FreeBSD® JOURNAL

January/February 2021

Case Studies:

- **Tarsnap**
- **Netflix**
- **BALLY WULFF**

Also:

FreeBSD for the Writing Scholar

New Faces of FreeBSD

Practical Ports

TrueNAS® M-SERIES
Powerfully Scalable Enterprise Storage

OPEN STORAGE FOR ENTERPRISE WORKLOADS

UTILIZES FLASH-OPTIMIZED ZFS TECHNOLOGY
IDEAL FOR LATENCY-SENSITIVE AND BUSINESS-CRITICAL VIRTUAL MACHINES AND PHYSICAL WORKLOADS.

- PERFORMANCE AND SCALE WITHOUT COMPROMISE
- INTELLIGENT STORAGE OPTIMIZATION
- SELF-HEALING DATA PROTECTION
- UNLIMITED SNAPSHOTS AND REPLICATION

Contact iXsystems to Learn More about what TrueNAS® can do for your business!

iXsystems.com/TrueNAS | (855) GREP-4-iX

iXsystems™

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Foundation

Welcome 2021!

Like many of you, the Foundation team was pretty happy to see that clock strike Midnight on January 1, 2021. Last year will definitely go down as one of the most memorable in the Foundation's 20-year history. However, we're happy to report it ended on a positive note. Thanks to your generous support, the Foundation was able to raise enough funds to not only continue our efforts to support the FreeBSD Project, but also expand our software development and advocacy efforts in 2021. From new hires in the software development group to increased online content and greater focus on FreeBSD in education, we're looking forward to all of the ways we can continue to help ensure FreeBSD is the high-performing, stable and secure operating system you've come to rely on.

Speaking of relying on FreeBSD, this issue of the Journal focuses on FreeBSD Case Studies. There's no better way to showcase the benefits of FreeBSD, than by hearing it directly from those using it to great success. Take a minute to see why companies have made FreeBSD their OS of choice, and be sure to share the always-free issue with your colleagues and friends.

Thank you again to readers for your continued support of the FreeBSD Project and Foundation. We wish you a very Happy New Year and look forward to all we are going to accomplish together in 2021.

On behalf of the FreeBSD Foundation,
Deb Goodkin
FreeBSD Foundation Executive Director

FreeBSD® JOURNAL
January/February 2021

WeGet letters

by Michael W Lucas

Oh Generous, Grandiloquent, Gratuitous *FreeBSD Journal* Letters-Answerer:

FreeBSD 13 comes out any time now. It has a whole bunch of features I'm eager to get my hands on, but I'm leery of a brand-new release. Any advice on when I should upgrade?

And do they really pay you in gelato?

Thanks,

—Doesn't Need Outages

DNO,

I know this story.

Early in your career you fell under the tutelage of a grizzled sysadmin, the sort who lost an eye in the Unix Wars, detonated a lobe of his liver bootstrapping the K&R C compiler, and kept a copy of the Alpha boot loader encoded in the knots of his flowing gray beard. You asked him this same question about some other new release. He picked up the *Free the Berkeley 4.4* coffee mug where he kept the knucklebones of the last Ultrix salesman who dared radiate body heat in his artisanally cooled datacenter, gave it a good shake, and cast the bones across his desk to read the wisdom therein. Between the roar of the racked servers all around and the way he'd wrecked his vocal cords screaming at the University of Minnesota's Gopher developers over changing their server to a paid license, you had to listen carefully to sieve his hoarsely whispered wisdom from the noise.

"Never, *never* install a .0 release."

That's the sort of thing that makes quite an impression on a young sysadmin.

In Old One-Eye's defense, that was unsurpassed wisdom in the Dialup Age. The fastest way to download an operating system release was to get a backup tape by mail order. Critical patches were distributed by Usenet, if you were lucky enough to have an account on a site with a mighty 1.544-megabit uplink. Inexpensive servers cost several thousand dollars, or a transplant-ready liver if you could find an insufficiently cautious salesman who hadn't already wrecked theirs.

It's a different world now. For one thing, we have salespeople. And they've all been warned about the liver thing.

If you're eyeing a .0 release today—you're already too late.

Modern operating systems are public, exactly like a sleazy Hollywood star's collection of intimate infections. The time to find problems is before the release. I don't care what flavor of Unix you run, they're public. Even closed-source Unix developers give their customers access to pre-release media, though I'm certain I don't know why you'd want to grant them more help than

your outrageous license fees. The developers have asked, begged, cajoled, pleaded, and threatened their user base to test release candidates, in-progress versions, snapshots, and patches for months or years. And here you are, asking if you can trust the finished product?

You selfish dweeb.

Grab the most recent snapshot, release candidate, or whatever's the latest and greatest, and try it in your environment. Configure it with all the debugging and prepare for kernel dumps. Test your applications under load. Tell the developers what worked and what doesn't.

If you're reading this right after 13.0 came out and are all sorts of relieved that you don't have to do this work, guess what? A pre-pre-release 14.0 is available this very moment! Or maybe all the good topics for PhD theses in Irrelevant American Authors have been taken, but desperation has driven you to delve into the moribund, unrecognizable text archives of a long-telepathic Journal to identify the moment when my descent into ferality crossed into forthright malignance, and release Ax67.πr^2 is now in development. Whatever the case, there's a forthcoming release available for testing.

No, I'm not saying that you should deploy the release candidates and development versions on every host across your environment. People should, but anyone asking this question shouldn't. You're not equipped.

Testing development releases requires not only sysadmin skills, but sysadmin practices. What's the difference? *Skill* means you know how to do the things you should do. Practice means you perform those things. You need backups. You need to know that those backups can be restored. You must not only know how to submit bug reports, you need to be comfortable submitting them. The whole point of running one of these early versions is to report on bugs.

For your own sanity, you need to deploy development versions intelligently.

Don't slam development releases onto every web server in the cluster. Pick one or two. Put them in the load balancing pool. See what happens. Compare responsiveness under similar loads. Configure them to automatically dump and reboot in case they panic. If they're running ZFS, keep known, good boot environments on hand. While "the kernel panics every one thousand sixty-three seconds, here's the text dump" is eminently valuable, there's no need to live with that once you've identified the problem.

If you keep working it, you'll eventually learn that you can run development versions everywhere. People do. You can become good enough to join them.

Yes, this requires allocating time and hardware. That's cheap. If you doubt me, go price sufficient Oracle Solaris licenses and servers to host your environment. In a big enough company, you can hire official testers and still save enough to feed the staff pizza and beer every day for lunch. Just be sure you give the Oracle rep a burner phone number and an email address in a burner domain name, because like the Terminator, they will never stop, never show mercy or pity, and never tire until they own your scrawny, underfed soul.

You know what improves your soul? What makes your soul blossom? What develops greatness of character and mind? What sharpens your sysadmin chops until none can stand against you?

Testing development versions of the software you depend on.

Deploying them. Using them day-to-day. Providing feedback for the developers. Bug reports are great, but so is "I'm running the latest in production to serve four and a half trajillion HamsterSoft queries a second, and it's going great." Negative results are still results.

You know what else improves your soul?

Paying your Letters Column guy his gelato. I had expected George to settle up at BSDCan in spring 2020, but he never showed up. Maybe he'll be there in 2021. If he doesn't come through, though, I'll have to write it off as a bad debt.

These columns might get a little cranky if that happens. Consider yourself warned.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS's most recent books include *SNMP Mastery*, *Cash Flow for Creators*, and *Drinking Heavy Water*, plus a bunch more at https://mwl.io. Under no circumstances is he allowed near users.

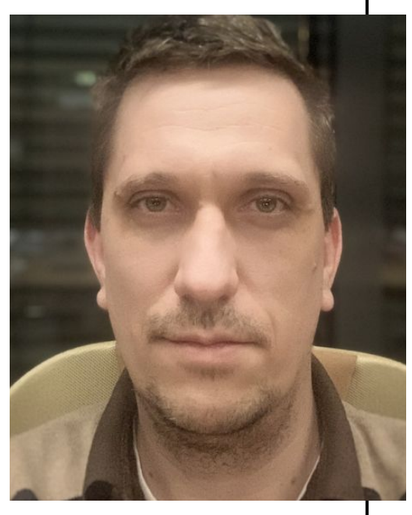
new faces of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community.

In this installment, the spotlight is on Juraj Lutter, who received his ports bit in December 2020.

Tell us a bit about yourself, your background, and your interests.



• Lutter: I was born in the former Czechoslovakia. My father is an electronics enthusiast and bought our first home computer (Sinclair ZX81), which was soon replaced by a more modern successor (ZX Spectrum). So as a boy, I got to know BASIC and later also the Z80 assembler. After the social changes that took place in our country in 1989, 16-bit computers, which were not available to private individuals until then, became more widespread in our country. This allowed me to get acquainted with DOS, PC hardware, Turbo Pascal, Turbo C, Turbo Assembler, and other tools.

I have been in the IT industry since the late 1990s. First as a PC service technician, then as a system administrator at an ISP and as a freelancer, I have been working in the system integration industry. In addition to FreeBSD, where I maintain various ports, I am also interested in other open source technologies such as SmartOS and illumos, various infrastructure programs (BIND, PowerDNS, Zabbix), databases (PostgreSQL), computer networks (switching, routing, firewalling), and data storage (either monolithic storage or ZFS). I also have a commit bit to pkgsrc for SmartOS and NetBSD.

I collect old computers (especially 8-bit computers from Sinclair) and also devote some of my time to electrical engineering and electronics.

I live in our capital, Bratislava, and have two children—an 8-year-old son who is starting to discover Python and a 10-year-old daughter who is more sympathetic to the LUA language.

How did you first learn about FreeBSD and what about FreeBSD interested you?

• Lutter: I first encountered UNIX around 1995, specifically SCO Unix 3.2, where one of my first tasks was to configure UUCP over an X.25 line. Shortly afterward, I received a 4-CD set from Walnut Creek CD-ROM that included among other things one of the first releases of Slackware Linux. And since I was already in contact with UNIX (SCO), I was interested in Linux because it was easy to install and use when compared to SCO UNIX. In my fourth year of high school in the fall of 1996, a classmate mentioned FreeBSD to me, and shortly thereafter I received my first user account on a FreeBSD server, which I used mainly for e-mail. Subsequently, around 1999, I started working as a system administrator in the Slovak branch of Nextra (Telenor Internet), where FreeBSD was deployed on several dozen

servers on i386 and Digital Alpha platforms and where we also had a Slovak FreeBSD mirror (www, ftp, CVSup).

