KUTE – A High Performance Kernel-based UDP Traffic Engine

Sebastian Zander, David Kennedy, Grenville Armitage
Centre for Advanced Internet Architectures (CAIA). Technical Report 050118A
Swinburne University of Technology
Melbourne, Australia
{szander, dkennedy, garmitage}@swin.edu.au

***Abstract*—Numerous tools have been developed for generating artificial traffic flows. These traffic generators are commonly used for emulating applications, measuring various network characteristics, or just generating traffic for performance tests. The performance of many applications, such as packet measurement tools, heavily depends on the packet rate of the network traffic under observation. The existing traffic generators are mostly user space implementations, which limits their performance, especially in high-speed networks such as Gigabit Ethernet. In this paper we present and evaluate KUTE, a UDP packet generator and receiver which runs entirely in the Linux kernel. We compare KUTE with a similar user space tool named RUDE/CRUDE and find that KUTE is able to send and receive much higher packet rates, produces more accurate inter-packet gaps at the sender, and more accurately measures inter-arrival times at the receiver.**

I. Introduction

A number of tools have been developed for generating artificial traffic flows. These tools are mainly used for the emulation of application traffic, and for testing and measurement purposes. Some possible usage scenarios are:

- Emulating the traffic of real applications when it is too difficult or infeasible to use real applications
- Measuring network characteristics such as delay, loss and jitter
- Testing and evaluating the performance of applications such as servers, traffic monitors and meters

The existing tools are mostly implemented in user space, which limits their performance. We focus on the following two performance aspects: packet throughput, and inter-packet time accuracy (jitter). We refer to the inter-packet time as inter-packet gaps (sender) and inter-arrival times (receiver).

For any kind of performance tests it is desirable to maximize the number of packets that can be sent and received. User space programs can only send and receive a small fraction of the theoretical possible packet rate achievable with small packets, especially if high-speed interfaces such as Gigabit Ethernet are used. For measuring network characteristics such as jitter, it is important that the generator sends with highly accurate inter-packet gaps, and the receiver measures the inter-arrival times as precisely as possible. Very accurate inter-packet gaps are also important for all applications attempting to emulate real application traffic (aggregates), something that requires that the inter-packet times be precisely chosen from the model distribution.

To improve the performance of user space tools, we have implemented a Kernel-based UDP Traffic Engine (KUTE, pronounced like "cute") that runs entirely within the Linux 2.6 kernel. We evaluate the performance of KUTE using a Netcom Systems Smartbits 2000 and compare it with the performance of similar user space tool, Real-time UDP Data Emitter (RUDE) and Collector for RUDE (CRUDE) [1], and tcpdump [2]. RUDE/CRUDE is a well-known tool for sending UDP test traffic across a network. It uses arbitrary packet rates and can perform delay, loss and jitter measurements. Our work is focused on UDP because the maximum achievable packet rate is higher than that of TCP, and the packet rate can be more precisely controlled. Furthermore the sending of TCP streams from within the kernel is far more complicated.

The rest of the paper is organized as follows. Section II presents related work. Section III describes the implementation of KUTE. Section IV evaluates the performance and compares the results with the two other tools. Section V concludes and outlines future work.

II. Related Work

Many tools for traffic generation exist and it is not possible to give a comprehensive overview in this paper. There are a number of places that provide a taxonomy and information about existing tools e.g. [3], [4] and [5]. We compare the performance of our tool against a similar user space application called RUDE/CRUDE [1]. We also compare the KUTE receiver against the popular network monitoring tool tcpdump [2].

To our best knowledge there are few tools that perform kernel-based traffic generation. The pktgen module in the Linux kernel [6] can be used to send multiple UDP flows over Ethernet. The click project developed a kernel-based UDP sender (udpgen) and receiver (udpcount) for Linux 2.2 [7] that can be used to send multiple UDP flows and measure packet inter-arrival times.

III. Implementation

KUTE [8] was initially based on the work of the click project. However, we have introduced substantial improvements in the sender and receiver routines, ported the code to Linux 2.6 and added more features. KUTE consists of two separate kernel modules for Linux 2.6 – the sender and the receiver. This section provides an overview of their functionality and implementation.

