns-2 [1] has been widely employed in the academic research community. However, in response to a number of deficiencies, the ns-3 discrete event network simulator [2] is proposed as its successor and is still under development. The ns-3 network simulator provides greater flexibility, evolvability, modularity, and the support of heterogeneity including hybrid wired and wireless models.

Despite its advantages, ns-3 is relatively new with some essential models missing from its release distribution [24]; existing built-in traffic generators are limited to *BulkSendApplication* for bulk data transfer and *OnOffApplication* for constant bit rate (CBR) application. To have a fairly complete set of traffic generators for simulations, we implemented this HTTP traffic generator [22] and contributed to a more complete ns-3 protocol list. There are a set of different methodologies supporting the development of network Web traffic generators: *page-based*, *behavior-based*, and *connection-based* [5, 9]. We chose the *connection-based* model for our traffic generator implementation. We extend and modify the source variable generation model from the Packmime-HTTP traffic generator [5] in ns-2 to work with ns-3. Furthermore, we add an extra working mode to our generator, which is the *user-defined* mode. In this working mode, users will be able to provide source variables to the HTTP transactions and be able to analyse the simulation scenarios they intend to test.

In this paper, we present our ns-3 implementation of the HTTP traffic generator and analyse its performance under different network characteristics. The rest of the paper is organised as follows: We present related work about application traffic models in Section II. Section III presents the implementation details of the HTTP traffic generator. We present validation results and performance results in Section IV. Finally, we conclude the paper in Section V.

2. BACKGROUND AND RELATED WORK

The HTTP protocol is now the de facto content delivery protocol, analysing its unique characteristics as well as its requirement on the network systems is required. However, to be able to develop an accurate HTTP traffic generator, we must first understand Web traffic characteristics. In the following section, we introduce the Web traffic features and the methodologies we use to develop our HTTP generator.

2.1 Web Traffic Characteristics

Web access is frequent, short request-response transactions based on bursts of many small Web requests and responses. Response messages are small to keep transmission time down; embedded objects per Web page and the frequency of smaller objects are increasing [21]. Although there are some very large responses, 85% of all responses are 10,000 bytes or less [12]; over 90% of the requests are between 100 and 1,000 bytes in size. There is also a year-to-year increase in the number of embedded objects on each Web page [21]. To make the burstiness worse, Web users tend to switch rapidly from site to site, as can be verified from both client [10] and server side traces [3]. In addition, there are other user interactions such as clicking the browser *stop* or *reload* buttons while one page is loading [21]. All these characteristics contribute to the bursty nature of Web traffic.

Several works have shown that Web traffic is statistically self-similar [9, 16, 19], that is bursty on several or all time scales. To be able to accurately represent Internet traffic, we need to generate traffic that carries self-similarity features. We choose the source variable generation model from PackMime-HTTP [5], which has been verified to be able to generate self-similar traffic that matches real trace data.

2.2 HTTP Protocol

HTTP is a stateless application protocol using the client-server paradigm. Its operation is transactional as the client sends one or a series of requests to the server, and the server responds to the request with response messages. HTTP is an application layer protocol and presumes a reliable TCP transport layer for end-to-end data transfer. The original version HTTP 1.0 [4] uses a separate connection for each request-response transaction. HTTP 1.1 [11] introduces three major performance enhancement mechanisms, with the first one using *persistent connections* to reduce the latency by allowing several request-response messages over a single TCP connection. The second performance enhancement in HTTP 1.1 is the *pipelining* of a series of requests on a persistent connection without waiting for responses for the previous requests. Note that the *pipelining* option can only be enabled when the *persistent connection* option is used. Thirdly, the *parallel connection* option is another important enhancement. It allows multiple transport connections opened for one Web page transaction and further lowers the loading latency. Our model does not support this later option in this release. We plan to incorporate it in the next release and compare its performance with that of other options.

2.3 Transactional Traffic Generators

There are mainly three types of transactional traffic generators: *page-based*, *behavior-based*, and *connection-based*. Page-based traffic generators only focus on the Web page details and fail to represent other major characteristics of HTTP traffic [21], such as the server delay time and Web request gap time. On the other hand, behavior-based generators [8] simulate the ON/OFF states in which the ON state represents active Web-request and OFF state represents the silent period after all objects are retrieved. However, these models fail to accurately represent the transport connection characteristics, such as the gap time among opening new transport connections.

