CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Enabling scalable emulation of differentiated services in mininet
Authors	Raca, Darijo;Salian, Meghana;Zahran, Ahmed H.
Publication date	2022-08-05
Original Citation	Raca, D., Salian, M. and Zahran, A. H. (2022) 'Enabling scalable emulation of differentiated services in mininet', MMSys '22: Proceedings of the 13th ACM Multimedia Systems Conference, June 2022, pp. 240-245. doi: 10.1145/3524273.3532893
Type of publication	Conference item
Link to publisher's version	10.1145/3524273.3532893
Rights	© 2022, Association for Computing Machinery. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in MMSys '22: Proceedings of the 13th ACM Multimedia Systems Conference, June 2022, https://doi.org/10.1145/3524273.3532893
Download date	2026-11-08 08:26:28
Item downloaded from	https://hdl.handle.net/10468/13601



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Enabling Scalable Emulation of Differentiated Services in Mininet

Darijo Raca
University of Sarajevo
Bosnia and Herzegovina
draca@etf.unsa.ba

Meghana Salian
Apple Inc.
Ireland

Ahmed H. Zahran
University College Cork
Ireland
a.zahran@cs.ucc.ie

ABSTRACT

Evolving Internet applications, such as immersive multimedia and Industry 4, exhibit stringent delay, loss, and rate requirements. Realizing these requirements would be difficult without advanced dynamic traffic management solutions that leverage state-of-the-art technologies, such as Software-Defined Networking (SDN). Mininet represents a common choice for evaluating SDN solutions in a single machine. However, Mininet lacks the ability to emulate links that have multiple queues to enable differentiated service for different traffic streams. Additionally, performing a scalable emulation in Mininet would not be possible without light-weight application emulators. In this paper, we introduce two tools, namely: QLink and SPEED. QLink extends Mininet API to enable emulating links with multiple queues to differentiate between different traffic streams. SPEED represents a light-weight web traffic emulation tool that enables scalable HTTP traffic simulation in Mininet. Our performance evaluation shows that SPEED enables scalable emulation of HTTP traffic in Mininet. Additionally, we demo the benefits of using QLink to isolate three different applications (voice, web, and video) in a network bottleneck for numerous users.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → *Public Internet*; *Wireless access networks*; • **Computing methodologies** → **Modeling and simulation**.

KEYWORDS

Mininet, Emulation, Differentiated Services, Web Behavioral Modelling, adaptive video streaming, web traffic, VoIP

ACM Reference Format:
Darijo Raca, Meghana Salian, and Ahmed H. Zahran. 2022. Enabling Scalable Emulation of Differentiated Services in Mininet. In *13th ACM Multimedia Systems Conference (MMSys '22), June 14–17, 2022, Athlone, Ireland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3524273.3532893

1 INTRODUCTION

Internet traffic exhibits a huge diversity in their network requirements in terms of throughput, delay, jitter, and loss. Ensuring the best user experience for a traffic mixture would be challenging without imposing traffic management mechanisms at network bottlenecks. Buffer bloat [9] is identified as a critical problem when elastic greedy applications, such as Video on Demand (VoD) or file transfer, share a bottleneck with latency sensitive applications, such as VoIP and web. The problem is more critical for evolving immersive multimedia applications that operate on a tight latency budget. Hence, queueing mechanisms and dynamic traffic steering using software-defined technologies represent key building blocks in future systems.

The performance evaluation of these evolving solutions requires reliable tools. Mininet [13] is currently one of the most widely-used performance evaluation tools for SDN solutions. Mininet API enables emulating a network consisting of various virtual hosts and switching devices using a single machine. Additionally, Mininet emulates links with specified link characteristics (bandwidth, delay, and loss) are defined using Linux traffic control command (tc). However, current Mininet implementation does not support creating multiple queues at network interfaces. While such queues can be created using Open Virtual Switch (OVS) control command (ovs-vsctl), this approach is limited to emulating link rate; i.e., no delay or loss emulation. Additionally, it requires a steep learning curve for traffic management and OVS control commands. Hence, extending Mininet to support Quality of Service (QoS)-enabled, fully-emulated links evolves as a crucial requirement to test differentiated service solutions in Mininet.

