Real-time Emulation of MASQUE-based QUIC Proxying in LTE networks using ns-3

Donát Scharnitzky
High Speed Networks Laboratory,
Department of Telecommunications and Media Informatics,
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics
Budapest, Hungary
dscharnitzky@edu.bme.hu

Zsolt Krämer
Ericsson
Budapest, Hungary
zsolt.kramer@ericsson.com

Sándor Molnár
High Speed Networks Laboratory,
Department of Telecommunications and Media Informatics,
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics
Budapest, Hungary
molnar@tmit.bme.hu

Attila Mihály
Traffic Analysis and Network Performance Laboratory
Ericsson Research
Budapest, Hungary
attila.mihaly@ericsson.com

***Abstract*—Tools for real-time emulation of mobile networks are valuable for researchers due to the high amount of time and resources it allows to save compared to carrying out measurements in live networks. In this paper we present the rationale, design and prototype implementation of a novel net device in the ns-3 open source network simulator that allows for end-to-end real-time emulation of LTE networks with real endpoints. We then show the performance evaluation of a QUIC proxy built on MASQUE using our emulated LTE setup. Our results confirm the intended behavior of the implementation, however, we also show the limitations of the real-time capabilities of ns-3.**

***Index Terms*—ns-3, QUIC, MASQUE, real-time emulation**

I. Introduction

ns-3 is a discrete-time network simulator written in C++ [1]. It is widely used in networking for studies related to routing, transport layer, testing prototypes, and a variety of other scenarios. It is also capable of simulating mobile networks including LTE and has an external mmWave module [2]. ns-3 has a real-time simulation mode, which can be used to work with real servers when emulating mobile networks, however, a real client is not supported (at the writing of this paper) and this limits the possible measurement scenarios.

QUIC is a novel transport protocol created by Google and under standardization by IETF [3]. In the design of QUIC a core concept is encryption, which makes it impossible to manage it from middleboxes. This feature increases privacy and security at the cost of functionality (e.g., performance enhancing proxies are hard to design and deploy). To overcome this limitation, MASQUE was proposed in IETF, which enables the creation of cooperative QUIC proxies, without compromising the encryption [4].

The main motivation of our work is to extend the emulation capabilities of the ns-3 simulator with real-time, end-to-end LTE emulation. For this purpose we designed and implemented a new net device module for the popular ns-3 simulator (a net device represents a NIC in ns-3). This enables a real client to communicate with a real server via an emulated LTE network. Such an environment provides an appropriate platform to easily perform performance evaluation of new protocols, such as QUIC, where the protocol is rapidly evolving and the continuous evaluation and testing is especially important.

The contribution of this paper is twofold. First, we present the design and implementation of our novel net device module, which enables LTE UE emulation with a real client node. We also show its working mechanism and how it can be integrated in a ns-3 environment. Second, we demonstrate its use by an important case study of a QUIC proxy built on MASQUE protocol using our emulated LTE setup. We also show our first performance evaluation results of this study focusing on metrics like the completion time and the round-trip time. For comparison purposes the study includes three cases, i.e., the IP forwarding case and also two modes when the MASQUE proxy is used (stream and data forwarding modes). For the sake of the comparison we also consider both low throughput and high throughput cases.

The paper is organized as follows. We discuss related works regarding ns-3, proxying and QUIC in Section II. We present the implementation highlights in Section III. Our net device and the measurement environment are presented In Section IV. The measurement results is demonstrated in Section V. Finally, Section VI concludes our paper.

II. Related Work

A. Evaluating the performance of QUIC in simulations

An implementation of the QUIC protocol in ns-3 has been presented in [5]. The authors describe the design and implementation in detail and validate the behavior of the protocol with various congestion control algorithms. The code has been made public, and has been used by the community in further research projects such as [6], where the authors study the performance of QUIC with interactive, low-latency traffic such as cloud gaming, 4K streaming and online gaming. This implementation has also been used in [7], which compares TCP and QUIC performance in LTE networks. The module since then has been extended in [8] where they added the BBR (Bottleneck Bandwidth and RTT) congestion control algorithm to the protocol and provided interfaces for further extensions. While the proper behavior of QUIC is validated and the module is immensely valuable for researchers, it takes considerable effort to keep the implementation aligned with the latest version of the protocol.