Connection-based models [5] provide a better alternative and their advantages have been shown [13, 23]. A connection-based implementation models TCP connections in terms of connection establishment rates, request/response data sizes, and gap time among objects within the connection. Synthetic traffic is injected into the simulation according to the distribution models of how users would behave in Web browsing activities. We developed our HTTP traffic model based on TCP connections between Web servers and clients, with each node acting as either server, client, or both. The model can run over both wired and wireless networks simulated with node mobility.

Scalability is another major factor for HTTP traffic generation to cope with ever-changing Web traffic characteristics. The *page-based* model [15] successfully captures Web traffic behavior when the model was implemented; however, it focuses on user-browsing behavior and constructs detailed Web pages. As Web traffic continues to change, the model fails to represent new Web browsing behavior as well as increasing size of embedded objects in Web pages. Therefore, a traffic model with scalability is necessary to capture Web-browsing behavior, as well as to predict its behavior in the future. Furthermore, peer-to-peer traffic is gradually taking a leading role in network traffic. The *connection-based* model [5] is scalable to Web traffic evolvement and and can incorporate a peer-to-peer traffic model when needed.

Based on the *connection-based* model [5], the PackMime-HTTP model in the ns-2 network simulator has been validated at packet-level by comparing synthetic traffic generated from simulation with measured. However, the model is problematic in that it treats a collection of clients as a single client and a collection of servers as a single server. For example, a campus network that contains 100s of clients and tens of servers will be abstracted to one server and one client in this model. This is a useful simplification in the case of wired network since it can easily replicate a campus local network connecting to the Internet. However, when simulating wireless network, especially after introducing mobility to the hosts, the simplification in this model is too restrictive.

Therefore, we use the *connection-based* model from the Pack-
Mime-HTTP generator with one modification to take individual Web servers and clients into consideration. One node can act as either server or client based on the start-time configuration of the simulation. This is advantageous as we can simulate both wired and wireless networks, with or without node mobility. For example, we can install the HTTP traffic generator in some of the nodes in the network, while running other traffic types in the remaining nodes, which gives the traffic generator more flexibility in simulation process.

3. TRAFFIC MODEL IN ns-3

This section describes the traffic model in our implementation of the HTTP traffic generator. This model is able to generate HTTP 1.0 traffic as well as HTTP 1.1 traffic with *persistent connection* and *pipelining*. The only feature lacking for HTTP 1.1 is the parallel TCP connection option planned the next code release. All the major attributes used in this implementation are listed in Table 1. The relationships among all of the classes implemented for this model

Table 1: HTTP model attributes and their default values

Attribute	Defaults	Summary
MaxSessions	10	Number of Web sessions for the http simulation
InternetMode	*true*	Working mode is Internet-like or user-defined
Persistent	*true*	Connection is using persistent or not
Pipelining	*true*	Connection is using pipelining or not
UserNumPages	2	User defined number of Web pages for each Web session
UserNumObjects	2	User defined number of Web objects for each Web page
UserServerDelay	0.1 s	User defined server delay time to send out the Web response
UserPageRequestGap	0.2 s	User defined request gap time between two adjacent Web page requests
UserObjectRequestGap	0.01 s	User defined request gap time between two adjacent Web object requests
UserRequestSize	100 B	User defined Web request size
UserResponseSize	2048 B	User defined Web response size

are shown in Figure 1. The two major operations of our model are *source variable generation* and *transactions handling*, which we will discuss later in this section.

The model is capable of generating both *Internet-like* traffic and *user-defined* traffic, which are the two working modes of this generator. The difference between them is how the source variables are provided. In the *Internet-like* mode, we generate each source variable automatically based on its relative stochastic distribution function. Furthermore, all the distributions are calculated to represent two real-world packet traces [6, 21]. By following the source variable generation module [5], this generator can simulate network traffic that replicates real-world Internet traffic.

The *user-defined* mode is designed for the users who want to generate specialised network traffic, in which all the source variables in this model can be provided as parameters. The mode is designed to run simulations with detailed scenario settings controlled by the users. For example, users can test how different Web object sizes affect the network performance by tuning them while fixing all the other parameters. The two working modes are designed to suit most of the traffic generation requirements.

