Performance Evaluation of KauNet in Physical and Virtual Emulation Environments

Thomas Hall, Per Hurtig, Johan Garcia and Anna Brunstrom

Faculty of Economic Sciences, Communication and IT
Computer Science
Karlstad University Studies | 2012:32
Performance Evaluation of KauNet in Physical and Virtual Emulation Environments

Thomas Hall, Per Hurtig, Johan Garcia and Anna Brunstrom
Performance Evaluation of KauNet in Physical and Virtual Emulation Environments

Thomas Hall, Per Hurtig, Johan Garcia and Anna Brunstrom

Research report

Karlstad University Studies | 2012:32

ISSN 1403-8099


© The authors

Distribution:
Karlstad University
Faculty of Economic Sciences, Communication and IT
Computer Science
SE-651 88 Karlstad, Sweden
+46 54 700 10 00

Print: Universitetstryckeriet, Karlstad 2012

WWW.KAU.SE
Abstract

Evaluation of applications and protocols in the context of computer networking is often necessary to determine the efficiency and level of service they can provide. In practical testing, three different options are available for the evaluation; using a physical network as a testbed, using an emulator to simplify the infrastructure, or using a simulator to remove reliance on infrastructure entirely. As a real network is costly and difficult or even impossible to create for every scenario, emulation and simulation is often used to approximate the behavior of a network with considerably less resources required. However, while a simulator is limited only by the time required to perform the simulation, an emulator is also limited by the hardware and software used. It is therefore important to evaluate the performance of the emulator itself, to determine its ability to emulate the desired network topologies.

The focus of this document is the KauNet emulator, an extension of Dummynet that adds several new features, primarily deterministic emulation of various network characteristics through the use of pre-generated patterns. A series of tests were performed using a testbed with KauNet in both physical and virtual environments, as well as a hybrid environment with both physical and virtual machines. While virtualization greatly increases the flexibility and utilization of resources compared to a pure physical setup, it may also reduce the overall performance and accuracy of the emulation.

From the results achieved, KauNet performs well in a physical environment, with a high degree of accuracy even at high traffic loads. Virtualization on the other hand, clearly introduces several issues with both processing and packet loss that may make it undesirable for use in experiments, although it may still be sufficient for scenarios where the requirements for accuracy are lower. The hybrid environment represents a compromise, with both performance and flexibility midway between the physical and fully virtualized testbed.
1 Introduction

In computer networking, it is often necessary to evaluate the performance of various protocols and applications in order to provide the best service possible. The three most common methods of testing is using a simulator for full control of the environment, a physical network for a scenario as realistic as possible, or an emulator that is a middle ground between the two others, sharing some of the advantages and drawbacks. Using an emulator, the required infrastructure can be greatly simplified when compared to a physical network as most of the network will be emulated, while also retaining a high degree of reproducibility and the ability to use the actual implementations of protocols and applications. Unlike the simulator, however, the emulator is still dependent on the available hardware and experiments are performed in real-time, meaning that the performance of the emulator itself as well as the environment in which it is used is also of interest to evaluate.

One available network emulator is KauNet. KauNet is an extension of the Dummynet emulator, adding several new features, but primarily the ability to control the emulated delay, bandwidth, packet losses, as well as introducing bit errors, packet reordering and a messaging mechanism through the use of patterns. Dummynet/KauNet filters traffic through different user-created pipes, an emulated network that can be configured with various network characteristics such as bandwidth, delay and packet loss. Filtering is done through the use of the ipfw application, that uses a rule-based matching mechanism to classify traffic based on various characteristics such as source and destination ip and port or protocol. In KauNet, a pipe can also be assigned several patterns for dynamic changes of the emulation effects during the experiment. A pattern is a file containing information about how and when a specified emulation effect should be applied. The use of patterns allows emulation effects to be applied in an exact and reproducible manner, on a per-packet (data-driven) or a per-millisecond (time-driven) basis.

The purpose of this document is to evaluate the performance of KauNet as well as several different environments in which it can be used. Specifically, the tests will evaluate the maximum bandwidth that the emulator and environments can generate and handle, and the accuracy of the delay mechanism. This will be done for both single pipes for cases where only a single path needs to be emulated, as well as multiple pipes for when the emulation is required to include several paths.

2 Experimental setup

The experiments are based on a typical setup for KauNet, using three nodes; A and B serving as the end points, and the gateway node G that routes traffic from A to B. G is also the emulator host and will represent a network topology between A and B. Nodes A and B use various versions of Ubuntu Server as the operating systems, depending on the environment, while node G uses 32-bit FreeBSD 7.3 with a custom kernel for KauNet. All nodes feature
a control network interface used for remote managing and monitoring of the nodes. In addition, an experiment network connecting A→G→B consists of a second interface for nodes A and B and two additional interfaces for node G.

A number of different environments are evaluated in order to determine the performance of both KauNet as well as the different ways in which it can be deployed. These environments include a physical environment in which separate physical machines are used for each of the three nodes, a virtual environment in which all three nodes are operating as guests in a virtualization host on a single machine, and a hybrid environment in which separate physical machines are running nodes A and B as virtualized guests. Using virtualization allows for greater flexibility and better use of available resources, but may also introduce overhead and other problems that negatively affect the performance. The different environments are presented in greater detail in the following subsections.