B. Extending the emulation capabilities of ns-3

The ns-3 simulator [1] and its models have proven to behave realistically and are generally considered to be capable of producing measurement results that accurately represent the behavior of real networks. However, increasing the fidelity of the simulations by using real implementations of protocols and algorithms instead of the ns-3 models remains a crucial area of research. Besides the increased accuracy of the findings, another advantage of these approaches is that it can mitigate the duplicate efforts of implementing both the real-world protocols and the simulation models. One important achievement in this field is the Direct Code Execution cradle for the ns-3 simulator [9], which enables the use of real Linux kernel network stacks on the endpoints with simulated networks. Another direction towards similar goals is the extension of the emulation capabilities of the simulator, such as by the authors of [10] and [11]. These works proposed enhancements to the real-time emulation mode of ns-3, making it easier to use and configure real network interfaces for the simulations.

The authors of [12] showed that it is possible to extend the emulation support based on the EmuFdNetDevice class of the simulator, by designing a novel, DPDK-based file descriptor net device. The measurements presented in the paper showed that DpdkNetDevice was capable of achieving much higher data rates with significantly lower CPU load. Moreover, the detailed description of the design and implementation of the new net device class is also available in the paper. An interesting use case of such an emulation-based testbed can be seen in [13] where a framework is presented for QUIC interoperability testing.

Real-time emulation of LTE networks with ns-3 has been studied by [14] and [15]. Both are complex solutions, with [14] using a modified LTE stack integrated with LabView and [15] creating an integrated approach by combining ns-3 and the CORE simulator in order to achieve the result.

C. MASQUE and Cooperative Proxying of QUIC Traffic

Multiplexed Application Substrate over QUIC Encryption (MASQUE) [16] is a new protocol for using QUIC as tunnel for IP and UDP traffic. It extends the HTTP CONNECT method, thus it requires explicit user request in order to work. The use case for such a protocol is similar to VPNs (Virtual Private Networks), a proxy is requesting objects from servers on behalf of the user, which can improve privacy depending on threat level (shifts the trust to the proxy provider from the web service provider in the context of the Internet). MASQUE can be used to tunnel a QUIC connection if the client supports it as well. The two different modes of MASQUE are datagram mode and stream mode. In datagram mode, the tunneling connection (also referred to as outer connection) does not acknowledge, and in stream mode, it does. Building on MASQUE makes it possible to implement and deploy middlebox functionalities with explicit consent and without compromising the integrity of the privacy or security context of the connection [4].

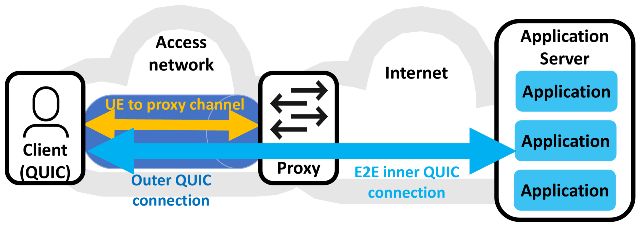

Fig. 1. Proxy functionality with QUIC tunnelling

The authors of [17] studied the performance implication of using MASQUE with different protocols (including QUIC). The throughput is a few percent less compared to traffic without tunneling, and in some cases MASQUE has better performance, e.g. when the client-proxy link is noisy but has low delay.

III. Implementation

In this section we describe our additions and modification to ns-3, focusing on the novel net device that enables LTE UE emulation with a real client node.