3.1 Source Variable Generation

The source variable generation model is responsible for generating HTTP parameters for the *Internet-like* mode. MaxSessions is a user-defined value that specifies the number of Web sessions in the entire simulation process. The other major variables are: NumPages, ObjectsPerPage, ServerDelay,PageRequestGap, ObjectRequestGap, RequestSize, and ResponseSize. We use source variable generation functions to sample them, with each function for one of the parameters. NumPages and ObjectsPerPage are the number of pages for each Web session and the number of objects within one Web page, respectively. Both of them are modeled by the *Discrete Weibull* distribution [5] but with different distribution parameters. PageRequestGap is the inactive interval between two intermediate Web pages, while ObjectRequestGap is the interval for intermediate Web objects within a page. Both of them are fitted well by a combination of *Normal* and *Gamma* distributions [5]. ServerDelay is the time for the Web server to process the request, modeled by the *Inverse Weibull* distribution [5]. The maximum server delay is set as 10 s to avoid generating large delay values. For the same reason, we set the maximum request gap time as 100 ms.

Both distributions for RequestSize and ResponseSize are implemented in the `HttpFileSize` class, and they are fitted by *Discrete Weibull* distributions [5]. The mean of ResponseSize is larger than that of RequestSize. We assume the parameters in our model are uncorrelated with one another so that the generation of one parameter is independent of the other ones; this assumption has been verified with experiments in previous work [5, 8]. The distribution functions for all the variables are shown in Table 2.

As shown in Figure 1, the `HttpClient` and `HttpServer` applications are responsible for the major functionalities, such as generating the transactional traffic, handling transactional processes, as well as recording results. When the HTTP model starts, the `HttpClient` and `HttpServer` applications are installed in client and server nodes, respectively. The `HttpClient` application starts when a new TCP connection is initiated. On the other hand, the `HttpServer` application starts from the beginning of the simulation. The `HttpController` class controls the source variable generations and schedules the sending events for both the client and server. The user can define the number of clients and servers, not restricted to one client interacting with one server. In other words, each client can communicate with multiple servers, and one server can respond to multiple clients.

`HttpClient` starts by running `HttpDistribution`, which we develop for generating HTTP parameters. This implementation of our HTTP traffic generator provides several ns-3 `RandomVariable` objects for specifying distributions of HTTP source variables. It is based on the source code provided by PackMimeHTTP in ns-2 [1] and modified to fit into the ns-3 `RandomVariable` framework. `HttpRandomVariable` includes several subclasses with each one responsible for sampling one variable. For each Web session, the client first samples the number of objects for a specific TCP connection from the distribution of `HttpNumPages` and `HttpObjsPerPage` and sums up all the objects for the pages in each Web session. There are two gap times including `HttpObjectRequestGap` and `HttpPageRequestGap` as mentioned before. For each one of the requests, the client node also samples the HTTP request and response sizes based on `HttpFileSize`.

3.2 Transactions Handling

There are two types of application data units (ADUs) in this generator, `RequestAdu` and `ResponseAdu`. After generating all the necessary source variables for both the `HttpClient` and `HttpServer` as we mentioned previously, this model saves the generated parameters into the respective ADUs. For example, the `HttpObjectRequestGap` and `HttpP-`

