Link Emulation on the Data Link Layer in a Linux-based Future Internet Testbed Environment

Martin Becke, Thomas Dreibholz, Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
{martin.becke,dreibh,rathgeb}@iem.uni-due.de

Johannes Formann
University of Duisburg-Essen
Institute for Computer Science
Schützenbahn 70, 45117 Essen, Germany
jformann@dc.uni-due.de

***Abstract*—Protocol design and development is not a straightforward process. Each approach must be validated for interactions and side-effects in the existing network environments. But the Internet itself is not a good test environment, since its components are not controllable and certain problem situations (like congestion or error conditions) are difficult to reproduce. Various testbeds have been built up to fill this gap. Most of these testbeds also support link emulation, i.e. using software to mimic the characteristic behaviour of certain kinds of network links (like bandwidth bottlenecks or error-prone radio transmissions). The most popular link emulation systems are the Linux-based NETEM and DUMMYNET, which are e.g. applied on the IP layer of Planet-Lab and various other testbeds. However, the restriction to the OSI Network Layer (here: IP) is insufficient to test new non-IP Future Internet protocols.**

In this paper, we first introduce DUMMYNET and NETEM. After that, we will present our approach of adapting DUMMYNET for Linux to support link emulation on the Data Link Layer. Finally, we evaluate the applicability and performance of DUMMYNET and NETEM for link emulation on the Data Link Layer, in a Planet-Lab-based testbed environment. Our goal is to outline the performance and limitations of both approaches in the context of Planet-Lab-based testbeds, in order to make them applicable for the evaluation of non-IP Future Internet protocols.[1]

Keywords: Link Emulation, Data Link Layer, Future Internet Testbed, NETEM, DUMMYNET

I. INTRODUCTION

The protocol development for the current Internet bases on a strictly hierarchical structure, which has been standardized as the OSI reference model [1]. This model covers classic Internet applications like e-mail and file transfer quite well. However, the rapid technological developments driven by the needs of new applications (e.g. mobility, e-commerce, VoIP) show the conceptual limitations of this approach. Solutions like cross-layer optimization weaken the hierarchical structure and make the resulting protocol implementations difficult to develop and maintain. This makes the realization and deployment of new features and protocols, e.g. multipath TCP [2], difficult. On the other hand, clean-slate service-oriented frameworks try to solve such conceptual issues by completely getting rid of the hierarchical structure. Multiple large research projects like FIRE and G-Lab in Europe, GENI and FIND in the U.S.A. and AKARI in Asia examine such approaches.

[1]Parts of this work have been funded by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung) and the German Research Foundation (Deutsche Forschungsgemeinschaft)

In every case, protocol design and development is more than a theoretical procedure. Over the years, protocol and network developers strike new paths to validate their approaches and concepts. Long-term experience shows that protocol development involves several more steps besides requirement formulation, specification, verification and documentation. In particular, it also includes the test of implementation and conformance as well as analysing and optimizing the performance for special applications and architectures. This requires the integration of the new ideas and approaches into real hardware, emulation and also into simulation testbed environments. That is, each new approach and proof of concept needs an adequate testbed. But each research community has its own specific requirements on its testbeds, resulting in many different variants. Examples of such testbeds are Emulab [3], VINI [4], One-Lab [5], G-Lab [6] and Planet-Lab [7].

Link emulation (i.e. using software to e.g. apply delay or bandwidth limitations of a satellite link to certain packet flows) is a widely used feature in such testbeds. Currently, the testbeds apply this link emulation feature on IP flows (i.e. on the Network Layer). While this is sufficient within IP networks – i.e. the current Internet protocols – it prevents the usage for new non-IP Future Internet approaches. But if projects are investigate on clean clean-slate service-oriented framework below IP – like German Lab project – emulation on a layer below is needed. Therefore, a link emulation solution on the Data Link Layer is desirable within testbeds. DUMMYNET [8], [9] (e.g. used by Planet-Lab, G-Lab and Emulab, see [10]) and NETEM [11] are popular tools for link emulation on Linux, which is the operating system used by the majority of the testbeds. NETEM already supports Data Link Layer usage, while DUMMYNET under Linux had lacked of this feature. In this paper, we first introduce our approach of extending the Linux version of DUMMYNET by Data Link Layer support. Then, we evaluate the performance of both approaches in a Planet-Lab-based testbed environment – used in the German Lab project – which we have extended by the support of link emulation on the Data Link Layer. The challenge here is to compare at first time link emulation an data link layer on one operation system.

II. Link Emulation

The intention of testbeds is to examine concepts and architectures under different conditions, e.g. nearly optimal conditions for a first proof of concept or more realistic conditions to analyse specific scenarios like communication over modem, satellite or other wireless links.