A. LteUeFdNetDevice

We aimed for a setup where we have a real client and a real server with a real proxy in front of it, with the proxy and the client being connected through ns-3 with LTE emulation, however this is not supported by ns-3. The main reason is that NAT is not implemented in the UE of LTE network and the LteUeNetDevice (C++ class of the net device that a host uses to connect to the LTE network in ns-3) does not support using file descriptors to communicate with the outside environment. The latter limitation could be potentially overcome by using two NetDevices (the C++ class of ns-3 net device), but we opted for creating a new NetDevice that is a mix of LteUeNetDevice and FdNetDevice (C++ class of the net device in ns-3 that supports file descriptors) to have finer control over it, called LteUeFdNetDevice. We use this new net device to mitigate the first limitation by implementing a simplified, limited NAT (with the help of another new net device on the other end of the LTE network).

This net device uses file descriptors in downlink and LTE in uplink directions, which enables the use of a real client to connect to it and then use the simulated LTE to connect to the proxy. LteUeFdNetDevice inherits from LteUeNetDevice, thus ns-3 can use it as the net device of the UE. The file descriptor capability was implemented based on FdNetDevice, however we do not inherit from it to reduce complications originating from the fact that both of these classes have the same base class (NetDevice) and implement (some of) its methods as shown in Figure 2. The classes with green background are added by us, they are based on existing implementations.

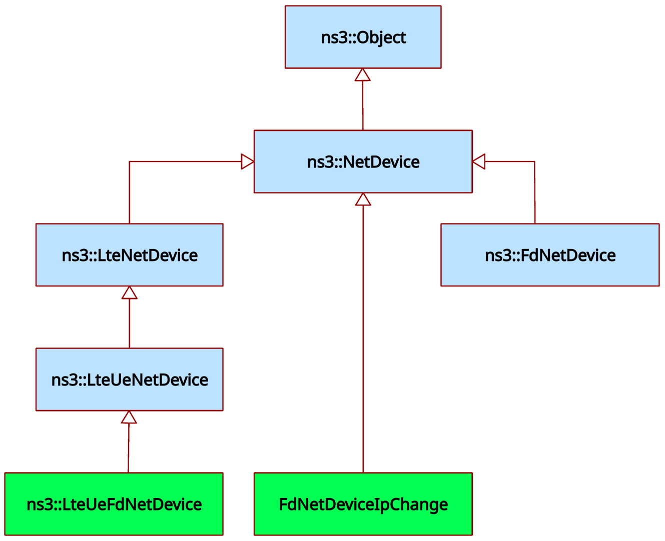

Fig. 2. Inheritance diagram of net devices. Our additions are shown with green background.

B. *Packet journey*

When a packet coming from the client arrives at the LteUeFdNetDevice, it will be handled by the ForwardUp C++ function at some point, where we call the LTE send function. This behavior basically routes the packet from the file descriptor to the LTE network. In the LTE network the eNodeB gets the packet via radio link, and sends it to the SGW node, which sends it to the PGW node, and then finally it is sent to the RightNode. This node is in ns-3 with a modified FdNetDevice (we call FdNetDeviceIpChange), and its task is to implement the communication between ns-3 and the proxy. It sends the packet to the proxy unmodified, the proxy handles it and replies. The packets coming back are handled by the ForwardUp C++ function, which implements the IP changing in FdNetDeviceIpChange, to change the destination address to that of the UEs and also adds a GTPU (GPRS Tunnelling Protocol, [18]) header with source address of the PGW node. This ensures that the LTE network will handle the packets correctly and the UE will receive them (to the SGW these packets appear as if the PGW sent them). On the side of the UE, in LteUeFdNetDevice when the packet is received, the destination address is changed back to the address of the client. Since the UDP checksum in the GTPU header is set to 0, UdpL4Protocol (the C++ class in ns-3 that represents the UDP layer) needed a modification to not drop packets with 0 checksums (this behavior is allowed by the standard, see [19]).

