;login:

WINTER 2016 VOL. 41, NO. 4

Your Cores Are Slacking Off
Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova

BeyondCorp Part III: The Access Proxy
Luca Cittadini, Batz Spear, Betsy Beyer, and Max Saltonstall

OpenLambda for Microservices
Scott Hendrickson, Stephen Sturdevant, Edward Oakes, Tyler Harter, Venkateshwaran Venkataramani, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau

Tuning ZFS for Databases
Allan Jude and Michael W. Lucas

Interview with Fyodor Vaskovich
Rik Farrow

Columns

Understanding Python Metaclasses
David Beazley

Monitoring Your Monitoring Systems
Dave Josephsen

Forecasting the Weather with Dark Sky
David N. Blank-Edelman

Extending Go Applications with Exec Plugins
Kelsey Hightower

Rising Tide of IoT Devices
Dan Geer

Idiot-Blockers for the Internet
Robert G. Ferrell

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

LISA16
December 4–9, 2016, Boston, MA, USA
www.usenix.org/lisa16

Co-located with LISA16

SESA '16: 2016 USENIX Summit for Educators in System Administration
December 6, 2016
www.usenix.org/sesa16

USENIX Journal of Education in System Administration (JESA)
Published in conjunction with SESA
www.usenix.org/jesa

Enigma 2017
January 30–February 1, 2017, Oakland, CA, USA
enigma.usenix.org

FAST '17: 15th USENIX Conference on File and Storage Technologies
February 27–March 2, 2017, Santa Clara, CA, USA
www.usenix.org/fast17

SREcon17 Americas
March 13–14, 2017, San Francisco, CA, USA
www.usenix.org/srecon17americas

NSDI '17: 14th USENIX Symposium on Networked Systems Design and Implementation
March 27–29, 2017, Boston, MA, USA
www.usenix.org/nsdi17

SREcon17 Asia/Australia
May 22–24, 2017, Singapore
www.usenix.org/srecon17asia

USENIX ATC '17: 2017 USENIX Annual Technical Conference
July 12-14, 2017, Santa Clara, CA, USA
Submissions due February 7, 2017
www.usenix.org/atc17

Co-located with USENIX ATC '17

HotCloud '17: 9th USENIX Workshop on Hot Topics in Cloud Computing
July 10-11, 2017, Santa Clara, CA, USA
Submissions due March 14, 2017
www.usenix.org/hotcloud17

HotStorage '17: 9th USENIX Workshop on Hot Topics in Storage and File Systems
July 10-11, 2017, Santa Clara, CA, USA
Submissions due March 16, 2017
www.usenix.org/hotstorage17

SOUPS 2017: Thirteenth Symposium on Usable Privacy and Security
July 12-14, 2017, Santa Clara, CA, USA
Paper registrations due March 1, 2017
www.usenix.org/soups2017

USENIX Security '17: 26th USENIX Security Symposium
August 16-18, 2017, Vancouver, B.C., Canada
Submissions due February 16, 2017
www.usenix.org/sec17

SREcon17 Europe/Middle East/Africa
August 30–September 2, 2017, Dublin, Ireland
www.usenix.org/srecon17europe

Do you know about the USENIX open access policy?

USENIX is the first computing association to offer free and open access to all of our conferences proceedings and videos. We stand by our mission to foster excellence and innovation while supporting research with a practical bias. Your membership fees play a major role in making this endeavor successful.

Please help us support open access. Renew your USENIX membership and ask your colleagues to join or renew today!

www.usenix.org/membership

www.usenix.org/facebook www.usenix.org/gplus www.usenix.org/linkedin
twitter.com/usenix www.usenix.org/youtube

;login:

WINTER 2016 VOL. 41, NO. 4

EDITORIAL

2 **Musings** *Rik Farrow*

SYSTEMS

6 **Your Cores Are Slacking Off—Or Why OS Scheduling Is a Hard Problem** *Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova*

14 **Serverless Computation with OpenLambda** *Scott Hendrickson, Stephen Sturdevant, Edward Oakes, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau*

20 **Standing on the Shoulders of Giants by Managing Scientific Experiments Like Software** *Ivo Jimenez, Michael Sevilla, Noah Watkins, Carlos Maltzahn, Jay Lofstead, Kathryn Mohror, Remzi Arpaci-Dusseau, and Andrea Arpaci-Dusseau*

SECURITY

28 **BeyondCorp Part III: The Access Proxy** *Luca Cittadini, Batz Spear, Betsy Beyer, and Max Saltonstall*

34 **Excavating Web Trackers Using Web Archaeology** *Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner*

41 **The Adblocking Tug-of-War** *Hamed Haddadi, Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E. Powles, Emiliano De Cristofaro, and Steven J. Murdoch*

44 **Interview with Gordon Lyon** *Rik Farrow*

SYSADMIN

48 **Tuning OpenZFS** *Allan Jude and Michael W. Lucas*

52 **Lessons from Iraq for Building and Running SRE Teams: From the Assembly Line to the Web** *Kurt Andersen*

57 **Interrupt Reduction Projects** *Betsy Beyer, John Tobin, and Liz Fong-Jones*

COLUMNS

64 **In Praise of Metaclasses!** *David Beazley*

70 **Practical Perl Tools: The Whether Man** *David N. Blank-Edelman*

75 **iVoyeur: Dogfood and the Art of Self-Awareness** *Dave Josephsen*

78 **For Good Measure: Implications of the IoT** *Dan Geer*

80 **Extending Go Applications with Exec Plugins** *Kelsey Hightower*

84 **/dev/random** *Robert G. Ferrell*

BOOKS

86 **Book Reviews** *Mark Lamourine*

USENIX NOTES

89 **USENIX: It's Not What You Might Think** *Cat Allman*

90 **Impressive Results for Team USA at 2016 International Olympiad in Informatics** *Brian C. Dean*

91 **Thanks to Our Volunteers** *Casey Henderson*

93 **USENIX Association Financial Statements for 2015**

usenix®
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Michele Nelson
michele@usenix.org

COPY EDITORS
Steve Gilmartin
Amber Ankerholz

PRODUCTION
Arnold Gatilao
Jasmine Murcia

TYPESETTER
Star Type
startype@comcast.net

USENIX ASSOCIATION
2560 Ninth Street, Suite 215
Berkeley, California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738

www.usenix.org

;login: is the official magazine of the USENIX Association. *;login:* (ISSN 1044-6397) is published quarterly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for nonmembers are $90 per year. Periodicals postage paid at Berkeley, CA, and additional mailing offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

EDITORIAL

Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

Four times a year, I sit down at my desk and try to come up with something different to write. I've been writing a column for *;login:* since 1996, although back then it was about Java [1], and I was not the editor. To be honest, I wouldn't still be editing *;login:* if I didn't love doing it. I enjoy working with the members and people who attend conferences, and the USENIX staff. Coming up with articles for each issue is always a challenge, one that I welcome you to help with.

Perhaps I should explain what I've been trying to do with *;login:*, since I was first asked to edit special issues on security in 1998. I actually had written a proposal to the USENIX Board of Directors in 1996 about how I would go about editing *;login:*, a proposal I found in 2006 when I was tasked with creating an updated version.

Goals

I strive to collect articles about emerging research, new tools, and techniques that I believe will benefit a broad cross-section of the USENIX membership. It's a challenging goal, as the membership ranges from industry to academic, from programmers to system administrators and SREs. USENIX has conferences on systems, security, file systems and storage, distributed systems, system administration, cloud computing, and site reliability engineering (SRE). Finding topics that may be useful to at least two of these categories, and hopefully more, has always been my goal.

Like any Program Committee member, I don't want to ever publish (or accept) bad research. In the case of *;login:*, I can often sidestep that issue because I ask the authors of accepted papers to write about their research. But that only works for topics that have been covered by papers.

When the author or authors haven't produced a paper, I look for other indications of competency in the topic at hand. I also check out references, by asking some of the many people I've gotten to know in attending USENIX conferences over the years, about the authors I have in mind.

And there's also the quality detector: you probably have one of these as well [2], and I highly recommend paying attention to any alerts it produces. For me, these alerts come when I notice that the draft I am reading fails to tell me anything useful, or covers the same territory as found online. Other times, the writer will contradict him or herself, or write things that sound just plain wrong. Again, the Web can be your friend.

Finally, timeliness is an important goal. There is lag time inherent in print publications, and I start searching for good topics six months before an issue comes out. That's a very long time when part of my goal is to cover emerging topics.

I plan on editing *;login:* into the foreseeable future and welcome new ideas and input. If you'd be interested in contributing to *;login:*'s process, and if you have suggestions for topics that should be covered, please do let us know by contacting us at login@usenix.org.

The Lineup

We start out this issue with several articles with the theme of systems. Lozi et al. had a paper at EuroSys 2016 that caught my attention, as the authors did a wonderful job of explaining how the Linux Completely Fair Scheduling system works. By adding in instrumentation and creating heat maps, they also uncovered ways in which it fails. (Those heat maps do look better in color, so you might want to view the online version of their article, or the paper it is based on.)

I ran into Tyler Harter during the 2016 USENIX Annual Technical Conference poster session. Tyler had presented a paper about OpenLambda during the HotCloud '16 workshop, and when I read the paper (I had missed his presentation), I decided that Lambdas, a way of supporting microservices, deserved a wider audience. OpenLambda is a research platform for exploring Lambdas, but Harter et al. also explain the AWS version of Lambdas and current shortcomings with this new way of providing services.

Carlos Maltzhan approached me during USENIX Security '16, wanting me to talk with some students at UC Santa Cruz about a new approach to creating papers that fosters reproducibility of CS research. He and his colleagues, Jimenez et al., explain how they have created a framework, named for Karl Popper [3], using a toolset that anyone familiar with DevOps will know about. Popper is a protocol that uses these tools, aiding in the research and paper writing process, but also creating a trail that others may follow later.

The December issue has traditionally covered security, and we do have four security-related articles.

Cittadini et al. explain another aspect of Google's BeyondCorp, a technique that does away with considering any "internal" network secure. Instead, BeyondCorp invests trust in managed devices and user authentication, and it permits access to specific services. This third article in a series on the topic explains BeyondCorp's Access Proxy, where access control lists (ACLs) control what services are available to which users using specific devices. The authors also detail the trickier aspects of making the Access Proxy work with non-HTTP protocols.

Lerner et al. had a paper at USENIX Security '16 about how Web tracking has changed over time. Their methodology involves using archive.org's Wayback Machine and examining just how accurate its record is. I was also interested in their results, but I can't say that I am surprised at just who is tracking our Web histories.

Haddadi et al. had a paper at FOCI '16 on anti-adblockers. I first thought this seemed a stretch for a workshop called Free and Open Communications on the Internet, where topics include Tor or the Great Firewall of China. If you're using adblockers, I'm sure you've encountered the manifestation of anti-adblockers; the authors explain both how these scripts work and why they are a privacy issue.

I interview Gordon Lyon, perhaps better known as Fyodor. Fyodor started working on the Nmap scanner back in the mid-'90s and has turned his passion into a successful open source business—plus a really useful tool that you should know about.

Switching over to the areas of sysadmin and SRE, we start out with an article that was adapted from a section of Allan Jude and Michael Lucas' book *FreeBSD Mastery* for inclusion in *;login:*. Jude and Lucas write about tuning ZFS for use with popular databases. While this might at first appear to be a pretty narrow topic, the authors cover the typical writing patterns used in several databases, something you may want to know if you use any of these databases, or wonder if you are using the right file system and have properly configured it.