A. Constituting Physical Constraints in a Testbed

Clearly, the physical model (e.g. the underlying transmission technology) is an important part of such tests, because it realizes the constraints set by physics and hardware implementations. The most realistic results on testing a system with certain hardware constraints is to actually run it on real hardware. This approach is e.g. used by the G-Lab testbed [6]. However, real network behaviour – like the effects caused by background traffic or BGP routing in the Internet – are not easily reproducible in such hardware-centric systems. Approaches like the Planet-Lab [7] interconnect a large number of virtualized systems over the real Internet, allowing for overlay network tests. However, in this case, the hardware itself is not under the control of the researcher.

For many upper-layer test cases, an emulation model of the physical constraints is already sufficient, e.g. a satellite link with its typical delay and bit errors may be emulated in software. Such an emulation is also easily adaptable to special cases and allows for easy reproduction of the results, e.g. to examine the communications protocol performance over a satellite link during solar wind. Particularly, software emulation is inexpensive, since special hardware (e.g. a real satellite link) are not needed. The most well-established emulation systems in the context of testbeds emulation systems are DUMMYNET [9] and NETEM [11]. These tools emulate links with variable delay, bandwidth, packet loss and jitter, i.e. they can be used to model the link characteristics of different network access technologies. We will introduce DUMMYNET and NETEM in the following subsections.

B. NetEm

The Linux Advanced Traffic Control Framework [12], which is part of the Linux kernel, uses filtering rules to map packets or frames – i.e. data on Data Link as well as Network Layers – to queuing disciplines (QDisc) of an egress network interface. QDiscs may be classful, i.e. contain a hierarchy of subclasses. Filtering rules (denoted as classifier) of the QDisc itself map packets or frames to the subclasses. Each subclass may have its own QDisc, which again may be classful or classless (i.e. no subclasses). NETEM [11] is a classful QDisc for Linux. This QDisc itself provides packet delay, loss, duplication and re-ordering.

If bandwidth limitations are required, a secondary queuing discipline – like the classless Token Bucket Filter (TBF) QDisc – has to be applied as sub-QDisc of NETEM to control the data rate. Bandwidth limitations are always based on the Data Link Layer frame sizes of the egress interface on which NETEM and subservient QDiscs are configured on. That is, using e.g. NETEM and TBF on an Ethernet interface, all bandwidth calculations for packets include the Ethernet headers and trailers.

C. Dummynet

DUMMYNET [9] provides link emulator functionality in the FreeBSD kernel. The packet filtering architecture of the kernel is used to pass packets through one or more queues. A hierarchy of the queues is realized by so-called pipes. A pipe represents a fixed-bandwidth channel; queues actually store the packets. Each queue is associated with a weight. Proportionally to its weight, it shares the bandwidth of the pipe it is connected to.

Originally, DUMMYNET had been realized on the Network Layer to control bandwidth, delay and jitter as well as packet loss rate, duplication and reordering of IP packet flows. A recent patch [13] for FreeBSD has added support for the Data Link Layer, i.e. the patched DUMMYNET implementation is able to handle frames on the Data Link Layer containing arbitrary Network Layer traffic. This allows for applying DUMMYNET to handle non-IP Future Internet protocols. [8] introduces a port of DUMMYNET to Linux, in order to apply it for Planet-Lab-based G-Lab Experimental Facility. However, this port does *not* support Data Link Layer traffic, due to the significantly different handling procedures of Data Link Layer frames in Linux. It is in the end an adaptation of a new packet filter mechanism on the Data Link Layer.

Bandwidth limitations realized by DUMMYNET always base on the Network Layer packet size only, even if DUMMYNET is used to restrict Data Link Layer traffic. That is, using DUMMYNET e.g. to shape IP traffic over an IEEE 802.11 WLAN does *not* include the WLAN headers and trailers.

III. Dummynet on Linux

The link emulation infrastructure of Planet-Lab-based G-Lab – as well as of its derived testbeds – is based on the Linux port of DUMMYNET [8]. In order to extend the G-Lab infrastructure by DUMMYNET-based Data Link Layer support, we had to extend DUMMYNET for Linux.

Figure 1 presents our concept for extending DUMMYNET on Linux by the support of link emulation on the Data Link Layer. The existing DUMMYNET hooks into the Network Layer packet chains of the Routing Subsystem. Chains are used by the packet filtering architecture of Linux and provide mechanisms to intercept and manipulate packets. Before a packet is routed, it traverses the PREROUTING chain. Packets to be forwarded to another system then pass through the FORWARD chain while packets destined for the system itself are handled by the INPUT chain. Packets sent from the system itself come from the OUTPUT chain. After routing, a packet passes through the POSTROUTING chain. The two hooks used by DUMMYNET are on the PREROUTING and POSTROUTING chains. DUMMYNET can intercept a packet, and eventually return it some time later into its original chain.