2.1 Physical

![Physical environment](image)

In the physical environment (Figure 1), each node runs on a separate physical machine. Two different setups were examined in order to determine the effects of different systems; the first using machines comparable to standard desktop computers, and the second using rack servers. The hardware of the desktop machines are identical, using Intel Core2Duo E8400 CPUs, 4GB of memory, and Intel Pro 1000GT network cards for the experiment network. Nodes A and G use 64-bit Ubuntu 10.10 as their operating system. In the server environment, the KauNet machine uses an Intel Xeon E5405 quad-core with 4GB of memory, while the two end nodes are less powerful than the desktop machines using dual-core Intel Xeon 2.8. Both end nodes use 64-bit Ubuntu 10.04.

The physical environment should produce the best results and perform better than any of the other environments, assuming identical hardware, but comes at the cost of management overhead and poor use of resources. Each setup will require three dedicated machines, and
switching operating systems or using the machines for any other tasks may require a complete reinstallation of the operating system and any other necessary software.

2.2 Virtual

The virtual environment (Figure 2) consists of a single physical virtualization host running a dedicated virtualization OS. For the host, both VMWare ESXi and Citrix XenServer are used for virtualization. Two different machines were used for testing the virtual environment. The first is a desktop machine slightly better than those used in the physical test; an Intel Core2Quad Q6600 CPU with 4GB of memory, with the nodes each being given one dedicated CPU core and 1.25GB of memory. The second machine is a server using dual Intel Xeon E5606 CPUs with 24GB of memory, each virtual node using 2 dedicated cores and 4GB of memory similarly to the physical environment. A single network card is also used for control traffic, while the experiment traffic is routed internally in the host machine using virtual network interfaces in the guests. In the virtualized environments, Ubuntu Server 11.04 was used as the guest OS of nodes A and B.

The virtual environment presents the best use of resources and the highest degree of flexibility possible, with the ability to configure an entire setup with few restrictions and only use a single physical computer for the entire setup. In addition, using virtual nodes allows for easy storage and restoration of setups through moving and cloning of virtual machines. Unfortunately, it is also expected to yield lower performance than a physical counterpart, as virtualization of several guests is bound to generate overhead, as well as the available processing power and memory of the physical host being shared between several nodes.
2.3 Hybrid

The hybrid environment (Figure 3) uses the same set of physical machines as in the physical desktop environment. The difference lies in that the end nodes run a dedicated virtualization OS, as in the virtual environment, with the end nodes running as Ubuntu Server 11.04 guests on separate machines. The guests are identical to those used in the virtual environment, although the network configuration has been slightly altered to use the physical network cards.

The hybrid environment is a compromise between the physical and the virtual environments. While it requires the same hardware as the physical environment and still suffers from some amount of virtualization overhead, the use of separate physical hosts for each guest node should reduce the potential performance issues. Having the guests virtualized also means that cloning, storing and replacing guests can be done easily, resources are more easily given and returned on demand, as well as the ability to use the available resources for other tasks (for example as computer clusters).
3 Evaluation

The goal of the evaluation was to determine the performance of both the KauNet emulator as a separate component, as well as provide an estimation of the performance one can expect to achieve using the various environments presented in the previous section. Specifically, the experiments examined a) the traffic rate/bandwidth that the sender, node A, could generate and transmit, b) the traffic that the receiver, node B, could receive and process, c) the throughput of the emulator, d) the amount of packet loss and where they occurred, and finally e) the accuracy of the delay mechanism at various loads. Tests were done for a single pipe for scenarios where only a single path is required, as well as a varying number of multiple pipes in order to determine if additional rules and pipes in the emulator affect overall performance in scenarios where more complicated networks must be emulated.

![Figure 4: Logging points](image)

The results of the experiments consists of several logs collected at various points during the experiment. Specifically, these logging points were the interfaces of each node, as illustrated in Figure 4. Logging at points A, G1 and G2 was performed using tcpdump. A modified version of tcpdump was implemented for use in the first experiment after an initial round of tests revealed that the standard implementation was unable to log all the packets fast enough at some of the traffic rates used in the tests, resulting in some packets being ignored. To alleviate this, tcpdump was modified to write considerably less data (16 bytes instead of 54 bytes per packet). In addition, a RAM disk was created on each node to write the logs to in order to avoid overhead from the disk, thus allowing tcpdump to perform considerably better. Logging at point B was done using a simple application for receiving and logging the received traffic. For all logs, each logged packet consisted of 16 bytes of data, containing a timestamp with microsecond resolution, a packet flow ID and a packet sequence number.

3.1 Experiment 1: Bandwidth, packet loss and latency

With increasing amounts of traffic comes an increased load on all machines involved. The first experiment was designed to evaluate the performance of the setup under different levels of load, to determine how much data the different environments can be expected to handle,
especially in terms of how much traffic the emulator can forward while still being able to perform the emulation accurately. The aspects analyzed are the end-to-end bandwidth, the throughput of the emulator, the accuracy of the delay mechanism of the emulator, and the potential amount and location of packet losses.

Testing was done using mgen, sending traffic with node A as the source and node B as the destination. While several traffic generators are available, mgen was chosen as it is well-known and implemented for a number of different operating systems. UDP was chosen to be used exclusively as it allowed for full control of the experiment traffic, unlike a protocol such as TCP that would heavily influence the results due to potential retransmissions, ACK traffic and the congestion control mechanism. Each test lasted 20 seconds each, during which traffic was continuously generated and sent from node A to node B, via the KauNet emulator on node G where the emulation effects were applied to the traffic. The tests filtered the traffic through a single pipe, configured with unlimited bandwidth to avoid unnecessary queuing problems (the traffic rate being limited by mgen), a fixed delay of 10 milliseconds, and a KauNet pattern. A total of four variable factors were used in the tests, specifically the generated traffic rate, the size of the packets, the KauNet pattern type (time- and data-driven), and the degree of pattern utilization.

