ns-2 Distributed Clients Emulation: Accuracy and Scalability

Stein Kristiansen
University of Oslo
Gaustadallèen 23
N-0371 Oslo, Norway
steikr@ifi.uio.no

Thomas Plagemann
University of Oslo
Gaustadallèen 23
N-0371 Oslo, Norway
plageman@ifi.uio.no

ABSTRACT

ns-2 is a well known network simulator, recently extended with improvements to its emulation facility. Real-time constraints and the boundary between real-world and simulated entities impose scalability and accuracy limitations, and distort the simulated network as perceived by the involved real-world applications. This paper presents results from a performance evaluation of the ns-2 emulation facility. Conducting emulation experiments of differing magnitudes, and under varying emulation environment set-ups, we unveil central types of scalability limitations and obtainable accuracy. We find throughput limits using high and low end computers, and a significant throughput decrease when increasing the number of involved real-world applications. We furthermore show how end-to-end delay increases both with traffic load and an increasing number of involved real-world applications. Moreover, during these conditions, we find that the system treats these applications increasingly unfair by distributing total throughput unevenly between them, and by imposing different amounts of end-to-end delay.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]; I.6.3 [**Simulation and Modeling**]: Applications—*ns-2*

Keywords

ns-2, Performance Evaluation, Network Simulation, Network Emulation

1. INTRODUCTION

New computer network solutions are continuously being developed. Testing and evaluating these has traditionally been performed with network simulators. However, the desire to utilize already existing real-world implementations within experiments has in the later years lead to the development of network emulation solutions, where real-world components can interact with synthetic models.

ns-2 is a well known network simulator, written as a combination of C++ and the scripting language OTcl [1]. Extensions written by Mahrenholz and Ivanov [2] [3] have recently improved ns-2's emulation facility, allowing real-world applications (RWApps) to be connected to specific nodes within a simulated ns-2 network (SimNet). This approach has the advantage of possible reuse of existing network models developed and refined for ns-2 over the years. The improved emulation facility is available in two variants: the *single host extension* (SHE) and *distributed clients extension* (DCE). The former allows RWApps to run alongside ns-2 utilizing one physical host only, while the latter distributes these across separate hosts (RWHosts) connected through a custom tailored UDP tunnel.

In lieu of the accuracy and scalability issues introduced by real-time constraints during emulation [4], this paper subjects the DCE to a performance evaluation by obtaining accuracy and resource measurements during execution of emulation experiments of differing scale. We define *accuracy of emulation* to be the degree to which our system is able to present to communication endpoints a SimNet conforming to the specifications defining it. This accuracy is determined by the SimNet model *and* the components and mechanisms enabling the distribution of RWApps across separate RWHosts. The latter include the UDP tunnel, the physical network and kernel scheduling of RWApps, and can be regarded as a *distribution layer* on top of the SimNet. In this paper, we investigate *how the distribution layer impacts scalability and accuracy of emulation for the DCE extension.* Our findings hence apply to experiments involving *any* SimNet as they only provide an indication on how this layer affects traffic on the path between RWApps and the SimNet. In our work we therefore employ a simple and deterministic SimNet to easier isolate the effects imposed by the distribution layer.

Realizing that accuracy decreases both with increased traffic workload and the number of communicating RWApps, we observe system behavior while varying experiment scale in both of these dimensions. Since our system is provided with only a limited amount of resources, system saturation is bound to occur at some point, in our case manifested as a limit in *throughput.* The *end-to-end delay* is another important metric which is affected by emulation mechanisms, and since we include several communicating RWApps, we inves-

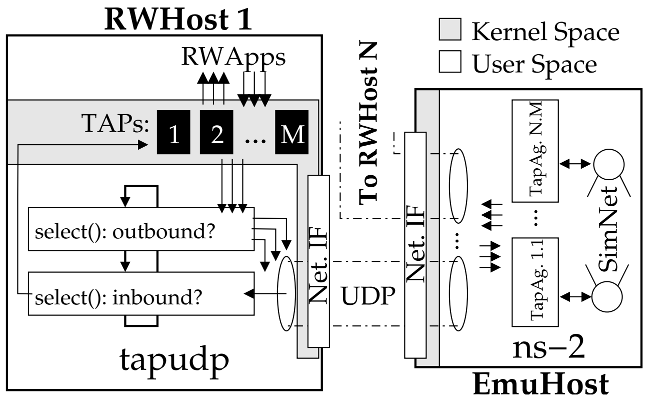

Figure 1: The DCE-Extension.

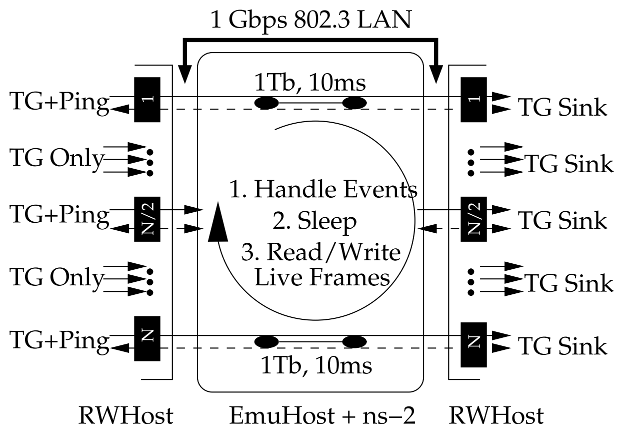

Figure 2: Emulation Environment Set-Up. Black Boxes Represent TAP Devices.

tigate how equally simultaneously flowing traffic streams are treated. The latter is here referred to as *fairness*. Our overall goal is partly to discover which considerations are important and general enough to be usefully applied to other distributed emulation solutions sufficiently similar to the DCE, and partly to aid researchers in considering the feasibility of the DCE for their own use. From the insight gained during the analysis of our results, we also propose a set modifications to the system believed to amount to improvements in performance.

Section 2 introduces the DCE-extension. Thereafter, in Section 3, we motivate our experiments and present their set-up, while Section 4 presents the results and improvement proposals based on these. Finally, Section 5 concludes this paper and elaborates on possible future work.

2. DISTRIBUTED CLIENTS EMULATION

This section provides the reader with an overview of the buildup and mechanisms of the DCE-extension. For further details, we refer to [2], [3], [5], and the ns-2 manual [6].

The DCE-extension dedicates one computer (EmuHost) for running ns-2 to perform network simulation, and a set of RWHosts for execution of the involved RWApps. The DCE-extension can be regarded as being composed of two separate parts: first, an extension of the ns-2 codebase, and second, mechanisms implemented at the RWHosts for (de)multiplexing and tunneling of traffic between the EmuHost and the involved RWApps.