IV. MEASUREMENT ENVIRONMENT

The setup is realized in Docker containers as shown in Figure 3. We are using a modified version of the setup from [17], which was created based on [13]. The client container contains the QUIC client, the sim container contains the simulated LTE environment implemented in ns-3 [1], the proxy and server containers contain the MASQUE proxy and QUIC server, respectively. The sim container has a pair of virtual network devices since the EmuEpc (the module in ns-3 that implements the LTE EPC network) requires them for the SGW-eNodeB communication. The direct communication between the two interfaces of the sim container is blocked via iptables firewall, thus the packets are forced to go through ns-3.

A. *ns-3 scenario*

We have implemented a scenario in ns-3 that creates the above setup. It is required to create routes between the RightNode and the PGW and the RightNode and the SGW since the RightNode will send packets to the SGW, pretending to be coming from the PGW. The RightNode also needs routes to the proxy, since it acts as a router between ns-3 and the proxy.

The eNodeB, the UE and the RightNode have mobility models (ConstantPositionMobilityModel class in ns-3). Neither of them are moving and they constitute a rectangular triangle. The distance between the eNodeB and UE is 99 meters, which is a typical distance in LTE networks (the position of the RightNode is actually irrelevant).

The scenario configures the LTE and the simulation parameters as shown in Table I. The modules are the following:

Global: sets the simulation type to real time and the checksum calculation to be off.

RealtimeSimulatorImpl: sets the hard limit parameter of the simulator. This is the maximum seconds that the simulator can lag behind the wallclock.

LteUePhy: sets the transmit power and noise level of all UEs.

LteEnbPhy: sets the transmit power and noise level of all eNodeBs.

LteSpectrumPhy: turns off error models of the LTE spectrum layer.

LteHelper: controls parameters of the LTE helper, which is used to set up the different parts of the LTE network.

LteEnbRrc: sets the default transmission mode of eNodeBs to SISO.

DropTailQueue<Packet>: default values for the drop tail queues. The max queue size is set to a low value to induce packet loss.

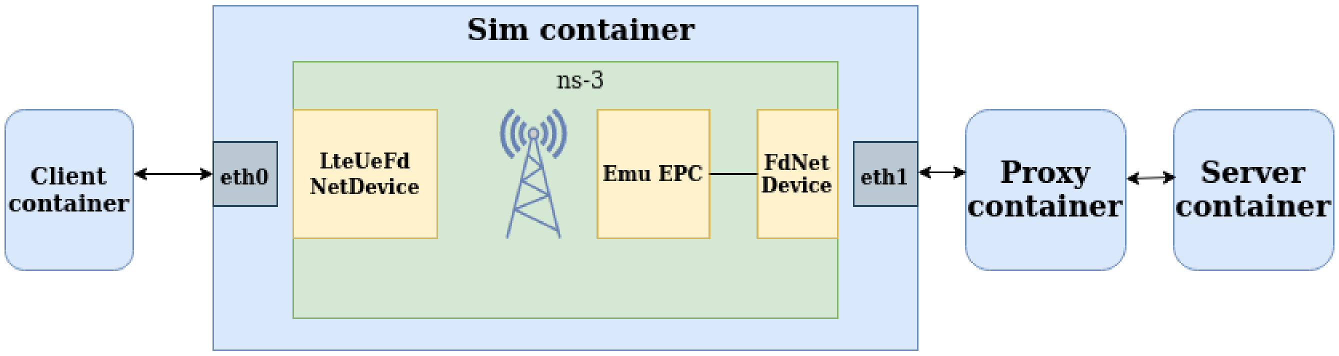

Fig. 3. Setup topology

LteAttribute: configures the scheduler type, the loss- and fading modes used in the simulations.

FadingModelAttribute: sets the file name to use and the number of RBs (the default is 100 also).

EnbDevice: configures the parameters of the net devices of the eNodeBs. The first 2 are the number of resource blocks allocated in downlink and uplink, respectively. The last 2 are the EARFC of the channel in downlink and uplink, respectively. The number of resource blocks are changed depending the measurement.