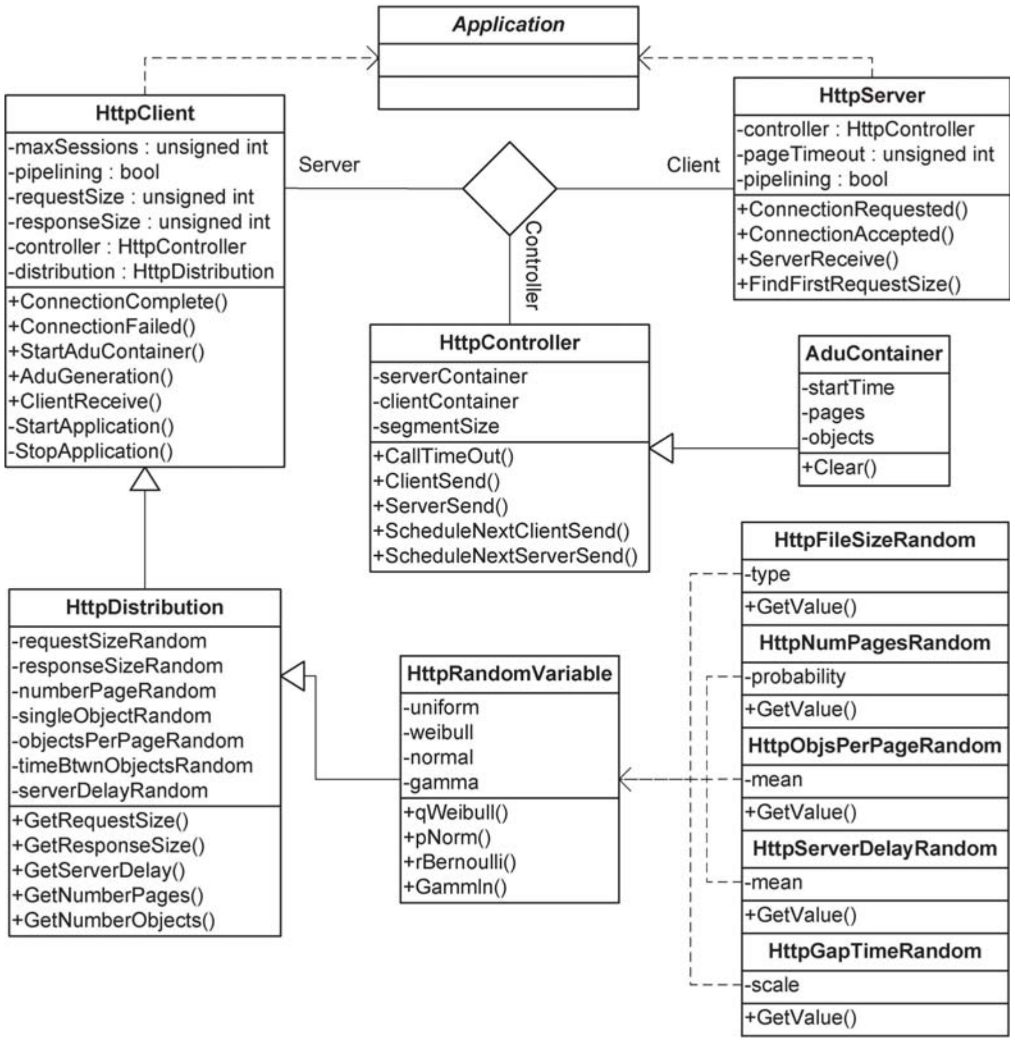

Figure 1: HTTP class diagram

Table 2: Transactional traffic model parameters

Parameters	Distributions	Model class
NumPages	Discrete Weibull	`HttpNumPages`
ObjectsPerPage	Discrete Weibull	`HttpObjsPerPage`
ServerDelay	Inverse Weibull	`HttpServerDelay`
PageRequestGap	Normal & Gamma	`HttpPageRequestGap`
ObjectRequestGap	Normal & Gamma	`HttpObjectRequestGap`
RequestSize	Discrete Weibull	`HttpFileSize`
ResponseSize	Discrete Weibull	`HttpFileSize`

ageRequestGap will be saved in the RequestAdu, while the HttpServerDelay be saved in ResponseAdu. The RequestAdu and ResponseAdu are saved to the AduContainer in sequence. After saving all the ADUs in AduContainer, we make two copies and give both client and server one copy. The reason for doing this is to keep track of both the sending and receiving events. For example, when the server receives 1500 B of the Web request, it knows which response corresponded to this request and sends the correct response back to the client. The HttpController is responsible for managing the two ADU containers and scheduling the data sending events. The AduContainer class is designed to work with both HTTP 1.0 and HTTP 1.1, with the latter having either *persistent connection* and/or *pipelining*. When each ADU is sent out from the client or server, HttpController removes it from the corresponding AduContainer and continues with the next ADU until the container for both sides are empty, which notifies the end of one Web session. The model continues with the next Web session.

As described before, each side of the node pair will be installed with either the client or server application. The server starts from the beginning of the simulation and listens for request ADUs from the associated clients, while the client starts when one transport connection is established. The client first checks the HTTP version defined by the user. If it is HTTP 1.0, the client sends the next request only after receiving the response for the previous request; for an HTTP 1.1 connection, the client also samples the inter-request gap times based on HttpGapTime and sends requests after the gap time without waiting for the previous responses. The model repeats this process until all the requested ADUs are sent or have a timeout without receiving any responses. This timeout value is defined as PageTimeout and it is a user tunable parameter. On the server side, when a RequestAdu arrives, the server locates it in server AduContainer. If the model finds one match, the Web request is deleted from the server AduContainer and the corresponding ResponseAdu is sent to the client after ServerDelay. This process will repeat until the requests are exhausted and all the responses are received by the client, following the next Web session. When all the Web sessions are finished transferring, the transport connection is closed and simulation is ended.