Each RWHost is connected to the EmuHost through their own UDP tunnel, the endpoints of which are realized by the dedicated user space application *tapudp* at the RWHosts' end, and a *UDP Network Object* at the ns-2 end. tapudp reads Ethernet-frames from a set of *virtual TAP interfaces* [7] in a round-robin manner, and writes them onto the UDP tunnel, thereby working as a multiplexer of outbound traffic. Conversely, all traffic arriving at an RWHost through the UDP tunnel is demultiplexed towards the correct TAP device according to destination address.

At the ns-2-end, network objects capture packets from their UDP-socket, inspecting the source address to obtain an index into an array of *tapagents*, and pass the packet towards that agent for further processing. Pointers to all agents are stored in an array of 64K entries, at the index corresponding to the combination of the associated RWHost number (one byte) and TAP interface number (one byte). When a packet arrives at the EmuHost-end of the UDP tunnel, the virtual source-tap-agent to receive the packet is found by a direct lookup in this array. This results in the frame emerging at its virtual source location within the SimNet, after being encapsulated into an ns-2 packet marked with a virtual destination agent's address. This address is composed of the third (destination TAP device) and fourth (destination RWHost) octets in the destination IP-address. Upon arrival at the virtual destination node, tap-agent and network object, the packet is written onto the UDP tunnel leading towards the destination RWHost, eventually forwarded through the receiving tapudp-application towards the correct destination TAP device. Figure 1 gives an overview of the DCE-extension.

3. EXPERIMENTS

Figure 2 gives an overview of the emulation environment used in our experiments. The main components are described below.

Hardware: We use 6 computers, three of which are equipped with 64-bit AMD dual-core 2.2Ghz processors and 2 GB RAM (called *high end*), and three of which have 32-bit Intel Pentium 4 1.6Ghz processors and 512 MB RAM (i.e. *low end*). During experiment runs our testbed always consist of 3 of these computers: two RWHosts running RWApps, and one EmuHost running ns-2. The RWHosts are furthermore assigned the roles *source* and *destination* depending on the direction of traffic flow. We interconnect the computers using a gigabit 802.3 Ethernet LAN, providing enough bandwidth to exclude the network as a prospective bottleneck.

RWApps: Instances of Traffic Generator (*TG*) v. 2.0 [8] make up our RWApps. These are configured to generate uniform UDP traffic, i.e. with packets of a constant size and a constant inter-departure time. We have N pairs of TG instances, communicating across separate paths (EmuPaths) across the SimNet, each identified by a unique path number. The TAP devices at both ends of the EmuPaths are identified by the combination of an *EmuPath ID* and the RWHost ID (both integers in the range $[1, 255]$). For round trip time (RTT) measurements, we use ping to transmit ICMP probes across Path 1, Path $N/2$ and Path N. By distributing three streams of probes uniformly across the paths we aim to obtain a good indication of any correlation between path number and RTT.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5595
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5595

Table 1: Experiment Details

Exp. 1	Packet Size (bytes)	Traffic Load (pps)
Low-End	58,500,1000,1472	300→20100
High-End	58,500,1000,1472	1000→85000
Exp. 2	**Nr. of EmuPaths**	**Traffic Load (pps)**
Low-End	10→250	2500→15000
High-End	10→250	5000→30000

SimNet: Our performance evaluation aims to be as SimNet independent as possible. We want to focus on the overhead of passing traffic between RWApps and the SimNet, rather than how accurately the SimNet conforms to its specifications. To make it easier to isolate this overhead, we utilize a SimNet as simple and deterministic as possible. Our SimNet is entirely built out of N point-to-point, 1Tb, 10 ms links (SimLinks), at the ends of which our RWApps are connected through ns-2 tapagents. We also avoid simulated traffic shaping and throughput limits for easier separation between effects imposed by the SimNet and those caused by external mechanisms. Using this SimNet set-up, we can regard any throughput limit, RTT above 20 ms, jitter and/or burstyness observed in the end-to-end path as deviations from the SimNet configuration, i.e. inaccuracies of emulation.

We conduct four sets of experiments summarized in Table 1, each tailored to accommodate specific sub-goals derived from the overall goals presented in the introduction. Experiments 1 and 2 measure RTT and throughput in the presence of *increasing traffic workload*, and *increased amount of RWApps*, respectively. Since pairs of RWApps communicate over EmuPaths, we hereafter refer to the latter factor as *the number of EmuPaths*. Furthermore, in Experiment 2 we compare measurements of different EmuPaths to unveil how similarly EmuPaths are treated. For the first experiment we only utilize one EmuPath, and in the second we keep the packet size constant at 500 bytes.

During preliminary experiments, we found that the throughput bottleneck wrt. *total amount of traffic flowing through the system* was ns-2. To add validity to our results beyond only one specification, we therefore run our first experiment with two different EmuHosts with differing hardware specifications. Similarly, since we found that the destination RWHost's tapudp became the main bottleneck when increasing the number of EmuPaths, we run our second experiment with high and low end destination RWHosts. Traffic generation rates and the number of EmuPaths were chosen based on results retrieved during preliminary experiments, to unveil system behaviour up to and beyond system saturation.

In each experiment, we vary *packet size*, *packet generation rate* (also called traffic load) and *the number of EmuPaths*. For each combination, we collect measurements from three identical 36 second emulation runs, where the initial and final three seconds are disregarded to avoid including prospective transient measurement variations. We present the average of these three runs, as we find a satisfyingly low standard deviation (SD) between runs[1]. The central measurements on which we base our analysis are the following:

- *TG Source and Receiver Logs:* Denotes departure/arrival times, sequence numbers and IP-addresses (thereby implicitly TAP device number) of UDP packets. Obtained only at source and destination RWHosts.

- *Ping report:* Denotes individual, average and SD of RTT across a specific EmuPath. Obtained only at source RWHost.

- *Sar Resource Measurements:* Reports CPU, memory and network device (both physical and virtual/TAP) utilization statistics during the runs. Obtained at all hosts.

- *ns-2 Event-Loop Instrumentation:* Reports utilization of the three main sections of the ns-2 real-time event loop, indicated in Figure 2. Obtained at the EmuHost only. These include portions of time consumed for

  1. handling of scheduled events (SimEvents)
  2. sleeping while waiting for the arrival of live packets or for the handling time of SimEvents[2]
  3. reading and writing packets to the physical network

[1]There is one exception, which is explicitly stated in Section 4.1.

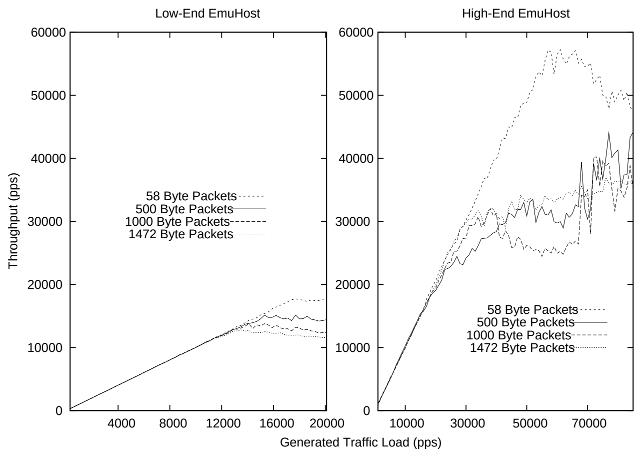

Figure 3: Throughput for Experiment 1.