This Network Layer concept for DUMMYNET can be extended to the Data Link Layer in the Bridging Subsystem. Note, that Linux uses the same chain naming (i.e. PREROUTING, FORWARD, INPUT, OUTPUT, POSTROUTING) as for the Routing Subsystem. Nevertheless, the Routing and Bridging Subsystems are completely independent. DUMMYNET has to be extended by the support for hooking also into the

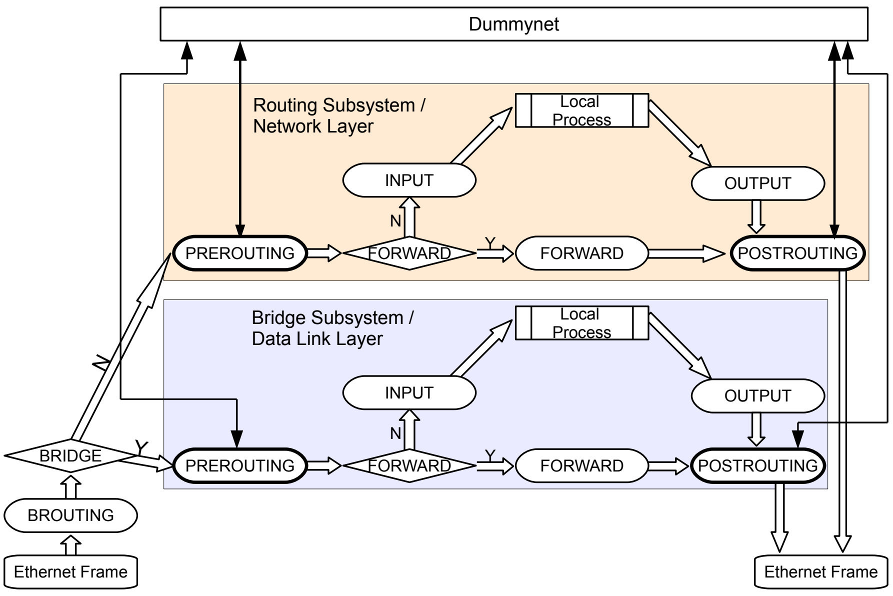

Figure 1. Our Concept of Extending DUMMYNET on Linux

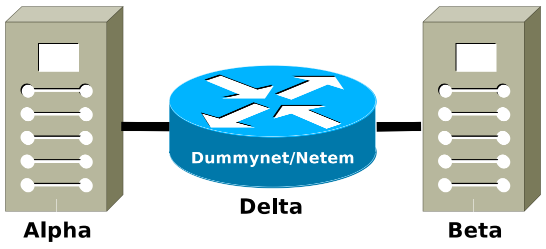

Figure 2. Testbed Setup

chains of the Bridging Subsystem to intercept frames on the Data Link Layer. Furthermore, the internal data handling of DUMMYNET has to be adapted to handle the frame structures.

The implementation of our approach has been realized as a patch for the Linux kernel, supporting all functionalities being necessary for our performance evaluation. After extending additional filtering features being necessary for real G-Lab deployment, we will contribute our patch to the G-Lab kernel development process. In this context it should be noticed, that this changes are independet from the infrastructure of planetlab, so that there should no drawbacks for this approach.

IV. TESTBED ENVIRONMENT

In order to evaluate the performance of DUMMYNET and NETEM, we have set up a G-Lab-based Linux test environment as shown in figure 2: the sender node "Alpha" is connected via router "Delta" to the receiver node "Beta" over 100 Mbit/s full-duplex Ethernet links. Table I provides the technical details of the systems; their hardware has been intentionally chosen to be "low performance", in order to illustrate the effects of high CPU load on the DUMMYNET/NETEM performance (which is a likely situation for G-Lab nodes hosting a large number of active slices). Router "Delta" is configured with both, DUMMYNET (including our Data Link Layer support extension as described in section III) and NETEM. The Planet-Lab-based test infrastructure has been extended to support link emulation on the Data Link Layer in addition to the already existing link emulation on the Network Layer. A configuration option decides whether to apply DUMMYNET or NETEM.