FreeBSD fascinated me because, compared to Linux, it is a compact system, where the kernel and userland are in harmony and symbiosis, and where ports make it very easy to maintain software packages for different servers (for example, using poudriere). I also still maintain our local FreeBSD mirror.

How did you end up becoming a committer?

• Lutter: The more I used FreeBSD, the more I ran into bugs in individual ports and the base OS. Over time (since 2004), I started to open bug reports and contribute patches. It wasn't until the end of last year that I spoke to Sergey A. Osokin, to whom I mentioned that sometimes the bug reports I open remain unresolved for a long time and that I would also like to contribute to the development. Sergei suggested that he ask about it. And one day, I received mail in which René Ladan informed me that I had become part of the FreeBSD development community as a ports committer. I was really very pleased and honored. I want to thank Sergey for the opportunity join this amazing community and also Steve Wills for his help and answers to my curious questions.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

• Lutter: Since I've been contributing patches for a long time, I've seen how committers and the commit process works and how code reviews work (everyone who contributes should learn phabricator). And that's why my impressions are still positive. Thanks to constructive discussion, sharp edges can always be smoothed out and there are many things to learn from the more experienced committers and vice versa.

It is also a good idea to read (and eventually remember) the information in the Porter's Handbook and Committer's Guide. For example, I found a lot of information there about Subversion and the processes built on it (like MFH and the like).

DRU LAVIGNE is the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

CASE STUDY

Tarsnap's FreeBSD Cluster

BY COLIN PERCIVAL

Tarsnap is an online backup service with an emphasis on security—indeed, when I started the company back in 2006, I quickly settled on the tagline "Online backups for the truly paranoid" to reflect the fact that as a cryptographer and FreeBSD Security Officer, my aim was to provide a service which was secure enough that I would trust it with my own secrets. Tarsnap is also one of the leading reasons that FreeBSD is available in Amazon EC2: Tarsnap needed to run in EC2 (among other things, so that it had cheap and fast access to the Amazon S3 storage service), but I needed an operating system which I trusted and knew I could administer easily—in other words, Tarsnap needed FreeBSD in EC2, and I scratched my itch.

Customized AMIs

While all of the EC2 instances Tarsnap uses run FreeBSD, none are ever launched from the "stock" FreeBSD Amazon Machine Images (AMIs) which the FreeBSD project publishes. Instead, I make use of a tool I created five years ago to build lightly customized FreeBSD images: An "AMI Builder" AMI, available for FreeBSD 12.2-RELEASE as `ami-085ee41974babf1f1` in the us-east-1 EC2 region. These AMI Builders are not currently provided by the FreeBSD project but are instead something I build and publish myself after each release; at some point I hope to integrate these into the builds performed by the release engineering team.

To create a "Tarsnap FreeBSD 12.2-RELEASE" image, I start by launching the aforementioned AMI Builder—and then I wait, roughly 20 minutes, while the virtual machine spins up and installs (stock) FreeBSD 12.2-RELEASE onto its virtual disk. This disk is then mounted on `/mnt/` while the FreeBSD system mounted at / runs from a memory disk and starts an `sshd` process.

Once I can SSH into the AMI Builder (like other FreeBSD images in EC2, using the SSH key I provided to EC2 and the user name `ec2-user`), I set to work with some standard configuration which I want on all of Tarsnap's systems:

- I build and install some packages from the FreeBSD ports tree: pkg, djbdns, qmail, spiped, and tarsnap.
- I enable some daemons I want (svscan and spiped), disable other code I don't want (sendmail and firstboot pkg installation), and lock down some settings (disabling network listening in syslogd and restricting sshd to IPv4) in `/etc/rc.conf`.
- I instruct pkg to use packages I build myself instead of the packages distributed by the FreeBSD project, by creating configuration files in `/usr/local/etc/pkg/repos/`.
- I add cron jobs to run `freebsd-update cron` and `pkg upgrade -qn` every morning—I don't want updates installed automatically, but I definitely want to get an email when they're available.
- I set up djbdns to provide a local DNS cache and set `resolv.conf` to point at it.

- I set up qmail to send outgoing email via my mail server.
- I set up spiped to create secure connections to my mail and package servers, and also to wrap incoming SSH connections.

Finally, after performing all the configuration I want on `/mnt/`, I unmount the disk and ask EC2 to "create an AMI from the running EC2 instance". Despite the description of this EC2 API call, the AMI created does not reflect what was running, but rather the current state on disk—in other words, it ignores the FreeBSD system running out of a memory disk and creates an AMI corresponding to the configuration I performed on the filesystem mounted at `/mnt/`.

Now I have a configured "Tarsnap FreeBSD" image from which I can launch instances with all of my preferred defaults set up, and there's one last step: I ask EC2 to copy this AMI from the us-east-1 region to the us-west-2 region. While almost all of Tarsnap's servers run in us-east-1 (which was the only EC2 region when I launched Tarsnap) I do have one system in us-west-2: A monitoring system which alerts me if anything breaks.

pkg Builder

The FreeBSD Project provides binary packages for software in the ports tree, and most users will want to make use of those rather than building from source. For Tarsnap's servers, I do my own builds, for two principal reasons:

- In some cases, I want non-default port options.
- As a FreeBSD developer, sometimes I commit updates and want to use them immediately, rather than waiting for the next scheduled package set.

It's possible that at some point in the future neither of these will apply—it may be that the ports where I set non-default options will gain "flavors" which provide what I need, and it may be that the FreeBSD Project will someday perform shockingly fast package builds every time a port is updated (but considering the rate at which new compilers are released with ever-worsening performance, this seems unlikely). In the meantime, running my own package builds is easy and convenient.

Tarsnap's "cluster" includes two very lightly used web servers.

I use poudriere for this purpose, and the process of setting up the builds is very easy: After installing poudriere, simply `poudriere ports -c` and `poudriere jail -c -j JAILNAME -v 12.2-RELEASE`. After that, I set a few options in `/usr/local/etc/poudriere.d/make.conf`, set a cron job to run `poudriere bulk -f /root/pkgs-wanted` (where I have a list of packages I want), and use lighttpd to serve up the resulting packages.

Building a complete set of the packages I use on Tarsnap's servers takes about 2 hours on the $15/month "t3.small" EC2 instance I use, but most package build runs complete far faster than that, thanks to poudriere operating incrementally and not recompiling unchanged packages. Indeed, I would use an even smaller EC2 instance but for one detail: Some of the C++ compiles fail on instances with only 1 GB of RAM.

Web Servers

Tarsnap's "cluster" includes two very lightly used web servers, running on EC2 "t3.nano" instances. In an era of web frameworks and rich web applications making javascript function calls, the Tarsnap website is perhaps exceptional mainly for its archaic design: The public portion of the

website is entirely static HTML—hand-written aside from a shell script which wraps content with a header and navigation section—and the account creation and management code consists of a handful of CGI scripts… written in C. While CGI scripts have inherent performance problems—forking a process is far more expensive than running code within the context of the web server or forwarding requests to another long-running daemon—as far as Tarsnap is concerned, I would love to be in the position of experiencing performance problems due to an excess of customers using the website.

There are a few slightly more modern aspects of the web servers, however: TLS traffic is unwrapped using hitch, in order to keep the cryptographic code segregated from the web server code; TLS certificates are obtained using Let's Encrypt and certbot; and TLS private keys are kept in an Amazon Elastic File System (aka. NFS) filesystem. While the use of hitch, Let's Encrypt, and certbot are very common, the use of Amazon EFS probably deserves some explanation.

The use of Amazon EFS solves a bootstrapping problem: I obtain TLS certificates via the "webroot" mechanism, wherein a certificate issuing request is authorized by placing files into a `/.well-known/acme-challenge` directory on the website. This only works once traffic for the website is directed to the server in question—but I don't want to direct traffic to a new host until it is ready to respond to HTTPS requests. Storing private keys in a manner which survives the replacement of a web server instance solves this problem.

Now, NFS is not known for having a stellar track record with regard to security, and operates in plaintext, neither of which seems ideal for cryptographic material—but the fact is that the guarantees made by TLS are now sufficiently weak that no security is lost here. If an attacker can intercept traffic on the Amazon EC2 network, they can trick Let's Encrypt into issuing them a new certificate allowing them to impersonate Tarsnap's web servers—so having another attack which would rely on intercepting traffic on the Amazon EC2 network does nothing to extend their capabilities.

Mailserver

All of Tarsnap's email is routed through a single mailserver. This includes:

- "Human" emails sent between me and the outside world.
- "Logging" emails generated by cron jobs (which land in my inbox).
- Transactional emails generated when users sign up for Tarsnap, confirm their email addresses, make payments, or need to be warned that their (prepaid) accounts are running out of money.
- Public mailing lists, both for Tarsnap announcements and discussions, and for open source software which originated from Tarsnap.

For historical reasons, the Tarsnap mail server runs qmail; specifically, "this is what I started with when I configured my first FreeBSD server, nearly 20 years ago." If I were starting from scratch, I would probably use postfix, but in the long-standing tradition of sysadmins everywhere, as long as it isn't broken, I'm unlikely to fix it.

Email arrives at the mailserver via two routes: Connecting to port 25 on the external network interface and connecting to spiped via port 8025—which then arrives at qmail via port 25 on the loopback interface. Using spiped in this manner not only secures "internal" network traffic, but also neatly solves the question of email relaying: Any email which arrives at qmail via the loopback interface can be safely relayed to other domains.

Since I use qmail, I naturally use Dan Bernstein's ezmlm (and its extension, ezmlm-idx) to manage Tarsnap's mailing lists. The "tarsnap announcements" list is moderated, but the rest are open to postings from any subscriber; fortunately, I have had no problems with trolling, and thus far

spambots don't seem to be smart enough to subscribe to mailing lists before attempting to send email through them.

Tarsnap's mailing lists have archives accessible via HTTP/HTTPS. I use mhonarc to take the mailing list posts from ezmlm-idx and convert them into HTML format; lighttpd to serve them up to the world; and hitch to add a layer of TLS for those who want it. It's hard to imagine what security is added by TLS here: The mailing list posts are public, and the number of bytes transferred is enough to uniquely identify the page being downloaded; nonetheless, some people strongly prefer to use TLS even when it serves no purpose.