All measurements are stored to ramdisk during experiment execution to avoid any interfering disk IO overhead. Jitter measurements were excluded in this paper due to spatial constraints, but elaboration on this can be found in [9].

4. RESULTS AND FINDINGS

This section presents the most important results from our performance evaluation. The two subsections present results obtained from Experiments 1 and 2 respectively, along with analysis, comments on the most important findings, and improvement proposals.

4.1 Effect of Increasing Traffic Load

In our first experiment, the main bottleneck on the end-to-end path is ns-2. The plots presented in Figure 3 show throughput in packets per second (pps), and Figure 4 shows the RTT values retrieved from ping-reports during increasing traffic load.

[2]In fact, the sleep returns shortly before the handling of the next scheduled event, as described in [3].

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5595
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5595

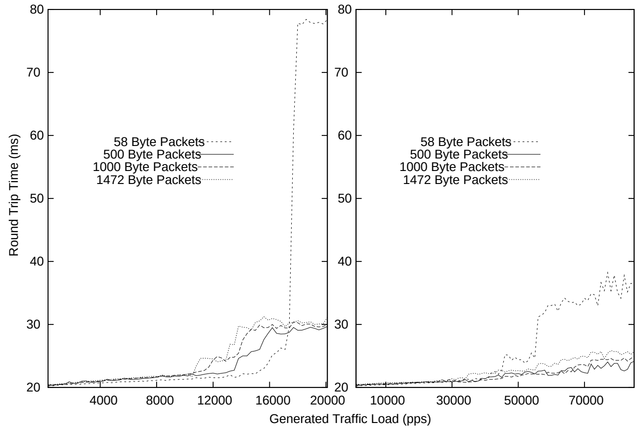

Figure 4: RTT overhead for Experiment 1.

The throughput plots show, as expected, higher pps values when utilizing the high end EmuHost. The maximum obtainable throughput is two to three times larger with the high end EmuHost than with the low end EmuHost, depending on packet size. Maybe not so expected is the difference between the development in throughput with increasing imposed traffic load. For the low end EmuHost, we see a clear point of saturation for all packet sizes. The plots indicate that *all* additional traffic load imposed beyond these points is dropped by the EmuHost. For the high end EmuHost, however, we can identify such a distinct point of saturation only for 58-byte packets. As we increase traffic load beyond certain values, we experience what appears to be a less predictable increase in the amount of packet drops for the three highest packet sizes. We furthermore have higher SD values between identically configured runs when utilizing the high end EmuHost, with SDs of up to almost 2% when generating less than 75000 pps, and at certain higher rates rising to up to just above 7%. In contrast, the corresponding values for the low end EmuHost never surpass 0.02%. Figure 3 also suggests a more consistent correlation between packet size and throughput for the low end EmuHost, as there is no immediately distinguishable difference between 500, 1000 and 1478-byte packets in the high end case, 50-byte packets being an exception.

Since our SimLinks are configured to impose a delay of 10 ms, our ping instances should never report an RTT lower than 20 ms; ideally, we would experience an RTT of just above 20 ms. However, because of the emulation boundary, this is not achievable. The plots in Figure 4 show the amount of RTT overhead imposed by the emulation extension in the presence of increasing traffic load. If we compare the throughput and the RTT plots, we notice a slightly and steadily increasing RTT across increasing traffic loads before reaching the capacity limit. We also observe a significant increase in RTT as we surpass EmuHost capacity, and that this increase is the most drastic for the smallest 58-byte packets, especially when utilizing the low end destination RWHost.

4.1.1 *Analysis*

Firstly, we consider the observed throughput limits at different imposed traffic loads. As expected, the cause of throughput limit can in most cases be attributed the saturation of the EmuHost CPU. Since we have such a simple SimNet, our event-loop utilization measurements show that the most clock-cycles are consumed handling events related to capturing and writing packets from/to the physical network. Compared to the handling of the relatively simple events of simulating the propagation of a captured packet across a SimLink, the events passing packets across the emulation boundary require system calls involving costly operations of copying complete packets between kernel and user space. The ns-2 event-loop instrumentation results presented in Figure 5 clearly show the difference in handling-time between the two types of aforementioned events. For the four packet sizes, it shows the distribution of event loop time on the low and high end machines. The lowest area represents the time used for read operations, and the area above for write. The area entitled "sleep" and the black area together make up unproductive event-loop time, the latter comprising the busy waiting introduced in [3]. The area in-between represents pure simulated events, constituting a relatively small fraction of available time compared to the read and write operations. These plots also show that the main reason for the throughput bottleneck is saturation of the ns-2 event-loop, in turn caused by exhaustion of available CPU clockcycles. They are furthermore coherent with the development seen in the throughput plots; we see a linear increase in event-loop utilization across increasing traffic loads for the low end EmuHost, while the high end instrumentation plots reflect the pre-saturation packet drops indicated by the throughput measurements.

For 500, 1000 and 1472-byte packets in the high end case, the sub-linear increase in event-loop utilization indicate that packets get dropped already before arriving at the UDP-socket for ns-2 to capture. Hence, for these three packet sizes, ns-2 no longer poses as the bottleneck. In addition, our SAR-measurements report that practically 100% of the traffic load in fact arrives at the network interface card (NIC). Hence, these packets must have been dropped after arriving at the NIC, but before arriving at the ns-2's UDP-socket; an effect observed for the low end EmuHost only during CPU-saturation. The reason may be that although both low and high end computers run on identically configured kernels, the different CPU-architectures (32-bit, single-core Intel versus 64-bit, dual-core AMD) may have an effect on scheduling between ns-2 and the kernels internal packet forwarding, affecting how packets are dropped within the kernel. The observed packet drop development, gradually increasing during increased packet arrival rates, may remind of a dropping scheme similar to random early detection. The fact that only the larger packets experience these packet drops, suggest that receive queues in the kernel, whose sizes are specified in number of bytes, may drop packets due to the kernel forwarding packets from the NIC's receive queue in bursts during high arrival rates. Another possibility may be that different policing rules are deployed in kernels due to the two hosts utilizing different NICs. Finally, because of the unreliable nature of UDP, the kernel may indeed drop packets at any suitable location in the stack as it sees fit, depending on the particular implementation. Although further studies are required to confirm or deny these speculations, our results show how different host system hardware and/or OS configurations can impact network emulation performance. Hence, in preliminary work during experiment design, one should not underestimate the importance of becoming properly familiar with key hardware- and kernel-specific mechanisms involved in traffic reception