Four different packet sizes were used; 100 bytes, 500 bytes, 1000 bytes and 1500 bytes. All packet sizes also included the 24 byte UDP header.

Several different bandwidths were used, ranging from 5-500 Mbps for all packet sizes except 100 bytes, which was limited to 300 Mbps as none of the environments could provide the packet rates required to go higher. 40 different bandwidths from this range (30 for 100 byte packets) were selected, with the interval between them increasing with higher bandwidths. Using the packet size, these bandwidths were then translated into an equivalent packet rate for use with mgen.

The patterns used in the test were all delay change patterns, using different modes and utilization. Both time- and data-driven patterns were used. Utilization was based on the frequency of which the patterns were invoked, ranging from 0% (a single invocation at the start) to 100% (an invocation at every packet/millisecond, depending on the pattern mode). The invocation positions were uniformly distributed across the patterns. Unique patterns were generated for each combination, resulting in a total of 202 different patterns.

3.1.1 Bandwidth
As one potential problem is mgen’s ability to generate traffic at the requested rate, the evaluation of the bandwidth consisted of two parts; the generated bandwidth on the sender and the received bandwidth on the receiver. The two were compared in order to determine the throughput of the emulator, as no more traffic than was generated can be received, as well as the rate at which traffic could be generated in the environment. While pipes can be configured to limit bandwidths to certain values, the accuracy of this mechanism was not evaluated, only how much stress the environment could endure.

Both the generated traffic and the received traffic could be calculated using log A and
log B respectively. From the logs, the total number of packets could be extracted. As the size of all packets were known, the number of packets and the size of each packet could be used to determine the amount of data that was sent and received at nodes A and B. As the time during which data was sent was fixed and known, the average bandwidth could also be calculated.

3.1.2 Packet loss

A potential result of the machines being overloaded is the occurrence of undesired packet loss. The purpose of the packet loss evaluation is to determine if this is a potential problem for KauNet, and if so, how severe and the correlation between packet loss and traffic load. When packets fail to arrive at the destination, it is also of interest to determine where the loss occurs. Especially so if this is caused by the emulator, or if another factor such as the network hardware is to blame.

Three loss points were checked, specifically the A→G link, the internal forwarding in G, and the G→B link. This was done by comparing the logs of the connected interfaces (i.e. log A and G1, G1 and G2, and G2 and B). Any packet present in one log but not in the log of the next interface is a lost packet, and the total number of packet losses for each link could be estimated.

3.1.3 Latency

Pipes in KauNet can be configured with a fixed delay with a millisecond resolution. The purpose of the latency evaluation was to determine just how accurate this delay mechanism is at various levels of traffic rates as the load on the emulator increases.

The latency of each packet was checked using the incoming and outgoing logs of node G (logs G1 and G2). Packets could be identified and matched between the logs using the flow ID and the sequence number in the logs, and the latency could be determined by comparing the corresponding timestamps. As the two logs were created on the same machine, no clock synchronization was necessary and clock drift should be a minor issue.

3.2 Experiment 2: Latency with no load

As in the latency tests under load in the first experiment, the second experiment was designed to determine the accuracy of the delay mechanism, but this time minimizing the load on the emulator so that the results will be influenced by other factors as little as possible.

As in the previous tests, each test used a single pipe with unlimited bandwidth, but with a fixed packet size, packet rate, and pattern utilization of 100%. Instead, only delays varied between tests. Packets were generated using ping, sending 1000 standard sized ICMP packets (64 bytes) at a rate of 10 packets per second, for a considerably lower traffic load than in the previous tests. With small packets and a relatively slow rate, queueing delays could be avoided so that the fixed delay mechanism in KauNet could be isolated and its accuracy measured. Using the standard tcpdump, timestamps were logged for the ping request packets
on either side of the emulator as they entered and left respectively, corresponding to logs G1 and G2. Tests were performed with varying values for the fixed delay, ranging from 0 to 10 milliseconds. For delays 1-10 milliseconds, a data-driven delay change pattern with a single invocation at the first position was used. For the 0 milliseconds delay test, no pattern was used as zero-values are as of yet not supported in KauNet patterns. The delay of each packet could be extracted by directly comparing the timestamps in the two log files.

3.3 Experiment 3: Multiple pipes

A feature of Dummynet is that traffic can be filtered through different pipes, each pipe having unique emulation effects, for example when is is desirable to emulate multiple clients connecting to a server. An additional experiment was created to evaluate the performance of KauNet while using several pipes, as opposed to a single pipe as used in the previous experiments.

The multi-pipe experiment is based on the first experiment, sharing many of the same parameters. Again 20 seconds of UDP traffic per test was generated using mgen, through identical pipes configured with unlimited bandwidth, a delay of 10 milliseconds, and a KauNet pattern. The difference is that the bandwidth is now fixed, and instead the number of different mgen traffic flows and pipes varies. The number of flows ranged from a single flow, up to 499 flows, as a single instance of mgen was unable to generate traffic for additional flows with unique ports. The total bandwidth was then evenly distributed across the flows. A unique pipe was also created and assigned to each flow, although beyond the traffic filtering, the pipes themselves were identical. In addition, due to results of experiment 1, only one pattern was used; a data-driven delay change pattern at 100% utilization.

While it was initially intended for both the throughput and the latency to be evaluated for multiple pipes as in the first experiment, initial testing showed that with an increasing number of pipes at high loads, the performance of the gateway node dropped and the logging was unable to keep up, missing a substantial portion of the packets. Therefore, the logging at the gateway node was disabled during the tests using the maximum stable bandwidth, as the delay could not be accurately measured with so many missing packets anyway, and to remove unnecessary load from the gateway that may lower end-to-end performance further. Initial attempts using end-to-end delay and clock synchronization proved unreliable when comparing the estimated delays seen in the first experiment, and was therefore not used as a substitute here. As a result, logging was only performed at the sender and the receiver in the same way as in the bandwidth tests, and only the throughput and end-to-end losses could be estimated. A second experiment was therefore performed, using a bandwidth value one magnitude lower, while otherwise identical to the previous experiment. This experiment did not suffer from the logging problem, and could log traffic at all points to include measurements for the delay.