A. *Sender*

The KUTE sender is activated by loading the kernel module (e.g. insmod or modprobe). It sends packets for a specified duration. The sender computes the inter-packet gap based on the specified sending rate (packets per second), and actively waits for the right time to send a packet. The timing is based on the CPU cycle counter [9], therefore the implementation may not work on other than Intel or AMD CPUs. The advantage of using the cycle counter over the gettimeofday function is a higher resolution (nanoseconds on 1+GHz machines whereas gettimeofday is limited to microseconds) and a higher performance when reading the current time. (An older version of KUTE uses the gettimeofday instead of the cycle counter and is therefore platform independent.) To avoid other processes from interfering with the sender, it basically blocks the kernel for the duration of the traffic flow. After the sending is complete, the kernel module has to be removed (e.g. rmmod).

In contrast to pktgen [6], KUTE can run on any link layer and the user does not need to specifiy link layer (L2) header information. This is achieved by constructing a UDP packet with an empty L2 header. The packet is then sent by injecting it into the kernel's output function that will properly set the L2 header. A copy of the packet with the L2 header generated by the kernel is kept and all subsequent packets are directly injected into the network interface driver to achieve maximum performance.

The following parameters can be specified: source and destination IP address, source and destination ports, packet rate, packet length, duration of the flow, packet payload, Time To Live (TTL), Type of Service (ToS), and whether UDP checksums and IP idenfication field should be used. The sender can create a number of different flows at the same time (up to four). The different flows can have different packet rates, but must have the same duration. At the moment, the sender can not be controlled from userspace while it is running.

B. *Receiver*

The KUTE receiver is activated by loading the kernel module (e.g. insmod or modprobe). It creates a packet inter-arrival histogram that can be accessed via the Linux proc file system. Furthermore, when the module is unloaded, it outputs the necessary information to compute the mean and standard deviation of the distribution into the kernel log file (/var/log/messages). Because the Linux kernel does not provide floting point arithmetic and we have not implemented floating point functions in our module it cannot compute the statistics directly.

The KUTE receiver can filter the traffic based on source IP address and port number, or it can simply measure all incoming UDP traffic. There are two different receiver routines and one must be selected at compile time. The first routine hooks into the Linux kernel UDP packet handler and can be used without any kernel modifications. After receiving a packet KUTE passes it on to the kernel's default UDP handler. The second alternative is faster but requires the kernel to be modified with a patch. A small code fragment has been written, that when inserted into the Linux kernel, immeditaly passes the packet to KUTE after it is received from the network interface driver (this is fast mode). In this mode, all packets received by KUTE are not passed to the usual kernel receive functions. Therefore it is essential to setup a proper filter, otherwise all UDP packets destined for some process on the receiver will never arrive. KUTE can receive packets destined to other machines if the network interface is switched into promiscuous mode (e.g. using ifconfig).

For the inter-arrival time measurement, KUTE uses the timestamps already present in the socket kernel buffer (skb) of each packet. The accuracy of this timestamp depends on whether it was generated in hardware or by the Linux kernel (the latter is the usual case with standard network interface cards). In addition to the filter, the following parameters can be specified: the number of histogram bins, and the size of the bins in micro seconds. At the moment the receiver can not be controlled from userspace while it is running and must be restarted in order to change parameters.

IV. Experimental Results

This section describes our testbed and the experiments we have done.

A. *Tesbed Setup*

We use two test setups. In the first setup a Linux PC is directly connected to a Smartbits via Fast Ethernet. In this setup we measure the maxmium sender and receiver packet rates over Fast Ethernet, and the accuraccy of the receiver inter-arrival time measurements, as we know the Smartbits inter-packet gap is very precise. Unfortunately we cannot use the Smartbits to measure the sender inter-packet gap accuraccy because our Smartbits model can only measure the inter-arrival times of 2048 consecutive packets, which is not enough for a meaningful analysis. At high packet rates 2048 packets represent only a very short time period (e.g. about 20ms at 100kpps) and we would risk overlooking any effects that may occur on larger timescales. Also measuring only a small number of packets would potentially increase the risk of biased results (e.g. RUDE generates some strange inter-packet gaps shortly after starting it).

In the second setup we connect two Linux PCs via Gigabit Ethernet. We measure the maximum sender and receiver packet rates over Gigabit Ethernet, and the inter-gap accuraccy using KUTE as the receiver.