The use of real application represents another advantage for Mininet as network hosts can run any application in the host. However, performing scalable experiments with numerous users can hit a computing bottleneck for a single machine. Hence, different light-weight tools have been developed to emulate the behaviour of different applications. For example, several HTTP adaptive streaming (HAS) players [7, 17, 21] emulates HAS streaming clients while excluding the demanding video processing component. Similarly, scalable emulation of delay-sensitive traffic. e.g., voice, is possible using [3]. For web traffic, browser automation tools, such as Selenium[1], allow automated controlled experimentation. However, such tools would not scale due to their reliance on underlying web browsers (e.g., Firefox or Google Chrome). Hence, there is a need for a light HTTP traffic emulator for scalable experimentation in Mininet.

The contribution of this paper can be summarized as follows,

- We extend Mininet to enable emulating QoS enabled links. Specifically, we extend Mininet API with an *addqLink* method that receives both link emulation and queuing parameters. Assigning traffic to queues is achieved through OpenFlow (OF) rules that can be installed through the application build on top of any SDN controller [2].

[1] http://www.seleniumhq.org/
[2] https://github.com/darijo/QLink

- We present SPEED[3] (a Scalable Python wEb bEhavioural moDel) as a novel emulation tool for web traffic. SPEED mimics both traffic patterns and user behaviour of modern web browsers. Additionally, it scales well by focusing on traffic exchange without performing demanding application-level user interface functions.
- We validate the presented tools using a Mininet testbed that includes multiple competing clients sharing a bottleneck link while running different applications. Our evaluation illustrates the benefits of priority queuing in scenarios in comparison to the widely-used First In First Out (FIFO) queueing.

The rest of this paper is organized as follows. Section 2 presents relevant background and related work. Sections 3 and 4 presents QLink and SPEED, respectively. Our experimentation is presented in Section 5 before concluding in Section 6.

2 BACKGROUND AND RELATED WORK

SDN is a networking paradigm that centralizes the network control plane, leading to simplified switching devices focused on data plane functions. The network intelligence is implemented as network applications running on a logically centralized controller that interfaces with switches to monitor and control data plane functionality. This interface is known as south-bound interface and OF [25] is the most common protocol on this interface. OF defines switch-controller communication procedure and exchanged messages for configuration, packet forwarding, and statistics collection. OF forwarding rules are defined by network applications and include packet fields to be matched and actions to be performed by the switch on matching received packets. Additionally, OF supports differentiated services framework through enqueue action to send a packet to a specific queue at the output interface. Furthermore, OF 1.3 introduced a meter feature that enables switches to mark or drop packets based on a pre-defined policy.

Mininet provides a powerful Python API to create end-hosts, switches, controllers, and links by leveraging Linux virtualization capabilities. Hosts are defined in Linux containers with configurable processing capacities and isolated namespace. Switches are nodes running a switch daemon that interacts with any SDN controller using OF protocol []. The switch daemon populates the switch forwarding table based on the received OF messages. Mininet leverages tc command to emulate bandwidth, delay, and loss for both TCLink and TCULink classes[4].

Mininet supports various switches, and its default is to OVS whose implementation extends over both user and kernel spaces. The kernel part includes a dump virtual switch and forwarding logic. The user space includes the switch daemon (ovs-vswitchd) and OVS Database (OVSDB) components. ovs-vswitchd interacts with the SDN controller and the kernel switch using OF and NetLink, respectively, to forward traffic and share switch information. OVSDB represents a network-accessible database system that is used for configuring and monitoring ovs-vswitchd. Furthermore, OVS provides a number of user space commands, such as ovs-vsctl and ovs-ofctl. ovs-vsctl configures ovs-vswitchd through interaction with OVSDB. ovs-ofctl is used to monitor and administer OF by directly interacting with ovs-vswitchd.