Finally, outgoing email is dispatched via a shell script which runs one of two "qmail-remote" programs: Either the original qmail-remote, which sends email directly via SMTP, or a "qmail-remote-ses" I wrote which sends email via Amazon's Simple Email Service. Unfortunately delivering email into inboxes is increasingly difficult, so for transactional emails I hand the job over to Amazon; at a cost of $0.10 per 1000 outgoing emails, Amazon doesn't need to be very much better at delivering email to pay for itself when it comes to customers signing up to give me money. On the other hand, people who have overly restrictive mail servers are unlikely to subscribe to Tarsnap's mailing lists in the first place—so I have no problem with relying on qmail and direct SMTP for that email traffic.

Monitoring

As mentioned earlier, while most of Tarsnap's systems are in Amazon's us-east-1 region, I have a monitoring system in the us-west-2 region. There are two reasons for having this instance in a different region from everything it is monitoring: First, to allow it to detect outages related to the AWS region's external network connectivity; and second, to minimize the likelihood that a failure disables both Tarsnap's systems and the monitoring simultaneously. Furthermore, even if an outage does knock two AWS regions offline simultaneously, it's (a) unlikely that there's anything I could do to respond to it, and (b) very likely that the world is facing far more important problems than Tarsnap being offline.

I use a handful of simple shell scripts to perform a range of monitoring tasks.

Rather than using any of the widely used monitoring frameworks, I use a handful of simple shell scripts to perform a range of monitoring tasks (pinging servers, fetching web pages, performing Tarsnap backups) from cron jobs; these each emit a state ("GOOD", "FAILED", or "TIMED OUT"), and another script uses Twilio to send SMS messages and make phone calls. The combination of a small number of notifications and Twilio's usage-based pricing makes this extremely affordable—indeed, most of the cost is the $1/month fee to rent a phone number, which is required in order to send SMS messages.

Holding Everything Together: spiped

The astute reader will have noticed that I've mentioned spiped a few times but without giving many details. Since this is a little-used tool—and one I wrote myself specifically for securing connections to and between Tarsnap systems—I think it deserves special attention.

The spiped utility, at its core, is a tool for connecting one socket address to another socket address in a cryptographically secure manner. In this sense it can be seen as a replacement for stunnel or "ssh -L" port forwarding; but whereas stunnel relies on the overly complex TLS protocol and

certificates, and ssh uses a persistent TCP connection over which tunneled connections are multiplexed, spiped uses a pre-shared secret and opens a new connection for each connection being relayed—making it much simpler and giving it the same behavior as TCP with regard to network failures. At present, spiped supports TCP sockets over IPv4 and IPv6, as well as local ("UNIX") sockets.

The name's origin, "secure pipe daemon," may give another hint to the intent, however: Whereas the introduction of pipes to UNIX in 1973 led to an explosion of functionality as utilities were combined in various ways, I wanted to develop software which used a multitude of daemons—and to be able to connect them without regard for whether they would end up running on the same host or across insecure networks.

As mentioned above, I use spiped to secure SMTP connections to Tarsnap's mailserver; I also use it to secure POP3 connections, which has the benefit of allowing me to use POP3 in its simplest and least secure mode—no TLS and plaintext passwords—while retaining the necessary security. (Hey, it works, ok? I've been meaning to switch over to IMAP for decades.)

Similarly, connections to the pkg builder are protected by spiped; while the contents of the packages are not sensitive (they all come directly from the FreeBSD ports tree) this allows me to ensure package integrity without needing the more heavy-weight option of using poudriere to sign packages and may also protect the pkg builder somewhat in the (unlikely) event that a vulnerability is found in lighttpd.

Finally, I use spiped to protect all of my SSH connections—while all of the systems I use have sshd enabled in order to allow me to easily administer them, port TCP/22 is blocked and the only way to reach sshd is via an spiped process which doesn't allow any traffic through until the incoming connection has been cryptographically authenticated.

Legacy Systems

Of course, in addition to the aforementioned infrastructure, there's the Tarsnap backup service itself. This runs on what one might call "legacy" systems: Since it is directly responsible for ensuring the safety of customer data—not to mention bringing in revenue—I allow the backup service to lag behind other systems (except where security is concerned, naturally). At present it uses an older version of FreeBSD on an older EC2 instance type—a fact which protected it from the stability problems in the (recently added) ENA network interface driver which were corrected in the FreeBSD-EN-20:11.ena Errata Notice—and the only third-party packages installed are those which I have in my "Tarsnap FreeBSD AMI"—i.e. those needed to allow me to securely connect for administration purposes, and those used to send outgoing email. Aside from those, the only code running is Tarsnap's proprietary code—a few daemons, plus cron jobs for internal performance monitoring and nightly billing purposes.

Over time I expect to use more open-source code in the backup service.

Over time I expect to use more open-source code in the backup service–or perhaps I should say, I have released open source code which I expect to be using in the backup service in the future. I designed the kivaloo data store around Tarsnap's needs for metadata storage, and it only made sense to release the code first and use it later. Tarsnap stores customer data in Amazon S3, but it's necessary to keep track of where each data block was stored in order to be able to retrieve it on demand. This leads to a data store usage pattern—tables with keys and values of roughly 40 bytes each—for which most data stores are poorly optimized.

CASE STUDY

Future Directions

I can't conclude without mentioning something Tarsnap isn't using yet, but which I'd like to use: "Graviton 2" arm64 EC2 instances. In all of my testing to date these have performed extremely well—and they're roughly 40% less expensive than the x86 EC2 instances currently used by Tarsnap.

Naturally, while FreeBSD images are available for arm64 EC2 instances, Tarsnap won't be making use of them (aside from development and testing) until the arm64 architecture is fully supported by the FreeBSD Project—including having binary updates available via FreeBSD Update. I understand that this is likely to arrive in time for FreeBSD 13.0; so, by this time next year I may have replaced many systems with new arm64 instances. (The core backup service, of course, will continue to run on x86 for a while longer.) Only time will tell, but it's safe to say that this is an area where I'm very excited about the future.

COLIN PERCIVAL has been a FreeBSD developer since 2004 and was the project's Security Officer from 2005 to 2012. In 2006, he founded the Tarsnap online backup service, which he continues to run. In 2019, in recognition of his work bringing FreeBSD to EC2, he was named an Amazon Web Services Hero.

CASE STUDY

BALLY WULFF

BY MATEUSZ PIOTROWSKI

BALLY WULFF Games & Entertainment GmbH is a prominent German company in the entertainment electronics segment that develops, produces and sells cash gaming machines. With its headquarters located in Berlin, BALLY WULFF operates not only in Germany, but also in Spain, and currently employs around 300 people.

Since the early 2000s, FreeBSD has been the platform of choice for BALLY WULFF products. Thanks to its invaluable stability and consistency, the system engineering team is able to meet the ever-changing market demand whether that is for small, disk-space footprint, higher security measures, or better graphics. Oftentimes, the team contributes code and documentation patches back to the community. In addition, many BALLY WULFF employees have served as FreeBSD committers.

BALLY WULFF is well known for its development process happening entirely in-house. The final products are the collective work of various BALLY WULFF teams. Collaboration among product designers, hardware engineers and game developers is the name of the game. Everything from the design of the machines, hardware integration and game development, to the final production and assembly of the machines takes place in-house. FreeBSD is at the heart of it all not only as the operating system in the final gaming machine, but also in the development workstations and production appliances.

This short case study provides a deeper insight into BALLY WULFF system engineering team's goals and explains how FreeBSD helps to achieve them.

FreeBSD is at the heart of it all not only as the operating system in the final gaming machine, but also in the development workstations and production appliances.

Goal 1: Limiting the Disk Footprint of the OS

A BALLY WULFF gaming machine can be thought of as a huge video game console. What may come as a surprise in the era of IoT is that it is not connected to the Internet, but instead ships with all the software, games and their assets preinstalled. Once it leaves the production site, the only way to change the software on a machine is via a manual update procedure that involves physical storage media like USB sticks. Although no longer a pressing issue, the older generations of gaming machines posed an interesting challenge for the system engineers. Disk quotas for each team had to be carefully balanced so that every game and administrative tool received its fair share of a disk. However, the disks always turned out to be just a little bit too small to fit all the desired data. As a result, the operating system

had to be stripped of all unnecessary bits. FreeBSD, like other, well-designed software projects, provides a great number of build options capable of excluding everything but essentials from being compiled.

Unfortunately, the standard build options were not enough. It turned out that in order to achieve the desired disk footprint, the system engineering team had to gain control of a greater granularity over the build process. Luckily, the FreeBSD build system has been engineered, maintained, and constantly improved with customizability and stability in mind. It is by design that downstream consumers (and appliance vendors in particular) are able to look under the hood of the build system, modify it as needed, and expect only a minimal maintenance overhead caused by the local changes to the source tree.

BALLY WULFF has maintained an internal patch set for the FreeBSD build system to exclude non-essential files from final OS images. This appliance-specific patch set has naturally fit into the build infrastructure and does not feel like an external add-on bolted on to an existing environment. At the same time, it has not caused a significant maintenance burden to the system engineering team. As a result, the disk footprint of the OS has been limited to a minimum, leaving more disk space for games—a real value to customers.

Goal 2: Shipping Modified Packages

The FreeBSD Ports Collection has proven to be an invaluable asset to BALLY WULFF over the years. It offers a standardized and expandable way of customizing and adding additional software to the OS. In fact, it is so straightforward that creating customized packages is one of the first things new FreeBSD users learn about.

FreeBSD ports developers make sure that the ports framework evolves steadily and stays backward-compatible for many years. So even though the FreeBSD Project has already switched to poudriere within its packaging infrastructure, users like BALLY WULFF can still migrate at their convenience. Ultimately, FreeBSD is all about stability with no unpleasant surprises. As a result, it is easy for the system engineering team at BALLY WULFF to keep up with the changes and plan ahead.

BALLY WULFF maintains—internally—a fork of the FreeBSD Ports Collection with a handful of additional company-specific ports and patches for the existing ports. Not only is backporting of the latest versions of ports incredibly easy, but maintaining a custom version of an existing port is also simple and painless. Another advantage of the FreeBSD Ports Collection is that the distribution of packages via an internal package repository is effortless and well-supported.

It is easy for the system engineering team at BALLY WULFF to keep up with the changes and plan ahead.