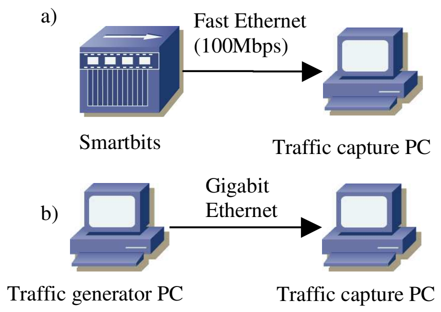

Figure 1: Testbed setup a) with one PC and Smartbits (Fast Ethernet) and b) with two PCs (Gigabit Ethernet)

Our Linux PCs are 2.4GHz machines with 256MB of RAM running Linux 2.6.4. Both have Intel 82540EM Gigabit Ethernet Controllers. Because Gigabit Ethernet cards perform interrupt bundling (also called interrupt batching or mitigation), which greatly influences the sender and receiver performance, we perform different measurements with different bundling parameters, including bundling turned off. For the tcpdump tests we have installed the MMAP version of libpcap [10] and tuned it, setting the number of ring buffer frames to 8000.

In all tests we send UDP packets of 64byte size including the Ethernet header. The packets are destined for the receiver because CRUDE can not operate in promiscuous mode (although tcpdump and KUTE can). A packet flow duration of 30 seconds is used. This duration seems to be very short but we performed a number of experiments using a duration of 5 minutes and found no significant difference. Therefore we use the shorter time interval because it makes the measurements faster and reduces the disk space needed for the CRUDE and tcpdump result files. We also compared multiple 30 second measurements made with the same settings and found the resulting differences to be very small e.g. different measurements for the standard deviation of the inter-packet times differ less than 0.1us.

B.*Maximum Sender Packet Rate*

To test the maximum sender rate on Fast Ethernet, we gradually increase the sending rate for RUDE and KUTE, measure the time needed for the sending, and count the packets received by the Smartbits. When evaluating KUTE with Gigabit Ethernet, we unfortunately have no receiver that is fast enough to receive all packets. In this case we can only 'verify' the sending rate by measuring the sending time and checking the network interface transmit counter on the sender (e.g. using ifconfig). Table 1 shows the approximate maximum packet rates that can be sent.

Table 1: Maximum sender packet rates

Sender	**Max Packet Rate [kpps]**	
	No bundling	Max 10,000 I/s
RUDE	~48	~85
KUTE (Fast Ethernet)	~122	~122
KUTE (Gig Ethernet)	~415	~415

The results show that KUTE clearly outperforms RUDE. KUTE is able to send with a much higher packet rate, especially when interrupt bundling is disabled. KUTE is not affected by the interrupt bundling setting.

C.*Maximum Receiver Packet Rate*

To test the maximum receiver rate, we use the Smartbits (Fast Ethernet) or KUTE (Gigabit Ethernet) as sender. We gradually increase the sender packet rate until the receiver does not receive all packets anymore. The rate at this point is the approximate maximum packet rate. We measure the maximum rate with different settings for the interrupt bundling (no bundling, and a limit of 10,000 interrrupts per second). CRUDE is run with per-packet output (timestamps) disabled and all tcpdump output is redirected to /dev/null to maximize their performance. The KUTE receiver is run in fast mode mode, which means the packets are received via the hook patched into the kernel. Table 2 shows the approximate maximum packet rates that can be received.

Table 2: Maximum receiver packet rates

Receiver	**Max Packet Rate [kpps]**	
	No bundling	Max 10,000 I/s
CRUDE	~46	~90
tcpdump	~50	~110
KUTE (Fast Ethernet)	~148 (max line rate)	~148 (max line rate)
KUTE (Gig Ethernet)	~220	~220

The table shows that KUTE can receive much more packets than CRUDE or tcpdump, especially when interrupt bundling is disabled. For Fast Ethernet, KUTE is able to receive the maximum rate the Smartbits can send, while for Gigabit Ethernet it is still far away from the maximum theoretical packet rate but much better than CRUDE or tcpdump. KUTE is unaffected by the interrupt bundling setting. (The maximum packet rates for CRUDE and tcpdump are very similar when the interface is switched to Gigabit Ethernet.)

D.*Receiver Inter-arrivalTime Accuracy*

Now we evaluate how accurate the different receivers can measure inter-arrival times. We use the Smartbits to send a packet flow with very precise inter-packet gaps. We disable interrrupt bundling. We measure the inter-arrival time mean and standard deviation (standard error) depending on the packet rate. The maxmimum packet rate used for CRUDE is much lower than the rate in Table 2 because in this experiment we need per-packet information (timestamps).