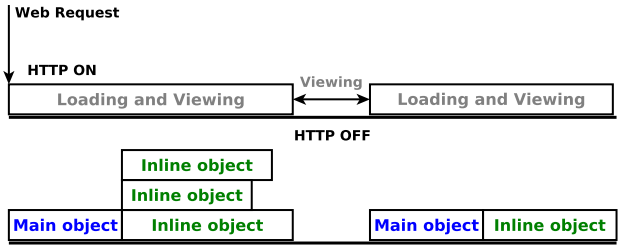


Figure 1: Simplified Web Behavioural Model

Web traffic is typically modelled as an on/off process [11]. The "ON" phase involves requesting and loading a webpage, while the "OFF" phase represents the page reading time. During the ON phase, the client fetches the main HTML file and any additional inline objects, such as images, scripts, and stylesheet. It is common that these objects are downloaded using multiple TCP connections, e.g., Firefox uses up to six parallel connections. This two-phase process typically repeats for subsequent pages. This process is illustrated in Figure 1. While many researchers analysed web content and user browsing behaviour [11, 14], few experimental tools are available. More critically, existing ones (e.g., SURGE [1]) are developed over two decades ago and do not represent the current webpage content structure. That represents a gap that SPEED is filling.

User experience is defined by different factors depending on the application. Voice Quality of Experience (QoE) are based on QoS impairments such as delay, jitter, packet loss rate and codec type [2, 10]. Web QoE is dominated by the delay component, specifically the User Perceived Page Loading Time (UPPLT). OnLoad is defined as elapsed time between sending a request and loading all objects on the web page. While onLoad is considered an over estimation of the visible part of the page (i.e., UPPLT), onLoad shows the highest correlation (0.85) with the UPPLT, outperforming more sophisticated Page Loading Time (PLT) metrics [27]. Video experience involves many application-level visual and temporal aspects. Several QoE models exist in the literature [5, 8, 12, 16, 20, 22, 24].

3 QLINK

3.1 QLink API

QLink uses a Hierarchical Token Bucket (HTB) queueing discipline with all traffic queues introduced as children to the root, as illustrated in Figure 2. QLink extends Mininet with two key method, namely addQSpec and addQLink. addQspec is used to define individual child queue specifications, including

- queue: a non-negative integer queue identifier. This identifier can be used in to direct the traffic to this queue.
- priority: an integer $\in \{0, .., 7\}$, where smaller values represent higher priorities. (default 7)
- minRate: a fraction $\in [0, 1]$ and represents the sustained ratio of the parent (root) bandwidth. (default 0.01)

[3] https://github.com/darijo/SPEED
[4] TCULink enables asymmetric link characteristics

- maxRate: a fraction $\in [0, 1]$ and represents the maximum allowable ratio of the parent bandwidth. (default 1)

A typical invocation in a Mininet Python script, where *net* is the instantiated Mininet network object, would be

```
q1 = net.addQspec(queue=0, priority=1,
        minRate=0.1, maxRate=0.5)
```

It is worth noting that HTB allows children queue to borrow tokens from their parent up to the maxRate as long as competing queues sustained rates are served.

addQlink is used to create a link between two Mininet nodes using the following parameters

- two mininet node names
- *params1* (*params2*): a dictionary defining the link parameters with *params1* (params2) representing the link parameters from the first (second) to the second (first) nodes.
- qspec1 (qspec2): a list of queues that are defined using addQspec. These queues are installed as children queues at the first (second) node.

A typical invocation for addQLink would be

```
net.addQlink( s1, s2,
    params1=\{'bw': 200, 'delay': '10ms'\},
    params2=\{'bw': 20, 'delay': '10ms'\},
    qspec1=[q1, q2], qspec2=[q1, q2])
```

In this example, the link uses asymmetric bandwidth but uses the same queue specifications in both directions.

It is worth noting that one of the HTB leaf queues is defined as a default queue that is automatically used for packets forwarded to the port without specifying a queue. In our implementation, this default queue is the first queue in qspec1 and qspec2. Hence, if the user would like to define a default queue, it should be the first item in qspec list. Otherwise, the first queue should have the lowest priority to ensure proper operation of prioritized forwarding.