The FreeBSD Project is constantly adding new improvements to ease the process of extending private collections of ports that benefit BALLY WULFF directly. The latest example is poudriere, which streamlines package building, testing, and publishing processes. Another important feature is an overlay support for ports trees, which is currently being tested by the system engineering team at BALLY WULFF. There is a high chance that it will alleviate the need to keep an internal fork of the ports tree, further reducing maintenance overhead.

Goal 3: Customizing System Startup

Typically, general-purpose operating systems feature a program to control the system start-up. It usually configures the newly booted system, for example, by mounting disks and starting essential system services like networking.

In the case of a BALLY WULFF gaming machine, the system start-up procedure is quite different from a typical desktop. The standard FreeBSD start-up procedure is configurable enough to cover most use cases for servers and desktops, but in the case of a gaming machine, it made sense to replace the standard rc(8) mechanism completely. Luckily, there is no black magic involved in replacing the standard rc(8) framework with a custom one. Actually, replacing the /etc/rc file is enough to get started. As a result, the system engineering team at BALLY WULFF maintains a dedicated system start-up script that prepares the OS environment for the games to launch.

At BALLY WULFF, the customized rc(8) framework has been in use for many releases and it continues to work flawlessly. This is certainly a benefit of FreeBSD's steady development practices and modularization of the base system—it is absolutely reasonable to customize a part of FreeBSD and expect the rest of the system to work just fine. It definitely gives the developers peace of mind and lets them focus on developing what is important rather than constantly catching up with backwards-incompatible, upstream changes.

Goal 4: Supporting Custom Update Procedures

It should come as no surprise that BALLY WULFF gaming machines require regular updates. Platform updates occur when there is a need to squash an annoying bug or add an important business functionality to an already-released and operating machine. Much more often, however, the machines are updated with new games. The update process of the gaming machines is one of the most important and rigorously tested procedures in the company. Thorough testing and QA checks guarantee that the updates applied to the machines already operating in the market are not going to cause any unnecessary downtime. The update process must allow for unsupervised and automatic installation of new software. Rendering the machine inoperable in the course of an update is out of the question.

Due to the architecture of the gaming machine's operating system, it would be unnecessarily complicated to update the system with freebsd-update(8) and pkg(8). Thankfully, FreeBSD's simplicity allows for implementing a completely custom update procedure.

Goal 5: Running the Same OS in Both Production and Development

One of the golden rules of software engineering is that development should happen in an environment identical or at least closely resembling the production environment. Developers do not have to debug their code twice when working in a unified environment.

BALLY WULFF game developers use FreeBSD workstations to test games before trying them out on the actual gaming machines. Amazingly, FreeBSD powers the gaming machines and the workstations equally well.

It is worth noting that the game development department is many times larger than the FreeBSD team at BALLY WULFF. Nevertheless, maintaining an internal distribution of FreeBSD tailored specifically to the game developers' needs is very doable. The same FreeBSD-based OS runs on both a gaming machine and on a development workstation, the difference being mainly the list of installed packages. This is a great benefit of FreeBSD being a general-purpose OS.

Goal 6: Staying Close to the Community

Being close to the community allows BALLY WULFF to both participate in the development of FreeBSD and to stay in contact with FreeBSD developers. For example, BALLY WULFF aims to keep the amount of local FreeBSD patches to a minimum. The maintenance burden of non-essential patches is simply not supportable. Not only is upstreaming patches a very sound business decision due to additional testing, but it is also a great way to give back to the FreeBSD community. Most of the time, however, the FreeBSD patches developed internally at BALLY WULFF are too vendor-specific and not suitable for inclusion in the FreeBSD source trees. Nevertheless, the company makes sure to contribute in other ways as well. BALLY WULFF developers regularly participate in FreeBSD Developer Summits and open-source conferences like FOSDEM and EuroBSDcon. In 2019, BALLY WULFF hosted a DevSummit organized at the company's headquarters in Berlin.

Summary

FreeBSD has been a great OS for BALLY WULFF thanks to its remarkable build system, which has been developed and maintained in a way that allows for effective adaptation of the system to specialized appliances. In the past, a major benefit of FreeBSD to BALLY WULFF was the small, yet functional, base system that could be stripped down even further by utilizing existing build(7) knobs or by introducing vendor-specific changes to precisely control what is included in the final OS image. The internal patches to FreeBSD base and ports build systems fit naturally into the consistent Makefile-based infrastructure. All those features allowed the BALLY WULFF system engineering team to minimize the size of the OS, which, in turn, left more space for games and their assets as the BALLY WULFF developers continued to push hardware and software limits.

Now that disk space is not as precious as it once was, the need for a special patch set reducing the final size of the OS is gone. The focus and energy at BALLY WULFF are directed toward other aspects of system development. It is no longer necessary to heavily modify FreeBSD sources to optimize for small disk footprint. The transition to building the OS from the unmodified FreeBSD sources is underway. So far, it has been painless and beneficial as it significantly simplified the build infrastructure. This is a result of a herculean effort by the FreeBSD community to maintain backward compatibility wherever feasible. Every major change to the FreeBSD system is implemented with downstream consumers workflows in mind.

The computing world has evolved quickly and the team's focus is no longer on making the system footprint as small as possible. The target now is to increase the robustness of the system and to keep maintenance costs low. Ultimately, the goal of the BALLY WULFF system engineering team is to provide game developers with a performant and stable gaming platform.

MATEUSZ PIOTROWSKI is a FreeBSD ports and documentation committer based in Berlin. He enjoys troubleshooting bugs, scripting automation, and designing robust software systems (always thoroughly documenting everything along the way). Recently, his interests have drifted toward tracing and performance engineering. When he is not hacking on the supposedly deterministic circuitry of modern software, he is exploring the ever-changing dynamics within society and culture.

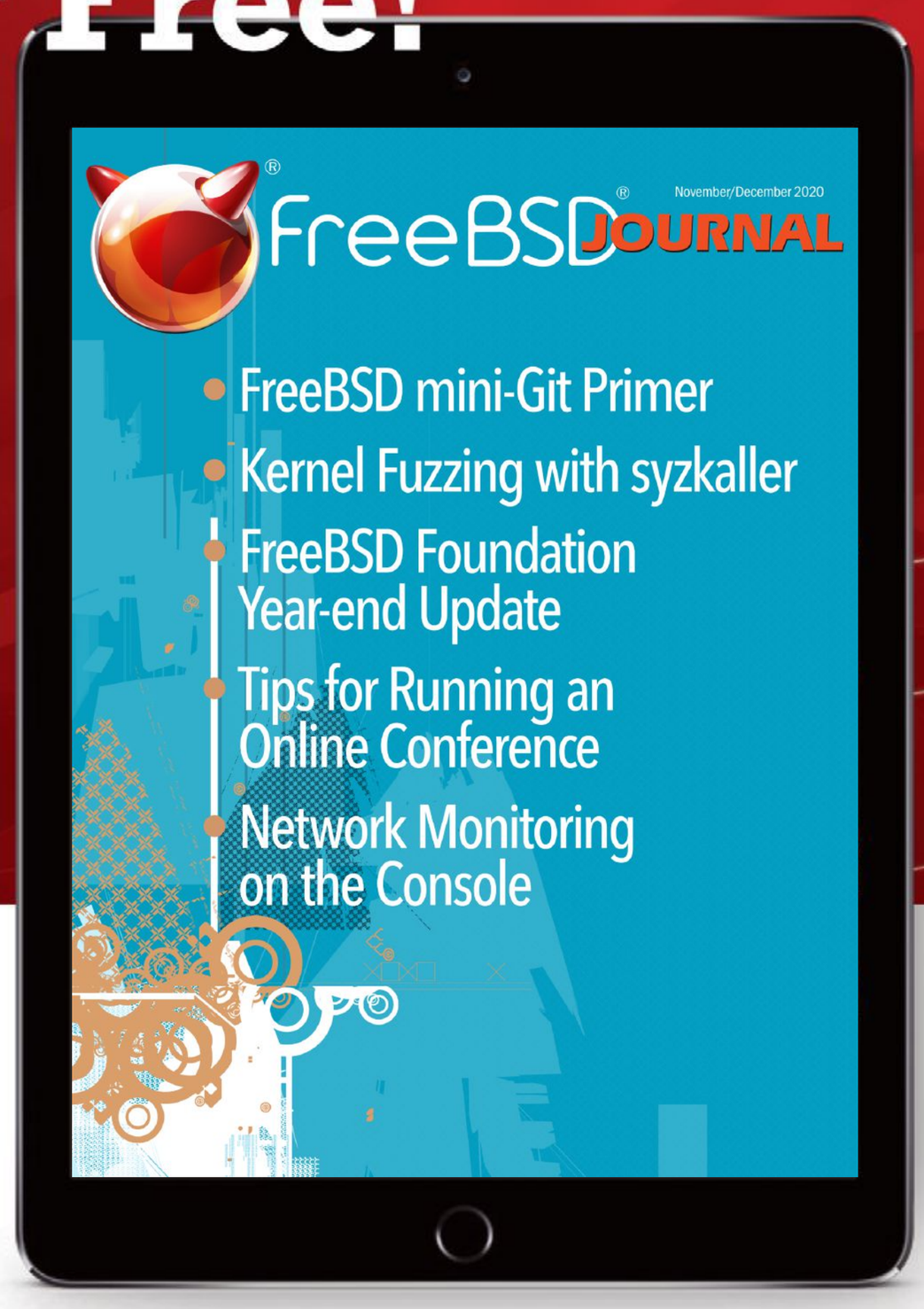
FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!



2021 Editorial Calendar

- Case Studies (January-February)
- FreeBSD 13 (March-April)
- Security (May-June)
- Desktop/Wireless/Graphics (July-August)
- Cloud (September-October)
- Embedded (November December)

Find out more at: freebsdfoundation/journal

NETFLIX | OPEN CONNECT

BY GREG WALLACE

Overview

Netflix (NASDAQ: NFLX) is the world's leading streaming entertainment service with 183 million paid memberships in over 190 countries enjoying TV series, documentaries and feature films across a wide variety of genres and languages. Members can watch as much as they want, anytime, anywhere, on any internet-connected screen. Members can play, pause and resume watching, all without commercials or commitments. www.netflix.com

Open Connect is the name of the global network that is responsible for delivering Netflix TV shows and movies to members world-wide. This type of network is typically referred to as a Content Delivery Network, or CDN, because its job is to deliver internet-based content (via HTTP/HTTPS) efficiently by bringing the content that people watch close to where they're watching it. Open Connect Appliances run a lightly customized version of FreeBSD. https://openconnect.netflix.com/Open-Connect-Overview.pdf

Netflix employs several FreeBSD committers and additional members of the Open Connect team also contribute code upstream.