V. Measurements

The measurements were done using the setup described above. We differentiate between lower and higher LTE throughput measurements, which we can control by setting the number of resource blocks (DlBandwidth and UlBandwidth) to either lower or higher values. The process consists of downloading a 10MB sized file. The RTT between the client and proxy containers is around 38ms as reported by the ping program (when no file download is happening). The one way delay between the RightNode and PGW is configured to be 10ms.

A. Lower performance

We set the LTE parameters to values that causes low throughput. To achieve this, the uplink and downlink resource blocks are each given the value 25.

Figure 4 shows the results of measurements with 3 different set of parameters where each of them ran 10 times. The first uses basic IP forwarding and the last two uses the MASQUE proxy. IP forwarding is accomplished via setting up DNAT in the proxy container and turning off proxy mode in the client and server (and not starting the MASQUE proxy in the proxy container). In proxy datagram mode, the outer QUIC layer does not acknowledge the packets, instead this is done in the tunneled connection (which also uses the QUIC protocol). In stream mode, the acknowledgement is done in the proxy (outer connection), and as can be seen in the figure, it increases the completion time significantly more (30.9% on average), than datagram mode (4.5% on average). This is the expected behaviour that results from the implemented infinite buffer size of the stream mode. Comparing this result to the prior work of [17] one can see the same effect.

Figure 5 shows the RTT for the different set of measurements. As can be seen, the delay between the client and the proxy (in proxy mode) in either datagram mode or stream mode is the same. The RTT is higher in IP forward mode and in proxy (datagram) mode (between the client and server), because of the 25ms one way delay between the proxy and the server, and the difference between them is small, the latter

TABLE I
The table shows the parameters and their values for the different modules in ns-3.

Global	
Simulator ImplementationType	RealtimeSimulatorImpl
ChecksumEnabled	False
RealtimeSimulatorImpl	
SynchronizationMode	HardLimit
HardLimit	0.2s
LteUePhy	
TxPower	10.0
NoiseFigure	7.0
LteEnbPhy	
TxPower	30.0
NoiseFigure	5.0
LteSpectrumPhy	
DataErrorModelEnabled	true
CtrlErrorModelEnabled	true
LteHelper	
UsePdschForCqiGeneration	false
UseIdealRrc	true
LteEnbRrc	
EpsBearerToRlcMapping	RLC_UM_ALWAYS
DefaultTransmissionMode	0
DropTailQueue<Packet>	
MaxSize	16
LteAttribute	
SchedulerType	PfFfMacScheduler
PathlossModel	FriisPropagation LossModel
FadingModel	TraceFadingLossModel
FadingModelAttribute	
TraceFilename	fading_trace_EPA_3kmph.fad
RbNum	100
EnbDevice	
DlBandwidth	{25,100}
UlBandwidth	{25,100}
DlEarfcn	9895
UlEarfcn	27785

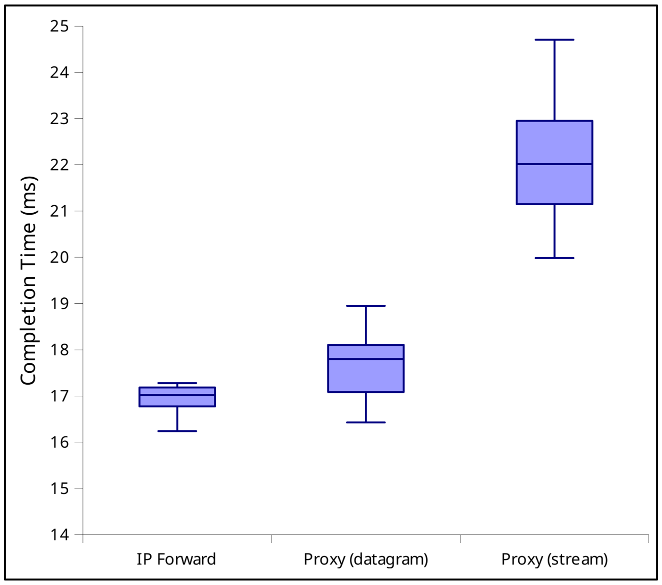

Fig. 4. Completion time shown for IP forward (left), proxy with datagram mode (middle) and proxy with stream mode (right) setups with lower bandwidth limit