11
4 Results

In this section, the results of the experiments for each environment are presented as graphs. While experiment 1 included varying pattern utilization and both time- and data-driven patterns, the results showed no noticeable difference between them. Therefore, all results shown here uses data-driven patterns at 100% utilization, the results of time-driven patterns and patterns of lower utilization being omitted to save space.

Also, while tests were performed using XenServer in the virtual environment, the results of the initial bandwidth tests showed less than 20 Mbit/s for the best case, and as such, the XenServer results are omitted entirely. There were indications that it was a computational limitation that caused the poor results, and it is possible that using a paravirtualized kernel would improve performance. However, as paravirtualization is not supported for all operating systems, including FreeBSD, this was not evaluated further.

Furthermore, it should be noted that the logging itself had an impact on performance. As an example, the achieved rate of generated traffic on the physical desktop would in one case drop to approximately 75% with logging enabled as compared to no logging. This is a problem especially for the virtual environment, where all logging takes place on the same machine and impact the performance of the entire environment. During a normal experiment, it may be unlikely that logging of all network interfaces is required, and performance may be significantly increased.

4.1 Experiment 1: Bandwidth, packet loss and latency

The first experiment was performed for all the available environments to provide a baseline to compare against.

4.1.1 Bandwidth

Here the results of the bandwidth analysis are presented. The results are illustrated as graphs, showing a) the actual generated traffic (Y-axis) as compared to the requested (X-axis), and b) the received traffic (Y-axis) as a percent of the actual generated traffic. The ratio was used as in some cases, mgen was unable to generate the requested traffic, and thus the results would be misleading as traffic that was not sent could not be received. Each graph features all the different packet sizes as a unique plot.

Physical

The results of the two physical environments are shown in Figure 5 and Figure 6 respectively. Both environments performed well overall, with the received traffic generally matching what was sent.

Of note is that mgen is somewhat unreliable, with the actual generated traffic varying slightly from what was requested. Both environments also reach a point where mgen is unable to generate traffic faster, where the curves for the different packet sizes plateaus. For
the server environment, this happens somewhat sooner, as a result of lower processing power of the sending machine. The packet rate is more relevant here than the bandwidth, with the different plateaus occurring at approximately the same packet rates for the different packet sizes.

Both environments perform fairly well, with the received traffic being close to or exactly what was sent for all packet sizes and rates in the server environment. The desktop environment shows an upper threshold however, at around 400 Mbps for packets at 500 bytes or larger, after which performance drops considerably.

**Virtual**

The results of the virtual ESXi environments are shown in Figure 7 and Figure 8. The performance of both virtual environments were far below that of both physical environments for all rates and packet sizes. Interestingly, the traffic generation appears to be more stable and capable of very similar traffic rates for the desktop machines compared to the considerably more powerful server. While the reason for this behavior is unknown, it is possible that configuration of the host may improve performance.

**Hybrid**

The results of the hybrid ESXi environment is shown in Figure 9. With the same machines being used as in the physical desktop environment, the results are also somewhat comparable. The traffic generation is slightly more unstable when done on a virtual guest than on a physical host, but the performance remains fairly similar. The most important difference is that losses are considerably more common, and while nearly all packets arrive at the receiver, there is still a drop by 0.1-0.3% at several points. Otherwise, the environment
performs similarly to the physical counterpart, with the same kind of threshold at 400 Mbps being present.
Figure 7: Virtual ESXi desktop environment - Throughput

Figure 8: Virtual ESXi server environment - Throughput
Figure 9: Hybrid ESXi environment - Throughput
4.1.2 Packet loss

The packet losses as a function of the expected bandwidth for the different packet sizes at the different points during the experiments are presented here. As most loss ratios tended to be fairly low but occasionally having much higher points, a logarithmic scale was used for the loss ratio. The ratio is calculated using the incoming and outgoing packets for each link, so losses shown are not cumulative but specific to each link.

Physical

![Graphs showing packet loss in different environments](image)

Figure 10: Physical desktop environment - Losses

The results of the two physical environments are shown in Figure 10 and Figure 11 respectively. Both physical environments show fairly few losses for most tests. For the desktop environment, significant losses only start occurring once the thresholds for the generated traffic has been reached, and especially once the bandwidth thresholds seen in the received graph.
has been exceeded. This is clearly seen in the internal gateway losses graph, where there is an area with a large cluster of high losses, caused by the sending network interface being unable to process the packets quickly enough. For most other rates, losses are either very few or non-existent. It can also be noted that in the few cases where isolated instances of packet loss does occur for the gateway (such as the 0.005% loss at 5, 100, 360 and 240 Mbps for packet sizes 100, 500, 1000 and 1500 bytes respectively), the packet losses correspond to the number of packets expected to be processed by KauNet during one millisecond (kernel tick).

The results are similar for the server environment, although noticeably with no losses at all for the A→G path. The gateway shows one point of 10% losses, but as this is not reflected in the bandwidth graphs it is most likely a logging error. Other than this, losses do occur, but only in very small numbers.
Virtual

The virtual environments, shown in Figures 12 and 13, showed significant losses for all links. While losses do increase as the bandwidth and packet rate increases, losses occur in large numbers at all rates for all packet sizes. Overall, the desktop performs slightly better than the server, but the difference is small and still suffers from significant losses.