Kurt Andersen of LinkedIn had a popular talk at SREcon16 Europe, and I asked him to reprise his talk for *;login:*. Kurt's topic is focused on how SRE teams are managed, but I think that the concepts presented would be useful in any organization that appreciates nimbleness.

Liz Fong-Jones also drew a large crowd at SREcon16 Europe with her presentation on Interrupt Reduction Projects. Working with Betsy Beyer and John Tobin, Liz created an article about how the Bigtable SRE teams worked to reduce interrupts. There are deep insights into how better to handle tickets and that may even include ignoring them while you fix underlying issues that aren't big enough to register as projects.

Dave Beazley wants to talk meta. Python has long had metaclasses, and Dave explains what metaclasses are and shows how they can be used to write cleaner Python code. Very cool ideas and mind-bending, as usual.

Dave Josephsen also goes meta, but in a different direction. Dave asks, if you rely on your monitoring, what monitors your monitoring system? Dave then describes how the people at the monitoring company he works for created a second site, one they call "dogfood," that eats the input from their main monitoring systems.

David N. Blank-Edelman wonders whether there will be weather and shows us how to use a RESTful service from Perl, not just for the current weather but for past *and* future weather, too.

Kelsey Hightower shows us how to extend Go applications with exec plugins. By creating Go programs that meet an interface, you can extend an application without editing the source.

Dan Geer considers the implications of the growth in connected devices. He projects that by 2020, there will be 6.5 IoT devices for every human alive, and that the security implications are, frankly, hard to imagine. For example, you might have heard of attack surfaces, that is, the features that make a computer vulnerable to attack. We think of attack surfaces because the goal is to reduce them. But with so many devices on the way, will this even make sense?

Robert G. Ferrell wants us to consider a new class of Web plugin: the idiocy blocker. Robert has found five main types of Internet idiots, and postulates a method for making it possible to read the comments associated with articles and other Web postings.

Mark Lamourine has written three reviews for this issue. The first two cover Single Program Applications (SPA), something I hadn't heard of before, but also something we've all encountered in the form of many online apps. Then he takes a look at a book on providing examples of practical Go programs.

Finally, Cat Allman shares her views on why she became a USENIX Board member. Cat had worked for USENIX in the noughts, and came back to help keep USENIX going. Casey Henderson has also written her yearly summary, including thank-yous to Program Chairs and the volunteers who make USENIX conferences interesting.

[1] R. Farrow, "Using Java": https://web.archive.org/web/19970606052503/http://www.usenix.org/publications/java/usingjava1.html.

[2] G. Pennycook, J. A. Cheyne, N. Barr, D. J. Koehler, J. A. Fugelsang, "On the Reception and Detection of Pseudo-Profound BS," *Judgment and Decision Making*, vol. 10, no. 6 (November 2015), pp. 549–563: http://journal.sjdm.org/15/15923a/jdm15923a.html.

[3] Wikipedia, "Karl Popper," last modified on Sept. 26, 2016: https://en.wikipedia.org/wiki/Karl_Popper.

usenix

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Publish and Present Your Work

USENIX ATC ’17 and its co-located events are seeking submissions.

USENIX ATC ’17: 2017 USENIX Annual Technical Conference

July 12–14, 2017, Santa Clara, CA

Paper submissions due: February 7, 2017

www.usenix.org/atc17

USENIX ATC ‘17 will bring together leading systems researchers for cutting-edge systems research and unlimited opportunities to gain insight into a variety of must-know topics, including virtualization, system administration, cloud computing, security, and networking.

Authors are invited to submit original and innovative papers to the Refereed Papers Track of the 2017 USENIX Annual Technical Conference. We seek high-quality submissions that further the knowledge and understanding of modern computing systems with an emphasis on implementations and experimental results. We encourage papers that break new ground, present insightful results based on practical experience with computer systems, or are important, independent reproductions/refutations of the experimental results of prior work.

SOUPS 2017: Thirteenth Symposium on Usable Privacy and Security

July 12–14, 2017, Santa Clara, CA

Paper registrations due: March 1, 2017

www.usenix.org/soups2017

The 2017 Symposium on Usable Privacy and Security (SOUPS) will bring together an interdisciplinary group of researchers and practitioners in human computer interaction, security, and privacy. The program will feature:

- technical papers, including replication papers
- workshops and tutorials
- a poster session
- lightning talks

We invite authors to submit previously unpublished papers describing research or experience in all areas of usable privacy and security. We welcome a variety of research methods, including both qualitative and quantitative approaches.

Calls for posters, workshop proposals, tutorial proposals, and proposals for lightning talks are also available.

HotCloud ’17: 9th USENIX Workshop on Hot Topics in Cloud Computing

July 10-11, 2017, Santa Clara, CA

Paper submissions due: March 14, 2017

www.usenix.org/hotcloud17

HotCloud brings together researchers and practitioners from academia and industry working on cloud computing technologies to share their perspectives, report on recent developments, discuss research in progress, and identify new/emerging “hot” trends in this important area. While cloud computing has gained traction over the past few years, many challenges remain in the design, implementation, and deployment of cloud computing.

HotCloud is open to examining all models of cloud computing, including the scalable management of in-house servers, remotely hosted Infrastructure-as-a-Service (IaaS), infrastructure augmented with tools and services that provide Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

HotStorage ’17: 9th USENIX Workshop on Hot Topics in Storage and File Systems

July 10-11, 2017, Santa Clara, CA

Paper Submissions due: March 16, 2017

www.usenix.org/hotstorage17

The purpose of the HotStorage workshop is to provide a forum for the cutting edge in storage research, where researchers can exchange ideas and engage in discussions with their colleagues. The workshop seeks submissions that explore longer-term challenges and opportunities for the storage research community. Submissions should propose new research directions, advocate non-traditional approaches, or report on noteworthy actual experience in an emerging area. We particularly value submissions that effectively advocate fresh, unorthodox, unexpected, controversial, or counterintuitive ideas for advancing the state of the art.

Submissions will be judged on their originality, technical merit, topical relevance, and likelihood of leading to insightful discussions that will influence future storage systems research. In keeping with the goals of the HotStorage workshop, the review process will heavily favor submissions that are forward looking and open ended, as opposed to those that summarize mature work or are intended as a stepping stone to a top-tier conference publication in the short term.

Your Cores Are Slacking Off—Or Why OS Scheduling Is a Hard Problem

JEAN-PIERRE LOZI, BAPTISTE LEPERS, JUSTIN FUNSTON, FABIEN GAUD, VIVIEN QUÉMA, AND ALEXANDRA FEDOROVA

Jean-Pierre Lozi is an Associate Professor at the University of Nice Sophia-Antipolis, in the French Riviera. When he's not reading OS papers on the beach, you can usually find him around Château Valrose, teaching multicore programming, big data, and other hot topics. jplozi@unice.fr

Baptiste Lepers is a postdoc at EPFL. His research topics include performance profiling, optimizations for NUMA systems, multicore programming, and proofs of concurrent programs. baptiste.lepers@gmail.com

Justin Funston is a PhD student at the University of British Columbia and is advised by Alexandra Fedorova. His research interests include contention management on multicore and NUMA systems, parallel and high performance computing, and operating systems in general. jfunston@ece.ubc.ca

Fabien Gaud is a Senior Software Engineer at Coho Data, focusing on performance and scalability. He received his PhD in 2010 from Grenoble University, and from 2011 to 2014 he was a postdoctoral fellow at Simon Fraser University. me@fabiengaud.net

As a central component of resource management, the OS thread scheduler must make sure that ready threads are scheduled on available cores. As surprising as it may seem, we found that this simple rule is often broken in Linux. Cores may stay idle for seconds while ready threads are waiting in run queues, delaying applications and wasting energy. This phenomenon is not due to an intentional design but to performance bugs. These bugs can slow down scientific applications many-fold and degrade performance of workloads like kernel compilation and OLAP on a widely used commercial database by tens of percent, particularly on machines with a large number of cores. The root cause of the bugs is the increasing scheduler complexity, linked to rapid evolution in modern hardware. In this article, we describe the bugs and their effects and reflect on ways to combat them.

Our recent experience with the Linux scheduler revealed that the pressure to work around the challenging properties of modern hardware, such as non-uniform memory access (NUMA) latencies, high costs of cache coherency and synchronization, and diverging CPU and memory latencies, resulted in a scheduler with an incredibly complex implementation. As a result, the very basic function of the scheduler, which is to make sure that runnable threads use idle cores, fell through the cracks. We have discovered four performance bugs that cause the scheduler to leave cores idle while runnable threads are waiting for their turn to run. Resulting performance degradations are in the range 13–24% for typical Linux workloads, and reach many-fold slowdowns in some corner cases. In this article, we describe three of the four bugs. For the complete description, please refer to our extended paper [7].

Detecting the aforementioned bugs is difficult. They do not cause the system to crash or hang, but eat away at performance, often in ways that are difficult to notice with standard performance monitoring tools. For instance, when executing the OLAP workload TPC-H on a widely used commercial database, the symptom occurred many times throughout the execution, but each time it lasted only a few hundreds of milliseconds—too short to detect with tools like htop, sar, or perf. Yet, collectively, these occurrences did enough damage to slow down the most affected query by 23%. Even in cases where the symptom was present for a much longer duration, the root cause was difficult to discover because it was a result of many asynchronous events in the scheduler.

In the rest of this article we provide relevant background on the Linux scheduler, describe the bugs and their root causes, demonstrate their performance effects, and finally reflect on ways to combat them, focusing especially on the tools that were crucial for the bug discovery.

The Linux Scheduler

Linux's Completely Fair Scheduling (CFS) is an implementation of the weighted fair queueing (WFQ) scheduling algorithm, wherein the available CPU cycles of each core are divided among threads in proportion to their weights. To support this abstraction, CFS time-slices the CPU cycles among the running threads.

Vivien Quéma is a Professor at Grenoble INP (ENSIMAG). His research is about understanding, designing, and building (distributed) systems. He works on Byzantine fault tolerance, multicore systems, and P2P systems. vivien.quema@grenoble-inp.fr

Alexandra Fedorova is an Associate Professor at the University of British Columbia. In her day-to-day life, she measures and hacks operating systems, runtime libraries, and other system software. In her spare time she consults for MongoDB. sashs@ece.ucb.ca

The scheduler defines a fixed time interval during which each thread in the system must run at least once. The interval is divided among threads proportionally to their *weights*. The resulting interval (after division) is what we call the *timeslice*. A thread's weight is essentially its priority, or *niceness* in UNIX parlance. Threads with lower niceness have higher weights and vice versa.

When a thread runs, it accumulates `vruntime` (the runtime of the thread divided by its weight). Once a thread's `vruntime` exceeds its assigned timeslice, the thread is preempted from the CPU if there are other runnable threads available. A thread might also get preempted if another thread with a smaller `vruntime` is awoken.

Threads are organized in *run queues*. As a matter of efficiency, there is one run queue per core. When a core looks for a new thread to run, it picks the thread in its run queue that has the smallest `vruntime`.

For the overall system to be efficient, run queues must be kept balanced. To this end, CFS periodically runs a load-balancing algorithm that will keep the queues roughly balanced.

CFS balances run queues based on a metric called *load*, which is the combination of the thread's weight and its average CPU utilization. Intuitively, if a thread does not use much of a CPU, its load will be decreased accordingly. Additionally, the load-tracking metric accounts for varying levels of multithreading in different processes.