Figure 2 presents the standard deviation of the inter-arrival times. The mean of the inter-arrival times always had the expected value (e.g. 40us for 25kpps). The standard deviation for CRUDE is much higher than for tcpdump or KUTE because CRUDE timestamps the packets in user space, whereas both other tools use the same timestamp generated in the Linux kernel shortly after the packet has been received from the device driver. KUTE is slightly better than tcpdump, probably because it uses less CPU time and therefore creates less jitter on the receiver. We conclude that at low packet rates no kernel-based receiver is required but tcpdump cannot handle high packet rates (see section IV.C). When interrupt bundling is enabled, the standard deviation is much higher because the inter-arrival time distribution is completely different. Instead of one peak at the mean value (e.g. 40us for 25kpps), there are two peaks: one close to 0us, and one at 100us (the time between interrupts). Therefore we do not provide these results here.

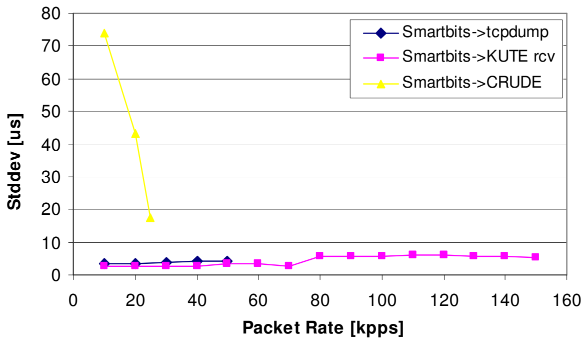

Figure 2: Receiver inter-arrival time accuracy using the Smartbits as sender

Figure 3 shows the inter-arrival time distributions of packets generated by the Smartbits with a rate of 25kpps and measured by the different receivers (25kpps is the maximum CRUDE can handle on our test machine). The distribution of tcpdump and KUTE are very similar (on top of each other) whereas the distribution measured with CRUDE is much wider and hence the standard deviation is much larger.

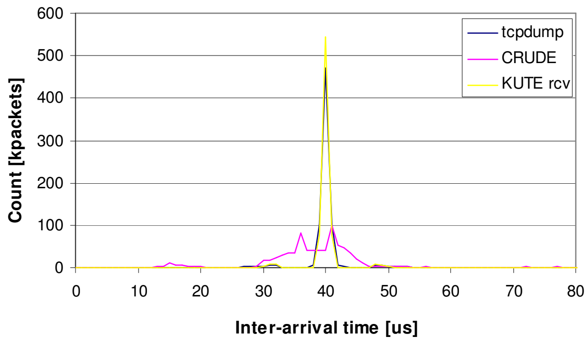

Figure 3: Inter-arrival time distributions for Smartbits sending with 25kpps as measured by KUTE, tcpdump and CRUDE

E.*Sender Inter-packet Gap Accuracy*

Finally we test the sending inter-packet gap accuracy of RUDE and the KUTE sender with a KUTE receiver. We cannot use the Smartbits as the receiver, because our particular unit can only measure the inter-arrival times of 2048 packets (which is not enough for a meaningful analysis). This means we cannot effectively measure the sender accuracy and can only compare the accuracy achieved by RUDE and KUTE at particular packet rates, presuming that the accuracy of the KUTE receiver is constant over time. We also estimate the standard error of the sender s_{sender} using the following equation:

$$s_{sender} = \sqrt{s_{total}^2 - s_{receiver}^2} \tag{1}$$

where s_{total} is the standard error using the sender and KUTE as receiver and $s_{receiver}$ is the standard error of the KUTE receiver assuming the standard error of the Smartbits is very small. We also assume that the sender and receiver process are completely independent. In all experiments the mean value of the inter-arrival times always had the expected value.

Figure 4 shows that the accuracy of KUTE is higher, even at much higher packet rates. Taking into account the inaccuracy of the receiver (see section IV.D), the KUTE sender is reasonably accurate, as the standard deviation increases only slightly when using the KUTE sender instead of the Smartbits. The standard error estimate for the KUTE sender is about 2us.