The result recording logic is triggered when one Web page is fully received. We consider the *object delivery ratio* and the *response latency* for each Web page as the performance metrics. If one Web page is not timed out, the *object delivery ratio* should always be one when using a reliable transport protocol such as TCP. The response latency is the time when client sent out the first request until the last byte of response for this specific Web page has been received. We have implemented our own result tracing system independent of the ns-3 built-in, which is included in our HTTP distribution. The reason behind this choice that for transactional traffic, users care more about the latency as well as the delivery ratio of the Web pages. It would be really difficult, if not impossible, to add tracing events in ns-3 notifying when a Web page is finished since the tracing events can only get information about how many TCP segments have been received by the node. There is no way to distinguish between two intermediate Web objects. This means that the result tracing logic is highly dependent on the HTTP model itself. When each page has finished receiving, we record how long it takes to successfully transfer the whole Web page. If TCP has timed out, we record what percentage of the Web page has been successfully delivered. All the tracing results are saved in different files and available to use after the simulation.

4. SIMULATION RESULTS

In this section, we present the simulation results conducted with the ns-3 network simulator [2] to analyse the performance of HTTP in different network conditions. Although the network characteristics data from [12] is aging, we still use it because this is the most detailed data public available and verified. Furthermore, this paper provides baseline results to compare against.

Several recent works have proposed Web traffic models based on current Web traffic data. One paper analyses Web traffic from 2006 to 2010 [14], capturing browsing behavior from more than 100 countries. Another recent paper [20] proposes their Web traffic models based on the top one million visited Web pages. We plan to test these datasets and models and may incorporate them into a future version of our ns-3 HTTP model if verified.

4.1 Scenarios and Metrics

Previous work [12] has defined characteristics for different networks including maximum segment size (MSS), round trip time (RTT), and the bandwidth for network links as shown in Table 3. Some of the networks are from measurement of the actual systems. For example, a 10 Mb/s Ethernet connection was measured between two Sun hosts. For some of the other networks, parameters were estimated; for example, Modem and ISDN used theoretical bandwidths. The networks with N-Modem and N-F-Internet [18] represents two similar network characteristics with Modem and Fast-Internet, respectively. They are just the same networks with a slight different characteristics. We include them to verify our model. The networks with the HTTP workloads are used to test how different size of Web pages and number of request-response transactions affect HTTP performance and they are shown as follows [12]:

- **Small Page**: single 5 kB Web page
- **Medium Page**: single 25 kB Web page
- **Large Page**: single 100 kB Web page
- **Small Cluster**: single 6651 B page with embedded 3883 B and 1866 B images
- **Medium Cluster**: single 3220 B page with three embedded images of sizes 57613 B, 2344 B, and 14190 B
- **Large Cluster**: single 100 kB page with 10 embedded 25 kB images

In addition to these workloads, we also use a varying workload with a range of 10 to 1000 Web objects with the fixed object size 10 kB, to test different HTTP versions: pipelining, persistent, and non-persistent connection. We will show the results in Section 4.3.

Our validation scenario consists of two nodes, one server and one client, with a point-to-point link connecting them. The transport protocol in this case is TCP. We use *user-defined* mode of our generator and use the following variables:

Table 3: Network characteristics

Network	RTT (ms)	BW (Mb/s)	MSS (B)
Fast-Internet	89	1.02	512
N-F-Internet	80	1.17	1460
ADSL	30	6	512
Ethernet	0.7	8.72	1460
Fast-Ethernet	0.7	100	1460
Modem	250	0.0275	512
DirecPC	500	1	512
Slow-Internet	161	0.102	512
ISDN	30	0.122	512
WAN-Modem	350	0.0275	512
WAN-ISDN	130	0.122	512
N-Modem	150	0.0275	1460

- UserObjectRequestGap: 0.01 s
- UserServerDelay: 0.1 s
- UserPageRequestGap: 0.2 s
- UserPageRequestSize: 256 B

We only consider a wired scenario in this paper, with only one server-client pair analysed. We plan to perform more detailed wireless simulations later with varying number of server-client pairs, including based on our previous work simulating wireless scenarios [7]. We choose `UserObjectRequestGap`, `UserServerDelay`, and `UserServerDelay` based on the most frequent values in our distribution model, and use 256 B of request size that would fit in one TCP segment for most real-world MTUs. This way the latency is mainly dependent on the response size. For the simulation metric, we use the Web page response latency and do not present object delivery ratio since TCP guarantees the delivery of data segments, and all the cases we tested do have 100% object delivery ratio. We use HTTP 1.1 with persistent connection and pipelining for all the simulation cases except for the last one in which we test how different HTTP versions would affect the response latency.

4.2 Distribution Module Validation

We verify that the source variable generation function can generate reasonable results compared to the distribution function described in previous work [5, 8, 17]. This is designed to test the operation of `HttpRandomVariable` in ns-3. We choose the `RequestSize`, `ResponseSize`, `RequestGap` and `ServerDelay` times as examples, and use a complementary cumulative distribution function (CCDF) to represent the results.