When a thread belongs to a group of threads (called a `cgroup`), its load is further divided by the total number of threads in its `cgroup`. `Cgroups` are used to group threads or processes that logically belong together, such as threads in the same application or processes launched from the same terminal (`tty`). This is done for fairness purposes, such that the CPU is shared among *applications* rather than individual instruction streams.

CFS implements a hierarchical load-balancing strategy. The cores are logically organized in a hierarchy, at the bottom of which is a single core. How the cores are grouped at the hierarchy's next levels depends on how they share the machine's physical resources. On the example machine illustrated in Figure 1, pairs of cores share functional units, such as the

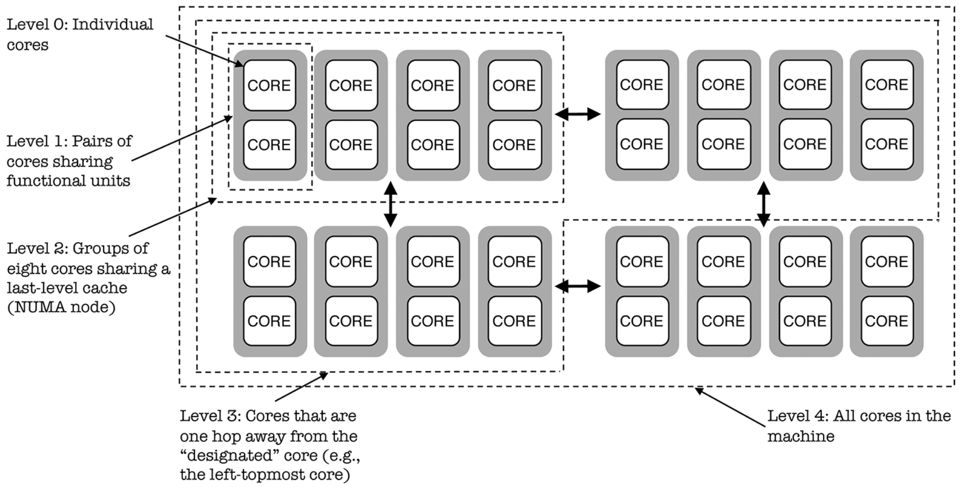

Level 0: Individual cores
Level 1: Pairs of cores sharing functional units
Level 2: Groups of eight cores sharing a last-level cache (NUMA node)
Level 3: Cores that are one hop away from the "designated" core (e.g., the left-topmost core)
Level 4: All cores in the machine
CORE

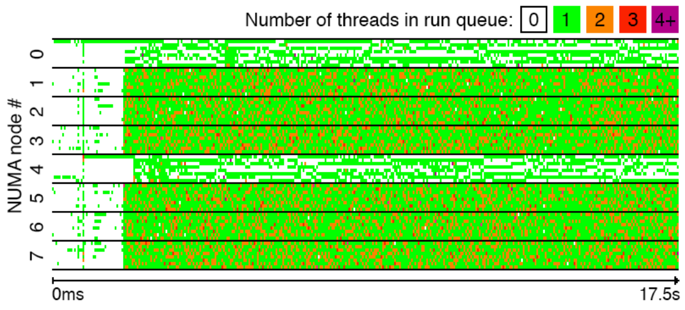
Figure 1: A machine with 32 cores, four NUMA nodes (eight cores per node sharing a last-level cache), and pairs of cores sharing a floating point unit. The dashed lines outline the scheduling domains as perceived by the left-topmost core. Level 3 of the hierarchy shows a group of three nodes: that is because these nodes are reachable from the left-topmost core in a single hop. (See Figure 3 for a detailed overview of node connectivity in our system.) At the fourth level, we have all the nodes of the machine, because they can be reached from the left-topmost core in at most two hops.

floating point unit, and groups of eight cores share a last-level cache; these groups of eight also form a NUMA node. As a result, at the second level of the hierarchy we have pairs of cores, and at the third level we have NUMA nodes. NUMA nodes are further grouped according to their level of connectivity. This is where things become a bit tricky, because the hierarchy is constructed from the point of view of a particular "designated" core; in the load-balancing algorithm it is the core that performs load balancing. In Figure 1 the hierarchy levels are shown as if the left-topmost core were designated. Hence, the third level of the hierarchy includes all nodes that can be reached from that designated core in one hop. The fourth level includes the nodes that can be reached from it in at most two hops, i.e., all nodes in the system.

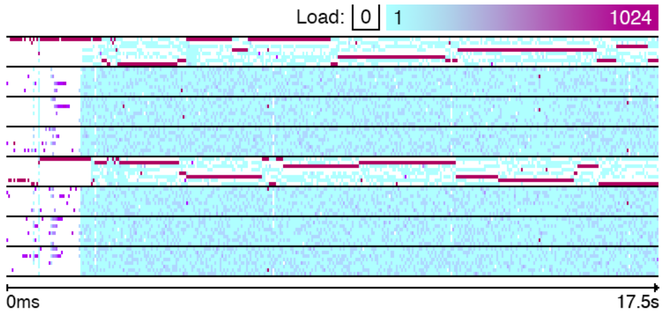
Each level of the hierarchy is called a *scheduling domain*. If a scheduling domain includes sub-domains, such as the NUMA-node domain including core-pair domains, those sub-domains are referred to in Linux terminology as *scheduling groups*.

At the high level, the load-balancing algorithm works as follows. Load balancing is run for each scheduling domain, starting from the lowest level of the hierarchy that contains more than a single core (the pair-of-cores level in our example) to the top. At each level, the algorithm is run by the designated core. The core is designated if it is either the first idle core of the domain or, if none of the cores are idle, the core with the lowest ID in the domain. The designated core computes the average load for each scheduling group of the domain and picks the busiest group, based on the load and on heuristics that favor overloaded and imbalanced groups. If the load of the busiest group is higher than the load of the designated core's home group, the designated core steals threads from the busiest group so as to balance the load.

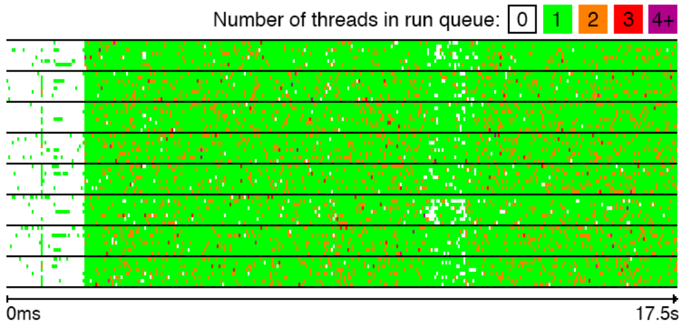
The scheduler implements a set of optimizations to improve the efficiency of the load-balancing mechanism, but as we will see later they increase complexity and nurture bugs. For example, in earlier versions of Linux, idle cores, which are typically transitioned into a lower power state, were always awoken on every clock tick; at this point they would run the load-balancing algorithm. Since version 2.6.21, Linux included the option, now enabled by default, to avoid periodically waking up sleeping cores. It is the responsibility of overloaded cores to wake up the sleeping cores when needed. Another set of optimizations has to do with the placement of threads that become unblocked. Normally when a thread wakes up, after sleeping or waiting for a resource like a lock or I/O, the scheduler tries to place it on the idlest core. However, when a thread is awoken by another "waker" thread, the scheduler will favor cores sharing a cache with the waker thread to improve cache reuse.



(a) # threads in each core's run queue over time



(b) Load of each core's run queue over time



(c) Same as (a), with fix applied

Figure 2: The Group Imbalance bug. The y-axis shows CPU cores. Nodes are numbered 0–7. Each node contains eight cores.

The Group Imbalance Bug

The first bug we encountered is illustrated in Figures 2a and 2b. The figures show the state of the scheduler when we execute a workload on an eight-node NUMA system summarized in Table 1. The x-axis shows the time and the y-axis shows the cores, grouped by their node number. In the time period shown in the figure, the machine was executing a compilation of the kernel (`make` with 64 threads) and running two `R` processes (each with one thread). The `make` and the two `R` processes were launched from three different `ssh` connections (i.e., three different `ttys`). Figure 2a is a heatmap showing the number of threads in each

CPUs	4 × 16-core Opteron 6272 CPUs (64 threads in total)
Clock rate	2.1 GHz
Caches	64 KB shared L1 i-cache
(Each core)	16 KB L1 d-cache
	2 MB shared L2
	8 MB shared L3
Memory	512 GB of 1.6 GHz DDR-3
Interconnect	HyperTransport 3.0 (see Figure 3)

Table 1: Hardware of our AMD Bulldozer machine

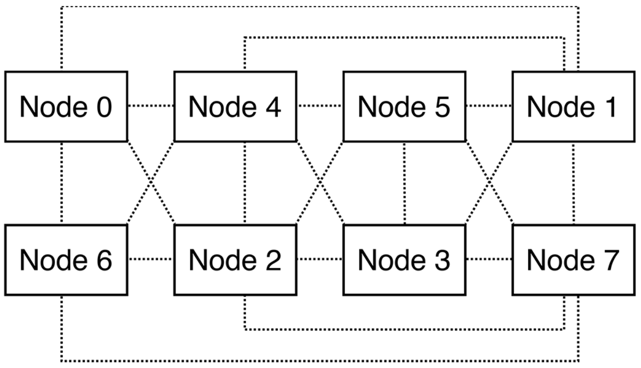


Figure 3: Topology of our 8-node AMD Bulldozer machine

core's run queue over time. The chart shows that there are two nodes (zero and four) whose cores run either only one thread or no threads at all, while the rest of the nodes are overloaded, with many of the cores having two threads in their run queue.

After investigation, we found that the scheduler is not balancing the load properly. Remember that a thread's load is a combination of its weight and its CPU utilization. Threads launched from the same `tty` belong to the same `cgroup`, and their load is thus divided by the number of threads in their `cgroup`. As a result, a thread in the 64-thread `make` process has a load roughly 64 times smaller than a thread in a single-threaded `R` process.

Discrepancies between threads' loads are illustrated in Figure 2b, which shows the combined load of threads in each core's run queue: a darker color corresponds to a higher load. Nodes 0 and 4, the ones running the `R` processes, each have one core with a very high load. These are the cores that run the `R` threads.

The Linux load balancer steals work from other run queues based on load; obviously the underloaded cores in Nodes 0 and 4 should not steal from the overloaded core in their own node, because that core runs only a single thread. However, they must be able to steal from the more loaded cores in other nodes. This is not happening for the following reason. Remember that to limit algorithmic complexity, the load-balancing algorithm uses a hierarchical design. When a core attempts to steal work from another node or, in other words, from another scheduling group, it does not examine the load of every core in that group, it only looks at the group's *average* load. If the average load of the victim group is greater than that of its own, it will attempt to steal threads from that group; otherwise it will not. In our case, the idle core looking for work is in the same group as the high-load `R` thread. So the average load for that group is actually the same as the load of the group with many overloaded cores. As a result, no stealing occurs, despite the victim group having overloaded cores with waiting threads.

To fix this bug, we changed the part of the algorithm that compares the load of scheduling groups. Instead of comparing the average loads, we compare the *minimum* loads. The minimum load is the load of the least loaded core in that group. Intuitively, if the minimum load of one scheduling group is lower than the minimum load of another scheduling group, it means that the first scheduling group has a core that is less loaded than *any* core in the other group, and thus a core in the first group must steal from the second.