Hybrid

For the Hybrid ESXi environment, the losses are shown in Figure 14. For the sender and the gateway, the graphs are fairly similar to those of the physical desktop, as is to be expected since they use the same hardware. Between the gateway and the receiver there are considerably more losses, however, mostly between 0 and 0.15%, for various rates and packet sizes with no specific pattern.
Figure 13: Virtual ESXi server environment - Losses
Figure 14: Hybrid ESXi environment - Losses
4.1.3 Latency

Here the results of the delay mechanism experiments under different traffic loads and using different packet sizes are presented. The results are illustrated as a bar chart, aggregating the delays of the packets into discrete steps of 1 millisecond each, with the decimal being truncated. A value of 9 milliseconds thus represents any packet with a delay between 9 and 10 milliseconds. This is the delay interval expected for a pipe configured with a 10 millisecond delay, due to an undercompensation in the delay mechanism which will be illustrated in the results of experiment 2. In order to conserve space, the results are grouped into an approximate behavior as several tests showed similar results. Only some selected graphs are included to illustrate examples of the different trends.

Physical

For the physical environments, the results are for the most part good, with the majority of packets (90% or higher) falling under the expected delay of 9-10 milliseconds, with the rest being shared between 8-9 and 10-11 milliseconds, with a bias for 10-11 milliseconds and sometimes without any packets at 8-9 milliseconds at all. However, for 100 byte packets, this behavior deteriorates at 165 Mbps but remains fairly stable after that, with a larger spread and worse ratios, corresponding to the limit for the generated traffic. For larger packets, a similar but also gradually deteriorating behavior can be found past the bandwidth thresholds shown in the figures for received bandwidth (see Figure 5).

The main difference between the desktop and the server environments was that the server performed slightly better. Nearly no packets were present in the 8-9 millisecond range, and a far larger ratio were found in the expected 9-10 millisecond range (above 95%, usually 98-99% of the total). Also as expected, the server did not show any signs of a maximum threshold, as none were found for the bandwidth tests, and performed just as well for all rates and packet sizes.

Examples of these behaviors are shown in Figure 15.

Virtual

In the virtual ESXi environments, results are significantly worse than in the physical (see Figure 16). While the majority of packets will still fall in the correct 9-10 millisecond range, there is a wider spread even from the start. With increasing bandwidth, the range of packets becomes considerably wider, with less packets with the correct delay. All packet sizes display an almost identical behavior, the only difference being that smaller packet sizes tend to deteriorate earlier, at around 35, 150, 240 and 380 Mbps for 100, 500, 1000 and 1500 byte packets respectively for the desktop environment. The server performs nearly identical to the desktop, although the deterioration starts somewhat earlier, at approximately 25, 110, 195 and 300 Mbps for the different packet sizes. Other than this, the behavior of the server is identical, with wider ranges and fewer packets in the correct range.
Hybrid

As the measurements used for the hybrid tests were collected from a machine identical in configuration to the one used in the physical environment, the expectation was therefore that the results would be identical to those previously seen. As this was also the case for the actual results achieved, the results of the hybrid environment will not be presented separately here. Instead, refer to the results for latency in the physical desktop environment.

Figure 15: Physical environments - Delay
(a) Good behavior (100 byte packets at 5 Mbps)  
(b) Poor behavior (100 byte packets at 35 Mbps)

Figure 16: Delays in the virtual VMWare ESXi environment
4.2 Experiment 2: Latency with no load

The results of the delay mechanism experiments without a traffic load are presented here. The graphs show the latency in milliseconds for each individual packet sent. The graphs contain one line for each of the different fixed delays from 0 to 10 milliseconds, for a total of 11 lines per graph.

Physical

![Graph showing latency under low load](image)

For the physical desktop environment (see Figure 17), the delays are for the most part stable at the specified rates, with a few spikes of 1 millisecond occurring periodically. For a configured delay of 1 millisecond, the latency shows a periodic pattern of increasing delay up to the upper bound, followed by a drop down to the lower bound. The server environment (see Figure 18) show similar results, but in comparison have few of the delay spikes and only small ones. It also shows a periodic pattern at the configured delay of 1 millisecond, but in this case a decreasing one.
Virtual
The same behavior as in the physical environment can also be observed in the virtual ESXi environment (see Figure 19 and 20), although considerably less stable. The virtual environment also shows a greater variation, with some spikes being much larger, when compared to the physical environments. Comparing the desktop and the server, the results are fairly similar. The desktop shows a greater variation, although this is mainly caused by many packets arriving at the millisecond change, resulting in the kind of behavior where the delay jumps back and forth between the upper and the lower limit. A periodic pattern of increasing delays and a sudden drop for the desktop, and of decreasing delays with a sudden increase for the server can be observed for configured delays of 9 and 10 milliseconds.

Hybrid
As with the latency under load, the same machine was used for the measurement as in the physical environment, with the same results obtained. See the results for the physical environment.
4.2.1 Delay undercompensation

What is of interest to note in these results is that Dummynet appears to undercompensate the delays, e.g., a configured delay of 1 millisecond will in most cases result in a delay between 0 and 1 milliseconds. The only exception is the packets with a fixed delay of 0 milliseconds, where they are almost perfectly aligned with the X-axis with only a few microseconds delay and with no delay spikes. This behavior can be explained by the Dummynet code in regards to how the fixed delay mechanism is implemented. As packets enter the emulator, the packets will be affected by a fixed delay immediately and given an output time based on the current time, assuming no bandwidth queuing takes place first, which is not the case in these experiments. Dummynet uses an internal tick counter with a resolution equal to that of the kernel tick rate to schedule events, including the release of packets from the delay queue. In this case, it means that the current time, and thus the output time of the packets, are only updated once per kernel tick. Any packets arriving at any point during a tick will be considered to have arrived at the same time, at the start of the tick. For a 1000Hz kernel as was used in these experiments, this means that the output time will lag behind by between 0-1 milliseconds, resulting in the fixed delay being undercompensated by 0.5 milliseconds.
on average. For a configured delay of 0 milliseconds, this mechanism is bypassed entirely, letting the packet move through the emulator as fast as it can be processed.