3.2 QLink Implementation

To support QoS-enabled emulated links, we could have modified OVS or Mininet code. We decided to extend Mininet because link emulation is a network feature rather than a switch function. Mininet emulates the link characteristics using tc-emnet. While defining a queuing discipline is possible using tc, these queues will not be registered in OVSDB. Hence, they will be invisible to SDN applications. Hence, creating the queues using ovs-vsctl is unavoidable. However, introducing the delay and loss components using mininet would not be possible at the same interface because both OVS and Mininet assume ownership for the root queue. Hence, we have split queuing and rate control from delay and loss emulation at two different interfaces using a hidden switch, as illustrated in Figure 2. Figure 2 illustrates our implementation for QLink. The end-points of QLink $(n1, n2)$ are connected through a hidden switch h. h is configured to automatically forward any incoming packet at a specific port to the second one. Hence, h will not induce any traffic to the network controller. Additionally, the egress interfaces of h are configured to emulate the delay and loss in every direction. The egress interface at each end-node is configured with an HTB whose child (leaf) queues are used for traffic isolation and the root queue is used to limit the link rate.

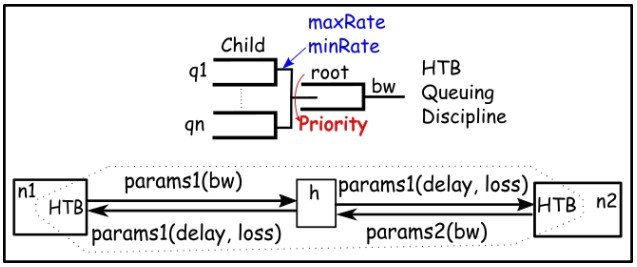


Figure 2: QLink Overview

The majority of the implementation sits in Mininet *net.py*. Additionally, changes are introduced *node.py* to add an additional member variable to the switch object to identify switches with QLink interfaces. Such distinction is required during network startup and destruction to ensure clean termination by restoring OVSDB to its initial state. Additionally, we introduced changes to *utils.py* to ensure the transparency of hidden switches when Mininet user explores the network using Mininet *dump* and *net* commands. As a final note, if one of QLink end-nodes is a host[5], queues are only created on the switch node. Note that SDN controller application can only steer traffic at network switches using OpenFlow rules.

4 SPEED

SPEED has two components including synthetic page generator and web client. The generator uses statistical models for page and embedded object characteristics from [18], which considered one million webpages. Table 1 summarises the model parameters used for the synthetic generation of webpages.

Generation of synthetic web content is carried in two steps, first creating JavaScript Object Notation (JSON) files containing all necessary metadata, followed by creation of actual web content. Script *speed_web_generator.py* implements the following logic:
JSON file generation: JSON file is created per user, containing all the information about each webpage, i.e., number of main and inline objects, size (in bytes) of each object. Also, this file stores information about user behaviour for each webpage, reading or dwell time (i.e., how much time user spends "reading" the content). Listing 1 shows the example of JSON file structure.

```
"Run_1": {
    "Page_1": {
        "num_main_objects": 1,
        "reading_time": 0.452,
        "main_1": 20881.0,
        "num_inline_for_main_1": 4.0,
        "inline_1_1": 16605.0,
        ...
    "Page_2": {
        "num_main_objects": 1,
        "reading_time": 17.027,
        "main_1": 7700.0,
        "num_inline_for_main_1": 128.0,
        "inline_1_1": 5918.0,
    ...
    "Run_10": {
```

[5]We assume that any host node starts with "h", which is a common convention.

Table 1: Webpage content parameters [18]

Parameter	Mean	Median	Max	Standard Deviation	Best Fit
Main object size	31,561 Byte	19,471 Byte	8*MB*	49,219 Byte	Weibull (28242.8,0.814944)
Num. of main objects	2.19	1	212	2.63	Lognormal μ= 0.473844, σ = 0.688471
Inline object size	23,915 Byte	10,284 Byte	8*MB*	128,079 Byte	Lognormal μ = 9.17979, σ = 1.24646
Num. of inline objects	31.93	22	1920	37.65	Exponential μ = 31.9291

```
        "Page_1": {
            "num_main_objects": 1,
            "reading_time": 1.35,
            "main_1": 15157.0,
            "num_inline_for_main_1": 35.0,
            "inline_1_1": 50644.0,
            ...
```

Listing 1: JSON File Example

Content generation: Followed by the previous step, JSON files are used for generation of actual web content matching the naming and size according to the JSON file. Files are created using *fallocate* command.