Figure 2c is a visualization of the same workload after we fixed the bug (showing a heatmap of run queue sizes, in the same fashion as Figure 2a). We observe that the imbalance disappears. With the fix, the completion time of the `make` job, in the `make/R` workload decreased by 13%. Performance impact could be much higher in other circumstances. For example, in a workload running `lu` from the NPB (NASA Advanced Supercomputing Parallel Benchmarks) suite with 60 threads, and four single-threaded `R` processes, `lu` ran 13x faster after fixing the Group Imbalance bug. `lu` experienced a super-linear speedup, because the bug exacerbated lock contention when multiple `lu` threads ran on the same core.

The Scheduling Group Construction Bug

Linux defines a command, called `taskset`, that enables pinning applications to run on a subset of the available cores. The bug we describe in this section occurs when an application is pinned on nodes that are two hops apart. For example, in Figure 3, which demonstrates the topology of our NUMA machine, Nodes 1 and 2 are two hops apart. The bug will prevent the load-balancing algorithm from migrating threads between these two nodes.

The bug results from the way scheduling groups are constructed, which is not adapted to modern NUMA machines such as the one we use in our experiments. In brief, the groups are constructed from the perspective of a specific core (Core 0), whereas they should be constructed from the perspective of the core responsible for load balancing on each node, the `designated` core. We explain with an example.

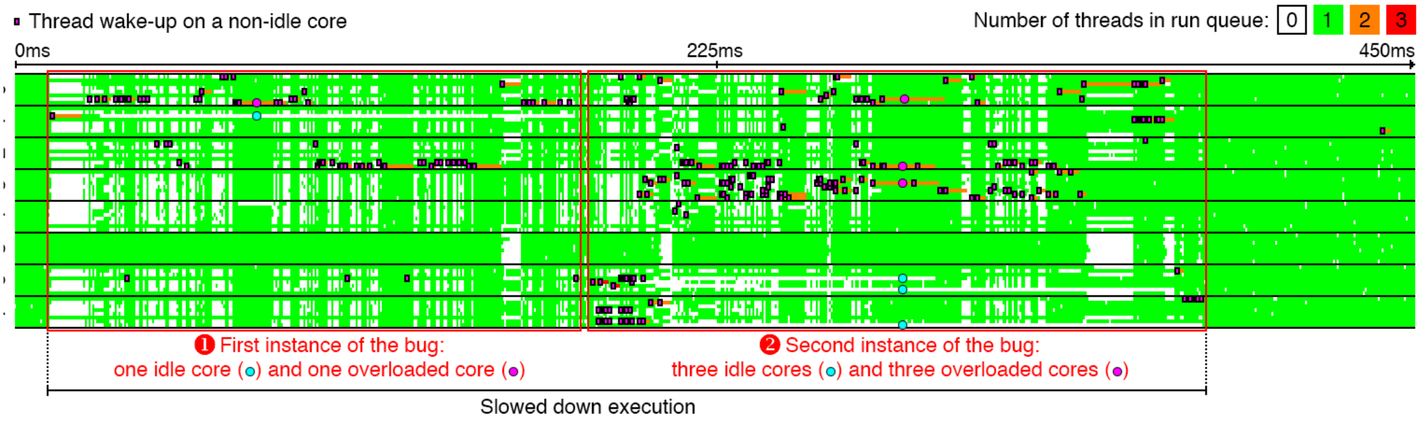


Figure 4: Several instances of the Overload-on-Wakeup bug

Let us walk through the key steps of the load-balancing algorithm when the balancing is performed at the top of the hierarchy, i.e., at the scheduling domain including all the machine's nodes. The algorithm will construct the scheduling groups (the sub-domains) included within that scheduling domain. The first scheduling group on the machine in Figure 3 will include Node 0 plus all the nodes that are one hop apart from Node 0, namely Nodes 1, 2, 4, and 6. The second group will include the lowest-numbered node that was not included in the first group: Node 3, in this case, and all nodes that are one hop apart from Node 3: Nodes 1, 2, 4, 5, 7. The two scheduling groups are thus: {0, 1, 2, 4, 6} and {1, 2, 3, 4, 5, 7}.

Suppose that an application is pinned on Nodes 1 and 2 and that all of its threads are being *created* on Node 1. Eventually we would like the load to be balanced between Nodes 1 and 2. However, when a core in Node 2 looks for work to steal, it will compare the load between the two scheduling groups shown earlier. Since each scheduling group contains both Nodes 1 and 2, the average loads will be the same, so Node 2 will not steal any work!

The bug originates from an attempt to improve the performance of Linux on large NUMA systems. Before the introduction of the bug, Linux would balance the load inside NUMA nodes and then across all NUMA nodes. New levels of hierarchy (nodes one hop apart, nodes two hops apart, etc.) were introduced to increase the likelihood for threads to remain close to their original NUMA node.

To fix the bug, we modified the construction of scheduling groups so that each core uses scheduling groups constructed from its own perspective. After the fix, the cores were able to detect the imbalance and to steal the work. Table 2 presents the performance difference in NPB applications with and without the Scheduling Group Construction bug. Applications are launched on two nodes with as many threads as there are cores. The maximum slowdown of 27x is experienced by `lu`. The slowdown is a lot more than the expected 2x because of locking effects.

Application	Time w/ bug	Time w/o bug	Speedup
	(sec)	(sec)	factor (x)
bt	99	56	1.75
cg	42	15	2.73
ep	73	36	2
ft	96	50	1.92
is	271	202	1.33
lu	1040	38	27
mg	49	24	2.03
sp	31	14	2.23
ua	206	56	3.63

Table 2: Execution time of NPB applications with the Scheduling Group Construction bug and without it

The Overload-on-Wakeup Bug

The gist of this bug is that a thread that was asleep may wake up on an overloaded core while other cores in the system are idle. The bug was introduced by an optimization in the wakeup code (`select_task_rq_fair` function). When a thread goes to sleep on Node *X* and the thread that wakes it up later is running on that same node, the scheduler only considers the cores of Node *X* for scheduling the awakened thread. If all cores of Node *X* are busy, the thread will wake up on an already busy core and miss opportunities to use idle cores on other nodes. This can lead to a significant under-utilization of the machine, especially on workloads where threads frequently wait.

The rationale behind this optimization is to maximize cache reuse. Essentially, the scheduler attempts to place the woken up thread physically close to the waker thread, e.g., so both run on cores sharing a last-level cache, in consideration of producer-consumer workloads where the woken up thread will consume the data produced by the waker thread. This seems like a reasonable idea, but for some workloads, waiting in the run queue for the sake of better cache reuse does not pay off.

This bug was triggered by and affected the runtime of a widely used commercial database configured with 64 worker threads (one thread per core) and running an OLAP (TPC-H) workload.

Figure 4 illustrates several instances of the Overload-on-Wakeup bug. During the first time period (1), one core is idle while a thread that ideally should be scheduled on that core keeps waking up on other cores, which are busy. During the second time period (2), there is a triple instance of the bug: three cores are idle for a long time, while three extra threads that should be scheduled on those cores keep waking up on other busy cores. The Overload-on-Wakeup bug is typically caused when a transient thread is scheduled on a core that runs a database thread. When this happens, the load balancer observes a heavier load on the node that runs the transient thread (Node *A*) and migrates one of the threads to another node (Node *B*). This is not an issue if the transient thread is the one being migrated, but if it is the database thread, then the Overload-on-Wakeup bug will kick in. Node *B* now runs an extra database thread, and threads of Node *B*, which often sleep and wake up, keep waking up on that node, even if there are no idle cores on that node. This occurs because the wakeup code only considers cores from the local node for the sake of better cache reuse and results in a core running multiple threads when some cores that are always idle are present on other nodes.

To fix this bug, we alter the code that is executed when a thread wakes up. We wake up the thread on the local core—i.e., the core where the thread was scheduled last—if it is idle; otherwise, if there are idle cores in the system, we wake up the thread on the core that has been idle for the longest amount of time. If there are no idle cores, we fall back to the original algorithm to find the core where the thread will wake up.

Our bug fix improves performance by 22.6% on the 18th query of TPC-H, and by 13.2% on the full TPC-H workload.

Discussions and Lessons Learned

The first question to ask is whether these bugs could be fixed with a new, cleaner scheduler design that is less error-prone and easier to debug, but still maintains the features we have today. Historically, though, this does not seem like a long-term solution, in addition to the fact that the new design would need to be implemented and tested from scratch. The Linux scheduler has gone through a couple of major redesigns. The original scheduler had high algorithmic complexity, which resulted in poor performance when highly multithreaded workloads became common. In 2001, it was replaced by a new scheduler with O(1) complexity and better scalability on SMP systems. It was initially successful but soon required modifications for new architectures like NUMA and SMT (simultaneous multithreading). At the same time, users wanted better support for desktop use cases such as interactive and audio applications, which required more changes to the scheduler. Despite numerous modifications and proposed heuristics, the O(1) scheduler was not able to meet expectations and was replaced by CFS in 2007. Interestingly, CFS sacrifices O(1) complexity for O(log (# threads)), but it was deemed worthwhile to provide the desired features.

As the hardware and workloads became more complex, CFS, too, succumbed to bugs. The addition of autogroups (i.e., the automatic grouping of threads from the same `tty` into a `cgroup`) coupled with hierarchical load balancing introduced the Group Imbalance bug. Asymmetry in new, increasingly complex NUMA systems triggered the Scheduling Group Construction bug. Cache-coherency overheads on modern multi-node machines motivated the cache locality optimization that caused the Overload-on-Wakeup bug.

The takeaway is that new scheduler designs come and go. However, a new design, even if clean and purportedly bug-free initially, is not a long-term solution. Linux is a large open-source system developed by dozens of contributors. In this environment, we will inevitably see new features and "hacks" retrofitted into the source base to address evolving hardware and applications.

The recently released Linux 4.3 kernel features a new implementation of the load metric. This change is reported to be "done in a way that significantly reduces complexity of the code" [1]. Simplifying the load metric could get rid of the Group Imbalance bug, which is directly related to it. However, we confirmed, using our tools (see [7] for a full description of our tools and the end of this article for a link to the code), that the bug is still present.

Kernel developers rely on mutual code review and testing to prevent the introduction of bugs. This could potentially be effective for bugs, like the Scheduling Group Construction bug, that are easier to spot in the code (of course, it still was not effective in these cases), but it is unlikely to be reliable for the more arcane types of bugs.

Catching these bugs with testing or conventional performance monitoring tools is tricky. They do not cause the system to crash or to run out of memory, but they do silently eat away at performance. As we have seen with the Group Imbalance and the Overload-on-Wakeup bugs, they introduce short-term idle periods that "move around" between different cores. These microscopic idle periods cannot be noticed with performance monitoring tools like `htop`, `sar`, or `perf`. Standard performance regression testing is also unlikely to catch these bugs, as they occur in very specific situations (e.g., multiple applications with different thread counts launched from distinct `ttys`). In practice, performance testing on Linux is done with only one application running at a time on a dedicated machine—this is the standard way of limiting factors that could explain performance differences.

One of the most important lessons we learned in the process of finding and diagnosing these bugs is that it was crucially important to have the tools that trace and visualize microscopic events during the execution, such as the state of the kernel run queues and the transitions of threads between the cores. The visualizations that we relied on for detection and diagnosis of the bugs are shown in Figures 2 and 4. We built our own tools that perform precise tracing of kernel events and plot them as the space/time charts shown earlier.

Although we found it convenient to build our own tools, there is also a variety of powerful third-party dynamic tracing tools, such as DTrace, LTTNG, Event Tracing for Windows, Ftrace, and SystemTap. Unfortunately, effective visual front-ends and trace analysis tools that are necessary to make the traces useful are lacking. Most of the user-friendly performance tools available today rely on sampling and display averages and aggregates, which is not powerful enough for detecting performance anomalies like those caused by the scheduler bugs. We strongly feel that the software engineering community must embrace dynamic tracing and visualization for efficient diagnosis of egregious performance anomalies.