INDUSTRY

STREAMING ENTERTAINMENT SERVICE

LOCATION

HEADQUARTERS IN LOS GATOS, CALIFORNIA

EMPLOYEES

6700 WORLDWIDE

Open Connect Pushes Over 100 Tb/s Peak

Those of us old enough to remember the dot com and telecom boom may recall the emblematic 1999 Quest Communications advertisement in which a weary traveler checks into a hotel in the middle of nowhere. The clerk promises a lackluster breakfast, but entertainment?

That they have in spades. "Every movie ever made, in any language, anytime day or night."

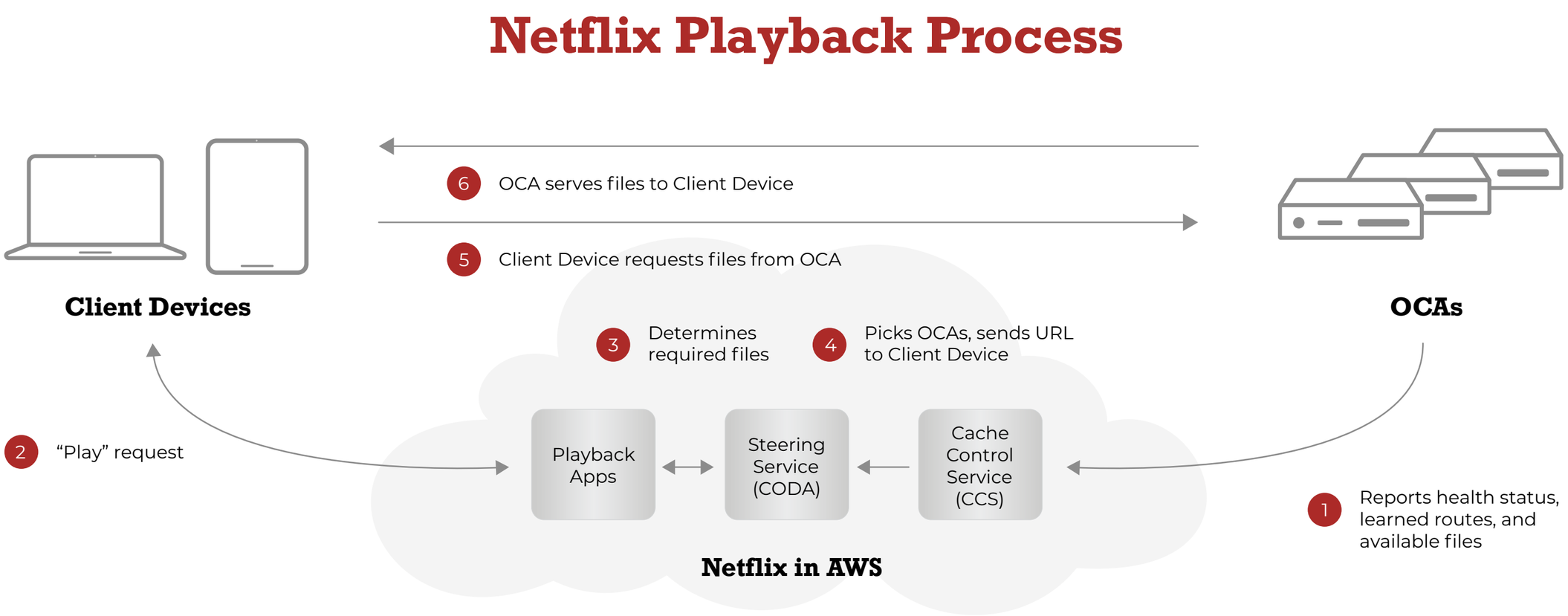
Flabbergasted, the guest wonders aloud "how is that possible?" How indeed! (read on). Twenty years later, and hotel TVs are some of the last devices to provide every movie ever made. Technology, it seems, is not without a sense of irony.

No discussion of the latest trends in streaming entertainment and the technology that makes it possible is complete without Netflix. As of April 2019, the Netflix U.S. catalog consisted of 47,000 TV shows and 4,000 movies. Netflix reports that the global Open Connect Network pushes over 100 Tb/s of traffic at peak. According to Sandvine, this represented about 15% of total internet traffic in 2019.

Open Connect: A Network And A Program

Netflix began the Open Connect initiative in 2011 as a response to the ever-increasing scale of Netflix streaming. Two primary reasons motivated the program:

1. As Netflix grew to be a significant portion of overall traffic on consumer Internet Service Provider (ISP) networks, it became important to be able to work with those ISPs in a direct and collaborative way
2. Creating a content delivery solution customized for Netflix allowed their engineers to design a proactive, directed caching solution that is much more efficient than standard demand-driven CDNs. The directed caching architecture reduces the overall demand on upstream network capacity by several orders of magnitude.



The Network

Most CDNs work in what's called a demand-driven way. This means that *what* the network caches and *where* is determined by what is requested in a particular area. For general purpose CDNs where there is limited ability to predict the content people will want, this works well.

Because Netflix controls the end user apps and has detailed information about viewing trends, they could achieve significant efficiencies moving to a directed CDN. In the Netflix directed CDN model, their fleet of Open Connect Appliances (OCAs), described in detail below, receive daily catalog updates during what are called Fill windows when viewing is very low.

The Program

Netflix has an open peering policy, meaning they will peer with any ISP that agrees with the terms of the program. Open peering improves internet user experience by localizing traffic. It also has the advantage of reducing transit costs, a benefit to Netflix, ISPs, and the internet as a whole.

In addition to OCAs in Netflix data centers and installed in Internet Exchange Points (IXPs), Netflix provides OCAs free of charge to qualifying ISPs for installation directly in the ISP's network. This increases localization and reduces upstream traffic even further.[1] Interestingly, the fact that these OCAs are owned by Netflix, but used by the ISP, raised some licensing considerations that initially drew the Open Connect engineers to FreeBSD for its permissive license.[2]

1 See https://openconnect.netflix.com/Open-Connect-Overview.pdf for program information.
2 https://www.nginx.com/blog/why-netflix-chose-nginx-as-the-heart-of-its-cdn/

Open Connect Appliances

The workhorses of the Open Connect CDN are the Open Connect Appliances, or OCAs for short. These appliances, of which there are three primary configurations, run a lightly customized version of FreeBSD head, or development, branch. That such a large and mission critical network would run the fast-moving development branch may at first blush seem risky. At the 2019 FOSDEM conference, Jonathan Looney, Netflix Engineering Manager on the team responsible for maintaining the OCA operating system, explained the rationale of tracking the FreeBSD head branch.

First, Jonathan and his team find FreeBSD code to be generally very stable and high quality. Second, they prefer to quickly find and fix the relatively infrequent and mostly low-impact bugs they do encounter. Otherwise, Jonathan explains, a development team that waits for the long-term, or Stable, branch, may end up in what he calls a vicious cycle of infrequent merges, many conflicts/regressions, and ultimately slower feature velocity.

"Running FreeBSD head lets us deliver large amounts of data to our users very efficiently, while maintaining a high velocity of feature development."
— Jonathan Looney, Netflix

Tracking the head branch helps Netflix add features more quickly. They also find that tracking the head branch makes collaborating with others in the development community easier.

40Gb/s OCA Storage Appliance with 248TB storage (2RU form factor)

- FreeBSD
- NGINX
- BIRD internet routing daemon

Throughput Efficiency

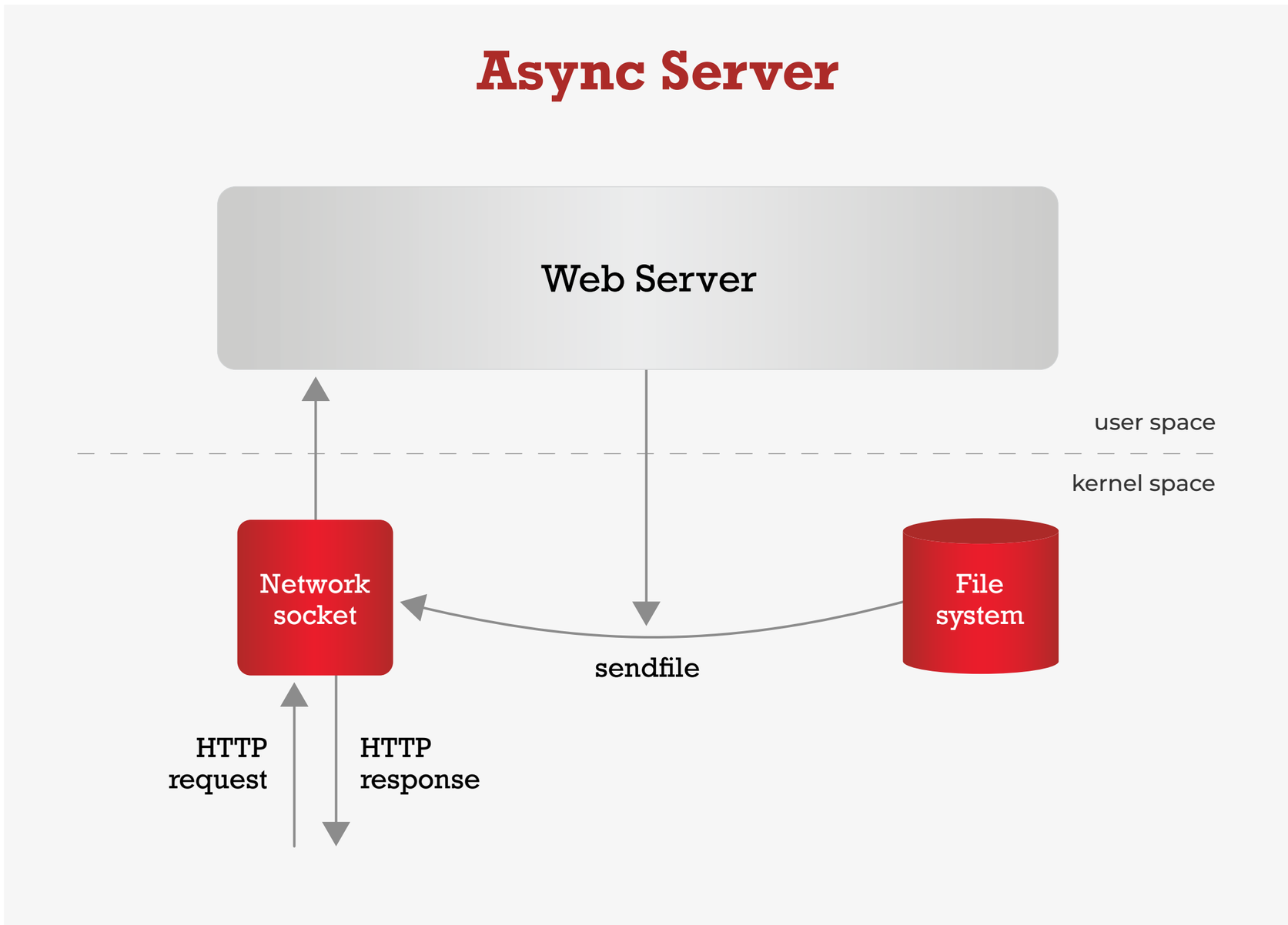
Just how efficient are these OCAs? Using FreeBSD and commodity parts, Netflix achieves 90 Gb/s serving TLS-encrypted connections with ~55% CPU on an Intel 6122 CPU, in 1 RU, with 96GB RAM, and 16TB of NVMe-attached flash storage.

