Enabling Dynamic Network Processing with ClickOS

Mohamed Ahmed, Felipe Huici, and Armin Jahanpanah

NEC Europe Ltd.

1. INTRODUCTION

Software Defined Networking (SDN) is improving how flexible networks are by allowing their basic components (e.g., switches and routers) to be easily programmable. Most of the work so far has focused on network routing and on managing SDNs; less attention has been paid to the packet processing that is actually done in the network.

The original Internet architecture was based on the end-to-end principle, with packets being forwarded through the network largely unmodified. In the current Internet this model is largely a thing of the past: NATs and firewalls were the first commonly deployed devices that placed layer 4 (or higher) functionality in the middle of the network, not just at the endpoints, and devices such as DPI boxes, rate-limiters, transparent web proxies, application accelerators and many others have followed. In fact, a recent study shows that as many as 33% of paths tested keep state and perform some level of L4+ functionality [2].

What's clear is that network processing, as embodied by such middleboxes, forms a crucial part of today's networks. As a result, any sensible SDN should have mechanisms to not only program the network, but also the packet processing that takes place in it. Such a system should be able to easily cover a wide range of middlebox functionality and be extensible. In addition, it should have the ability to be quickly and dynamically instantiated so that when the network changes, the middlebox processing that takes place in it can change along with it.

Finally, a critical requirement is that the system provide isolation. In a SDN world where slices of networks are given to different entities and users, it becomes increasingly important that network processing from such users that happens to run on common hardware do not affect each other, both from a security and performance point of view.

Towards these requirements we introduce ClickOS, a minimalistic network operating system that runs on top of the Xen hypervisor [1] and that is based on the Click modular router software [3].

2. SYSTEM DESCRIPTION

Ideally we would like to have an OS that can do middlebox processing without all the overheads of a full-fledged OS like Linux. While several minimalistic OSes exist, their limited driver support would present a difficult obstacle for a network-focused system. ClickOS takes advantage of the best from two worlds: the small overheads of a minimalistic OS (mini-os, which comes with the Xen source code) and the driver support and isolation mechanisms provided by Xen. In the rest of this section we discuss ClickOS's design, including its control and data planes.

2.1 Control Plane

The system consists of a set of ClickOS virtual machines (vms), each composed of Click version 2.0.1 running on top of mini-os (figure 1). In addition, Xen's privileged domain, called dom0, contains the ClickOS CLI and the Xen store. The Xen store keeps control information about running vms, such as their ids and what virtual interfaces they have.

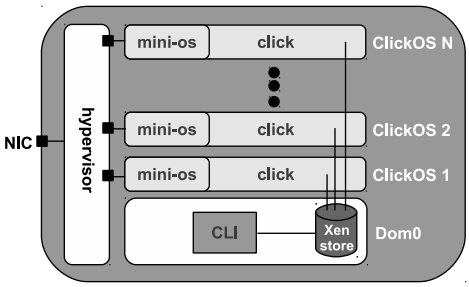

Figure 1: ClickOS Architecture Overview

In addition to these basic parts, ClickOS's control plane consists of two mechanisms. The first is the control thread, a mini-os thread that gets created at system start-up. This thread adds an entry to the Xen store and then watches for changes to it. When a Click configuration string is written to it, it takes care of creating and running a Click instance, where by instance we mean a running Click configuration such as, for example, a firewall or NAT. The second mechanism is a new Click element called *ClickOSControl* which takes care of reading and writing to element handlers from the CLI (an element handler is a Click element internal variable, e.g., a packet count for the element *AverageCounter*).

2.2 Data Plane

Xen has a split driver model, where a netback driver running in a driver domain (usually dom0) talks to hardware devices; and a netfront driver running in a guest domain

(e.g., ClickOS) talks to the netback driver via shared memory, more specifically a ring.

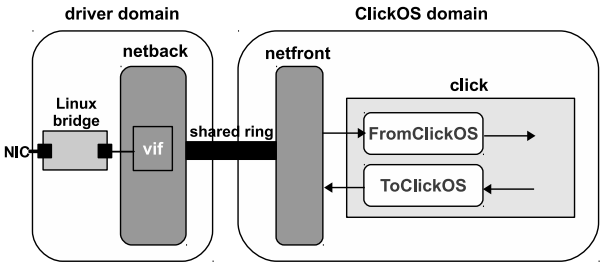

Figure 2: ClickOS Data Plane.

Under a typical Xen set-up, a network card (in our case an Intel X520-T2 dual-port 10Gb card using the ixgbe driver) is linked to a virtual network device called a *vif* via a regular Linux bridge (figure 2). When a packet is received, the Linux kernel hands it to the vif, which queues it at the netback driver. At a later point in time, one of the netback driver threads picks up the packet and puts it on the shared ring, notifying the netfront driver in the process.