Reflection on the Future of OS Scheduling

The bugs we described resulted from increasingly more optimizations in the scheduler, whose purpose was mostly to cater to complexity of modern hardware. As a result, the scheduler, that once used to be a simple isolated part of the kernel, grew into a complex monster whose tentacles reached into many other parts of the system, such as power and memory management. The optimizations studied in this paper are part of the mainline Linux, but even more scheduling optimizations were proposed in the research community.

Since 2000, dozens of papers have described new scheduling algorithms catering to resource contention, coherency bottlenecks, and other idiosyncrasies of modern multicore systems. There were algorithms that scheduled threads so as to minimize contention for shared caches, memory controllers, and multithreaded CPU pipelines [2, 6, 8]. There were algorithms that reduced communication distance among threads sharing data [10]. There were algorithms that addressed scheduling on asymmetric multicore CPUs [4, 9] and algorithms that integrated scheduling with the management of power and temperature [3]. Finally, there were algorithms that scheduled threads to minimize communication latency on systems with an asymmetric interconnect [5]. All of these algorithms showed positive benefits, either in terms of performance or power, for some real applications. However, few of them were adopted in mainstream operating systems, mainly because it is not clear how to integrate all these ideas in the scheduler safely.

If every good scheduling idea is slapped as an add-on to a single monolithic scheduler, we risk more complexity and more bugs, as we saw from the case studies in this paper. Rapid evolution of hardware that we are witnessing today will motivate more and more scheduler optimizations. Instead of producing yet another monolithic scheduler design, what we may need is to switch to a more modular architecture..

One possible avenue is to decouple time management from space management. Historically, on single-core systems, the scheduler was tasked with managing time, that is, sharing the CPU cycles among the threads. On multicore systems, the scheduler evolved to also manage space, that is to decide where to place the threads. Several researchers postulated that space management need not be done at as fine a time granularity as time management, and this idea becomes more and more feasible in the age where machines are evolving to have more cores than most applications need. Perhaps space management could be done at coarse time intervals by placing groups of threads on subsets of cores such that each thread always has a free core whenever it needs one. Then time management would be largely out of the picture, and the space manager would deal with issues like NUMA locality and resource contention. It would adjust the mapping of threads to cores somewhat infrequently to reflect the changes in application behavior over time. Still, understanding how to combine different, perhaps conflicting, space optimizations and reason about how they interact remains an open research problem.

Our bug fixes and tools are available at http://git.io/vaGOW.

References

[1] Michael Larabel, "The Linux 4.3 Scheduler Change 'Potentially Affects Every SMP Workload in Existence,'" Phoronix, September 2015: https://www.phoronix.com/scan.php?page=news_item&px=Linux-4.3-Scheduler-SMP.

[2] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A Case for NUMA-Aware Contention Management on Multicore Systems," in *Proceedings of the 2011 USENIX Annual Technical Conference (USENIX ATC '11):* https://www.usenix.org/legacy/events/atc11/tech/final_files/Blagodurov.pdf.

[3] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI),* 2004: https://engineering.purdue.edu/~vijay/papers/2004/heat-and-run.pdf.

[4] D. Koufaty, D. Reddy, and S. Hahn, "Bias Scheduling in Heterogeneous Multi-Core Architectures," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10)*: http://eurosys2010.sigops-france.fr/proceedings/docs/p125.pdf.

[5] B. Lepers, V. Quéma, and A. Fedorova, "Thread and Memory Placement on NUMA Systems: Asymmetry Matters," in *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC '15):* https://www.usenix.org/system/files/conference/atc15/atc15-paper-lepers.pdf.

[6] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, "Efficient Operating System Scheduling for Performance-Asymmetric Multi-Core Architectures," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC '07):* http://happyli.org/Tong/papers/amps.pdf.

[7] J. P. Lozi, B. Lepers, J. R. Funston, F. Gaud, V. Quéma, and A. Fedorova, "The Linux Scheduler: A Decade of Wasted Cores," in *Proceedings of the 11th European Conference on Computer System (EuroSys 2016),* pp. 1:1–1:16: http://www.ece.ubc.ca/~sasha/papers/eurosys16-final29.pdf.

[8] K. K. Pusukuri, D. Vengerov, A. Fedorova, and V. Kalogeraki, "FACT: A Framework for Adaptive Contention-Aware Thread Migrations," in *Proceedings of the 8th ACM International Conference on Computing Frontiers (CF '11):* https://www.cs.sfu.ca/~fedorova/papers/cf150-pusukuri.pdf.

[9] J. C. Saez, M. Prieto, A. Fedorova, and S. Blagodurov, "A Comprehensive Scheduler for Asymmetric Multicore Systems," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10):* https://www.cs.sfu.ca/~fedorova/papers/eurosys163-saez.pdf.

[10] D. Tam, R. Azimi, and M. Stumm, "Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07):* http://www.cs.toronto.edu/~demke/2227/S.14/Papers/p47-tam.pdf.

Serverless Computation with OpenLambda

SCOTT HENDRICKSON, STEPHEN STURDEVANT, EDWARD OAKES, TYLER HARTER, VENKATESHWARAN VENKATARAMANI, ANDREA C. ARPACI-DUSSEAU, AND REMZI H. ARPACI-DUSSEAU

Scott Hendrickson is a Software Engineer at Google, working on the future of wireless Internet infrastructure. Formerly, he was a student at the University of Wisconsin-Madison where he participated in the OpenLambda project while working on his Bachelor of Science in computer engineering and computer science. research@shendrickson.com

Stephen Sturdevant is a student at the University of Wisconsin-Madison, where he has received his bachelor's degree and is currently pursuing a computer science PhD. Professors Andrea and Remzi Arpaci-Dusseau are his advisers. Stephen has recently participated in Google Summer of Code and is a contributor to the OpenLambda project. stephensturdevant@gmail.com

Edward Oakes is a student at the University of Wisconsin-Madison, where he is pursuing an undergraduate degree in computer science and mathematics. He is advised by Professor Remzi Arpaci-Dusseau. This fall, Edward will be continuing his work on OpenLambda at the Microsoft Gray Systems Lab in Madison, WI. oakes@cs.wisc.edu

Tyler Harter is a recent PhD graduate from the University of Wisconsin-Madison, where he was co-advised by Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau. He has published several storage papers at FAST and SOSP, including an SOSP '11 Best Paper. He has joined the Microsoft Gray Systems Lab, where he will continue contributing to OpenLambda (https://github.com/open-lambda). tylerharter@gmail.com

Rapid innovation in the datacenter is once again set to transform how we build, deploy, and manage Web applications. New applications are often built as a composition of microservices, but, as we will show, traditional containers are a poor fit for running these microservices. We argue that new serverless compute platforms, such as AWS Lambda, provide better elasticity. We also introduce OpenLambda, an open-source implementation of the Lambda model, and describe several new research challenges in the area of serverless computing.

In the preconsolidated datacenter, each application often ran on its own physical machine. The high costs of buying and maintaining large numbers of machines, and the fact that each was often underutilized, led to a great leap forward: virtualization. Virtualization enables tremendous consolidation of services onto servers, thus greatly reducing costs and improving manageability.

However, hardware-based virtualization is not a panacea, and lighter-weight technologies have arisen to address its fundamental issues. One leading solution in this space is *containers*, a server-oriented repackaging of UNIX-style processes with additional namespace virtualization. Combined with distribution tools such as Docker [8], containers enable developers to readily spin up new services without the slow provisioning and runtime overheads of virtual machines.

Common to both hardware-based and container-based virtualization is the central notion of a *server*. Servers have long been used to back online applications, but new cloud-computing platforms foreshadow the end of the traditional backend server. Servers are notoriously difficult to configure and manage [3], and server startup time severely limits elasticity.

As a result, a new model called *serverless computation* is poised to transform the construction of modern scalable applications. Instead of thinking of applications as collections of servers, developers define applications as a set of functions with access to a common datastore. An excellent example of this *microservice*-based platform is found in Amazon's Lambda [1]; we thus generically refer to this style of service construction as the Lambda model.

The Lambda model has many benefits as compared to traditional server-based approaches. Lambda handlers from different customers share common pools of servers managed by the cloud provider, so developers need not worry about server management. Handlers are typically written in languages such as JavaScript or Python; by sharing the runtime environment across functions, the code specific to a particular application will typically be small and easily deployable on any worker in a Lambda cluster. Finally, applications can scale up rapidly without needing to start new servers. The Lambda model represents the logical conclusion of the evolution of sharing between applications, from hardware to operating systems to (finally) the runtime environments themselves (Figure 1).

Venkat Venkataramani is the CEO and co-founder of Rockset, an infrastructure startup based in Menlo Park building a high performance cloud-first data service. Prior to Rockset, he was an Engineering Director in the Facebook infrastructure team responsible for all online data services that stored and served Facebook user data. Before Facebook, he worked at Oracle on the RDBMS for 5+ years after receiving his master's degree at UW-Madison. venkat@rockset.io

Andrea Arpaci-Dusseau is a Full Professor of Computer Sciences at the University of Wisconsin-Madison. She is an expert in file and storage systems, having published more than 80 papers in this area, co-advised 19 PhD students, and received nine Best Paper awards; for her research contributions, she was recognized as a UW-Madison Vilas Associate. She also created a service-learning course in which UW-Madison students teach CS to more than 200 elementary-school children each semester. dusseau@cs.wisc.edu

Remzi Arpaci-Dusseau is a Full Professor in the Computer Sciences Department at the University of Wisconsin-Madison. He co-leads a group with his wife, Professor Andrea Arpaci-Dusseau. They have graduated 19 PhD students in their time at Wisconsin, won nine Best Paper awards, and some of their innovations now ship in commercial systems and are used daily by millions of people. Remzi has won the SACM Student Choice Professor of the Year award four times, the Carolyn Rosner "Excellent Educator" award, and the UW-Madison Chancellor's Distinguished Teaching award. Chapters from a freely available OS book he and Andrea co-wrote, found at http://www.ostep.org, have been downloaded millions of times in the past few years. remzi@cs.wisc.edu

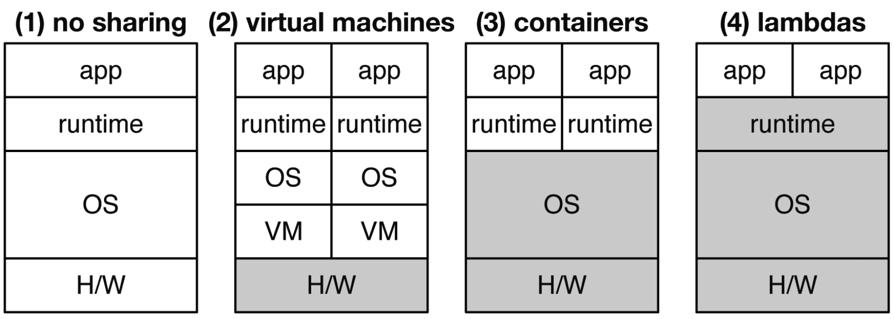


Figure 1: Evolution of sharing. Gray layers are shared.

There are many new research challenges in the context of serverless computing, with respect to efficient sandboxing, cluster scheduling, storage, package management, and many other areas. In order to explore these problems, we are currently building OpenLambda, a base upon which researchers can evaluate new approaches to serverless computing. More details can be found in Hendrickson et al. [5]. Furthermore, while research is a primary motivation for building OpenLambda, we plan to build a production-quality platform that could be reasonably deployed by cloud providers.