Because it's their intention to upstream as much code as they can, all FreeBSD users benefit from the many enhancements that help Netflix achieve this kind of performance. Some of these contributions include NUMA enhancements, Asynchronous send file, Kernel TLS, Pbuf allocation enhancements, "Unmapped" mbufs, I/O scheduling, TCP algorithms, and TCP logging infrastructure.

In order to achieve this kind of performance cost-effectively, Netflix engineers realized they need to reduce context switching between Kernel and user space as much as possible. Async sendfile is one key technique that helps with this.

The new implementation of the sendfile(2) system call, which is a drop-in replacement for the previous one, speeds up TCP data transfers because it avoids copying file data into a buffer before it's sent. The new version of sendfile further speeds up and simplifies large data transfers by supporting asynchronous I/O.

The new sendfile is a product of a development partnership between NGINX and Netflix, and was released in tandem with a 2016 Netflix service expansion to nearly 200 countries.



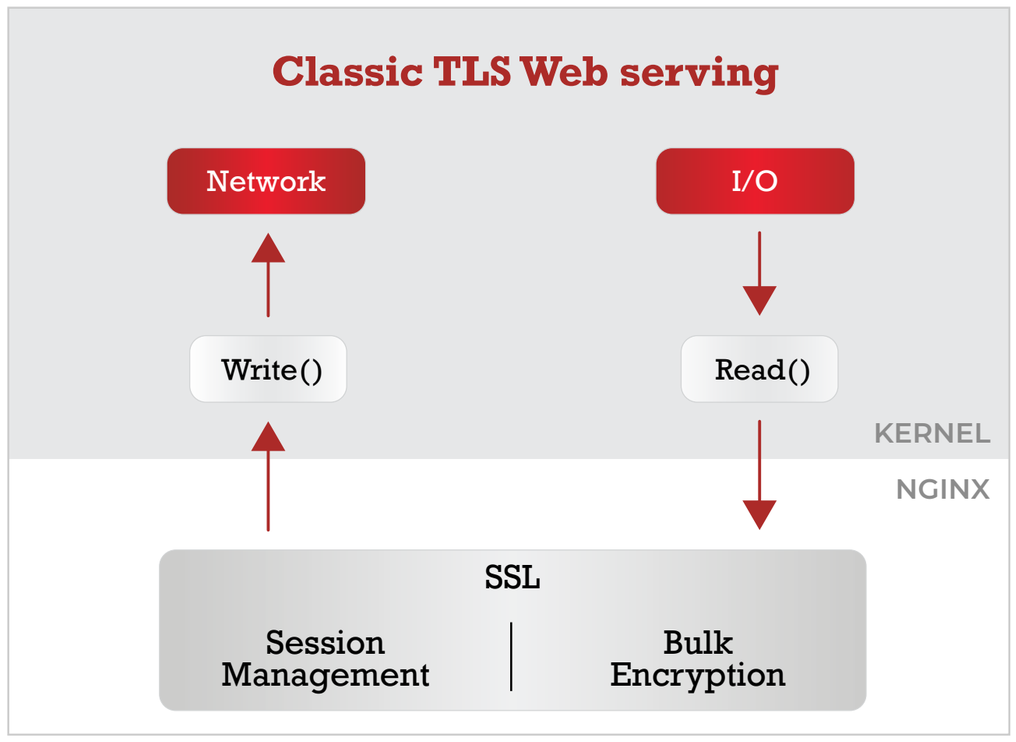
Increasing Efficiency and Privacy — Kernel TLS

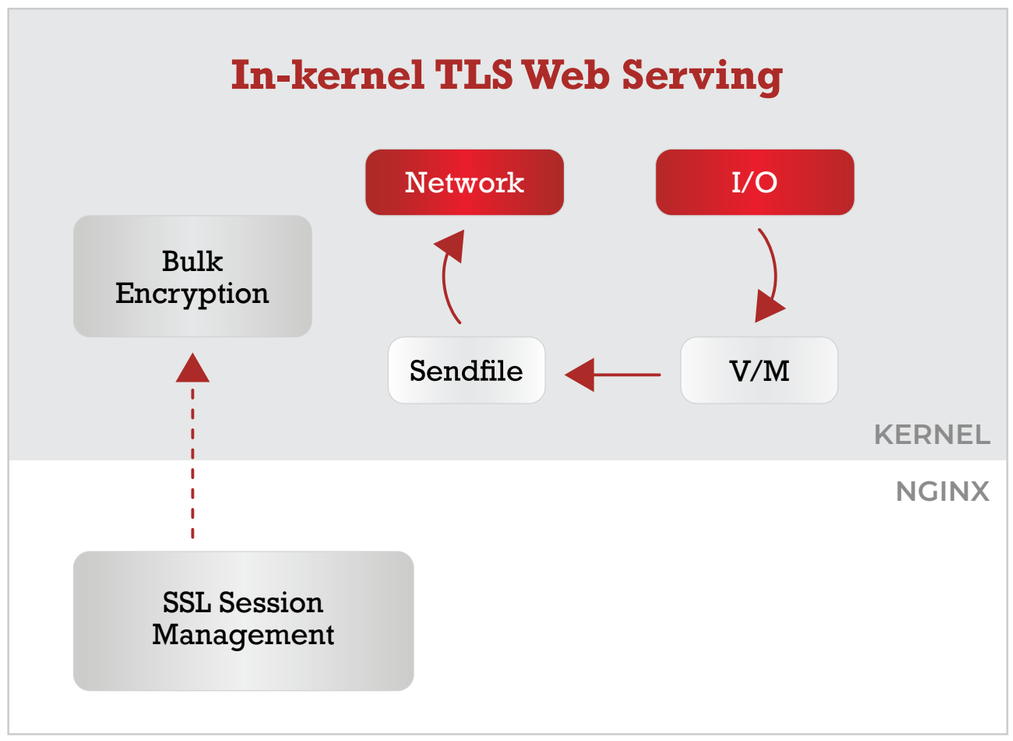
To protect the privacy of end users, in 2016 Netflix added Transport Layer Security (TLS). Jan Ozer summarized this move well in his Streaming Media article:

> Netflix had long deployed DRM to prevent piracy, and it protects customer data during account login and any administration via HTTPS. However, the actual transfer of the movie data was not protected, so any information contained in the communications between the server and client could be accessed by hackers, or by network administrators or ISPs. This information could be used to determine which content the viewer was watching, and perhaps other details.

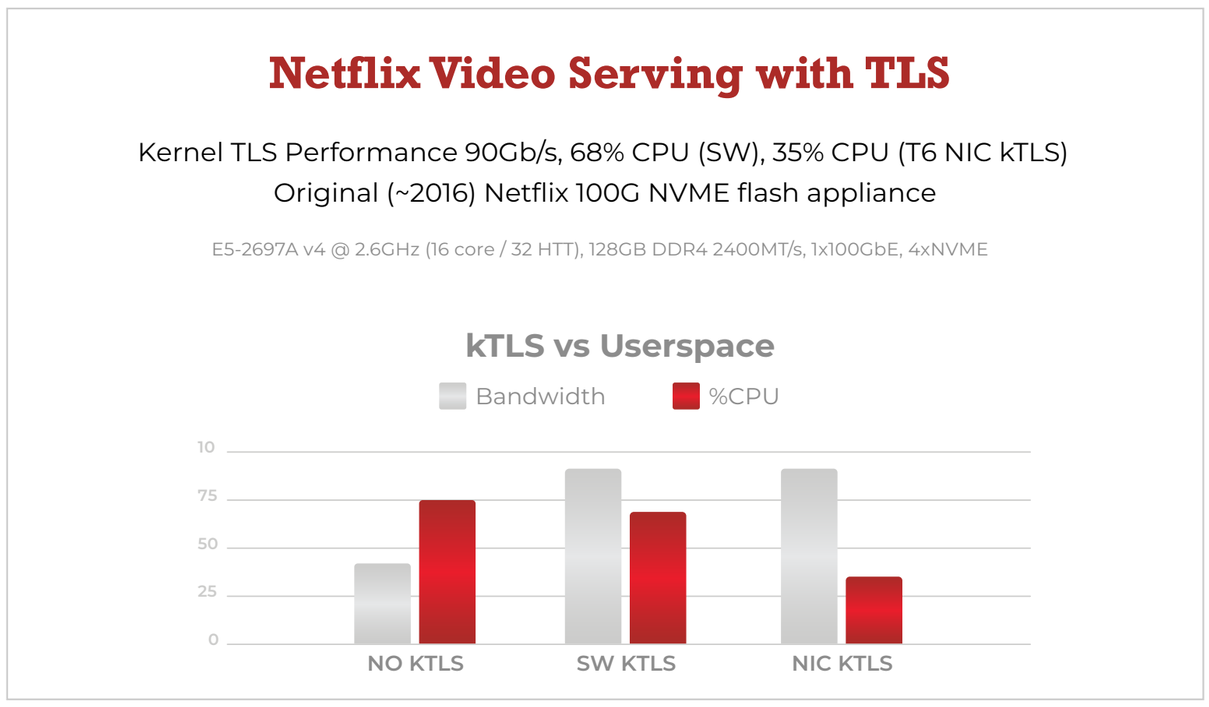
Adding TLS encryption *efficiently* required additional performance enhancements to the OCA software stack. That's because existing TLS techniques relied on the web server — an approach that Netflix's Director of Streaming Standards Mark Watson reported in 2014 would diminish capacity "between 30-53%."