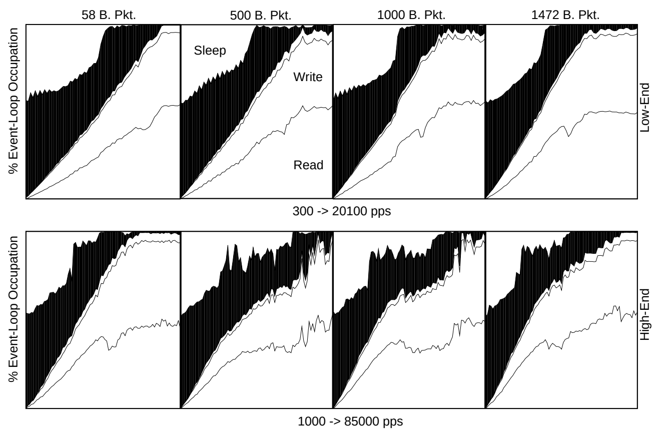

Figure 5: Event-Loop instrumentation measurements.

and transmission. Tools such as iptables [10] and tc [11] are among those that can be utilized to modify and gain valuable information about routing and shaping policies employed by the kernel.

The correlation between increasing packet size and decreasing throughput is partly caused by packet copying time between kernel and user-space. On the end-to-end packet journey, we rely on three such copy operations per packet per RWHosts, namely between RWApps and TAP devices, TAP devices and tapudp, and between tapudp and the UDP tunnel. At the EmuHost, two such copy operations are required between the UDP tunnel and ns-2. Another reason for packet drops lies in the forwarding of outbound traffic by tapudp. As soon as tapudp is scheduled for execution, all packets waiting on TAP devices are written to the tunnel as fast as possible, hence being forwarded in bursts. Furthermore, since receive queue sizes at the EmuHost are specified in bytes, the smaller the incoming packets are, the more of them fit within receive queues. Hence, it is probable that large packets are dropped since the bursts grow too big for the receive queues. This effect is more prevalent in our last experiment, presented and analyzed in Section 4.2.

Looking at the RTT-plots in Figure 4, we notice a steady, nearly linear pre-saturation increase of about 1 ms and 2 ms for high and low end configurations, respectively. Since there is an increasing number of packets to forward across the EmuPath, the average duration ping probes have to wait for expedition at intermediate buffers increases, especially in lieu of the aforementioned bursty forwarding. Since the high end hosts have more processing power, and because their two cores open for the possibility of running packet reception code in parallel with ns-2, receive queues may not grow as much, and can be emptied faster, resulting in lower RTT during non-saturating conditions.

We furthermore see how RTT increases drastically in correspondance to the amount of packet drops performed. One of the main reasons for dropping packets are filling queues. Upon EmuHost saturation, the rate at which packets arrive at the NIC surpasses the rate at which the host can capture them, resulting in a sudden filling of receive queues. Ping probes that are not dropped must in these cases suddenly wait in receive queues for a significantly longer time while all preceding packets are dequeued and forwarded, resulting in the observed jump in RTT measurements. Also, the smaller the packet, the more fit within the queues, and since we experience a maximum queue utilization during saturation, the corresponding waiting times are much higher for the smallest packets.

4.1.2 Summary and Possible Improvements

This experiment unveils throughput limits and RTT overhead imposed by DCE while utilizing EmuHosts with two hardware specifications. We also found that the CPU-power at the EmuHost is the main limiting factor for throughput, that receive queues at the EmuHost cause increased extra RTT with higher traffic rates, and that their sizes probably have a significant effect on packet drops internally in the kernel for very high packet arrival rates. Taking into consideration these results when designing experiments across hardware with certain specifications, one can make more precise assumptions about achievable accuracy of emulation, and more easily avoid the unwanted effects of system saturation. However, our results show how the type of hardware utilized and its configuration have a major impact on performance, indicating the importance of proper familiarization and configuration of ns-2 external mechanisms such at receive queues and kernel scheduling.

Based on our findings, this section proposes a few changes to the system believed to improve performance in different ways. Our focus lies on the EmuHost, since it was found to be the throughput bottleneck and main source of RTT-overhead.

With regard to throughput, it is clear that increasing computational resources on the EmuHost results in higher throughput and lower RTT. Aside from increasing CPU power, distributed or parallel execution of ns-2 may be an alternative [12] [13]. On the other hand, one could attempt to lower the computational cost associated with passing packets across the emulation boundary. Since network models often inspect only a small fraction of packets (i.e. headers) during emulations, lowering packet read and write overhead could be accomplished by passing only the portions of interest through the SimNet, while routing the complete packets directly to the destination RWApp. Another approach could be to utilize memory-mapped ring buffers for exchange of packets between user and kernel space, eliminating both packet copy and system call overhead. Finally, increased receive queue sizes residing within the EmuHost kernel should have a positive impact on performance.

4.2 Effect of Increasing the Number of EmuPaths

In this section we focus on achievable throughput, accuracy and flow fairness across increasing numbers of involved EmuPaths. We impose traffic at 6 different rates for both experiment sets, all rates at any time configured to be equally distributed across all available EmuPaths. Total throughput is defined as the sum of the throughput of all EmuPaths, and is in Figure 6 denoted in relation to an increasing number of included EmuPaths. The grayscale plots indicate how total throughput is distributed across the available EmuPaths, with increasing the number of EmuPaths along their x-axes and EmuPath ID along the y-axes. From black to white, our gray scales represent 100% to 0% throughput, relative to the traffic load injected onto the particular EmuPath. Practically all packet drops are performed between the NIC and

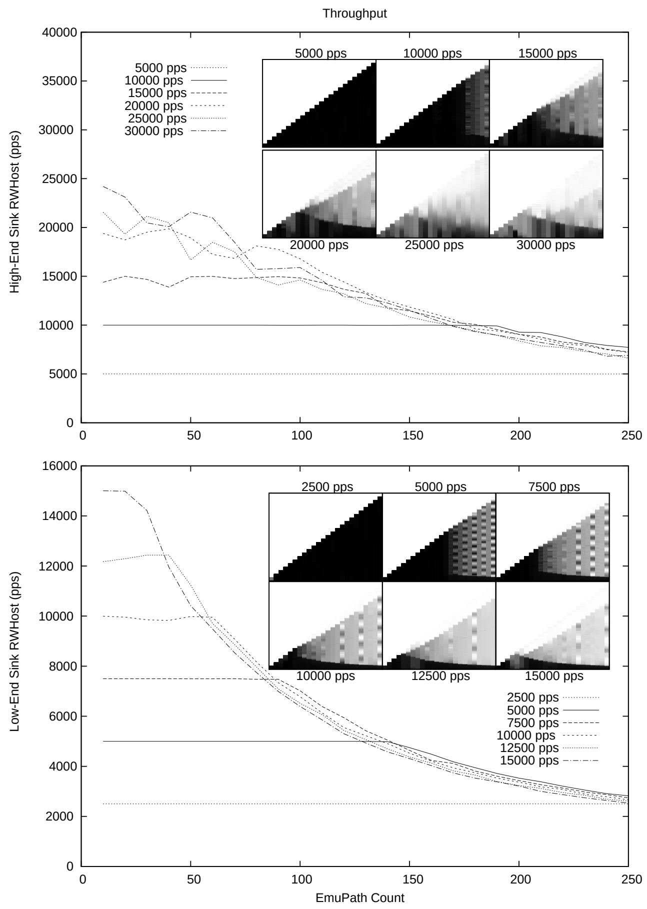

Figure 6: Throughput for Experiment 2.