Arguments passed to the scripts are the following:

- numClients: number of clients (a separate JSON file is generated for each client)
- numRuns: the number of planned runs to generate random page sequences for subsequent runs.
- webPages: number of webpages per client
- save_json: path/location where to save JSON files
- save_content: path/location where to save web content

Web content can be served using any of existing web servers (e.g., Apache, Caddy, flask).

SPEED web-client relies on the JSON file for page and user behaviour information. The web client requests the webpage sequence for a given run and persists on every page for the corresponding reading time, which is randomly generated following lognormal distribution with $\mu = -0.495204$ and $\sigma = 2.7731$ [14]. If a page loading time exceeds 15 seconds[26], the user abandons the page and requests another page. Also, we limit maximum reading time to 70 seconds [15]. The web-client uses *aiohttp*[6] asynchronous HyperText Transfer Protocol (HTTP) library, which allows opening multiple parallel TCP connections to a web server. SPEED web client generates a log file for every run including the page-loading time for each webpage object, total page load time, and reading time. Listing 2 depicts the example of log file structure.

```
1645296312.626835 - PLT main object_1: 164
1645296312.873558 - PLT main object_2: 410
1645296313.284074 - PLT main object_3: 821
1645296313.367373 - PLT Page_3: 904 Reading Time: 7.068
```

Listing 2: Example of Log file from web client

SPEED web-client is emulated using *httpClient.py* script, which requires the following arguments:

- file: path/location of JSON file for the client
- server_ip_addr: IP address of the server storing web content
- run_num: run identifier
- out_path: path/location where to save log files

[6]https://aiohttp.readthedocs.io/en/stable/

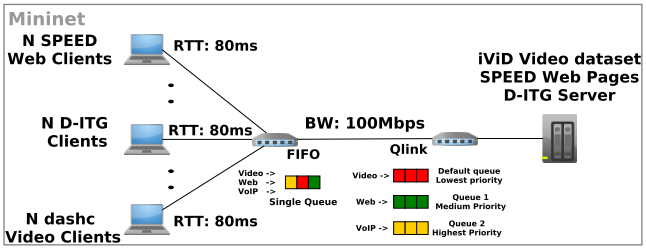


Figure 3: Dumbbell Topology used for Experiments

5 EXPERIMENTATION

In this section, we present our experimental methodology, including details of our platform and key performance metrics.

5.1 Experiment Setup

We experimented both Mininet default FIFO and QLink using the dumbbell topology in which a number of users share a network bottleneck as depicted in Figure 3. The bottleneck link capacity and round trip delay are set to 100*Mbps* and 80*ms*, respectively.

In our experiments, we tested different number of users including 30, 45, 60, and 75 users, equally split into three classes including voice, web, and video users.

Voice traffic is generated using D-ITG distributed traffic generator [4]. We set D-ITG to generate VOIP with codec G.711.

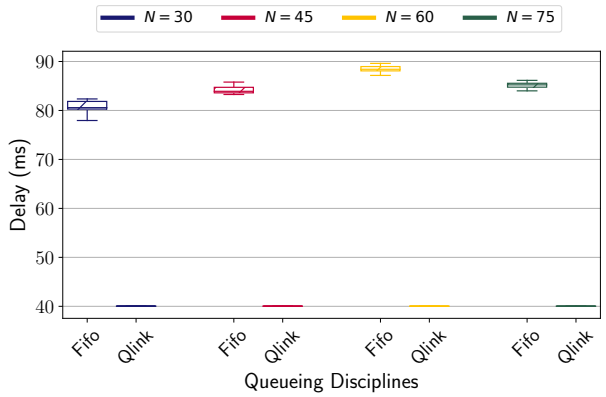
Web traffic is generated by SPEED as described in Section 4. The pages are stored locally, and we use Flask[7] for serving the webpages. We create 40 webpages per client across 10 runs (in total 400 webpages per client).