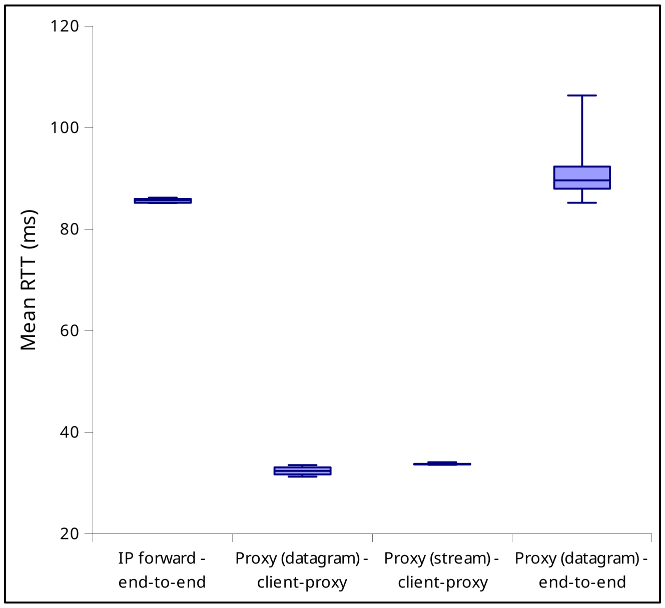

Fig. 5. From the perspective of the client, the RTT shown for (from left to right): 1. IP forward between the client and server, 2. Proxy (datagram mode) between the client and the proxy, 3. Proxy (stream mode) between the client and proxy, 4. Proxy (datagram mode) between the client and server.

being slightly higher. In these configurations the simulator's lag behind real time adds to the overall RTT, which causes the delay to be higher than expected (10ms one way delay between the RightNode and PGW, and the LTE networks delay should be less than 35ms). The proxy stream mode client to server RTT is not shown in this figure because the results were erroneous.

B. *Higher performance*

We set the LTE parameters to values that causes higher throughput to simulate adequate reception conditions. To achieve this, the uplink and downlink resource blocks are each given the value 100.

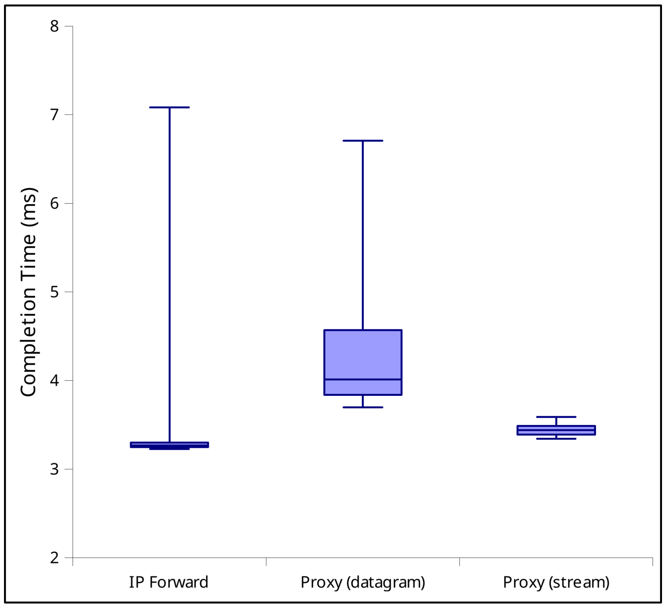

Fig. 6. Completion time shown for IP forward (left), proxy with datagram mode (middle) and proxy with stream mode (right) setups with higher bandwidth limit