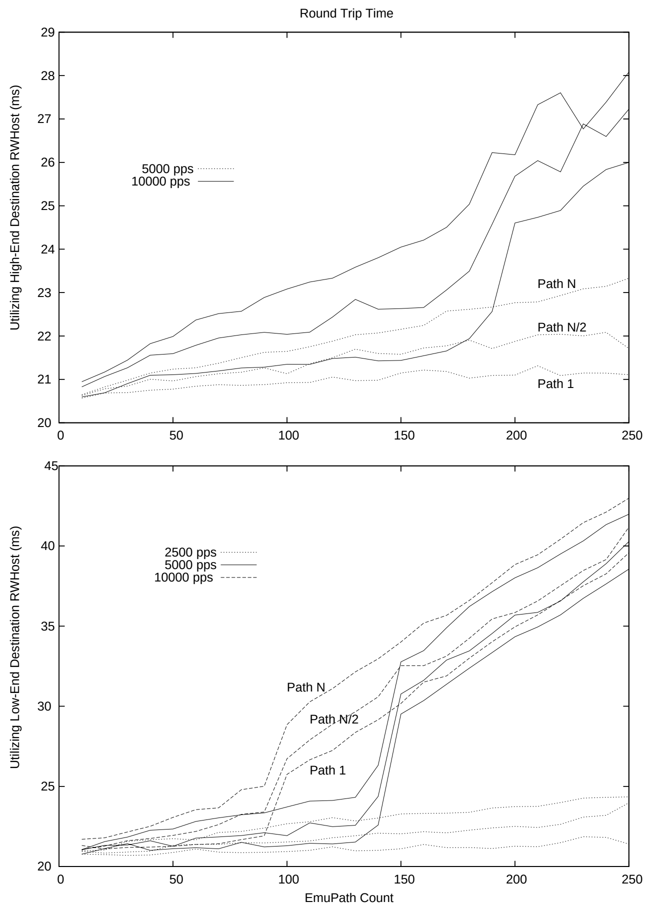

Figure 7: RTT for Experiment 2.

TAPs at the destination RWHost. For an accurate representation of throughput, we therefore decide to present the average number of packets per second arriving at the destination TAP devices.

The graphs outline an overall exponential decline in total throughput with increasing the number of EmuPaths. In our experiments, maximum throughput drops from 24200 pps to 7700 pps (68.18% decrease) and 15000 to 2800 pps (81.33% decrease)[3] for the low and high end set-ups, respectively. This observation is particularly interesting as increasing the number of EmuPaths and RWApps in real-world experiments often imply *increased* amounts of injected traffic. This effect should therefore be kept in mind when considering the feasibility of the DCE for large-scale experiments.

Secondly, our throughput distribution plots show an uneven distribution of packet drops across EmuPaths; fewer packets are dropped for the lower numbered EmuPaths. Hence, if one does not carefully restrict imposed traffic workload during emulation experiments including multiple EmuPaths, traffic flowing across these will experience unfair packet dropping. However, as mentioned above, care should be taken to restrict the experiment magnitude to avoid any packet drops in the distribution layer. These throughput distribution plots during system overload are nevertheless beneficial for the purpose of analysis to gain understanding of system behaviour and underlying mechanisms in the work towards prospective improvements (see Section 4.2.2).

Figure 7 illustrates the amount of RTT over 20ms observed across our three selected EmuPaths while increasing the total number of EmuPaths across the environment. The main purpose of these is to show the trends in RTT increase. To avoid overfilling the plots and because of very high ping-probe losses at high rates, we include only measurements for the lower rates. Notice that the y-axes have different dimensions. At a first glance we immediately notice how RTT increases with increased number of EmuPaths. We furthermore experience larger delays at higher numbered EmuPaths, and an increasing difference between them as we add more of them to the environment. During pre-saturation conditions, we see that EmuPath 1 shows a relatively modest increase of less than 1ms for all reported rates, while the middle and highest numbered ones increase with up to 2-3 ms. We also see that higher numbered EmuPaths are more affected by increasing traffic rates, adding to the difference in RTT between the paths. These observations suggest that even if we keep imposed traffic load below the throughput limit, increasing the number of EmuPaths involved in the experiment cause increased RTT emulation overhead and larger differences between them. Finally, we see a sudden increase at certain points, similar to that found in RTT measurements obtained in our first experiment (see Figure 4).

[3]Note that maximum obtainable throughput with 10 EmuPaths is in fact somewhat larger, but due to shortage of time we could could not pinpoint this exact value.

4.2.1 Analysis

In this section, we first look at the reasons behind the throughput development found in Figure 6. In our analysis, we focus on the throughput distribution plots, in which we have identified three particular characteristics:

1. EmuPaths are unfairly differentiated into three distinct groups wrt. packet drop rate:
    - *Group 1:* A small set of the lowest numbered ones experiencing relatively low packet drop rate.
    - *Group 2:* An equally small set of the highest numbered ones experiencing a close to 100% packet drop rate.
    - *Group 3:* A larger set of the remaining EmuPaths experiencing increasing drop rates with increasing the number of EmuPaths.
2. At certain numbers of EmuPaths (for instance, 220 EmuPaths, 7500 pps in the low end case), we see how EmuPaths in the above mentioned Group 3 are grouped in clusters wrt. the packet drop rates they experience. This is more clearly visible for the measurements from the low end plots, where we also notice how this effect, visible as "stripes of waves" in the grayscale plots in Figure 6, occur more often for lower rates.
3. We experience higher packet drop rates with increasing packet generation rates for all three groups.

Packets are mainly dropped at two locations: at EmuHost receive queues and at destination RWHost receive queues before arriving at the tapudp instance. At the EmuHost, we find that packets are dropped independently of the number of EmuPaths, while increasing total traffic load results in drops according to what is seen in Figure 3. This is the cause of Characteristic 3 above. Upon packet arrival, ns-2 locates the virtual source to receive the packet through a simple lookup in the tapudp-array (see Section 2), the computational workload of which is independent of the number of occupied slots in the array. Further propagation of the packet between the virtual source and destination, and back onto a UDP tunnel, is with our simple SimNet also independent of the number of EmuPaths. This explains why increasing the number of EmuPaths does not affect ns-2 throughput. How more complex SimNets are affected by increasing emulation experiment magnitudes, are out of the scope of this paper (see [2] and [3] for some investigations).