In the graphs, this can be seen as most packets tend to lie between two milliseconds, with the upper being the specified fixed delay. The occasional spikes occur when a packet misses a tick, and it can be noted that these spikes are generally close to or exactly 1 millisecond high, corresponding to a kernel tick. If a packet lies close to the threshold, it may sometimes arrive before and sometimes during a specific tick, resulting in the delays alternating between the two millisecond limits. This kind of behavior is particularly common for the virtual desktop (Figure 19). By comparison, packets arriving in the middle of a tick show few of these spikes, although occasional spikes may still be present due to a missed tick.

A second test was performed on the physical desktop in order to verify this (see Figure 21). Here the kernel tick rate was increased from 1000Hz to 2000Hz in order to improve the resolution to 0.5 milliseconds instead of 1 millisecond. The expected behavior here would be similar to that of a 1000Hz kernel, except with thresholds and spikes being 0.5 milliseconds apart instead. As shown in the graph, this was also what happened. Higher accuracy may be possible by increasing the tick rate further, although it may also impact overall system
Figure 21: Physical desktop environment - Latency under low load using a 2000Hz kernel performance negatively and will not be included in this evaluation.
4.3 Experiment 3: Multiple pipes

The results when using multiple mgen flows and pipes in the emulation are presented here. The graphs show both the total generated and received traffic across all flows, with the X-axis representing the number of flows/pipes used, and the Y-axis the measured bandwidth. Only the two physical environments were used in this experiment, as the purpose was only to compare the performance of using multiple pipes versus a single pipe, with less focus on the type of environment used. However, as problems with mgen overshadowed the potential issues of the gateway for the server machines and no performance degradation could be seen, these results are omitted.

![Figure 22: Physical desktop environment - Multipipe throughput](image)

In the physical desktop environment, the fixed rates used was 120, 360, 400 and 400 Mbps for packets of 100, 500, 1000 and 1500 bytes respectively. Behavior for 100 byte packets is erratic, with mgen showing a decreased performance with the number of flows. Throughput starts dropping at around the 200 pipe mark and then decreases steadily (the fluctuations for the receiver is caused by the fluctuations of the generated traffic). For all other packet sizes, packet generation appears to be stable. For 500 byte packets, throughput starts dropping at around 375 pipes and decreases steadily from there. No decrease in performance can be seen for the remaining packet sizes, though this would likely occur with a higher packet rate or additional pipes.

As previously mentioned, a second experiment was performed using a bandwidth one magnitude lower than the maximum in order to determine the influence of multiple pipes on the delay mechanism. The throughput results shown in Figure 23 offer no surprises, with mgen generating traffic at the requested rate (although some issues with rounding in the conversion between packet rate and bandwidth can be seen where the generated traffic exceeds 100%) and all traffic being received. The results for the latency however, shows degrading performance with the number of pipes (see Figure 24). Here a clear trend can be
seen, with more pipes decreasing the accuracy of the delay mechanism noticeably. With more pipes, delays tend to spread out wider, favoring higher delays more and more as the number of pipes increases with less packets falling under the correct delay of 9-10 milliseconds (i.e. 9 in the graph). Although not shown here, there does not appear to be any bias between the flows, with the distribution of different delays being fairly equal between them.

Figure 23: Physical desktop environment - Multipipe throughput, low load
Figure 24: Physical desktop environment - Multipipe latency, low load
5 Conclusion

All experiments have seen limited bandwidth, varying with the traffic rate, as well as losses occurring. There are several factors that may influence the results, specifically mgen’s ability to generate packets at the correct rate, the processing power of the machines, the performance of the network hardware, and the logging mechanism. In the virtual environment, the network hardware is not a factor as all traffic is routed internally in the same machine through virtual interfaces. On the other hand, with resources being shared, processing power and memory may be limited instead. In addition, the virtual environment suffers from virtualization overhead that may further reduce its performance. For all environments and for all the different factors evaluated, performance seems to be primarily related to the packet rate, the size of the packets, and thus the bandwidth, being of less importance.

5.1 Physical

Both physical environments perform well overall. The desktop suffers from a hard threshold at around 400 Mbps regardless of the packet sizes and rates involved. Past this threshold, significant losses occur in the gateway, as well as a lower performance of the delay mechanism. This threshold is likely caused by the specific network hardware configuration of the gateway, as it occurs at approximately the same bandwidth regardless of the packet rate, is repeatable, and is apparently unrelated to the traffic load. In addition, a different machine with less memory and a slower CPU running the same operating system but using a server-grade network card had no similar problem. No such threshold is present for the server environment, and the results are overall more stable, with less variation in delay and very few packet losses at all packet sizes and rates. The KauNet machine in the server environment is considerably more powerful than its desktop counterpart, which in combination with higher quality network hardware may explain the results. On the other hand, the desktop environment manages to generate more traffic than the server environment, in particular at lower packet sizes, most likely because the sending node in the server environment is not as powerful as that in the desktop environment.