NUTTCP [14] has been used for throughput and packet loss measurements using the UDP protocol. Due to the different traffic measurement bases of NETEM (Data Link Layer frame size, see subsection II-B) and DUMMYNET (Network Layer packet size, see subsection II-C), we have configured the packet output rate R^{nuttcp} of NUTTCP appropriately to achieve a desired on-network Data Link Layer rate R_{Network} for a given payload message size M and IP header size H_{IP}, UDP header size H_{UDP} and Ethernet header/trailer size H_{Ethernet}:

$$R^{\text{nuttcp}}_{\text{NetEm}} = \frac{M * R_{\text{Network}}}{M + H_{\text{IP}} + H_{\text{UDP}}} \quad (1)$$

$$R^{\text{nuttcp}}_{\text{Dummynet}} = \frac{M * R_{\text{Network}}}{M + H_{\text{IP}} + H_{\text{UDP}} + H_{\text{Ethernet}}} \quad (2)$$

Our bandwidth results always show the achieved Data Link Layer throughput at the receiver side.

For measuring delay, we have utilised the standard PING tool (which uses ICMP Echo Requests and Replies).

V. PERFORMANCE ANALYSIS

In the following analysis, we examine the performance of DUMMYNET and NETEM based on the studies in [8], [11], [15]. But unlike former studies, our interest is in the performance of link emulation on the Data Link Layer, which has not been examined before – but which becomes crucial when examining Future Internet protocols on top of it. In the following subsections, we evaluate Planet-Lab-based setups

Node Name	Processor	Memory	Role
Alpha	700 MHz AMD Duron	512 MiB	Sender
Beta	700 MHz AMD Duron	512 MiB	Receiver
Delta	1666 MHz AMD Athlon	1024 MiB	Router

Table I
TECHNICAL DETAILS OF OUR G-LAB-BASED TESTBED

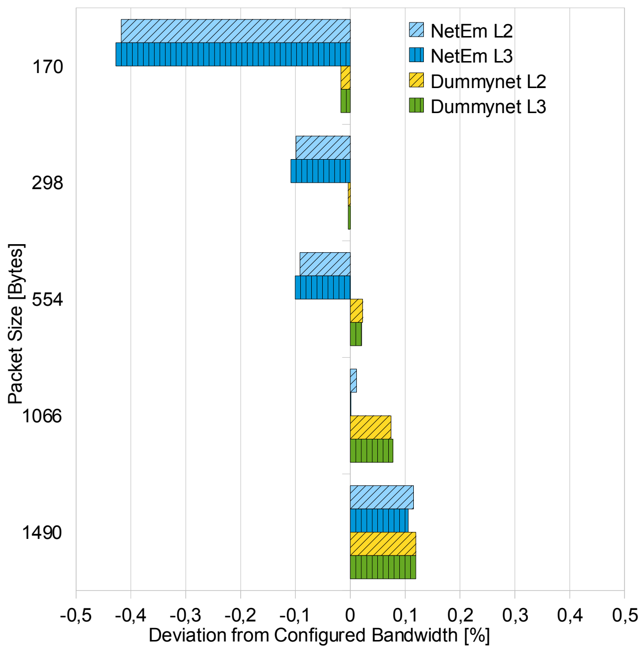

Figure 3. Derivation from Configured Bandwidth

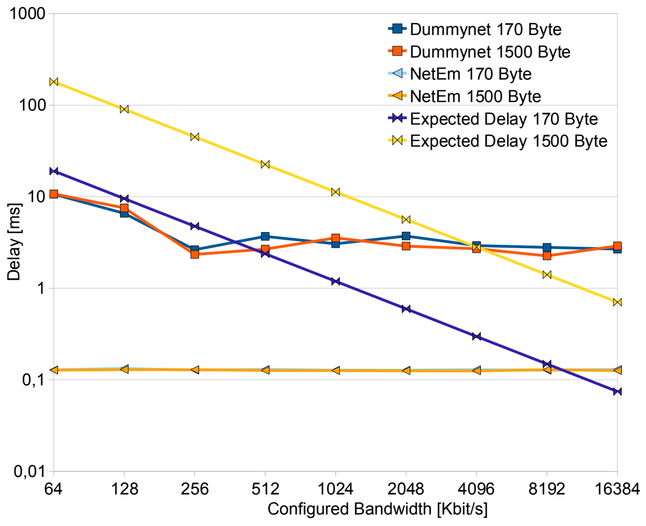

Figure 4. Expected and Measured Delays for Different Packet Sizes

with the link emulation varying the basic QoS measures: error rate, bandwidth limitation, delay and jitter.

A. Bandwidth

Bandwidth is probably the most crucial QoS measure. Therefore, the accurate adherence of configured bandwidth limitations – for any kind of traffic pattern – is a highly important feature of a link emulation system. Figure 3 shows the deviations from the desired bandwidth of 8389 Kbit/s (setting based on a DSL media streaming scenario) for DUMMYNET and NETEM on Data Link (L2 – Layer 2) and Network (L3 – Layer 3) Layers when varying the packet size. Small packets particularly occur in multimedia scenarios, leading to a high per-byte routing/bridging overhead. DUMMYNET and NETEM handle different packet sizes on both layers quite well, with an increased deviation for NETEM when using small packets (about -0.4% for 170 byte packets). However, this deviation still remains small and should be uncritical for most use cases.