The packet drops causing throughput distribution Characteristics 2 and 3 are performed at the receive queues of the receiving RWHost, and mainly increase with the number of EmuPaths. This behaviour is rooted in the interaction between RWApp multitasking and tapudp's manner in which to forward traffic between TAP devices and the UDP tunnel. Between every time tapudp is scheduled to execute, a set of RWApps have time to transmit packets to TAP devices. Depending on transmission rate, scheduling order and the number of executing RWApps, certain TAP devices will have packets waiting to be forwarded when tapudp is scheduled to execute. Upon selection of outbound packets, tapudp runs through all available TAP interfaces, in the order of their associated number, forwarding one packet, if available, per interface onto the UDP tunnel. We refer to a complete cycle through the TAPs as a *round.* A key observation is that the order in which packets are transmitted from

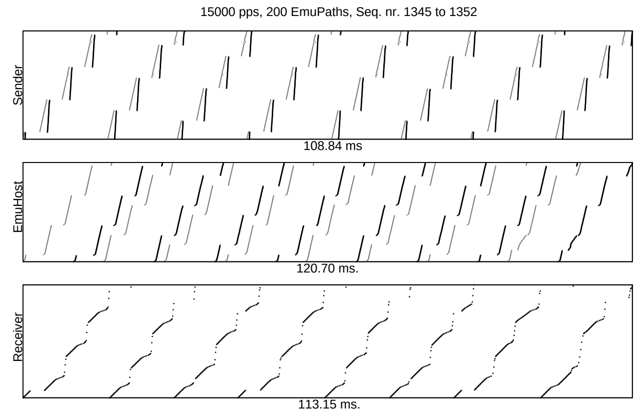

Figure 8: Snapshot of a run in Experiment 2.

RWApps are not preserved upon tunneling; lower-numbered TAPs are always serviced before higher-numbered ones, regardless of the order in which they arrive at the TAPs. Furthermore, the resulting train of outbound packets are written onto the UDP tunnel as fast as possible, restricted only by the bandwidth of the network link. In effect, outbound traffic is forwarded in bursts in which packets are ordered by increasing EmuPath id, and the burst sizes depend on the number of involved EmuPaths and individual RWApps' packet transmission rates. At the receiving RWHost, *only one packet is captured per round regardless of the number of EmuPaths*, while as increased number of EmuPaths implies more work per round for polling TAP devices, even in the absence of outbound traffic. Therefore, tail dropping occurs at the destination RWHost during increasing number of EmuPaths due to more slowly emptying receive queues. The added overhead of scheduling an increased number of RWApps also matters, although our measurements suggest this to be of less impact than the added tapudp polling.

For clarification, we employ the instrumented version of tapudp in a 30 second experiment run, from which Figure 8 presents a snapshot of 8 sequence numbers, drawn from the middle of the run, i.e. after 18 seconds. We generate 15000 pps distributed over 200 EmuPaths, and enable our simplified ns-2 tracing (the overhead of this tracing is neglectable for the following analysis). The snapshots in Figure 8 represent EmuPath IDs along the y-axes and time along the x-axes. TG and tapudp timestamps are denoted in gray and black pixels, respectively. The middle plot denotes timestamps for packets arriving at and departing from the SimNet as gray and black pixels, respectively. At the source RWHost, notice the steep slopes present in the tapudp measurements, and how the TG instances and tapudp take turns at execution. This slope indicates the rapid forwarding of subsequent packets, resulting in the aforementioned bursts. We also see how the SimNet, not configured to perform any traffic shaping, preserves the burstyness of passing traffic, even though an added computational effort of sustaining the SimNet results in a modest dispersion of packets within the bursts. Upon arrival at the destination RWHost, we see how tail dropping causes a significant loss of packets at the end of the bursts. As mentioned, these are always the ones belonging to the highest numbered EmuPaths, explaining the appearance of Group 2 from Characteristic 1. Low numbered EmuPaths (Group 1) are on the other hand always

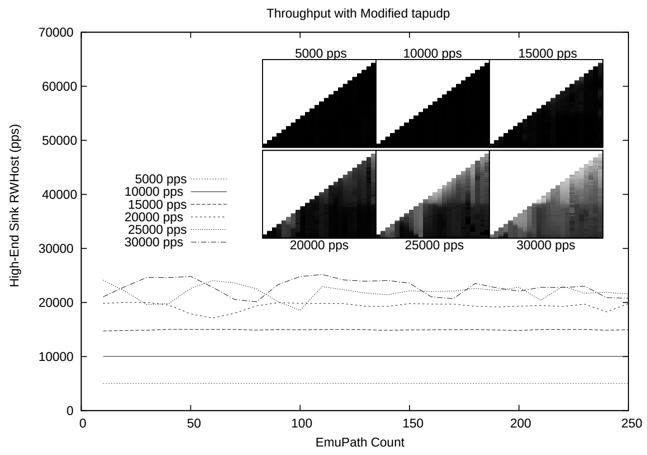

Figure 9: Throughput for Experiment 2 with modified tapudp

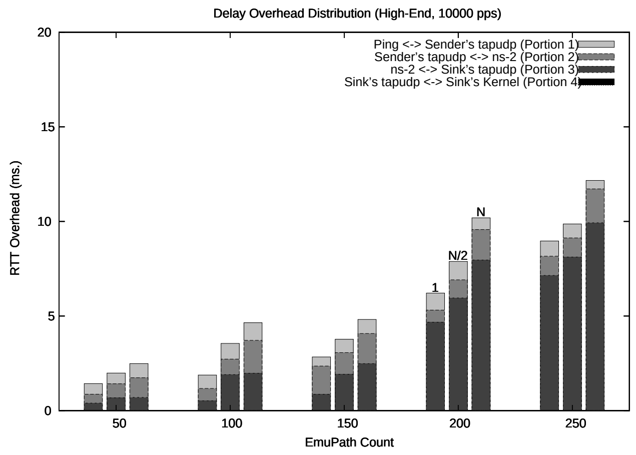

Figure 10: Distribution of RTT overhead.

favoured, as early packets never get dropped from bursts arriving at empty queues. Notice also how TG and tapudp timestamps overlap in the plot for the destination RWHost. This indicates that the kernel schedules the destination TG instance to capture its packet immediately after tapudp has forwarded it from the tunnel. We find this to be the case during high packet generation rates and/or high numbers of EmuPaths.