Regardless, both alternatives perform adequately for a typical use case and KauNet does not appear to be the limiting factor in these experiments. Traffic rates are primarily limited by what can be sent, and KauNet can be expected to process the packets in a timely manner, with few of the packets outside the specified delay. Better network hardware and a sender capable of generating traffic faster will likely yield better results.

With multiple pipes, an additional factor in performance becomes apparent. With several pipes to filter the traffic through, performance is noticeably reduced as more pipes are added once the load reaches a certain threshold. This affects both the throughput of the emulator, as well as the accuracy of the emulation effects, although the problems are fairly minor unless high loads or a fairly large number of pipes are used. It does not appear to be a problem in Dummynet however, at least not exclusively, as a large number of regular ipfw rules will also result in reduced performance, indicating that it is an issue with the rule matching.
5.2 VMWare ESXi

The fully virtualized ESXi environment performed considerably worse than the physical environments, with lower bandwidth, more unpredictable delay, and a generally less stable performance. This held true for both the desktop and the server, despite the latter in particular having plenty of resources to draw upon.

What causes the problems in ESXi is difficult to pin down. While the network hardware is eliminated as a factor, the fact that resources are shared will instead cause difficulty in identifying the problem. For instance, if the sender is unable to generate packets fast enough, it may also drain the available processing power for the emulator and receiver, causing problems for the other two machines as well. Further indication of the processing power being the problem is the severe impact on performance that enabling logging has, with bandwidths being reduced and losses increasing dramatically. In addition, problems may arise from the virtualization environment itself. With the network hardware being unused and the packets being routed internally, it may put additional strain on the software of either the guest or the virtualization host itself that would otherwise be offloaded on the hardware, thus decreasing performance. Furthermore, while ESXi offers several features for improved performance of both computation and networking, not all of these are readily available for all operating systems. It is also possible that some operating system features interact poorly with virtualization, such as interrupt handling of network devices. FreeBSD especially seems to suffer from several problems when virtualized, with some tests resulting in the virtual machine becoming unresponsive to network traffic and forcing the guest to be rebooted, more frequently occurring at higher packet rates, though not in a reliably reproducible manner.

Although outside the scope of this document as the comparisons was intended to be between virtual guests and physical hosts with identical settings, the problems with ESXi were explored further in an attempt to determine the cause of the performance issues. For an overview of what was tried, see Appendix A.

Regardless of the exact cause of the poor performance, a fully virtualized environment performs poorly compared to a physical environment. The virtualization itself brings several issues that will reduce performance, at least without careful tuning of both the guests and the host which in itself may be an undesirable overhead. While it would serve as a development environment, performing actual tests that rely on a reasonable degree of accuracy would be inadvisable.

5.3 Hybrid

The hybrid environment offers a compromise between the physical and the virtual environment in that the endpoints are relatively easy to replace. It also performs relatively well and in most respects similar to the physical environment as the gateway remains unchanged, although the virtualization of the end points introduces additional end-to-end packet losses. Another issue is the limited virtualization support of some operating systems which makes the hybrid solution somewhat less flexible than one would desire. Also, guests appear to
suffer from timing issues, showing a more unstable packet generation of mgen and also a considerably greater clock drift. Still, while some issues can be seen, the hybrid environment can perform quite well under the right circumstances, and should be sufficient for most scenarios where absolute reliability is not required.

Acknowledgments

This work was supported by Nordforsk through the project “A Living Labs network for user-driven innovation of ICT services”.

35
A Further tests

In this appendix, various miscellaneous tests are listed that were performed in an attempt to determine the cause of the problems encountered for the different virtual environments. Few of these tests were directly related to the performance of KauNet, but may be still relevant when attempting to configure a virtual environment.

A.1 FreeBSD configuration

As noted previously, the gateway node running FreeBSD in particular appears to suffer from problems when running in a virtual environment, even when running as the only guest on a dedicated physical host. In particular, these problems can be seen as packet losses for all links, even under relatively low traffic loads. In addition, when attempting to generate traffic at higher rates, FreeBSD reports that the send buffers are full, sometimes even resulting in the network becoming unresponsive and forcing a reboot of the system. By comparison, a physical installation of FreeBSD with the default configuration on the same machine does not suffer from any of these problems, and in fact performs even better than an Ubuntu installation in terms of both maximum and accuracy of the mgen traffic. As KauNet is designed to operate on a FreeBSD platform primarily and as this platform would be responsible for both the emulation as well as routing of the traffic, it would be beneficial to optimize the performance of FreeBSD under virtualization.

Specific to the FreeBSD guest, two different modifications were attempted. The first was to modify the packet RX checksum calculation. Per default, FreeBSD attempts to offload this calculation on the network card in order to spare the CPU. As the interaction with the hardware in a virtual environment is somewhat different (or even non-existent as in the case of a fully virtualized topology), this offloading was disabled to see if the emulation of the network hardware could be avoided by instead relying on the processor which, as most modern processors, has built-in support for virtualization. However, whether the offloading was enabled or not, there was no noticeable difference in either the maximum throughput or the amount of packet losses.

A second test was to modify the handling of arriving packets, by switching FreeBSD to polling the network interface rather than the default interrupt-driven behavior. Although the throughput did improve with polling, it came at the cost of reduced accuracy of the delay mechanism in Dummynet as packets will not be processed immediately but at discrete intervals. While the throughput was improved, it did not increase dramatically and packet losses remained unaffected.

A.2 VMWare configuration

In addition to tuning the guest systems, the ESXi platform itself can make use of different settings in order to alter the performance of the machines. In particular these options include the resource allocation of the guests, the operating system used, and the type of network
adapter used by the experiment network interface.