Video clients stream five-minute four-second-segment 4K clips from a publicly available HAS dataset [19]. The dataset includes video resolutions up to 3840x2160 with thirteen quality representation rates, shown in Table 2. The video content is streamed from a Caddy[8] HTTP server. We use dashc [21] as a light HAS client emulator. We set the streaming algorithm to ELASTIC [6], which shares bottlenecks in a friendly manner.

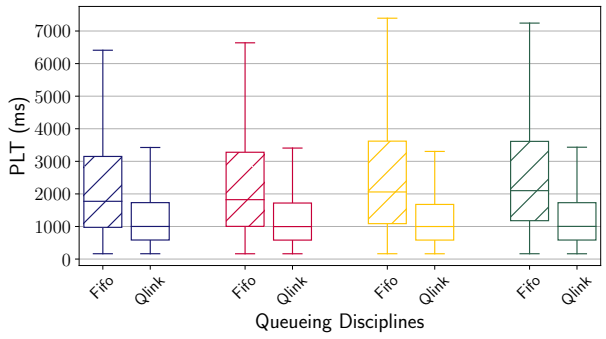
In QLink experiments, we employ three queues for voice, web, and video traffic with the priority set to 1, 2, and 3, respectively. QLink minRate and maxRate are set to 0.1 and 1 for all the queues. Additionally, the switches are configured using ovs-ofctl to enqueue packets of different applications in their corresponding queues.

Our performance metrics include packet loss and delay for voice, PLT for web traffic, and QoE for video [28]. The video QoE score is

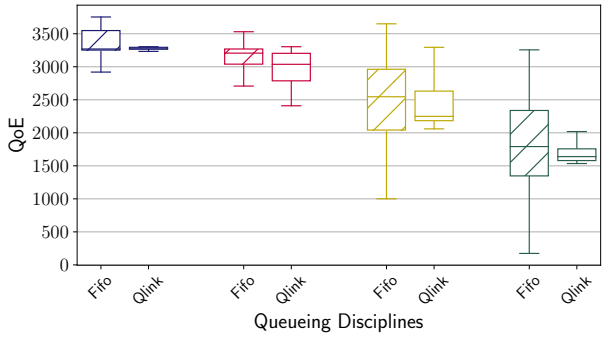
[7]https://flask.palletsprojects.com/en/2.0.x/
[8]https://caddyserver.com/



(a) VoIP Performance



(b) Web Performance



(c) Video Performance

Figure 4: Performance of video, web, and voice clients across different queuing disciplines and number of competing users (N - total number of competing users)

calculated as follows:

$$QoE = \sum_{n=1}^{M} q(R_n) - \lambda \times \sum_{n=1}^{M-1} |q(R_{n+1}) - q(R_n)| - \theta \times S_D - \theta_S \times S_I, \quad (1)$$

where M is the number of streamed segments per session, q is a mapping function between bitrate and quality; S_D represents the total stall duration per run (seconds), and S_I is the initial delay (seconds). $\lambda, \theta, \theta_S$ represents non-negative penalty factors for video quality variations, rebuffering time, and startup delay, respectively. We employ similar values for these factors as used in [28], $\lambda = 1$, $\theta = \theta_S = 4200$.

The presented results represent the outcome of 10 different runs that last for 6 minutes to allow for streaming a whole video.

5.2 Experiment Results

Figure 4a plots the delay of VOIP packets for all tested scenarios when FIFO and QLink are used. With QLink, VOIP packets only encounter insignificant queuing delay, equals the transmission time of the largest packet in the worst case. Hence, the observed delay of the highest priority traffic would be defined by the RTT. On the contrary, when VOIP competes with other traffic in the FIFO queue, the packet delay is roughly doubled for all tested scenarios. Additionally, VOIP packets experienced packet losses $0.2 - 0.5\%$ with FIFO in comparison to zero loss when QLink is used.

Figure 4b depicts the web PLT across all tested scenario. With QLink, the average PLT is approximately halved in comparison to when FIFO is used for all user configurations. Additionally, the variance in the experienced PLT significantly drops when QLink is used. The observed PLT variance in case of FIFO is due to the competition between the bursty web traffic and bandwidth-hungary DASH streams in the queue.