The Web object sizes include both request and response file sizes; the generation function for both is the *Discrete Weibull* distribution. However, the size of responses is significantly larger than that of requests. As we can see from Figure 2, for response sizes, 78% are smaller than 10 kB, and only 1% larger than 100 kB. While for request sizes, 97.5% are smaller than 1460 B, which would fit in one TCP segment. These two source variables follow the self-similar distribution as the number of large file sizes is small, while the number of small file sizes is large. This phenomena matches the real-world Internet traffic [9].

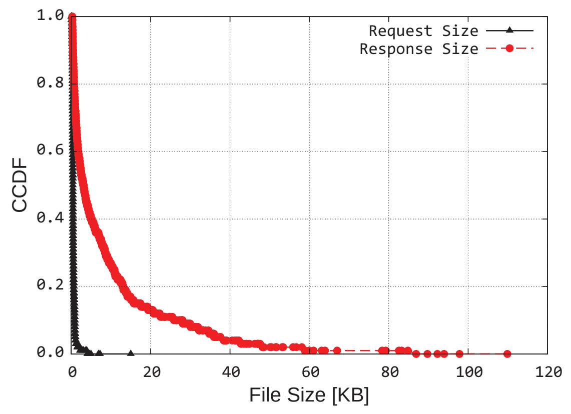

Figure 2: CCDF of HTTP file sizes

Two major time values in this model are the request gap time and server delay time as shown in Figure 3. The request gap time is the delay between two subsequent Web objects, while the server delay time is the latency for the server to process the ADU from clients and trigger the response sending mechanism. The request gap time follows a mixture of *Normal* and *Gamma* distributions [5]. Based on the generation functions for each time variable; 90% of the request gap time is below 10 ms, while 1% are larger than 25 ms. The server delay time follows an *Inverse Weibull* distribution, 90% are below 500 ms, with only 1% larger than 1000 ms. Both of the two time values follow self-similar distributions. The maximum request gap time is set as 100 ms, which explains the cutoff of the request gap time at that time value.

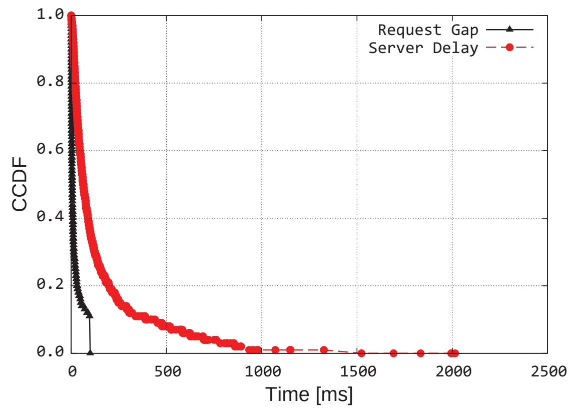

Figure 3: CCDF of HTTP delay times

4.3 Simulation Results

The first simulation scenario is to test how different workloads affect network performance in terms of response latency. The result is shown in Table 4 with the time scale in milliseconds. As the Web page sizes increase, the latency for all the networks increases as expected. Although some of the values are not exactly the same as the theoretical values [12], they are of the same order of magnitude. Furthermore, we additionally take server processing delay into consideration, which is another factor that contributes to the overall latency. When plotting the performance of all the networks, we did not include the WAN-Modem and N-Modem curves because their performance are very similar to the Modem curve. For the same reason, we do not present WAN-ISDN here and show the ISDN curve.

We further test how the response size would affect HTTP performance. We use the same network configuration as previous simulations. As the response size increases, the response latency increases for both of the fast network curves in Figure 4 as well as the slow networks in Figure 5. While for the slow network, the increase degree is larger, they are more sensitive to the increasing size of responses. For example, as shown in Figure 5, the latency of Modem increases from 15 s to 33 s when response size increases from 40 kB to 100 kB; however the latency only increases from 10 s to 15 s for DirecPC.

The latency for the fast networks in Figure 4 is within the delay range tolerable for everyday use. We can see that for Fast-Internet, the latency increases to 1.5 s when the response size is 100 kB. The latency is a little bit larger than we normally desire in current Web browsing. This is because all the network data from [12] is estimated 10 years ago and dated, as mentioned before.