Figure 6 shows the results of measurements with the same setup as above, with the only difference of higher resource block resulting in higher bandwidth limit. In this case, the stability of the IP forward and datagram proxy mode is degraded (as indicated by the outlier higher completion time). The average completion time in datagram proxy mode is 19.3% higher compared to IP forward mode. The completion time in stream proxy mode is 5.8% less than in IP forward mode on average, which is caused by the individual high value in the latter, as the median is actually 5.4% higher.

Figure 7 shows the RTT for the different set of measurements. Comparing with the previous results (Figure 5) the RTT increased overall, which can be explain by the higher throughput, thus higher computation requirements by the simulator, even though the completion times are much lower. The delay, however, not increased equally. In case of IP forwarding, where previously it was slightly lower than proxy in datagram mode (client-server delay), now it is slightly higher. We can also see in this figure that the proxy in stream mode has the

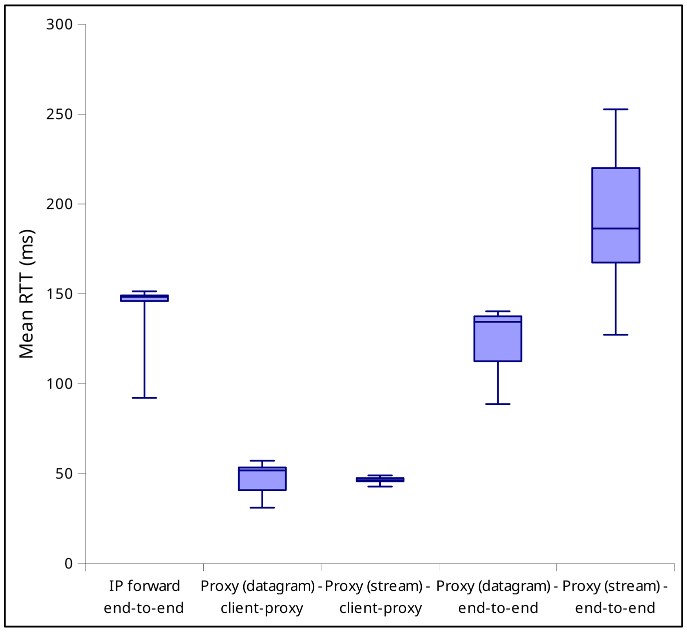

Fig. 7. From the perspective of the client, the RTT shown for (from left to right): 1. IP forward between the client and server, 2. Proxy (datagram mode) between the client and the proxy, 3. Proxy (stream mode) between the client and proxy, 4. Proxy (datagram mode) between the client and server, 5. Proxy (stream mode) between the client and the server.

highest RTT. The increase is 57ms for IP forward mode, and 15ms for proxy datagram mode (client-proxy delay), which hints that the added delay by ns-3 increased, but it is not the sole cause of this, and the RTT between the proxy and server is higher as well. A possible cause for the latter is higher processing requirement in the server, necessitated by the higher throughput.

VI. Conclusions

Adding a new net device to ns-3 that enables a real client to communicate with a real server via an emulated LTE network has two main benefits. It makes it easier to test new protocols in a simulated network environment, since there is no need to implement it in the simulator and a real LTE environment is not needed.

The implementation is a proof of concept, there are multiple ways to enhance it. Future works could expand on it to enable full NAT support, which in turn would enable multiple real clients to connect through one LTE UE node. A study could assess the feasibility of a similar modification to the mmWave module. This would further increase the testing capabilities on ns-3.

It is clear that simulating an LTE network have limitations caused by the resource (memory, CPU speed) hungry nature of the process. In case of ns-3 it manifests as additional delay. For our low throughput setups this is not pronounced, however, in high throughput scenarios this can be significant. It is for further study to understand the achievable cellular throughput in real time using more powerful computing resources. If real-time ns-3 emulation is not strictly required, our implementation provides a valuable tool since it does not require any radio specific equipment.