In this context, it has to be mentioned that tests configured with relatively short traffic duration time and large buffers – over particularly low-bandwidth emulated links – can lead to distortions of the measurements by the time necessary to fully transmit the buffered packets. Furthermore, due to the bandwidth-delay product, packets with larger size require a proportionally longer transmission time on the link. This delay – depending on the configured bandwidth and used packet sizes – is *not* being emulated by DUMMYNET or NETEM. An example is provided in figure 4, which shows the expected and measured delays for varying bandwidth using packets of 146 bytes and 1494 bytes. The user should be aware of this fact if packet sizes differ significantly.

In case of bandwidth limitation, it is important to figure out the limitation of the testbed hardware. Depending on the needs of the applications, the traffic pattern could differ in the size of the messages, e.g. small messages in a multimedia setup or full MTUs in a download scenario. The resulting loss rate for enforcing a maximum bandwidth of 100 Mbit/s using packet sizes of 146 bytes and 1494 bytes is shown in figure 5; the left-hand plot presents the DUMMYNET results, the right-hand plot the NETEM results. Since the data rate generated by NUTTCP is less or equal to the enforced data rate, no loss should occur. But obviously, small packet sizes result in packet losses [16], since the hardware (CPU, but also network interface cards and buffers) are incapable of handling the high number of packets per second. This side effect – which is caused by interrupt frequency, timer resolution, the latency of context-switch operations by the operating system and also limited queue sizes [17] – has to be considered carefully when planning experiments in the testbed. These limitations apply for both, Data Link as well as Network Layer link emulation. But in comparison to NETEM, DUMMYNET shows these side effects earlier: at a rate of about 30 Mbit/s and a packet rate

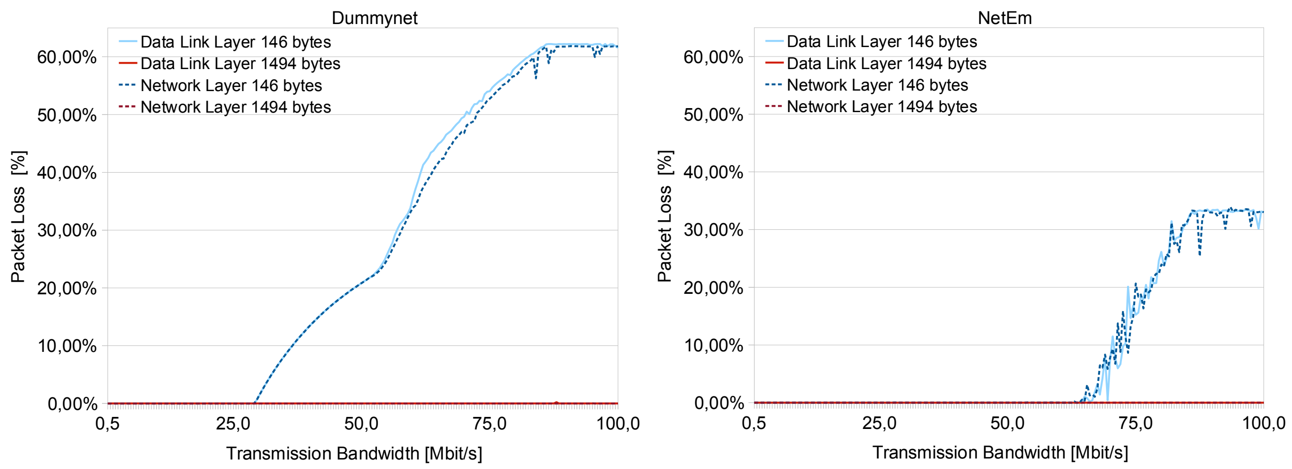

Figure 5. Hardware Limitations of DUMMYNET and NETEM for Different Packet Sizes

of 146 bytes, DUMMYNET starts losing packets while NETEM is capable of handling more than 60 Mbit/s without losses.

B. Error Rate

The loss rate is another important QoS measure for link emulation, e.g. to emulate lossy wireless or satellite connections. Therefore, it is useful to examine the accuracy of DUMMYNET and NETEM to adhere to a configured loss rate. For our measurement, NUTTCP has generated the configured data rate, while the link emulation has applied a configured loss rate. Figure 6 shows the absolute deviation from the configured loss rates for DUMMYNET and NETEM, using Data Link Layer (Layer 2) as well as Network Layer (Layer 3) link emulation. Since small packets are the most performance-critical (due to the high rate of packets/s), we only present the results for a packet size of 146 bytes here.