AWS Lambda Background

AWS Lambda allows developers to specify functions that run in response to various events. We focus on the case where the event is an RPC call from a Web application and the function is an RPC handler. A developer selects a runtime environment (for example, Python 2.7), uploads the handler code, and associates the handler with a URL endpoint. Clients can issue RPC calls by issuing requests to the URL.

Handlers can execute on any worker; in AWS, start-up time on a new worker is approximately one to two seconds. Upon a load burst, a load balancer can start a Lambda handler on a new worker to service a queued RPC call without incurring excessive latencies. However, calls to a particular Lambda are typically sent to the same worker(s) to avoid sandbox reinitialization costs [10]. Developers can specify resource limits on time and memory. In AWS, the cost of an invocation is proportional to the memory cap multiplied by the actual execution time, as rounded up to the nearest 100 ms.

Lambda functions are essentially stateless; if the same handler is invoked on the same worker, common state may be visible between invocations, but no guarantees are provided. Thus, Lambda applications are often used alongside a cloud database.

Motivation for Serverless Compute

A primary advantage of the Lambda model is its ability to quickly and automatically scale the number of workers when load suddenly increases. To demonstrate this, we compare AWS Lambda to a container-based server platform, AWS Elastic Beanstalk (hereafter Elastic BS). On both platforms we run the same benchmark for one minute: the workload maintains 100 outstanding RPC requests and each RPC handler spins for 200 ms.

Figure 2 shows the result: an RPC using AWS Lambda has a median response time of only 1.6 sec, whereas an RPC in Elastic BS often takes 20 sec. While AWS Lambda was able to start 100 unique worker instances within 1.6 sec to serve the requests, all Elastic BS requests were served by the same instance; as a result, each request in Elastic BS had to wait behind 99 other 200 ms requests.

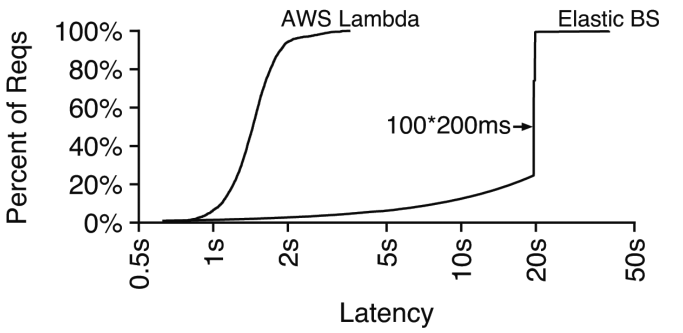


Figure 2: Response time. This CDF shows measured response times from a simulated load burst to an Elastic BS application and to an AWS Lambda application.

AWS Lambda also has the advantage of not requiring configuration for scaling. In contrast, Elastic BS configuration is complex, involving 20 different settings for scaling alone. Even though we tuned Elastic BS to scale as fast as possible (disregarding monetary cost), it still failed to spin up new workers for several minutes.

OpenLambda Overview

We now introduce OpenLambda, our open-source implementation of the Lambda model. Figure 3 illustrates how various servers and users interact in an OpenLambda cluster during the upload of a Lambda function F and a first call to that function. First, a *developer* uploads the Lambda code to the Lambda service, which stores it in a *code store*. Second, a *client* may issue an RPC to the service, via AJAX, gRPC, or some other protocol. A load balancer must decide which worker machine should service the request. In order to implement certain locality heuristics, the balancer may need to request the RPC schema from the code store in order to perform deep inspection on the RPC fields.

The *OpenLambda worker* that receives the request will then fetch the RPC handling code from the code store if it is not already cached locally. The worker will initialize a sandbox in which to run the handler. The handler may issue queries to a distributed database; if the balancer and database are integrated, this will hopefully involve I/O to a local shard.

There are different ways to implement worker sandboxes, but OpenLambda, like AWS Lambda, currently uses containers. Figure 4 shows how the Lambda model avoids common overheads faced by standard container use cases. Normally, each application runs inside a container, with its own server and runtime environment. Thus, application startup often involves deploying runtime engines to new machines and starting new servers. In contrast, servers run outside the containers with the Lambda model, so there is no server spinup overhead. Furthermore, many applications will share a small number of standard runtime engines. Although multiple instances of those runtime engines will run in each sandbox, the runtime engine code will already be on every worker, typically in memory.

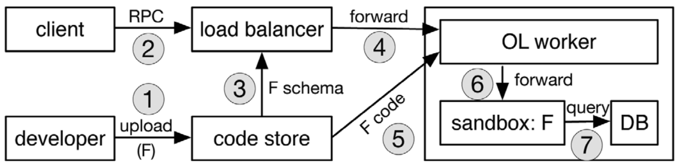


Figure 3: OpenLambda architecture. A new Lambda handler is uploaded, then called.

Tutorial: Running OpenLambda in development mode (with only a worker and no load balancer or code store) is relatively simple in our current pre-release:

```
# build and run standalone OL worker
curl -L -O https://github.com/open-lambda/open-lambda/archive
/v0.1.1.tar.gz
tar -xf v0.1.1.tar.gz
cd open-lambda-0.1.1
./quickstart/deps.sh
make
./bin/worker quickstart/quickstart.json

# from another shell, issue AJAX w/ curl
curl -X POST localhost:8080/runLambda/hello -d
'{"name":"alice"}'
```

Code for new handlers can be written in the `./quickstart/handlers` directory, but the worker must be restarted upon a handler update. RPC calls can be issued via AJAX `curl` POSTs, with the URL updated to reflect the handler name. Directions for running a full OpenLambda cluster are available online: https://www.open-lambda.org.

Research Agenda

We now explore a few of the new research problems in the serverless-computing space.

Lambda Workloads

Characterizing typical Lambda workloads will be key to the design of OpenLambda and other serverless compute platforms. Unfortunately, the Lambda model is relatively new, so there are not yet many applications to study. However, we can anticipate how future workloads may stress Lambda services by analyzing the RPC patterns of existing applications.

In this section, we take Google Gmail as an example and study its RPC calls during inbox load. Gmail uses AJAX calls (an RPC protocol based on HTTP requests that uses JSON to marshal arguments) to fetch dynamic content.

Figure 5 shows Gmail's network I/O over time, divided between GETs and POSTs. Gmail mostly uses POSTs for RPC calls and GETs for other requests; the RPC calls represent 32% of all requests and tend to take longer (92 ms median) than other requests (18 ms median).

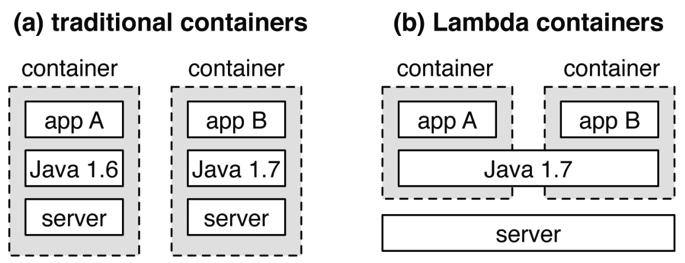


Figure 4: Container usage. The dashed lines represent container boundaries.

The average time for short RPCs (those under 100 ms) is only 27 ms. Since we only trace latency on the client side, we cannot know how long the requests were queued at various stages; thus, our measurements represent an upper bound on the actual time for the RPC handler. On AWS Lambda, charges are in increments of 100 ms, so these requests will cost at least 3.7x more than if charges were more fine-grained.

We also see a very long request that takes 231 seconds, corresponding to 93% of the cumulative time for all requests. Web applications often issue such long-lived RPC calls as a part of a *long polling* technique. When the server wishes to send a message to the client, it simply returns from the long RPC [2]. Unless Lambda services provide special support for these calls, idle handlers will easily dominate monetary costs.

Execution Engine

In the motivation section, we saw that Lambdas are far more elastic than containers. Unfortunately, under steady load, containers tend to be faster. In our experiments [5], Elastic BS request latencies are an order of magnitude shorter than AWS Lambda latencies. If Lambdas are to compete with VM and container platforms, base execution time must be improved.

Optimizing sandbox initialization and management is key to improving Lambda latencies. For example, AWS Lambda reuses the same sandbox for different calls when possible to amortize startup costs; between requests, containers are maintained in a paused state [10].

Unfortunately, there are difficult tradeoffs regarding when to garbage-collect paused containers. Resuming a paused container is over 100x faster than starting a new container, but keeping a container paused imposes the same memory overheads as an active container [5]. Reducing the time cost of fresh starts and reducing memory overheads of paused containers are both interesting challenges.

Interpreted Languages

Most Lambdas are written in interpreted languages. For performance, the runtimes corresponding to these languages typically have just-in-time compilers. JIT compilers have been built for Java, JavaScript, and Python that optimize compiled code based on dynamic profiling or tracing.

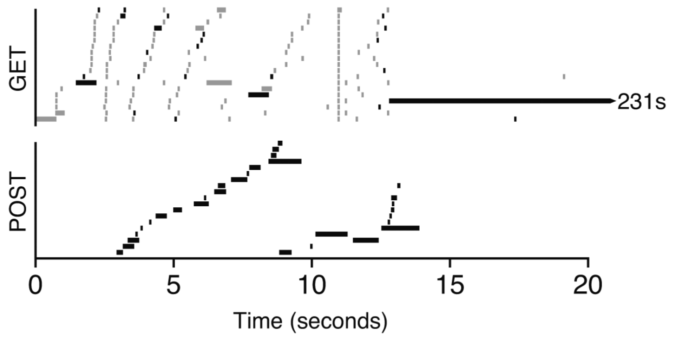


Figure 5: Google Gmail. Black bars represent RPC messages; gray bars represent other messages. The bar ends represent request and response times. The bars are grouped as POSTs and GETs; vertical positioning is otherwise arbitrary.

Applying these techniques with Lambdas is challenging because a single handler may run many times over a long period in a Lambda cluster, but it may not run long enough on any one machine to provide sufficient profiling feedback. Making dynamic optimization effective for Lambdas may require sharing profiling data between different Lambda workers.

Package Support

Lambdas can rapidly spin up because customers are encouraged to use one of a few runtime environments; runtime binaries will already be resident in memory before a handler starts. Of course, this benefit disappears if users bundle large third-party libraries inside their handlers, as the libraries need to be copied over the network upon a handler invocation on a new Lambda worker. Lazily copying packages could partially ameliorate this problem [4].

Alternatively, the Lambda platform could be package aware [7] and provide special support for certain popular package repositories, such as `npm` for Node.js or `pip` for Python. Of course, it would not be feasible to keep such large (and growing) repositories in memory on a single Lambda worker, so package awareness would entail new code locality challenges for scheduling.

Cookies and Sessions

Lambdas are inherently short-lived and stateless, but users typically expect to have many different but related interactions with a Web application. Thus, a Lambda platform should provide a shared view of cookie state across calls originating from a common user.

Furthermore, during a single session, there is often a two-way exchange of data between clients and servers; this exchange is typically facilitated by WebSockets or by long polls. These protocols are challenging for Lambdas because they are based on long-lived TCP connections. If the TCP connections are maintained within a Lambda handler, and a handler is idle between communication, charges to the customer should reflect the fact that the handler incurs a memory overhead, but consumes no CPU. Alternatively, if the platform provides management of TCP connections outside of the handlers, care must be taken to provide a new Lambda invocation with the connections it needs that were initiated by past invocations.