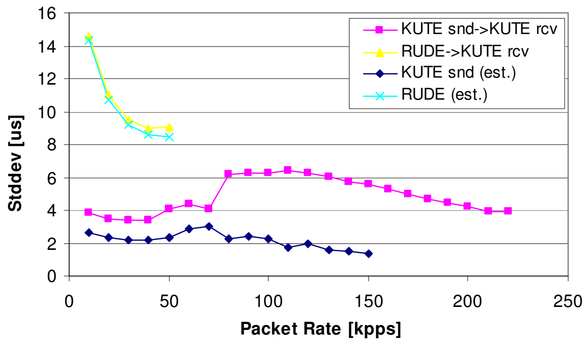

Figure 4: Sender inter-packet gap accuracy using KUTE as receiver

Figure 5 shows the inter-arrival time distributions measured by the KUTE receiver at 25kpps for all senders. The distribution generated by the KUTE sender is very close to that from the Smartbits. Again RUDE has a much wider distribution indicating a higher standard deviation.

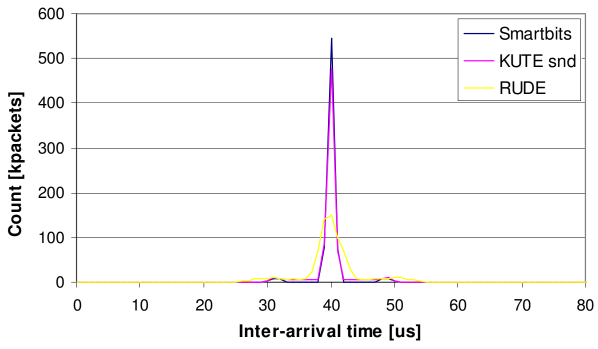

Figure 5: Inter-arrival distributions measured with KUTE for the different senders at 25kpps: Smartbits, KUTE and RUDE

Figure 6 shows the quotient of standard deviation and mean (relative error) for KUTE used as sender and receiver. The relative error increases with increasing packet rate and the difference between the Smartbits and the KUTE sender is fairly small. The estimated relative standard error of the KUTE sender has a similar value as the KUTE receiver until 70kpps. Then the sender relative error stays almost constant while the receiver relative error increases.

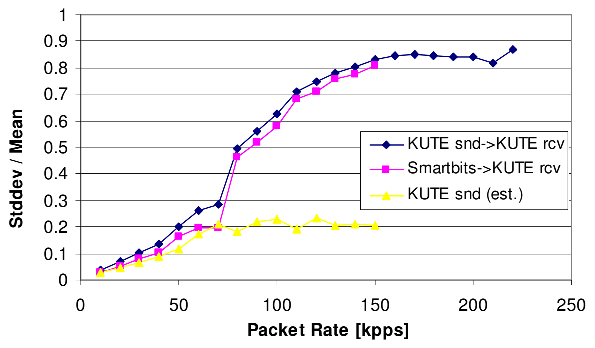

Figure 6: Standard deviation divided by mean using KUTE as receiver and Smartbits and KUTE as sender

V. Conclusions and Future Work

In this paper we presented a performance analysis of KUTE, a kernel-based tool for sending and receiving UDP packets. KUTE has been implemented to send and receive higher packet rates, especially when using high-speed network interfaces (e.g. Gigabit Ethernet), and to handle inter-packet times more accurately than traditional user space tools. Our evaluation shows that KUTE clearly outperforms a similar user space tool in all measurements.

In the future we plan to extend the KUTE implementation with a better interface towards user space, allowing runtime control. We also plan to extend the functionality of the sender by supporting more complex flow definitions and different inter-arrival time distributions. The sender should also optionally keep track of its accuracy, producing a histogram of the packet departure times. This would allow for better accuracy measurements without a Smartbits, but would also decrease the performance of the sender. It would be interesting to perform similar measurements running user space applications on real-time Linux (RTLinux). This combination could be an alternative to using kernel-based tools.

References

[1] RUDE/CRUDE: http://rude.sourceforge.net/ (as of November 2004)

[2] tcpdump: http://www.tcpdump.org (as of November 2004)

[3] CAIDA: http://www.caida.org/tools (as of November 2004)

[4] IP measurement: http://www.ip-measurement.org (as of November 2004)

[5] Traffic Generator Overview: http://www.grid.unina.it/software/ITG/link.php (as of November 2004)

[6] pktgen: http://www.kernel.org (as of November 2004)

[7] click project: http://www.pdos.lcs.mit.edu/click/ (as of November 2004)

[8] KUTE: http://caia.swin.edu.au/genius/tools/kute-1.0.tar.gz

[9] CPU cycle counter: e.g. http://www.scl.ameslab.gov/Projects/Rabbit/ (as of November 2004)

[10] MMAP libpcap: http://public.lanl.gov/cpw/ (as of November 2004)