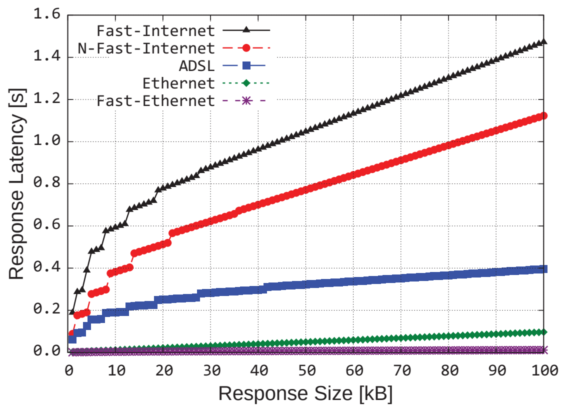

Figure 4: Fast network with different response sizes

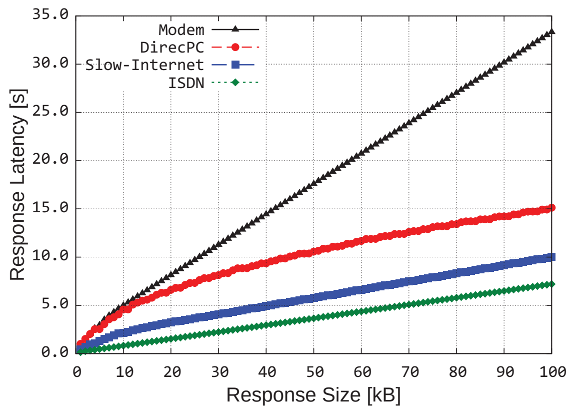

Figure 5: Slow network with different response sizes

Furthermore, we compare the performance among persistent connections with or without the pipelining option, representing the HTTP 1.1 version, and the non-persistent connections representing HTTP 1.0. We are able to demonstrate the performance improvement of HTTP 1.1 with different options. The workload is from 0 to 1000 Web objects with 10 kB in size. The network we use is Fast-Internet. The result is shown in Figure 6 with response latency as the metric.

As expected, we can see that HTTP with non-persistent connections performs worst. When the number of Web objects reaches 1000, the response latency is 800 s. The persistent connection without pipelining improves the performance greatly: when carrying same number of Web objects, the latency is only 300 s. Furthermore, when the pipelining option is included in the persistent connection, the latency further drops to 100 s. All the three curves are linear, because all the Web objects have the same size, so as the number of Web objects increases, the latency increases linearly. The slope of the three curves for non-persistent, persistent, and persistent with pipelining are 0.82, 0.28, and 0.09 respectively. The persistent connection option has increased the performance more as it drops the slope from 0.82 to 0.28, while the pipelining option further drops the slope from 0.28 to 0.09. We plan to incorporate the *parallel connection* option in the future performance comparison cases to have a complete list of HTTP options.

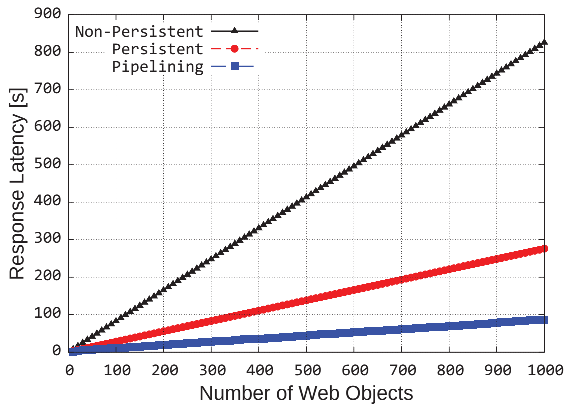

Figure 6: Performance of different HTTP versions

5. CONCLUSIONS

In this paper we presented the implementation details of our HTTP traffic generator for the ns-3 network simulator and validated its performance. Our results confirm both the source variable generation functions and latency for different networks when carrying HTTP traffic. We analyse HTTP performance over different network conditions with different response sizes. Our results demonstrate that latency is inversely proportional to response size. The larger the response size is, the larger the latency required to transfer. The performance comparison case among different HTTP versions confirms the improvements of HTTP 1.1 options. As part of future work, we plan to include HTTP with *parallel connection* option.