Acknowledgment

The authors are thankful for the fruitful discussions and guidance from Mirja Kühlewind, Marcus Ihlar, Magnus Westerlund and Zaheduzzaman Sarker (Ericsson).

References

[1] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*, pp. 15–34, Springer, 2010.

[2] M. Mezzavilla, S. Dutta, M. Zhang, M. R. Akdeniz, and S. Rangan, "5g mmwave module for the ns-3 network simulator," in *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 283–290, 2015.

[3] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, pp. 183–196, 2017.

[4] Z. Krämer, M. Kühlewind, M. Ihlar, and A. Mihály, "Cooperative performance enhancement using quic tunneling in 5g cellular networks," in *Proceedings of the Applied Networking Research Workshop*, pp. 49–51, 2021.

[5] A. De Biasio, F. Chiariotti, M. Polese, A. Zanella, and M. Zorzi, "A quic implementation for ns-3," in *Proceedings of the 2019 Workshop on ns-3*, pp. 1–8, 2019.

[6] A. Bujari, C. E. Palazzi, G. Quadrio, and D. Ronzani, "Emerging interactive applications over quic," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–4, IEEE, 2020.

[7] A. I. Kyratzis and P. G. Cottis, "Quic vs tcp: A performance evaluation over lte with ns-3," *Communications and Network*, vol. 14, no. 1, pp. 12–22, 2021.

[8] U. Paro, F. Chiariotti, A. A. Deshpande, M. Polese, A. Zanella, and M. Zorzi, "Extending the ns-3 quic module," in *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 19–26, 2020.

[9] H. Tazaki, F. Urbani, and T. Turletti, "Dce cradle: Simulate network protocols with real stacks," in *Workshop on NS3 (WNS3)*, 2013.

[10] G. Carneiro, H. Fontes, and M. Ricardo, "Fast prototyping of network protocols through ns-3 simulation model reuse," *Simulation modelling practice and theory*, vol. 19, no. 9, pp. 2063–2075, 2011.

[11] H. Fontes, R. Campos, and M. Ricardo, "Improving ns-3 emulation support in real-world networking scenarios," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 3, no. 9, pp. e5–e5, 2016.

[12] H. Patel, H. Hiraskar, and M. P. Tahiliani, "Extending network emulation support in ns-3 using dpdk," in *Proceedings of the 2019 Workshop on ns-3*, pp. 17–24, 2019.

[13] M. Seemann and J. Iyengar, "Automating quic interoperability testing," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pp. 8–13, 2020.

[14] R. Gupta, B. Bachmann, R. Ford, S. Rangan, N. Kundargi, A. Ekbal, K. Rathi, M. I. Sanchez, A. De La Oliva, and A. Morelli, "Ns-3-based real-time emulation of lte testbed using labview platform for software defined networking (sdn) in crowd project," in *Proceedings of the 2015 Workshop on ns-3*, pp. 91–97, 2015.

[15] A. Sabbah, A. Jarwan, I. Al-Shiab, M. Ibnkahla, and M. Wang, "Emulation of large-scale lte networks in ns-3 and core: A distributed approach," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pp. 1–6, IEEE, 2018.

[16] D. Schinazi, "Proxying UDP in HTTP," Internet-Draft draft-ietf-masque-connect-udp-11, Internet Engineering Task Force, May 2022. Work in Progress.

[17] M. Kühlewind, M. Carlander-Reuterfelt, M. Ihlar, and M. Westerlund, "Evaluation of quic-based masque proxying," in *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC*, pp. 29–34, 2021.

[18] E. U. T. R. A. Network, "S1 application protocol (s1ap)(release 10)," *Technical Specification*, vol. 36, 2011.

[19] J. Postel *et al.*, "User datagram protocol," 1980.