Allocating additional processor cores and memory to a virtual machine appears to make little difference. Additional cores may be of use for multi-threaded applications but in general, over allocation of resources does not improve performance, with a virtual machine performing below that of a physical machine with the same resources allocated. Also of note is the difference in performance between the two physical environments, where the server is considerably more powerful than the desktop yet actual performance is actually lower. Although this was not further examined, it may be caused by context switching between the two separate processors on the server as opposed to the single one on the desktop. It could therefore be of interest to determine whether virtual guests can be given a processor affinity in ESXi and check whether this would improve performance.

Several tests sending network traffic were also performed on different operating systems and using different network adapters. Specifically, the operating systems used were Ubuntu 11.10, FreeBSD 7.3 and FreeBSD 8.0. CentOS 6.0 was also tested, but the results were nearly identical to those of Ubuntu, presumably as both use a fairly similar Linux kernel, and are therefore omitted. For these operating systems, the E1000 and VMXNet 3 adapter was used, except for FreeBSD 7.3 which does not support VMXNet 3, and instead used VMXNet 2 (Enhanced).

The setup for these tests were similar to those previously used, though now using only two nodes. Three tests were made; a virtual machine sending packets to a physical machine, a virtual machine receiving packets from a physical machine, and two virtual machines running on the same physical host sending/receiving packets between each other (internal virtual routing). For tests involving both a physical and a virtual host, two dual core machines were used, one running a virtual guest on top of ESXi and with the evaluated operating system, the other running a physical FreeBSD 7.3 installation that has previously been verified to work well without packet losses or limited performance of mgen. For internal routing tests, a single quad core machine was used, running two virtual guests with an identical operating system. UDP traffic was generated and sent between the two nodes and recorded using the custom logging mechanism from the previous experiments.

The results of these experiments are presented in the following subsections, in Figures 25 through 42. From the results, only a few general conclusions can be drawn; the performance of the different adapters varies considerably depending on the operating system used and the setup of the environment. Furthermore, virtual guests appear to perform better when sending packets compared to when receiving them, at least in terms of packet losses. Generating packets on a virtual machine, even when running the same operating system and on an identical machine to the physical counterpart with all available resources being allocated, is also less efficient.
A.2.1 FreeBSD 7.3
E1000

Figure 25: Bandwidth: FreeBSD 7.3 E1000, physical → virtual

Figure 26: Bandwidth: FreeBSD 7.3 E1000, virtual → physical
Figure 27: Bandwidth: FreeBSD 7.3 E1000, internal virtual routing
VMXNet 2 (Enhanced)

Figure 28: Bandwidth: FreeBSD 7.3 VMXNet 2 (Enhanced), physical → virtual

Figure 29: Bandwidth: FreeBSD 7.3 VMXNet 2 (Enhanced), virtual → physical
Figure 30: Bandwidth: FreeBSD 7.3 VMXNet 2 (Enhanced), internal virtual routing
A.2.2 FreeBSD 8.1

E1000

Figure 31: Bandwidth: FreeBSD 8.1 E1000, physical → virtual

Figure 32: Bandwidth: FreeBSD 8.1 E1000, virtual → physical
Figure 33: Bandwidth: FreeBSD 8.1 E1000, internal virtual routing
VMXNet 3

Figure 34: Bandwidth: FreeBSD 8.1 VMXNet 3, physical $\rightarrow$ virtual

Figure 35: Bandwidth: FreeBSD 8.1 VMXNet 3, virtual $\rightarrow$ physical
Figure 36: Bandwidth: FreeBSD 8.1 VMXNet 3, internal virtual routing
A.2.3 Ubuntu 11.10

E1000

Figure 37: Bandwidth: Ubuntu 11.10 E1000, physical $\rightarrow$ virtual

Figure 38: Bandwidth: Ubuntu 11.10 E1000, virtual $\rightarrow$ physical
Figure 39: Bandwidth: Ubuntu 11.10 E1000, internal virtual routing
Figure 40: Bandwidth: Ubuntu 11.10 VMXNet 3, physical → virtual

Figure 41: Bandwidth: Ubuntu 11.10 VMXNet 3, virtual → physical
Figure 42: Bandwidth: Ubuntu 11.10 VMXNet 3, internal virtual routing
Performance Evaluation of KauNet in Physical and Virtual Emulation Environments

Evaluation of applications and protocols in the context of computer networking is often necessary to determine the efficiency and level of service they can provide. In practical testing, three different options are available for the evaluation; using a physical network as a testbed, using an emulator to simplify the infrastructure, or using a simulator to remove reliance on infrastructure entirely. As a real network is costly and difficult or even impossible to create for every scenario, emulation and simulation is often used to approximate the behavior of a network with considerably less resources required. However, while a simulator is limited only by the time required to perform the simulation, an emulator is also limited by the hardware and software used. It is therefore important to evaluate the performance of the emulator itself, to determine its ability to emulate the desired network topologies.

The focus of this document is the KauNet emulator, an extension of Dummynet that adds several new features, primarily deterministic emulation of various network characteristics through the use of pre-generated patterns. A series of tests were performed using a testbed with KauNet in both physical and virtual environments, as well as a hybrid environment with both physical and virtual machines. While virtualization greatly increases the flexibility and utilization of resources compared to a pure physical setup, it may also reduce the overall performance and accuracy of the emulation.

From the results achieved, KauNet performs well in a physical environment, with a high degree of accuracy even at high traffic loads. Virtualization on the other hand, clearly introduces several issues with both processing and packet loss that may make it undesirable for use in experiments, although it may still be sufficient for scenarios where the requirements for accuracy are lower. The hybrid environment represents a compromise, with both performance and flexibility midway between the physical and fully virtualized testbed.