Acknowledgments

We would like to acknowledge the assistance of Justin P. Rohrer, and the members of the ResiliNets research group for their advice and suggestions which helped us with this

Table 4: Latency of different networks [ms]

Network	Small page	Medium page	Large page	Small cluster	Medium cluster	Large cluster
Fast-Internet	477.5	819.6	1471.7	808.5	1456.3	3852.8
N-Fast-Internet	277.4	587.2	1123.0	510.3	1141.2	3158.0
ADSL	155.5	258.3	395.3	256.4	411.2	935.4
Ethernet	7.1	26.0	96.7	16.5	71.0	262.3
Fast-Ethernet	2.5	5.4	11.6	4.4	11.7	25.7
Modem	2941.3	9700.3	33314.6	6829.5	25468.8	91694.2
DirecPC	2546.5	7329.5	15125.6	5501.0	12969.6	27129.3
Slow-Internet	1293.4	3653.1	10019.7	2769.2	8185.2	26695.3
ISDN	465.7	1884.6	7207.5	1263.4	5359.9	20106.1
WAN-Modem	3441.3	10580.1	34194.4	7669.4	26648.6	93574.0
WAN-ISDN	922.2	2341.1	7664.03	1922.7	6116.4	21562.6
N-Modem	3466.9	9456.1	31897.5	6880.8	24191.6	86515.7

implementation. We would also like to thank Tom Henderson and the ns-3 development team for their responsiveness to issues with ns-3 platform. We acknowledge Asifuddin Mohammad for an early version of the code that provided a starting point for this work.

6. REFERENCES

[1] The network simulator: ns-2. `http://www.isi.edu/nsnam/ns/`, December 2007.

[2] The ns-3 network simulator. `http://www.nsnam.org`, July 2009.

[3] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *ACM SIGMETRICS*, pages 126–137, 1996.

[4] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[5] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle. Stochastic Models for Generating Synthetic HTTP Source Traffic. In *IEEE INFOCOM*, volume 3, pages 1546–1557, Mar. 2004.

[6] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun. On the Nonstationarity of Internet Traffic. In *ACM SIGMETRICS*, pages 102–112, New York, NY, Jun. 2001.

[7] Y. Cheng, E. K. Çetinkaya, and J. P. Sterbenz. Performance Comparison of Routing Protocols for Transactional Traffic over Aeronautical Networks. In *Intl. Telemetering Conf. (ITC)*, Oct. 2011.

[8] H. Choi and J. O. Limb. A Behavioral Model of Web Traffic. In *IEEE ICNP*, pages 327–334, Oct. 1999.

[9] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Trans. Netw.*, 5:835–846, Dec. 1997.

[10] T. D. Dyer and R. V. Boppana. Routing HTTP traffic in a mobile ad hoc network. In *IEEE MILCOM*, volume 2, pages 958–963, Oct. 2002.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[12] J. Heidemann, K. Obraczka, and J. Touch. Modeling the Performance of HTTP over Several Transport Protocols. *IEEE/ACM Trans. Netw.*, 5(5):616–630, 1997.

[13] F. Hernandez-Campos, K. Jeffay, and F. D. Smith. Modeling and Generating TCP Application Workloads. In *BROADNETS*, pages 280–289, Sep. 2007.

[14] S. Ihm and V. S. Pai. Towards Understanding Modern Web Traffic. In *ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 295–312, New York, NY, 2011. ACM.

[15] L. Le, J. Aikat, K. Jeffay, and F. D. Smith. The Effects of Active Queue Management on Web Performance. In *ACM SIGCOMM*, pages 265–276, New York, NY, 2003. ACM.

[16] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-similar Nature of Ethernet Traffic. *IEEE/ACM Networking*, 2(1):1–15, Feb. 1994.

[17] B. A. Mah. An Empirical Model of HTTP Network Traffic. In *IEEE INFOCOM*, volume 2, pages 592–600, Kobe, Japan, 1997.

[18] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *ACM SIGCOMM*, pages 155–166, New York, NY, Oct. 1997.

[19] V. Paxson and S. Floyd. Wide Area Traffic: the Failure of Poisson Modeling. *IEEE/ACM Networking*, 3(3):226–244, Jun. 1995.

[20] R. Pries, Z. Magyari, and P. Tran-Gia. An HTTP Web Traffic Model based on the Top One Million Visited Web Pages. In *EURO-NGI*, pages 133–139, June 2012.

[21] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott. What TCP/IP Protocol Headers Can Tell Us About the Web. In *ACM SIGMETRICS*, pages 245–256, 2001.

[22] J. P. Sterbenz. Ns3-models. `https://wiki.ittc.ku.edu/resilinets/Ns3-Models`, September 2010.

[23] M. Weigle. Improving Confidence in Network Simulations. In *Winter Simulation Conference*, pages 2188–2194, Dec. 2006.

[24] E. Weingartner, H. vom Lehn, and K. Wehrle. A Performance Comparison of Recent Network Simulators. In *IEEE ICC*, pages 1–5, Jun. 2009.