Databases

There are many opportunities for integrating Lambdas with databases. Most databases support *user-defined functions* (UDFs) for providing a custom view of the data. Lambdas that transform data from a cloud database could be viewed as UDFs that are used by client-side code. Current integration with S3 and DynamoDB also allows Lambdas to act as *trigger* handlers upon inserts.

A new *change feed* abstraction is now supported by RethinkDB and CouchDB; when an iterator reaches the end of a feed, it blocks until there is more data, rather than returning. Supporting change feeds with Lambdas entails many of the same challenges that arise with long-lived sessions; a handler that is blocked waiting for a database update should probably not be charged the same as an active handler. Change-feed batching should also be integrated with Lambda state transitions; it makes sense to batch changes for longer when a Lambda is paused than when it is running.

Relaxed consistency models should also be re-evaluated in the context of RPC handlers. The Lambda compute model introduces new potential consistency boundaries, based not on what data is accessed, but on which actor accesses the data. For example, an application may require that all RPC calls from the same client have a read-after-write guarantee, but weaker guarantees may be acceptable between different clients, even when those clients read from the same entity group.

Data Aggregators

Many applications (search, news feeds, and analytics) involve search queries over large datasets. Parallelism over different data shards is key to efficiently supporting these applications. For example, with search, one may want to scan many inverted indexes in parallel and then gather and aggregate the results.

Building these search applications will likely require special Lambda support. In particular, in order to support the scatter/gather pattern, multiple Lambdas will need to coordinate in a tree structure. Each leaf Lambda will filter and process data locally, and a front-end Lambda will combine the results.

When Lambda leaves are filtering and transforming large shards, it will be important to co-locate the Lambdas with the data. One solution would be to build custom datastores that coordinate with Lambdas. However, the diversity of aggregator applications may drive developers to use a variety of platforms for preprocessing the data (for example, MapReduce, Dryad, or Pregel). Thus, defining general locality APIs for coordination with a variety of backends may be necessary.

Load Balancers

Previous low-latency cluster schedulers (such as Sparrow [9]) target tasks in the 100 ms range. Lambda schedulers need to schedule work that is an order of magnitude shorter, while taking several types of locality into account. First, schedulers must consider *session locality*: if a Lambda invocation is part of a long-running session with open TCP connections, it will be beneficial to run the handler on the machine where the TCP connections are maintained so that traffic will not need to be diverted through a proxy.

Second, *code locality* becomes more difficult. A scheduler that is aware that two different handlers rely heavily on the same packages can make better placement decisions. Furthermore, a scheduler may wish to direct requests based on the varying degrees of dynamic optimization achieved on various workers.

Third, *data locality* will be important for running Lambdas alongside either databases or large datasets and indexes. The scheduler will need to anticipate what queries a particular Lambda invocation will issue or what data it will read. Even once the scheduler knows what data a Lambda will access and where the replicas of the data reside, further communication with the database may be beneficial for choosing the best replica. Many new databases (such as Cassandra or MongoDB) store replicas as LSM trees. Read amplifications for range reads can range from 1x to 50x [6] on different replicas; an integrated scheduler could potentially coordinate with database shards to track these varying costs.

Cost Debugging

Prior platforms cannot provide a cost-per-request for any service. For example, applications that use virtual machine instances are often billed on an hourly basis, and it is not obvious how to divide that cost across the individual requests over an hour. In contrast, it is possible to tell exactly how much each individual RPC call to a Lambda handler costs the cloud customer. This knowledge will enable new types of debugging.

Currently, browser-based developer tools enable performance debugging: tools measure page latency and identify problems by breaking down time by resource. New Lambda-integrated tools could similarly help developers debug monetary cost: the exact cost of visiting a page could be reported, and breakdowns could be provided detailing the cost of each RPC issued by the page as well as the cost of each database operation performed by each Lambda handler.

Legacy Decomposition

Breaking systems and applications into small, manageable subcomponents is a common approach to building robust, parallel software. Decomposition has been applied to operating systems, Web browsers, Web servers, and other applications. In order to save developer effort, there have been many attempts to automate some or all of the modularization process.

Decomposing monolithic Web applications into Lambda-based microservices presents similar challenges and opportunities. There are, however, new opportunities for framework-aware tools to automate the modularization process. Many Web-application frameworks (for example, Flask and Django) use language annotations to associate URLs with handler functions. Such annotations would provide an excellent hint to automatic splitting tools that port legacy applications to the Lambda model.

Conclusion

We have seen that the Lambda model is far more elastic and scalable than previous platforms, including container-based services that autoscale. We have also seen that this new paradigm presents interesting challenges for execution engines, databases, schedulers, and other systems. We believe OpenLambda will create new opportunities for exploring these areas. Furthermore, we hope to make OpenLambda a platform that is suitable for actual cloud developers to deploy their serverless applications. The OpenLambda project is online at https://www.open-lambda.org.

Acknowledgments

Feedback from anonymous reviewers has significantly improved this work. We also thank the members of the ADSL research group for their helpful suggestions and comments on this work at various stages.

This material was supported by funding from NSF grants CNS-1421033, CNS-1319405, CNS-1218405, CNS-1419199 as well as generous donations from EMC, Facebook, Google, Huawei, Microsoft, NetApp, Seagate, Samsung, Veritas, and VMware. Tyler Harter is supported by an NSF Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and may not reflect the views of the NSF or other institutions.

References

[1] "AWS Lambda": https://aws.amazon.com/lambda/, May 2016.

[2] A. Russell, Infrequently Noted, "Comet: Low Latency Data for the Browser" (blog entry), March 2006: https://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/.

[3] J. Gray, "Why Do Computers Stop and What Can We Do About It?" in *Proceedings of the 6th International Conference on Reliability and Distributed Databases,* June 1987: http://www.hpl.hp.com/techreports/tandem/TR-85.7.pdf.

[4] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast Distribution with Lazy Docker Containers," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST 16),* pp. 181–195: https://www.usenix.org/system/files/conference/fast16/fast16-papers-harter.pdf.

[5] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless Computation with OpenLambda," in *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16):* https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_hendrickson.pdf.

[6] L. Lu, T. Sankaranarayana Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "WiscKey: Separating Keys from Values in SSD-Conscious Storage," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16),* pp. 133–148: https://www.usenix.org/system/files/conference/fast16/fast16-papers-lu.pdf.

[7] M. Boyd, "Amazon Debuts Flourish, a Runtime Appliction Model for Serverless Computing": http://thenewstack.io/amazon-debuts-flourish-runtime-application-model-serverless-computing/, May 2016.

[8] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal,* 2014, no. 239: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.

[9] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, Low Latency Scheduling," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (ACM, 2013):* https://people.csail.mit.edu/matei/papers/2013/sosp_sparrow.pdf.

[10] T. Wagner, AWS Compute Blog, "Understanding Container Reuse in AWS Lambda," December 2014: https://aws.amazon.com/blogs/compute/container-reuse-in-lambda/.

Standing on the Shoulders of Giants by Managing Scientific Experiments Like Software

IVO JIMENEZ, MICHAEL SEVILLA, NOAH WATKINS, CARLOS MALTZAHN, JAY LOFSTEAD, KATHRYN MOHROR, REMZI ARPACI-DUSSEAU, AND ANDREA ARPACI-DUSSEAU

Ivo Jimenez is a PhD candidate at the UC Santa Cruz Computer Science Department and a member of the Systems Research Lab. His current work focuses on the practical reproducible generation and validation of systems research. Ivo holds a BS in computer science from University of Sonora and a MS from UCSC. ivo@cs.ucsc.edu

Michael Sevilla is a computer science PhD candidate at UC Santa Cruz. As part the Systems Research Lab, he evaluates distributed file system metadata management. At Hewlett Packard Enterprise, he uses open-source tools to benchmark storage solutions. He has a BS in computer science and engineering from UC Irvine. msevilla@soe.ucsc.edu

Noah Watkins is a PhD candidate in the Computer Science Department at UC Santa Cruz and a member of the Systems Research Lab. His research interests include distributed storage systems and data management, with a focus on programmable storage abstractions. Noah holds a BS in computer science from the University of Kansas. jayhawk@cs.ucsc.edu

Independently validating experimental results in the field of computer systems research is a challenging task. Recreating an environment that resembles the one where an experiment was originally executed is a time-consuming endeavor. In this article, we present Popper [1], a convention (or protocol) for conducting experiments following a DevOps [2] approach that allows researchers to make all associated artifacts publicly available with the goal of maximizing automation in the re-execution of an experiment and validation of its results.

A basic expectation in the practice of the scientific method is to document, archive, and share all data and the methodologies used so other scientists can reproduce and verify scientific results and students can learn how they were derived. However, in the scientific branches of computation and data exploration the lack of reproducibility has led to a credibility crisis. As more scientific disciplines are relying on computational methods and data-intensive exploration, it has become urgent to develop software tools that help document dependencies on data products, methodologies, and computational environments, that safely archive data products and documentation, and that reliably share data products and documentations so that scientists can rely on their availability.

Over the last decade software engineering and systems administration communities (also referred to as DevOps) have developed sophisticated techniques and strategies to ensure "software reproducibility," i.e., the reproducibility of software artifacts and their behavior using versioning, dependency management, containerization, orchestration, monitoring, testing and documentation. The key idea behind the Popper Convention is to manage every experiment in computation and data exploration as a software project, using tools and services that are readily available now and enjoy wide popularity. By doing so, scientific explorations become reproducible with the same convenience, efficiency, and scalability as software reproducibility while fully leveraging continuing improvements to these tools and services. Rather than mandating a particular set of tools, the Convention requires that the tool set as a whole implements functionality necessary for software reproducibility. There are two main goals for Popper:

1. It should be usable in as many research projects as possible, regardless of domain.
2. It should abstract underlying technologies without requiring a strict set of tools, making it possible to apply it on multiple toolchains.

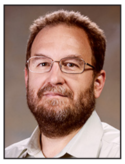
Popper
falsifiable.us

Common Experimental Practices

Ad hoc personal workflows: A typical practice is the use of custom bash scripts to automate some of the tasks of executing experiments and analyzing results.

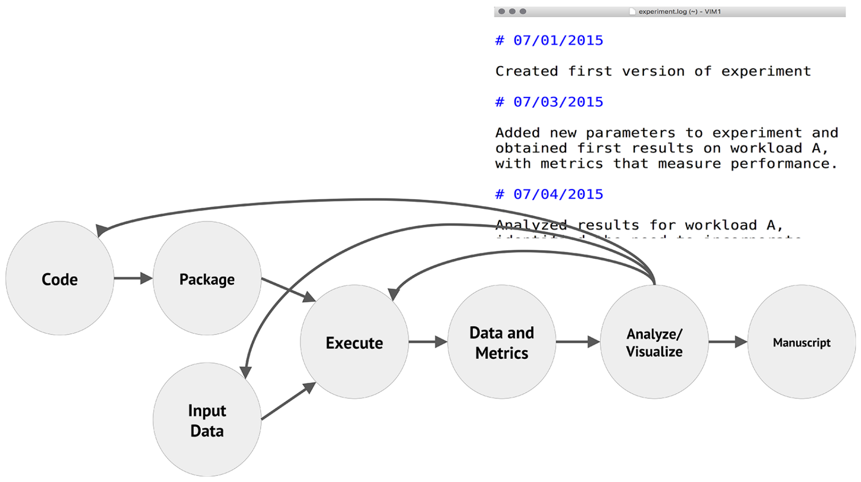
Sharing source code: Version-control systems give authors, reviewers, and readers access to the same code base, but the availability of source code does not guarantee reproducibility [3]; code may not compile, and even if it does, results may differ due to differences from other components in the software stack. While sharing source code is beneficial, it leaves readers with the daunting task of recompiling, reconfiguring, deploying, and re-executing an experiment.