While the deviations for a NUTTCP bandwidth of 1 Mbit/s remain small for both, DUMMYNET and NETEM, a significant deviation is observable already for DUMMYNET at 40 Mbit/s. Even higher deviations can be found at a bandwidth of 70 Mbit/s. In this case, also NETEM shows an increased loss rate deviation, but this is still significantly smaller than for DUMMYNET. The reason of this deviation for DUMMYNET is again the limitation of the system resources, as already observed for the bandwidth limitation in subsection V-A. Again, NETEM can cope significantly better with these limited resources and still achieve a reasonable performance in parameter ranges where DUMMYNET is unable to work properly any more. This property applies for Data Link as well as Network Layer link emulation.

C. Delay

In order to examine the accuracy of the link emulation to adhere to a configured packet delay, we have varied the desired delay in a 100 Mbit/s setup for packet sizes of 170 bytes and 1500 bytes (i.e. full MTU on the Network Layer). The results are presented in table II for DUMMYNET and NETEM on Data Link (L2 – Layer 2) and Network (L3 – Layer 3) Layers. For both layers, the differences between the two packet sizes (i.e. small packets vs. full MTU) are quite small. However, it is observable that the delay results achieved by NETEM more accurately reach the configured target delay. For example, the difference to a target delay of 100 ms is almost 2 ms for DUMMYNET, but only 0.01 ms for NETEM. Also, the delay achieved by DUMMYNET is a little bit smaller than the actual target delay in most cases. This may distort measurements expecting a hard lower bound on the packet latency.

D. Jitter

While NETEM provides a configuration option to apply certain jitter distributions to the traffic, jitter is not directly supported by DUMMYNET. In DUMMYNET, jitter can be mimicked by configuring a set of pipes with different delays. Traffic is mapped to these pipes appropriately to reach a certain delay distribution. Due to these differences, it is not possible to directly compare the jitter performance of both approaches. We therefore show the Data Link and Network Layer performances of both systems separately.

For our jitter examination, we have configured a 100 Mbit/s setup using 1500 byte packets (i.e. full MTU). NETEM has been configured with a normal delay distribution of 100 ms average, while DUMMYNET has been set up with 11 pipes to mimic a similar distribution. The delay distributions are presented in figure 7; the left-hand plot shows the DUMMYNET results, the right-hand plot the distribution for NETEM. The Network Layer (Layer 3) values are displayed by the bars, the line depicts the Data Link Layer (Layer 2) values. As expected, the behaviour for Data Link and Network Layers is quite similar. Due to the different capabilities of NETEM, the distribution for NETEM is quite smooth while for DUMMYNET the 11 pipes are clearly observable as large peaks. To achieve a smoother distribution, DUMMYNET could be set up with a larger number of pipes. However, the complexity and resource consumption of such a kind of configuration would be extraordinarily high.