In testing this theory, we conducted Experiment 2 again with tapudp modified to address the issue of differing amounts of workload associated with in- and outbound traffic. Instead of polling for only one inbound packet per round, our modified tapudp polls for one inbound packet for each poll for an outbound packet from a TAP device. That is, in the presence of N TAP devices, the number of polls for both inbound and outbound packets per round becomes equal to N. The results presented in Figure 9 indicates a significant increase in throughput of up to 172% for 250 EmuPaths compared to Figure 7, and that the modified tapudp treats the EmuPaths much more fair. Note that the dimension of the y-axis is different from that in Figure 6. Now, ns-2 reclaims the role as the main bottleneck, which is not affected by the number of EmuPaths. This modification is meant for proof of concept purposes only, as we have not yet performed any proper evaluation to unveil any other prospective consequences of this modification.

Next, we look at the RTT emulation overhead found in Figure 7. In explaining the different characteristics of RTT overhead, we employ our instrumented tapudp and ns-2 tracing experiment runs across 50 to 250 EmuPaths, generating 500 byte frames at a total rate of 10000 pps. These were repeated 5 times to obtain a satisfyingly low SD in our results. We use this combination of values because Figure 7 shows how the corresponding obtained measurements clearly contain the most important characteristics new to this experiment. We use the high end setup (but the underlying mechanisms responsible for RTT overhead is the same in the low end case). We collect RTTs at the ping instances, the source tapudp instance, the ns-2 exit-point and at the destination's tapudp instance. By combining these, we are able to differentiate between contributions to the RTT overhead belonging to four portions of the end-to-end path. These portions are numbered 1 through 4 in Figure 10 where we present our five-run averages. In comparing these results with those for the high end setup in Figure 7, we see overall larger values due to the extra measurement overhead. However, in this analysis the most important information lies in the difference in values between portions of the end-to-end path.

First, we observe how the major difference in RTT measurements between EmuPaths appear in Portion 3 of the end-to-end path, especially for higher numbers of EmuPaths. This is in agreement with our throughput distribution discussion above. An increased number of EmuPaths implies larger bursts in which packets are ordered increasingly wrt. their EmuPath IDs. Therefore, packets belonging to higher numbered EmuPaths are preceded by a higher number of packets in the destination RWHost's receive queue, and must hence wait longer to be serviced by tapudp. Once captured by tapudp, a packet covers Portion 4 of the path relatively quickly, causing the corresponding bars in Figure 10 to be invisibly small. This graph also suggests a similar, but less prevalent, relationship between EmuPaths for Portion 2, caused by the same queuing effect occurring within EmuHost receive queues. Here, the difference is smaller because less work has to be done per packet. Although future experiments have to confirm this, we believe a more complicated SimNet may increase this difference also for Portion 2.

4.2.2 *Summary and Improvement Proposals*

Experiment 2 shows how the mechanism for reception of traffic from the UDP tunnel is responsible for the performance degradation with increased number of EmuPaths, shifting the bottleneck from ns-2 to tapudp when including many EmuPaths. Increasing the number of 1-to-1 mappings between TAP devices and tapagents decreases maximum obtainable throughput and increases end-to-end delay, an effect one should be particularly aware of when designing large-scale experiments. Furthermore, we see how tapudp treats EmuPaths increasingly unfair when increasing the number of them. Hence, when analysing results from large-scale experiments one should consider how the system differentiates the involved EmuPaths.

Taking our findings into consideration, we believe performance improvements can be gained both by configurational means and by utilizing alternative approaches for traffic forwarding between RWApps and ns-2. One optimization regards the manner in which to distribute RWApps across RWHosts. We see how increasing the number of EmuPaths causes an increased difference between RWHosts in the amount of instructions required by tapudp per communicated packet. Hence, given that the approximate amount of traffic to be generated by each RWApp included in an experiment is known, a manner in which to increase total obtainable throughput would be to distribute RWApps so as to balance the amount of inbound and outbound traffic at each RWHost. This way, the polling of TAP devices for outbound traffic becomes more efficient, since this traffic will more often be available. Note that in this respect, our experiments represent the opposite extremity involving only simplex traffic travelling from one RWHost to another.

To address the problem of tail dropping upon reception of large packet bursts, one could increase the size of receive queues to at least $max(Nr_{TAP}) \times Nr_{RWHost} \times MTU$, $max(Nr_{TAP})$ being the highest number of TAP devices present at any RWHost, MTU being the maximum packet size involved in the experiment and Nr_{RWHost} the number of RWHosts involved. This way, receive queues will always be large enough to contain bursts created by tapudp, and moreover any train of bursts resulting from simultaneous transmission among RWHosts. For large-scale experiments, however, this approach could become unfeasible in requiring very large receive queues.

Other possible optimizations involve employing alternative mechanisms for traffic forwarding. Upon being scheduled for execution, tapudp forwards the burst of available outbound packets in the order of the TAP device at which they appear, regardless of the order in which they left RWApps. We propose adding a single FIFO queue between the TAP devices and tapudp, such that references to all packets arriving at TAP devices are placed into this queue immediately after arriving at the device. tapudp may then retrieve outbound packets from such a queue in the order in which they left their RWApps, rather than according to TAP device numbers, effectively resolving the unfairness observed in our results. Also, if one enables tapudp to poll the FIFO queue once per round regardless of the number of TAP devices present, the difference in instructions required at a source and a destination RWHost disappears, allowing RWApps to be distributed more freely across RWHosts wrt. the amount of traffic they generate and receive. For maximum efficiency, this FIFO queue should be implemented as part of the kernel.

In our final proposal we suggest exchanging the user space tunneler with the kernel space ethernet switch emulator Net:Bridge [14]. By attaching all TAP devices along with the physical network interface to such a switch, and by denoting the EmuHost as the gateway for all outbound traffic, we can achieve forwarding of packets between RWApps and ns-2 without a UDP tunnel. A similar approach was outlined in [2] for the SHE, but does not seem to be properly evaluated. This solution avoids the extra packet copying between kernel and user space, along with having the same properties as the above mentioned FIFO-queue. Furthermore, because of the absence of extra UDP-headers, the network interconnecting RWHosts and the EmuHost could more realistically be considered as part of the emulation experiment (further elaboration in [9]). This approach however requires the EmuHost and RWHosts to reside in the same private physical network, as the source and destination TAP devices' addresses still adhere to the imposed addressing scheme involving only private 10.*.*.* addresses.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we expose the DCE extension to a performance evaluation. The extension enables the 1-to-1 mapping between real world applications (RWApps) and dedicated locations within a simulated ns-2 network (SimNet), where RWApps and ns-2 can execute on different physical hosts connected through a UDP tunnel. We obtain measurements from experiments where we vary the number of communicating RWApp pairs (EmuPaths) and the amount of traffic generated by these. By including only a minimal SimNet and uniform traffic streams, we unveil how ns-2 external emulation mechanisms (the *distribution layer*) impose throughput limits, unfairness and end-to-end delay overhead.