Figure 4c plots the video QoE for all tested scenarios. Clearly, the video QoE drops as the number of users increases due to the drop in the share of every user in the link bandwidth. Generally, QLink slightly reduces the average video QoE due to the higher priority of VOIP and web traffic. On the other hand, QLink generally features smaller variance of QoE due to reducing the interaction between different traffic streams in the bottleneck link.

To illustrate scalability of SPEED, we compare CPU and memory utilisation of SPEED and Selenium, a browser-based testing framework. Our webpage content is not suitable for testing with Selenium as our webpages only mimic the structure of the webpage. We use results obtained from [23] for comparison, where 10 users were simulated for different configurations of Selenium. We compare SPEED resource usage against Selenium running in the headless browser mode as it gives the lowest CPU and memory utilisation.

Table 3 indicates that SPEED has significantly less impact on CPU and memory utilisation, requiring 65x, and 3x less CPU and memory resources, respectively. Furthermore, running 50 SPEED clients increases resource requirements to 4.3%, and 7.2%, for median CPU and memory utilisation, respectively.

6 CONCLUSIONS

Ensuring performance guarantees to delay-sensitive application is challenging when they share a network bottleneck with bandwidth hungry applications, such as video. Achieving these guarantees could be possible by adopting classful queuing disciplines and dynamic traffic management technologies, such as SDN. In this paper, we have presented QLink that extends Mininet API to enable emulating links with HTB queuing discipline. Additionally, we present SPEED as a novel lightweight web traffic emulator to support large-scale experimentation. Our performance evaluation compares the standard FIFO and three-class QLink in the presence of voice, web, and video traffic from numerous users sharing a network bottleneck. We illustrate that traffic isolation enabled the highest priority

Table 2: Rates (kbps) and resolutions for each representation

1	2	3	4	5	6	7	8	9	10	11	12	13
235	380	568	760	1065	1777	2387	3046	3906	4361	15000	25000	40000
320 × 240	384 × 288	512 × 384		640 × 480	720 × 480	1280 × 720		1920 × 1080		3840 × 2160		

Table 3: The resource usage comparison between SPEED and Selenium

Type	Median CPU	Median Memory
SPEED	0.75%	1.7%
Selenium headless	49%	5%

traffic to achieve zero packet loss and queuing delay. The second class (web) also experience a noticeable reduction in the PLT.

ACKNOWLEDGMENTS

This publication has emanated from research supported in part by a Grant from Science Foundation Ireland under Grant number 18/CRT/6222. The authors acknowledge the support of the Ministry of Education, Science and Youth of Sarajevo Canton.

REFERENCES

[1] Paul Barford and Mark Crovella. 1997. An Architecture for a WWW Workload Generator. In *World Wide Web Consortium Workshop on Workload Characterization (Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems)*.

[2] J. A. Bergstra and C. A. Middelburg. 2003. *ITU-T Recommendation G.107 : The E-Model, a computational model for use in transmission planning.* Technical Report.

[3] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. 2012. A Tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios. *Comput. Netw.* 56, 15 (oct 2012), 3531–3547.

[4] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. 2012. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks* 56, 15 (2012), 3531–3547.

[5] Manri Cheon and Jong-Seok Lee. 2018. Subjective and Objective Quality Assessment of Compressed 4K UHD Videos for Immersive Experience. *IEEE Transactions on Circuits and Systems for Video Technology* 28, 7 (2018), 1467–1480.

[6] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2013. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *2013 20th International Packet Video Workshop*. 1–8.

[7] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2014. TAPAS: A Tool for RApid Prototyping of Adaptive Streaming Algorithms. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming* (Sydney, Australia) *(VideoNext '14)*. Association for Computing Machinery, New York, NY, USA, 1–6.

[8] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. 2013. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 1288–1293.

[9] Jim Gettys and Kathleen Nichols. 2011. Bufferbloat: Dark Buffers in the Internet: Networks without Effective AQM May Again Be Vulnerable to Congestion Collapse. *Queue* 9, 11 (nov 2011), 40–54.