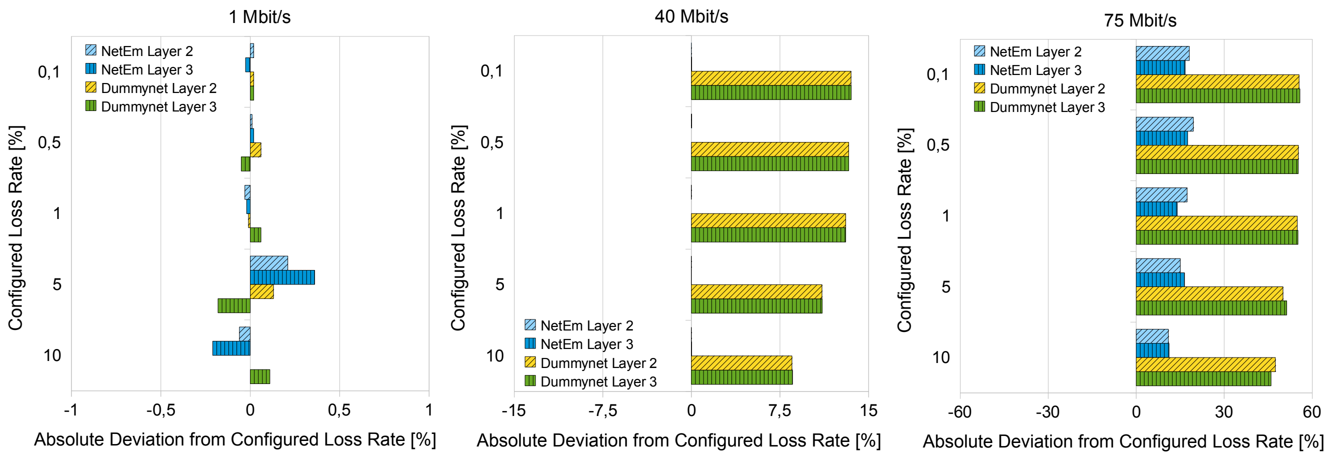

Figure 6. Derivation from Configured Error Rate for using DUMMYNET and NETEM

	Delay in ms for packet size of 1500 Byte							
Emulation\Parameter	5	10	20	50	100	200	500	1000
Dummynet L2	5.9	10.5	18.3	50	98.3	198.1	498.1	998.1
Dummynet L3	5.9	10.5	18.3	50.2	98.4	198.2	498,1	998.2
NetEm L2	5	10	20	50	100	200,1	500	1000
NetEm L3	5	10	20	50	100	200.2	500.1	1000.1

	Delay in ms for packet size of 170 Byte							
Emulation\Parameter	5	10	20	50	100	200	500	1000
Dummynet L2	5,97	11.69	18.4	50.84	98.24	198.4	498.25	998.35
Dummynet L3	5,93	11.69	18.62	50.54	98.2	198.41	498.26	998.22
NetEm L2	5	10	20.01	50	100.01	200.01	500.01	1000.01
NetEm L3	5	10	20.01	50.01	100.01	200.01	500.01	1000.01

Table II
DELAY ON DATA LINK AND NETWORK LAYER

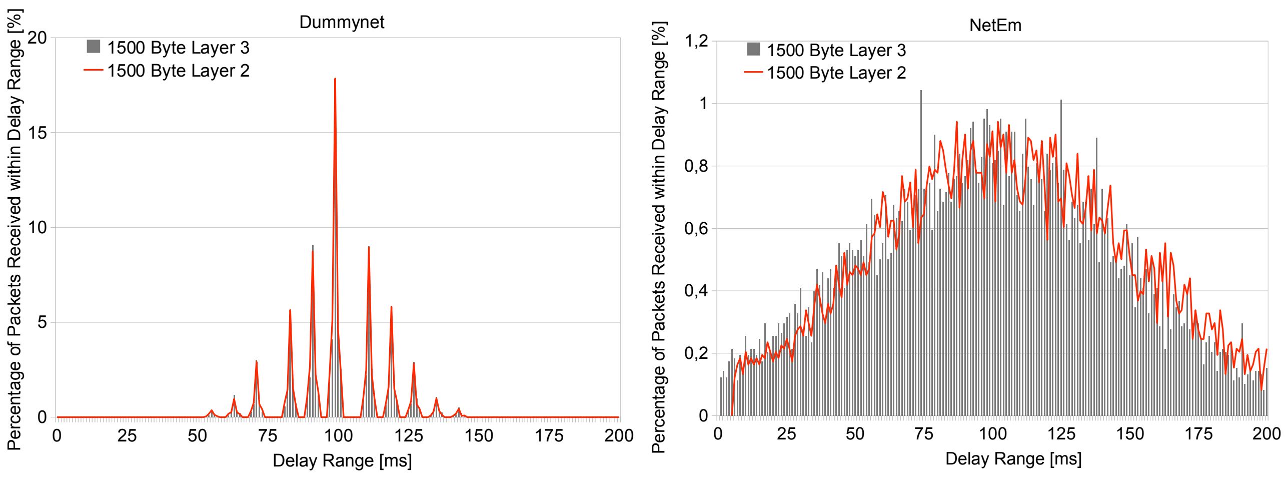

Figure 7. Emulating Jitter with DUMMYNET and NETEM

	146 Byte Packets					1500 Byte Packets	
Bandwidth	Baseline	L2 NETEM	L2 DMYNET	L3 NETEM	L3 DMYNET	L2 DMYNET	L3 DMYNET
1 Mbit/s	0.03%	0.04%	0.96%	0.04%	0.96%	0.10%	0.10%
15 Mbit/s	0.28%	0.39%	13.83%	0.50%	14.51%	1.56%	1.52%
30 Mbit/s	0.75%	0.88%	28.26%	1.01%	30.02%	2.93%	3.07%
45 Mbit/s	1.12%	1.71%	52.44%	1.58%	54.33%	4.46%	4.56%
60 Mbit/s	1.78%	2.15%	60.72%	2.18%	62.97%	5.18%	5.29%
75 Mbit/s	1.57%	2.15%	69.74%	2.19%	73.74%	5.71%	6.02%

Table III
CPU UTILIZATION FOR USING DUMMYNET AND NETEM WITH DIFFERENT PACKET SIZES

E. *CPU Load*

Beside the four network QoS measures, it is also important to examine the CPU load caused by the link emulation. In particular, Planet-Lab nodes are usually highly loaded by their slices already. The link emulation should therefore save CPU resources and – even more important – not exceed the CPU capacity (which would cause undesired frame/packet loss).