Aside from pinpointing the throughput limit with two different hardware specifications, we locate and analyze the two main traffic bottlenecks. First, ns-2 limits the total obtainable throughput due to CPU saturation, and second, the design of the custom tailored UDP tunnel limits the number of involved RWApps. We unveil how increasing the the number of RWApps causes the instructions required per received packet to increase at the destination RWHost's end of the UDP tunnel, in effect significantly decreasing maximum obtainable throughput. Packet drops were furthermore found to be unevenly distributed between the EmuPaths, due to the UDP tunnel implicitly prioritizing lower numbered EmuPaths over higher numbered ones. In addition, we show how increased traffic load and the number of RWApps cause higher end-to-end delays, and how these values also differ between EmuPaths.

To the best of our knowledge, the only earlier studies reported in literature on the performance of ns-2 extended for emulation was those performed by the authors of the extensions themselves [2] [3]. Their main focus lies on the accuracy of wireless ns-2 SimNets under real-time constraints, and they compare measurements from emulations and equivalent experiments with real networks. Following a single host approach (SHE), they utilize only one physical Linux computer for their experiments, emulating RWHosts as instances of User Mode Linux (*UML*) running alongside ns-2. Our performance evaluation differs from the above mentioned in two aspects. First, our work evaluates the DCE rather than the SHE, and second, our focus lies on the *distribution layer* rather than the SimNet model.

Besides ns-2 there are numerous other distributed network emulators. Marenholz and Ivanov discuss in [3] and [2] how some of these emulators relate to their ns-2 extension. Furthermore, we recommend the surveys from Göktürk [15] and Kroppf [16] for a classification of the different emulators. For each of these emulators, their authors have performed some kind of evaluation. The common concerns in these evaluations are whether the emulating machinery can keep up with the real-world (mostly limited by CPU), and how precisely network emulation is performed. In comparing our work with these, we realize that our particular focus on ns-2 specific mechanisms make direct comparisons unfeasible. Differing hardware and operating systems (OS) utilized further complicate such comparisons.

NetBed [17], a descendant of EmuLab, should nevertheless be explicitly mentioned as it utilizes ns-2 emulation to integrate simulated networks with real-world elements. Their performance evaluation includes measurements of the accuracy and scalability of ns-2 emulation utilizing an 850Mhz PC for emulation interposed between two PCs, each transmitting a UDP traffic flow to the other (i.e. a duplex flow). They obtain an aggregate throughput of 8000pps with 64 and 1512-byte packets, which, taken the difference in experiment set-up, OS and hardware into account, is in agreement with our results. It should be noted that they also utilize an older and simpler version of ns-2 emulation [18], to which they made several fixes and improvements for the particular integration into NetBed. They do however lack a proper evaluation of fairness between concurrent traffic flows, and utilize only one hardware configuration in their experiments.

Based on our measurement analysis, we propose a few modifications to the system believed to result in different kinds of performance gains. We demonstrate with a simple proof of concept modification how to improve throughput and fairness. However, proper implementation and evaluation of our suggestions remain as future work. In this respect, the evaluation of the emulator Trellis [19] may be of particular interest, as they include investigations of different approaches for forwarding packets between distributed applications. They demonstrate how avoiding user space forwarding of packets can result in significant throughput gains, and how tunneling and software bridging of physical and virtual network interfaces impact performance. On the other hand, an approach similar to our proposed *packet distilling* is taken by the ModelNet emulator [20] to maximize throughput. Here, the emulating machinery consists of a set of *Core Nodes* over which the simulated network is distributed, and between which only packet headers are passed during emulation.

For other future work, it would be interesting to see how the system performs with larger and more complex SimNets and/or with differing characteristics of traffic load (such as bursty or duplex traffic). Finally, performing a similar evaluation of the SHE extension would be a natural second step.

6. ACKNOWLEDGEMENTS

This work was funded by the VERDIKT program of the Norwegian Research Council (DT-STREAM, project number 183312/S10). We want to thank Erek Gökturk for his guidance, support and many good ideas during the course of the masters project on which this paper builds. We also thank Ovidiu Valentin Drugan for his valuable suggestions and feedback while writing this paper.

7. REFERENCES

[1] D. Wetherall. Otcl, online software: http://otcl-tclcl.sourceforge.net/otcl.

[2] D. Mahrenholz and S. Ivanov. Adjusting the ns-2 emulation mode to a live netowork. In *Proceedings of KiVS*. Springer, 2005.

[3] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns-2. *Distributed Simulation and Real-Time Applications, 2004. Eighth IEEE International Symposium on*, pages 29–36, Oct. 2004.

[4] E. Göktürk. Emulating ad hoc networks: Differences from simulations and emulation specific problems. New Trends in Computer Networks conference, July 2005.

[5] D. Mahrenholz and S. Ivanov. *HOWTO: Wireless Network Emulation Using ns-2 and Distributed Applications*. University of Magdeburg, Germany, December 2004.

[6] USC/ISI A Collaboration Between Researchers at US Berkley, LBL and Xerox PARC. *ns-2 Manual*. The VINT Project, October 2006.

[7] M. Krasnyansky and M. Yevmenkin. Universal tun/tap driver, online software: http://vtun.sourceforge.net/tu.

[8] Traffic generator, online software: http://www.postel.org/tg.

[9] S. Kristiansen. ns-2 emulation: Performance evaluation and improvement proposals. Master's thesis, University of Oslo, Dep. of Informatics, 2008.

[10] P. Russel and P. McHardy. netfilter/iptables, online software: http://www.netfilter.org.

[11] W. Almesberger and E. Ica. Linux network traffic control - implementation overview. In *Proceedings of 5th Annual Linux Expo*, 1999.

[12] G. F. Riley, R. M. Fujimoto, and M. H. Ammar. Parallel/distributed ns, online software: http://www.cc.gatech.edu/computing, 2000.

[13] G.F. Riley, R.M. Fujimoto, and M.H. Ammar. A generic framework for parallelization of network simulations. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999. Proceedings. 7th International Symposium on*, pages 128–135, 1999.

[14] Net:bridge, online software: http://www.linux-foundation.org/en/net:bridge.

[15] E. Göktürk. A stance on emulation and testbeds, and a survey of network emulators and testbeds. In *Proceedings of ECMS*, 2007.

[16] M. Kropff et al. A survey on real world and emulation testbeds for mobile ad hoc networks. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, pages 6 pp.–453, 2006.

[17] B. White et al. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[18] K. Fall. Network emulation in the vint/ns simulator. Univ. of California, Berkley, 1999.

[19] S. Bhatia et al. Hosting virtual networks on commodity hardware. Technical report, Georgia Tech Computer Science, January 2008.

[20] K. Yocum et al. Scalability and accuracy in a large-scale network emulator. *SIGCOMM Comput. Commun. Rev.*, 32(3):28–28, 2002.