Linux as a Switch Operating System: Five Lessons Learned

Kenneth Duda

Arista spent the last nine years building a switch operating system based on Linux, including nearly six years of field experience. Here are five lessons we learned along the way.

Lesson 1. It's okay to leave the door unlocked.

We weren't the first to put Linux on a switch, but we were the first to leave it open. By "open", I mean that an administrator can get a root shell on the switch, with access to all of the standard Linux tools from awk to zip. A few naysayers believed that customers would run with scissors, hurt themselves, and blame us, yet this has literally never happened, because (a) Linux provides such good isolation between processes, (b) there is a high degree of Linux expertise among customers who exploit Arista's open Linux, and (c) the administrator retains control (through AAA) over who can access these facilities.

Lesson 2. Preserve the integrity of the Linux core.

We went to a fair amount of trouble to preserve the standard Linux networking abstractions so that standard tools, such as ifconfig, ping, tcpdump, and sshd, would work out of the box. Am I glad we did! It has made it much easier for us to get things like Microsoft's nanowbem or VMware's ovsdb running on our switch. It has also enabled customers to run all sorts of software that we hadn't imagined running on a switch: tftpd, named, dhcpd, sendmail, nfsd/portmap, collectd, chef, puppet, cfengine, ansible, mrtg, nagios, ganglia… the list goes on and on. By preserving Linux interfaces, we preserved our connection to the Linux community, capturing the benefits of community development for our customers.

Lesson 3. Focus on state, not messages.

Aggregating switching state in a database process (Sysdb) has worked out very well for us. It makes the system more resilient, because all processes restart statefully, and the restart code path is the same as the initial process-start code, meaning it actually works. Other Linux-based switch operating systems are based on message passing, where processes coordinate by sending each other messages (add route, delete ACL entry, etc). The problem is in recovery: when a process fails and restarts, getting it caught up with everyone else is a non-trivial exercise, and the special recovery code may not get a lot of testing.

Lesson 4. Keep your hands out of the kernel.

Unlike those who propose that the Linux kernel should be where all the state resides, we believe that the kernel should hold only the minimum state required for the software control-frame transmit path. Why? First, that's all you actually need — software forwarding is uninteresting in the modern data center. Second, if you don't touch it, you don't break it, and keeping our hands out of the kernel has led to world-class stability for our customers. Third, if you make changes, you either wind up with a kernel fork (which makes it harder to keep up with the community) or you have to convince the Linux kernel maintainers to accept your changes, which isn't always easy especially if those changes are specific to the environment of a switch/router. We expect those who take a kernel-centric approach will question that decision when they take a hard look at what is required to implement features like private VLANs, storm control, VRF, VLAN translation, q-in-q, etc.

Lesson 5. Provide familiar interfaces to ease adoption.

Learning the lesson from others' attempts at integrating Linux into switching, we wrote a standard CLI that runs on top of Sysdb. It retains the familiar management commands, keeping configuration in a familiar startup-config file. Of course, you don't have to use our CLI if you don't want to; the native Linux interfaces are all there, and about 20% of our customers adopt a pure-Linux approach. The remaining 80% tell us they appreciate the way they can leverage their deep IOS experience, as they can easily upgrade an aging Catalyst infrastructure to Arista. Plus, by providing APIs around familiar networking abstractions, we enable automation in a more traditional networking environment.

In summary, it is possible to be both a true modern networking operating system, and also run true Linux. The best of both worlds is achievable with proper, disciplined engineering.