[10] ZhiGuo Hu, HongRen Yan, Tao Yan, HaiJun Geng, and GuoQing Liu. 2020. Evaluating QoE in VoIP networks with QoS mapping and machine learning algorithms. *Neurocomputing* 386 (2020), 63–83.

[11] Hyoung-Kee Choi and J. O. Limb. 1999. A behavioral model of Web traffic. In *Proceedings. Seventh International Conference on Network Protocols*. 327–334.

[12] S. Shunmuga Krishnan and Ramesh K. Sitaraman. 2012. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. In *Proceedings of the 2012 Internet Measurement Conference* (Boston, Massachusetts, USA) *(IMC '12)*. Association for Computing Machinery, New York, NY, USA, 211–224.

[13] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (Monterey, California) *(Hotnets-IX)*. Association for Computing Machinery, New York, NY, USA, Article 19, 6 pages.

[14] Jeongeun Julie Lee, Maruti Gupta, and Intel Corp. 2007. A NEW TRAFFIC MODEL FOR CURRENT USER WEB BROWSING BEHAVIOR.

[15] Chao Liu, Ryen W. White, and Susan Dumais. 2010. Understanding Web Browsing Behaviors through Weibull Analysis of Dwell Time. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Geneva, Switzerland) *(SIGIR '10)*. Association for Computing Machinery, New York, NY, USA, 379–386.

[16] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. 2015. Deriving and Validating User Experience Model for DASH Video Streaming. *IEEE Transactions on Broadcasting* 61, 4 (Dec 2015), 651–665.

[17] John O'Sullivan, Darijo Raca, and Jason J. Quinlan. 2020. Godash 2.0 - The Next Evolution of HAS Evaluation. In *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 185–187.

[18] R. Pries, Z. Magyari, and P. Tran-Gia. 2012. An HTTP web traffic model based on the top one million visited web pages. In *Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012*. 133–139.

[19] Jason J. Quinlan and Cormac J. Sreenan. 2018. Multi-Profile Ultra High Definition (UHD) AVC and HEVC 4K DASH Datasets. In *Proceedings of the 9th ACM Multimedia Systems Conference* (Amsterdam, Netherlands) *(MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 375–380.

[20] A. Raake, M. Garcia, W. Robitza, P. List, S. Göring, and B. Feiten. 2017. A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1. In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. 1–6.

[21] Aleksandr Reviakin, Ahmed H. Zahran, and Cormac J. Sreenan. 2018. Dashc: A Highly Scalable Client Emulator for DASH Video. In *Proceedings of the 9th ACM Multimedia Systems Conference* (Amsterdam, Netherlands) *(MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 409–414.

[22] Zaixi Shang, Joshua P. Ebenezer, Alan C. Bovik, Yongjun Wu, Hai Wei, and Sriram Sethuraman. 2021. Assessment of Subjective and Objective Quality of Live Streaming Sports Videos. arXiv:2106.08431 [eess.IV]

[23] Shahnaz M. Shariff, Heng Li, Cor-Paul Bezemer, Ahmed E. Hassan, Thanh H. D. Nguyen, and Parminder Flora. 2019. Improving the Testing Efficiency of Selenium-Based Load Tests. In *Proceedings of the 14th International Workshop on Automation of Software Test* (Montreal, Quebec, Canada) *(AST '19)*. IEEE Press, 14–20.

[24] Babak Taraghi, Abdelhak Bentaleb, Christian Timmerer, Roger Zimmermann, and Hermann Hellwagner. 2021. Understanding Quality of Experience of Heuristic-Based HTTP Adaptive Bitrate Algorithms. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video* (Istanbul, Turkey) *(NOSSDAV '21)*. Association for Computing Machinery, New York, NY, USA, 82–89.

[25] The Open Networking Foundation. 2012. OpenFlow Switch Specification.

[26] I. Tsompanidis, A. H. Zahran, and C. J. Sreenan. 2014. Mobile network traffic: A user behaviour model. In *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*.

[27] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. 2016. EYEORG: A Platform For Crowdsourcing Web Quality Of Experience Measurements. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies* (Irvine, California, USA) *(CoNEXT '16)*. ACM, 399–412.

[28] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 325–338.