MiniStack: Operating System Support for Fast User-space Network Protocols

Michio Honda[*], Felipe Huici[*] and Luigi Rizzo[†]
NEC Europe Ltd.[*], Università di Pisa[†]
{michio.honda,felipe.huici}@neclab.eu, rizzo@iet.unipi.it

Motivation: Networking stacks have been implemented in the kernel since the inception of Unix to ensure good performance, security and isolation between user processes. As networks have gotten faster and faster, there has been a trend towards offloading certain tasks to hardware (e.g., checksumming and segmentation) in order to accelerate packet processing. However, this speed-up results in reduced flexibility: we are essentially embedding the functionality of today's protocols in hardware at the expense of future ones or even to the detriment of extensions to current ones [2]; this has the end result of making the network harder to evolve.

In this abstract we argue that instead of moving network functionality down to the kernel and hardware, it should be shifted *up*, into user-space. Such a change would make development and deployment of new protocols easier, and would allow optimizations of the network protocol suited to the specific needs of an application [1]. Indeed, we could envision several variations of a commonly-used protocol such as TCP to be running concurrently.

The question now is: is it possible to move the network stack into user-space while still achieving high throughput and retaining the necessary security and isolation guarantees provided by the kernel? Recent developments show that using the *netmap* API it is possible to achieve high performance when moving packets to and from a NIC and user-space, while removing a lot of common in-kernel overheads such as `skbuf/mbuf` management and system calls [3].

To confirm this, we generated a TCP flow on top of *netmap* on a system with a 10Gb Intel NIC and a Core i7 CPU. The set-up yielded line-rate for packets larger than 128-bytes, 7.84 Mp/s (out of 8.22) for 128-byte packets, and 11.24 Mp/s (out of 14.2) for 64-byte packets. These results show that we can saturate a 10Gb pipe from user-space without having to rely on hardware offloading.

In the rest of the abstract we describe *MiniStack*, a proof-of-concept system that allows applications to use fast, secure, user-level network stacks.

MiniStack Overview: Besides high performance, allowing unprivileged applications to run either their own or a general-purpose stack in user-space means that the operating system must isolate them from each other and prevent them from sending spurious packets out into the network. The *netmap* framework provides a simple memory-mapped I/O model between the NIC and user-space, meaning that applications can (1) snoop or overwrite each other's packet buffers and (2) spoof header fields such as the IP source address.

To address the first issue, we need separate packet buffers for each application. For this, MiniStack extends VALE [4], a kernel-level, Ethernet switch which exposes netmap-based virtual ports. MiniStack assigns one or more VALE ports to an application, and forwards (copies) packets between the packet buffer of the application's port and the port that the NIC is connected to. As an initial test, we measured the throughput between a single VALE application port and a NIC for both outgoing and incoming packets, and achieved 13.7 Mp/s (outgoing) and 13.6 Mp/s (incoming) for 64-byte packets, and line-rate for larger packets.

Regarding the second problem, MiniStack once again extends VALE's functionality in order to implement a packet filter. Applications register the desired source IP address (which must match one of the IP addresses on the system), protocol and port numbers. Note that this assumes IP (both IPv4 and IPv6 are supported), and that the protocols running on top of it use port numbers; this is likely to be true in most cases (e.g., TCP, UDP, SCTP, DCCP, etc), but if needed different mechanisms can be implemented.

Once registered, the application sends packets and MiniStack verifies that the fields in them agree with those that the application had registered, otherwise drops them. On the receive-side, MiniStack implements a demultiplexing mechanism, delivering the packets to the "right" application/port based on the registered port numbers.

With all of this in place, MiniStack achieves 9.69 Mp/s for 64-byte packets and line-rate for most other packets sizes on receive; and 7.93 Mp/s for 64-byte packets and line-rate for packets larger than 256 bytes on transmit. We will accompany the poster with a demo.

References

[1] M. Fiuczynski et al. An Extensible Protocol Architecture for Application-Specific Networking. In *Proc. USENIX ATC*, 1996.

[2] M. Honda et al. Is it Still Possible to Extend TCP? In *Proc. ACM IMC*, pages 181–192, 2011.

[3] L. Rizzo. Revisiting Network I/O APIs: The netmap Framework. *ACM Queue*, 10(1):30–39, Jan. 2012.

[4] VALE, a switched ethernet for virtual machines. L. Rizzo URL `http://info.iet.unipi.it/~luigi/vale/`.