In order to receive packets, our modified netfront driver (see section 3) has a dedicated mini-os thread per vif. This thread takes a packet from the shared ring and puts it in the *FromClickOS* element's queue (this is a new element we created that knows how to talk to the netfront driver).

In addition to these per-vif threads, ClickOS creates one mini-os thread for each Click instance. Such a Click thread schedules *FromClickOS* to run, taking the packet from its queue and sending it to the next element down the line. The transmit process is similar but simpler: when scheduled, the *ToClickOS* element takes a packet from an upstream element and sends it directly to the shared ring via the netfront's transmit function.

3. EVALUATION

Image Size: ClickOS is compiled with most of the available Click elements (221/267), the remaining ones requiring a file system to work.[1] The uncompressed ClickOS image is 12MB, the compressed one 2.9MB, and the virtual machine needs a minimum of 5MB to run. This shows ClickOS's small footprint: in a quick test we were able to have as many as 1,010 dummy virtual machines without network devices before errors (not related to memory exhaustion) occurred.

Start-up Times: ClickOS boots quickly. Creating a ClickOS virtual machine and starting a Click instance within in takes roughly 7.9 seconds. Instantiating a Click instance in an existing ClickOS vm takes only 1 second.

Migration Time: ClickOS's small image size and fast boot-time mean that migrating a virtual machine without much state (e.g., only a few forwarding rules for an IP router or a few firewall rules) takes about 7.5 seconds (0.5 to stop, 0.9 to copy, 6.1 to restart the vm).

Netfront Performance: Out of the box, the mini-os netfront driver performs rather poorly. To improve it, we introduce three mechanisms: (1) *poll*: we change the driver's receive function to poll for packets from a dedicated mini-os thread, rather than be interrupt driven; (2) *GRU*: we re-use the grants that receive buffers are given and keep them for the lifetime of the network device (a grant is Xen's way of allowing two domains to share memory); and (3) *FR*: we do fast re-fill of request entries in the shared ring, since the netback driver cannot send packets to the netfront unless requests are available on the shared ring.

[1]We are in the process of porting a simple file system to increase the number of compiled elements.

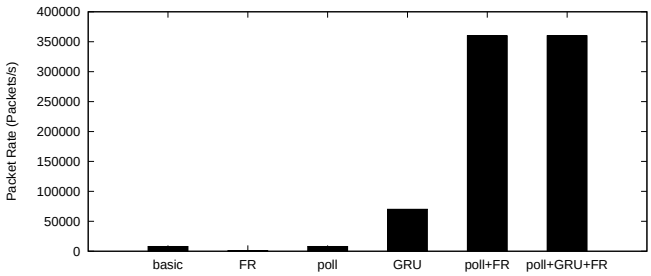

Figure 3: Netfront Performance.

Figure 3 shows netfront receive performance results for maximum-sized packets under different set-ups. As mentioned, without improvements ("basic") the netfront achieves only 8,000 pkt/s, which equates to 96Mb/s, or less than 1% of the 10Gb/s that our system's NIC can achieve. The next three bars are for rates when only one of the three mechanisms is used, and show that an important gain can be had by including only GRU (70,000 pkt/s). Using poll and FR yields the best performance, equivalent to when all three mechanisms are in place. We hypothesize that this is because at this point the bottleneck is somewhere else (possibly the netback driver), and so the gains that adding GRU gives go unnoticed. The final rate of about 360,000 packets per second, equivalent to 4.3Gb/s, represent a 45x increase with respect to the basic netfront driver; this is better than the 2.9Gb/s reported in [4] for a Linux vm.

4. CONCLUSION

We presented ClickOS, a tiny network operating system based on Xen and the Click modular router system. ClickOS's small size and quick start-up and migration times allow for fast and dynamic instantiation of network processing in programmable networks such as SDNs. Further, we have shown that ClickOS provides good performance, receiving packets at rates of 4.3Gb/s on a single ClickOS vm. As future work, we are partly through extending the netfront driver to support multiple vifs per ClickOS vm. We're further working to scale this performance with increasing number of ClickOS vms, as well as seeing what the performance is like when large number of vms are running in the system. We are also in the process of implementing middlebox functionality in ClickOS such as firewalls and carrier-grade NATs.

5. ACKNOWLEDGMENTS

We would like to thank Adam Greenhalgh for the original idea behind ClickOS.

6. REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. ACM SOSP, 2003.*

[2] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proc. ACM IMC*, 2011.

[3] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems, August 2000.*

[4] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner. Achieving 10 gb/s using safe and transparent network interface virtualization. In *Proc. ACM VEE, 2009*, VEE '09, 2009.