Table III shows the CPU load caused by DUMMYNET and NETEM for applying bandwidth limitation on Data Link (L2 – Layer 2) and Network (L3 – Layer 3) Layers for packet sizes of 146 bytes and 1500 bytes (i.e. full MTU). The link emulation is just used to enforce the configured bandwidth by dropping out-of-profile packets. For comparison, also a baseline measurement for 146 bytes packets (i.e. the performance-critical case) without link emulation is shown. Clearly, without link emulation, the CPU utilization remains quite small: about 1.5% for a 75 Mbit/s bandwidth limitation regardless of the packet size. Also, using NETEM on Data Link or Network Layer only slightly increases the CPU utilization to about 2.2% – regardless of the packet sizes. On the other hand, the CPU load for DUMMYNET is significantly influenced by the packet size: for 1500 byte packets, it requires 5.7% (Data Link Layer) and 6.0% (Network Layer) of the CPU, while the load rises to 69.74% (Data Link Layer) and 73.74% (Network Layer) for the small 146 bytes packets. That is, DUMMYNET requires significantly more CPU power for the same task.

VI. CONCLUSIONS

In this paper, we have evaluated our approach of extending Planet-Lab-based network testbeds – with fokus on G-Lab – by emulation on the Data Link Layer of the OSI model. Unlike the already existing link emulation supported on the Network Layer (i.e. for the IP protocol) only, our approach also allows for testing new non-IP Future Internet protocols. Two popular link emulation approaches have been considered: DUMMYNET and NETEM. The DUMMYNET approach – which is currently used by Planet-Lab for Network Layer link emulation – first had to be extended by us to support link emulation on the Data Link Layer of Linux-based Planet-Lab setups.

In our evaluation, we have shown that both Data Link Layer link emulation approaches are usable for Planet-Lab. The resulting performance of NETEM and DUMMYNET for the Data Link Layer emulation is quite similar to the performance of the Network Layer emulation. However, NETEM provides a slightly better accuracy for delay emulation and requires significantly less CPU power in comparison to DUMMYNET. Also, NETEM is able to emulate jitter much more accurately. As part of our future work, we are therefore going to contribute a configurable link emulation solution to Planet-Lab, allowing for switching between the currently used DUMMYNET for backwards compatibility and NETEM – in order to let experiments choose the best-suitable approach for their specific requirements.

REFERENCES

[1] International Telecommunication Union, "Open Systems Interconnection – Base Reference Model," ITU-T, Recommentation X.200, Aug. 1994.

[2] C. Raiciu, M. Handley, and D. Wischik, "Practical Congestion Control for Multipath Transport Protocols," University College London, London/United Kingdom, Tech. Rep., 2009.

[3] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," Boston, Massachusetts/U.S.A., Dec. 2002, pp. 255–270.

[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI Veritas: Realistic and Controlled Network Experimentation," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa/Italy, 2006, pp. 3–14.

[5] T. Friedman, S. Fdida, P. Duval, N. Lafon, and X. Cuvellie, "OneLab: Home," 2009.

[6] R. Steinmetz, J. Eberspächer, M. Zitterbart, P. Müller, H. Schotten, and P. Tran-Gia, "G-Lab Phase 1 - Studien und Experimentalplattform für das Internet der Zukunft," White paper, www.german-lab.de, Jan. 2009, available online (16 pages).

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[8] M. Carbone and L. Rizzo, "Dummynet Revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, 2010.

[9] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[10] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale Virtualization in the Emulab Network Testbed," in *Proceedings of the USENIX Annual Technical Conference on Annual Technical Conference*, Boston, Massachusetts/U.S.A., 2008, pp. 113–128.

[11] S. Hemminger, "Network Emulation with Netem," in *Proceedings of the Linux Conference Australia (LCA)*, April 2005.

[12] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy, "Linux Advanced Routing and Traffic Control HOWTO," 2010.

[13] G. Kurtsou, "Layer 2 FreeBSD Dummynet Patch," 2009.

[14] B. Fink, "Manpage of NUTTCP," 2007.

[15] M. Carbone and L. Rizzo, "Adding Emulation to Planetlab Nodes," in *Proceedings of the ACM CoNEXT Workshop*, Rome/Italy, December 2009.

[16] J. J. Dongarra and T. Dunigan, "Message-Passing Performance of Various Computers," Knoxville, Tennessee/U.S.A., Tech. Rep., 1995.

[17] L. Nussbaum and O. Richard, "A Comparative Study of Network Link Emulators," in *Proceedings of the Spring Simulation Multiconference (SpringSim)*, San Diego, California/U.S.A., 2009, pp. 1–8.