The answer is kernel-side TLS, or kTLS for short, which marries TLS with the new sendfile model. This hybrid TLS scheme (described by John Baldwin in this vBSDCon 2019 session) keeps session management in the application space, and inserts the bulk encryption into the sendfile data pipeline in the kernel. TLS session negotiation and key exchange messages are passed from Nginx to the TLS library, and session state resides in the library's application space. Once the TLS session is set up and appropriate keys are generated and exchanged with the client, those keys become associated with the communication socket for the client and are shared into the kernel.





In their EuroBSD 2019 presentation, Drew Gallatin and Slava Shwartsman show how kTLS gives a 50 Gb/s boost to bandwidth performance while reducing CPU%. The next frontier in TLS performance improvement is something called NIC TLS, where the encryption is done in hardware. As the graph on below shows, this promises to reduce CPU utilization significantly.



Getting to 200 Gb/s with NUMA

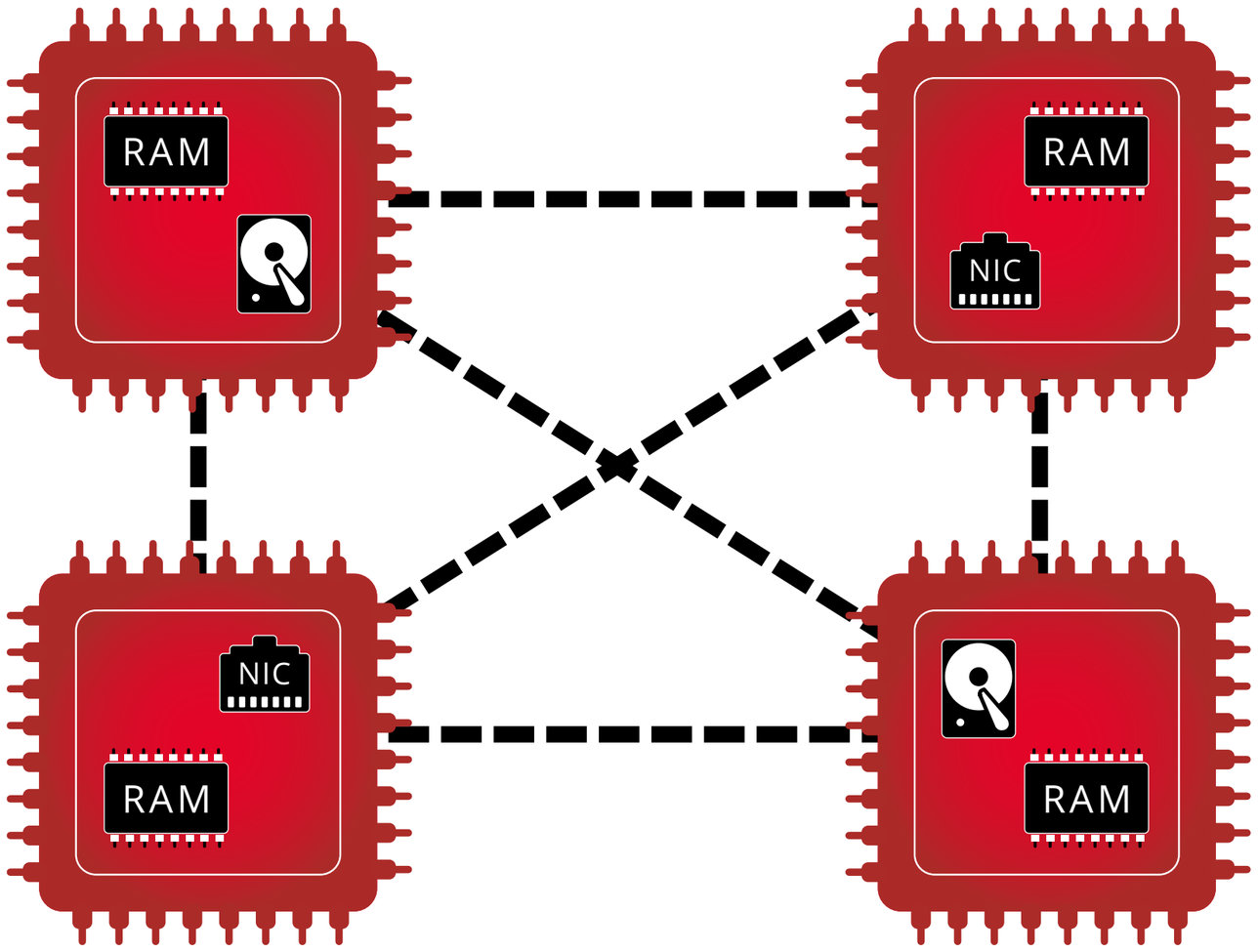
With no end in sight to members' demand for more shows and higher definition, Netflix continues to look for ways to increase the throughput of OCAs. With the evolution of high core count systems, the team has been developing and testing Non Uniform Memory Architecture, or NUMA, support since 2014, and that is now beginning to show results. Where a typical system has a single CPU, disk and memory, a NUMA system can have many more. As with sendfile and TLS, this can present throughput-sapping bottlenecks that Netflix engineers have been hard at work to minimize.

NUMA makes it cheaper for a CPU to access local resources (e.g. memory) and more expensive for it to access resources attached to another node. Consequently, memory and I/O locality impacts performance. For Netflix to take advantage of NUMA's greater computation density, they had to come up with a way to keep as much of the disk-to-CPU-to-network traffic local to a node and minimize performance-sapping NUMA bus transfers. This led to enhancements, which are in various stages of being merged upstream, including:

- Allocating NUMA local memory to back files sent via sendfile(9)
- Allocating NUMA local memory for Kernel TLS cryptobuffers
- Directing connections to TCP Pacers and kTLS workers bound to the local domain
- Directing incoming connections to Nginx workers bound to the local domain via modifications to SO_REUSEPORT_LB listen sockets

In tests, these enhancements have improved Xeon performance from 105Gb/s to 191Gb/s While reducing NUMA fabric utilization from 40% to 13%. For AMD EPYC, performance increased from 68Gb/s to 194Gb/s.

Four Node Configurations are Common on AMD EPYC



FreeBSD Gives NETFLIX Three Kinds Of Efficiency: Throughput, Development, and Operations

In response to the FAQ "why FreeBSD?" Jonathan says they came for the license and stayed for the efficiency — efficiency that Netflix measures in three ways:

1. Throughput, or performance, efficiency described in the previous section
2. Development efficiency
3. Operational efficiency

From a development perspective, the ease of working with the FreeBSD community helps Netflix upstream their enhancements for ongoing maintenance by the community. They also enjoy collaborating with others in the community that are working on the same area. Sharing code with these other community members can improve the code all parties are developing.

Finally, the huge fleet of OCAs requires sophisticated tooling for monitoring and operations. Some of the tools they've needed already existed, and the rest they have written. For the latter, Jonathan has found FreeBSD does a good job surfacing the necessary hooks and, where not, the team has been able to implement them.

What's Coming Next from the Open Connect Brain Trust

In addition to NUMA and ongoing exploration of NIC TLS, the team is working on upstreaming some enhancements to kTLS and on UFS enhancements.

In closing, the massive scale of Open Connect combined with the team's focus on efficiency and their commitment to open source means that every FreeBSD user with a similar use case can reap the same performance benefits. The ability to turn on kTLS and take advantage of Async Sendfile allows anyone serving static content over HTTPS to extend their hardware lifetime, reduce density, and deliver a great user experience more efficiently.

GREG WALLACE is a freelance technology marketer who has been working with open source software and communities since 2005. In addition to his current work with the FreeBSD Foundation, Greg dabbles in Kubernetes, security, DevOps, and routing. Previously, he led marketing for Node.js, ODPi, and Hyperledger.

FreeBSD for the Writing Scholar

BY COREY J. STEPHAN

Perhaps it is natural that the sort of mind that thrives on piecing together minutia within one (not computer-related) academic discipline tends to be different from the sort of mind that thrives on learning to use cutting-edge technology. If all that an information technology team provides (or allows) at a workstation is either bloatware from Redmond or spyware from Cupertino, then one might (fairly) assume that nothing better exists. Each week, the typical academic must balance preparing lessons, lecturing, grading, attending meetings, and holding office hours—all while struggling to reserve blocks of time (inevitably too small) for personal research and writing. The stereotype of the absent-minded scholar often holds true (I write while looking in the mirror) not only because of a personal propensity for aloofness but also because workdays are disorderly by institutional design. Normally, the Frankenstein Monster-esque computer setup that I notice while chatting in a windowless, book-filled office is only one piece in a particular scholar's chaotic work life. With so much pressure to produce materials for publication, who has time to build a better computer workflow?

I think that all the traits that a scholar needs in a desktop operating system fit within three broad categories: documentation, stability, and security.

Scholars like documentation atop documentation (atop documentation). If scholars cannot verify something for ourselves, then we are unlikely to trust it. Poorly documented operating systems cannot withstand the skepticism that is standard in the academy. Organization and documentation go hand-in-hand. An operating system whose developers prioritize consistency is probably going to be intelligible for the person who takes the time to learn a bit about how it works. Good man pages, a clean handbook, and a wiki that a dedicated userbase actively maintains—this trifecta is the minimum of order that a scholar's OS must have.

Scholars do not need the same kind of stability for our workstations as, say, freebsd.org needs for its servers, but we do need stability. Twice, my Ryzen-based desktop computer with 16GB of RAM has crashed because I was using 13GB in one program to manipulate a manuscript facsimile. On another occasion, I almost entered cardiac arrest because an update for LibreOffice's Fresh branch rendered all of my work-in-progress dissertation chapters un-editable until I paused and realized that I simply needed to downgrade to LibreOffice Still. To say the least, neither moment was pleasant. Scholars work with many long, complicated documents and databases, and our success depends on as few of those failing during crunch time as possible. If tools in scholars' toolboxes are not broken, we will not wish for anyone to try to fix them.

Security, like stability, means something else for the scholar's workstation than it does for the kinds of environments that a system administrator or software developer has in mind. For the scholar, security means privacy. The best scholars share their work with others for feedback and expect others to do the same with them. However, academics are also the sort of people to be quite selective about those who are or are not able to see what they are doing. Many academics work with confidential materials. Any operating system that reports what its end users are doing to someone somewhere else or that can be hacked or monitored easily is not suitable for scholarly work.

FreeBSD is not the only major open-source operating system with superb documentation, stability, and security (we must never overlook our siblings in the other *BSDs or cousins working on Debian and other great GNU/Linux distributions), but it is undoubtedly one of the best. Information technology professionals often are the only people writing about FreeBSD. They might not realize that many of the things that make FreeBSD fantastic for servers and embedded systems also make it outstanding as a desktop operating system for the writing scholar.

The documentation and organization of FreeBSD are splendid. As a scholar particularly ever in search of historical documentation, trying to find order amidst a diversity of ancient voices, I find the internal coherency of FreeBSD to be downright refreshing. Even kernel building in FreeBSD mainly involves the use of human-readable, plain text configuration files. Things make sense because the people creating them aim for them to make sense.

> All the traits that a scholar needs in a desktop operating system fit within three broad categories: documentation, stability, and security.

Few people are going to question FreeBSD's stability. When the FreeBSD Wiki lists certain hardware as being well supported, FreeBSD most likely will not be the cause of a system crash for anyone doing (even esoteric) desktop work on that hardware. My system crashes were in a different operating system. I doubt the same would have happened if I had been using a properly configured FreeBSD-STABLE system and the long-term service branches of my main third-party software applications.

Finally, FreeBSD is fully open and markedly secure. With FreeBSD, scholars can trust that our intellectual property is not monitored by a suit at some hypothetical FreeBSD headquarters in Silicon Valley. FreeBSD is sensibly secure by default and sensible enough to secure.

I proudly recommend FreeBSD for fellow writing scholars. Install FreeBSD. Install the X Window System. Install a dynamic tiling window manager with plain text configuration like FreeBSD consistently uses, such as spectrwm or i3wm, or else a simple traditional desktop environment (XFCE is trustworthy). Install LibreOffice Still and a decent citation management tool that integrates with LibreOffice and handles materials from your discipline (Zotero runs flawlessly in Wine, and FreeBSD has its own native options). Install discipline-specific command-line interface tools. The pkg system makes a good deal of third-party desktop software simple to install. Enjoy researching and writing with a logical, stable, and overall solid operating system. Your publishing schedule no longer will be subject to setbacks from your software.

COREY STEPHAN is a Ph.D. candidate in the Department of Theology at Marquette University in Milwaukee, Wisconsin. He proudly makes exclusive use of F.O.S.S. tools to assist his research in the history of Christian theology. His professional website is coreystephan.com.

PRACTICAL PORTS

On Top of the World

BY BENEDICT REUSCHLING

This column covers ports and packages for FreeBSD that are useful in some way, peculiar, or otherwise good to know about. Ports extend the base OS functionality and make sure you get something done or, simply, put a smile on your face. Come along for the ride, maybe you'll find something new.

Ever since I began working with Unix, I remember using `top(1)`. For those of you who have been living under a Unix rock, you should really climb on top. With its continuing display of processes, it gives you an instant view of what is going on in your machine—process wise. Compared to the static output of `ps(1)`, it decorates at least one corner of any serious Unix sysadmin screen to make it look busy. Fancy screensaver or not, a quick glance can help in an early evaluation of system load. Of course, the BSDs are no different and they even provide some extra features that other top implementations don't have. For I/O,

```
top -m io
```

gives you reasons every second to finally replace that hard disk with something, well, flashier.

If you want to just list the ten most active processes that hog your system? Simply type

```
top 10
```

into your terminal to get it. Very intuitive! Also, exiting out of top is easier for beginners than getting out of vi (hint: q is involved). Perhaps that is one of the reasons this program has been cloned and rewritten for other purposes over the years. Whenever there is a need to figure out why things are sluggish, top is one way to figure it out. Users tend to complain about it only to discover that their own processes are grinding the system to a near standstill.

The most popular rewrite is probably `sysutils/htop`, which extends the base top with colors and a customizable display. From adding a remaining battery display on your laptop to a display of the hostname for the ssh system-to-system hoppers among us, it all can be configured. Process views offer a neat tree view of threads spawned by your shell or system daemons

processing data. My years of playing DOS games and hitting the spacebar all too often has me confused in htop, though. On the FreeBSD top implementation, this causes the display to refresh. And on htop, it selects the process under the cursor. I should study the man page to see what it does instead of relying on top being the same everywhere.

Speaking of old DOS games: after installing `sysutils/bashtop` and invoking it for the first time, I can't shake the feeling that I'm running in professional protected mode runtime again. It freezes completely and the homepage redirects me to bpytop instead. Just looking at the github page [https://github.com/aristocratos/bpytop], I see it not only lists FreeBSD support, but a whole lot of others, too. My fingers quickly type

```
pkg install bpytop
```

into my terminal to pull down the latest ported version. The application runs and I'm stunned for a second: another initialization routine. My trained gaming fingers are quick enough to capture it in a screenshot for readers before the main display appears. You may think I'm taking this to the top, but what follows is as close to a drug trip that I have probably ever come to in my life—the colors are just too much for me. Nevertheless, it shows a lot of good information about your system. The little dots in the top left corner give me a combined view of each of the 24 CPUs I have on this system. Selecting a process and hitting enter yields more details about it. Network and memory are also shown on the same screen. Certainly neat, but now I need to get my eyeballs replaced—I think I still have some in the fridge...

OK, Dilbert's Topper would say "That's nothing..." so I'm searching freshports.org looking for other top-like utilities. Sure enough, nearly every letter of the Alphabet seems to have been put in front of top in one way or the other. From `sysutils/atop` that works quite well on FreeBSD despite the man page claiming it is a resource monitor for Linux, `databases/mtop`

or `mytop` to view your MySQL processes, to `pgtop` (same for Postgres) there is plenty to select from. The only thing I'm missing is stop—stop a process. What happened to "programs should do one thing and do it well?" Maybe I'm old fashioned in this regard!

Network admins will probably take a look at `dns/dnstop` to capture and peek into DNS traffic flowing by. Simpler views than bpytop for sure but it has all one needs. Or try out `net-mgmt/bandwhich` to pinpoint where your bandwidth is going all day.

One thing I found that has not been ported yet and has nothing to do with a process viewer is topgrade [https://github.com/r-darwish/topgrade]. It upgrades not only from one, but all package managers you have on your system. Perhaps playing the Top Gun theme while doing it, the logo certainly brings that to mind. The server I was running this on has no sound chip, so I can't tell. Imagine all the datacenter workers scrambling to figure out which machine this comes from in a rack of machines!

What happened to "programs should do one thing and do it well?" Maybe I'm old fashioned in this regard!

I also remember (off the top of my head, of course) other top-like programs not yet ported to FreeBSD. The movie TRON seems to have inspired eDEX-UI [https://github.com/GitSquared/edex-ui] with its futuristic design. Can we get that one ported, pretty please (with sugar on top)?

There is a base RUST library called tui-rs [https://github.com/fdehau/tui-rs] which provides the building blocks for many other flexible and dynamic window-like displays in the terminal. I'll mention just one of them here which seems to be what `tail(1)` is to top's `head(1)`: bottom [https://github.com/ClementTsang/bottom]. While there, `sysutils/gotop` may interest you with its squiggly line display of CPU usage. On laptops and servers, it tries to determine CPU temperatures as well.

I would refer you to www.unixtop.org to get some history about this utility if it were not down. Luckily, archive.org has a copy from 2017 that can be used. Wikipedia also has enlightening information, so I'll leave you to that. Here's hoping this column wasn't too much over the top and you could add some utilities to your Unix toolbox.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Together with Allan Jude, he is host of the weekly bsdnow.tv podcast.

Thank you!

The FreesBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.

FreeBSD FOUNDATION

Are you a fan of FreeBSD? Help us give back to the Project and donate today! **freebsdfoundation.org/donate/**

Please check out the full list of generous community investors at freebsdfoundation.org/donors/

Uranium

Koum Family Foundation

Iridium

arm NGINX NetApp

Platinum

NETFLIX

Gold

Juniper Networks

Silver

BECKHOFF Microsoft

mozilla vmware

STORMSHIELD Tarsnap

Write For Us!

Contact Jim Maurer with your article ideas.

(**jmaurer@freebsdjournal.com**)

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

Events Calendar

BSD Events taking place through March 2021

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.

Users with organizational software that uses the iCalendar format can subscribe to the FreeBSD events calendar which contains all of the events listed here.

APRICOT 2021

March 1-4, 2021
VIRTUAL

Representing Asia Pacific's largest international Internet conference, Asia Pacific Regional Internet Conference on Operational Technologies (APRICOT) draws many of the world's best Internet engineers, operators, researchers, service providers, users and policy communities from over 50 countries to teach, present, and do their own human networking. The ten-day summit consists of seminars, workshops, tutorials, conference sessions, birds-of-a-feather (BOFs), and other forums with the goal of spreading and sharing the knowledge required to operate the Internet within the Asia Pacific region. This year, the conference will happen virtually.

FOSSASIA

March 13-21, 2021
VIRTUAL

Join the FreeBSD Foundation at the FOSSASIA Virtual Summit 2021. Taking place March 13-21, 2021, FOSSASIA provides the opportunity to learn about open technologies and to connect with lead developers and technologists from Free and Open Source software (FOSS) and Open Hardware projects from around the world. Registration is free of charge.

FreeBSD Fridays

https://freebsdfoundation.org/freebsd-fridays/
Our next FreeBSD Fridays session will be March 26.
Past FreeBSD Fridays sessions are available at: https://freebsdfoundation.org/freebsd-fridays/

FreeBSD Office Hours

https://wiki.freebsd.org/OfficeHours
Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions. There will be a FreeBSD Core Team Office Hours session on March 17.

Past episodes can be found at the FreeBSD YouTube Channel
https://www.youtube.com/c/FreeBSDProject.