Carlos Maltzahn is an Adjunct Professor of Computer Science at UC Santa Cruz and the Director of the Center for Research in Open Source Software (CROSS) and the Systems Research Lab (SRL). Carlos graduated with a PhD in computer science from the University of Colorado at Boulder. carlosm@soe.ucsc.edu



Jay Lofstead works on issues related to workflow, I/O, storage abstractions and the infrastructure necessary to support these system services. His other projects include the SIRIUS storage system project and the Decaf generic workflows project. Jay is a three-time graduate of Georgia Tech in computer science. gflofst@sandia.gov

Kathryn Mohror is a Computer Scientist on the Scalability Team at the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. Her research focuses on scalable fault-tolerant computing and I/O for extreme scale systems. Kathryn holds a PhD, MS, and BS from Portland State University (PSU) in Portland, OR. kathryn@llnl.gov



```
# 07/01/2015

Created first version of experiment

# 07/03/2015

Added new parameters to experiment and
obtained first results on workload A,
with metrics that measure performance.

# 07/04/2015

Analyzed results for workload A,
```

Figure 1: A generic experimentation workflow typically followed by researchers in projects with a computational component. Some of the reasons to iterate (backwards-going arrows) are: fixing a bug in the code of a system, changing a parameter of an experiment, or running the same experiment on a new workload or compute platform. Although not usually done, in some cases researchers keep a chronological record of how experiments evolve over time (the analogy of the lab notebook in experimental sciences).

Experiment repositories: An alternative to sharing source code is experiment repositories [4]. These allow researchers to upload artifacts associated with a paper, such as input datasets. Similar to code repositories, one of the main problems is the lack of automation and structure for the artifacts.

Virtual machines: A Virtual Machine (VM) can be used to partially address the limitations of only sharing source code. However, in the case of systems research where the performance is the subject of study, the overheads in terms of performance (the hypervisor "tax") and management (creating, storing, and transferring) can be high and, in some cases, cannot be accounted for easily [5].

Data analysis ad hoc approaches: A common approach to analyze data is to capture CSV files and manually paste their contents into Excel or Google spreadsheets. This manual manipulation and plotting lacks the ability to record important steps in the process of analyzing results, such as the series of steps that it took to go from a CSV to a figure.

Eyeball validation: Assuming the reader is able to recreate the environment of an experiment, validating the outcome requires domain-specific expertise in order to determine the differences between original and recreated environments that might be the root cause of any discrepancies in the results.

Goals for a New Methodology

A diagram of a generic experimentation workflow is shown in Figure 1. The problem with current practices is that each practice only partially covers the workflow. For example, sharing source code only covers the first task (source code), experiment packing only covers the second one (packaging), and so on. Based on this, we see the need for a new methodology that:

- Is reproducible without incurring any extra work for the researcher and requires the same or less effort than current practices but does things systematically.
- Improves the personal workflows of scientists by having a common methodology that works for as many projects as possible and that can be used as the basis of collaboration.
- Captures the end-to-end workflow in a modern way, including the history of changes that are made to an article throughout its life cycle.

Remzi Arpaci-Dusseau is a Full Professor in the Computer Sciences Department at the University of Wisconsin-Madison. He co-leads a group with his wife, Professor Andrea Arpaci-Dusseau. They have graduated 19 PhD students in their time at Wisconsin, won nine Best Paper awards, and some of their innovations now ship in commercial systems and are used daily by millions of people. Remzi has won the SACM Student Choice Professor of the Year award four times, the Carolyn Rosner "Excellent Educator" award, and the UW-Madison Chancellor's Distinguished Teaching award. Chapters from a freely available OS book he and Andrea co-wrote, found at http://www.ostep.org, have been downloaded millions of times in the past few years. remzi@cs.wisc.edu

Andrea Arpaci-Dusseau is a Full Professor of Computer Sciences at the University of Wisconsin-Madison. She is an expert in file and storage systems, having published more than 80 papers in this area, co-advised 19 PhD students, and received nine Best Paper awards; for her research contributions, she was recognized as a UW-Madison Vilas Associate. She also created a service-learning course in which UW-Madison students teach CS to more than 200 elementary-school children each semester. dusseau@cs.wisc.edu

- Makes use of existing tools (don't reinvent the wheel!); the DevOps toolkit is already comprehensive and easy to use.
- Has the ability to handle large datasets and scale to an arbitrary number of machines.
- Captures validation criteria in an explicit manner so that subjective evaluation of results of a re-execution is minimized.
- Results in experiments that are amenable to improvement and allows easy collaboration; makes it easier to build upon existing work.

A DevOps Approach to Conducting Experiments

The core idea behind *Popper* is to borrow from the DevOps movement the idea of treating every component as an immutable piece of information [6] and provide references to scripts and components for the creation, execution, and validation of experiments (in a systematic way) rather than leaving to the reader the daunting task of inferring how binaries and experiments were generated or configured. Version control, package management, multi-node orchestration, bare-metal-as-a-service, dataset management, data analysis and visualization, performance monitoring, continuous integration, each of these categories of the DevOps toolkit has a corresponding open source software (OSS) project that is mature, well documented, and easy to use (see Figure 2 for examples). The goal for Popper is to give researchers a common framework to systematically reason about how to structure all the dependencies and generated artifacts associated with an experiment by making use of these DevOps tools. The convention provides the following unique features:

1. Provides a methodology (or experiment protocol) for generating self-contained experiments.
2. Makes it easier for researchers to explicitly specify validation criteria.
3. Abstracts domain-specific experimentation workflows and toolchains.
4. Provides reusable templates of curated experiments commonly used by a research community.

Self-Containment

We say that an experiment is Popper-compliant (or that it has been "Popperized") if all of the following are available, either directly or by reference, in one single source code repository: experiment code, experiment orchestration code, reference to data dependencies, parametrization of experiment, validation criteria and results. In other words, a Popper repository contains all the dependencies for one or more experiments, optionally including a manuscript (article or tech report) that documents them.

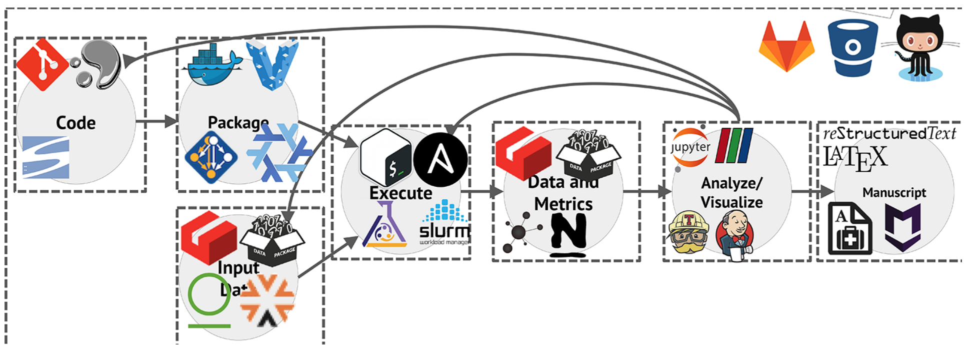


Figure 2: The same workflow as in Figure 1 viewed through a DevOps looking glass. The logos correspond to commonly used tools from the "DevOps toolkit." From left-to-right, top-to-bottom: git, mercurial, subversion (code); docker, vagrant, spack, nix (packaging); git-lfs, datapackages, artifactory, archiva (input data); bash, ansible, puppet, slurm (execution); git-lfs, datapackages, icinga, nagios (output data and runtime metrics); jupyter, paraview, travis, jenkins (analysis, visualization and continuous integration); restructured text, latex, asciidoctor and markdown (manuscript); gitlab, bitbucket and github (experiment changes).

An example paper project is shown in Listing 1. A paper repository is composed primarily of the article text and experiment orchestration logic. The actual code that gets executed by an experiment and all input datasets are not part of the repository; instead, they reside in their own repositories and are stored in the `experiments/` folder of the paper repository as references.

```
paper-repo
| README.md
| .git/
| .popper.yml
| .travis.yml
| experiments
|   |-- myexp
|   |   |-- datasets/
|   |   |       |-- input-data.csv
|   |   |-- figure.png
|   |   |-- process-result.py
|   |   |-- setup.yml
|   |   |-- results.csv
|   |   |-- run.sh
|   |   |-- validations.aver
|   |    -- vars.yml
| paper
|   |-- build.sh
|   |-- figures/
|   |-- paper.tex
|    -- references.bib
```

Listing 1: Sample contents of a Popper repository

With all these artifacts available, the reader can easily deploy an experiment or rebuild the article's PDF. Figure 3 shows our vision for the reader/reviewer workflow when reading a Popper for a Popperized article. The diagram uses tools we use in the use-case presented later, like Ansible and Docker, but as mentioned earlier, these can be swapped by equivalent tools. Using this workflow, the writer is completely transparent, and the article consumer is free to explore results, rerun experiments, and contradict assertions made in the paper.

A paper is written in any desired markup language (LaTeX in our example), where a `build.sh` command generates the output (e.g., PDF file). For the experiment execution logic, each experiment folder contains the necessary information such as setup, output post-processing (data analysis), and scripts for generating an image from the results. The execution of the experiment will produce output that is either consumed by a post-processing script, or directly by the scripts that generate an image.

The experiment output can be in any format (CSV, HDF, NetCDF, etc.), as long as it is versioned and referenced. An important component of the experiment logic is that it should

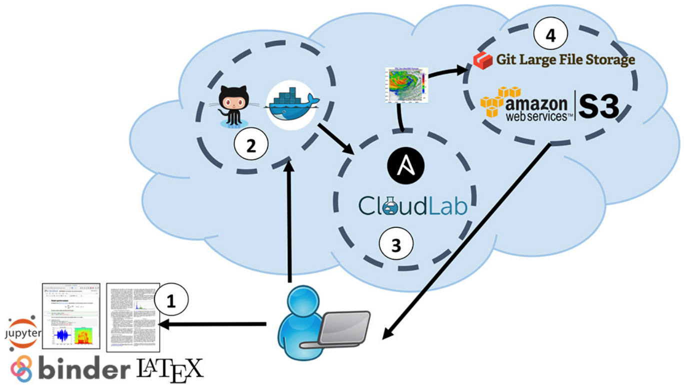


Figure 3: A sample workflow a paper reviewer or reader would use to read a Popperized article. (1) The PDF, Jupyter, or Binder are used to visualize and interact with the results postmortem on the reader's local machine. (2) If needed the reader has the option of looking at the code and cloning it locally (GitHub); for single-node experiments, they can be deployed locally too (Docker). (3) For multi-node experiments, Ansible can then be used to deploy the experiment on a public or private cloud (NSF's CloudLab in this case). (4) Lastly, experiments producing large datasets can make use of cloud storage. Popper is tool agnostic, so GitHub can be replaced with GitLab, Ansible with Puppet, Docker with VMs, etc.

assert the original assumptions made about the environment (`setup.yml`): for example, the operating system version (if the experiment assumes one). Also, it is important to parametrize the experiment explicitly (`vars.yml`) so that readers can quickly get an idea of what is the parameter space and what can be modified in order to obtain different results. One common practice we follow is to place in the caption of every figure a `[source]` link that points to the URL of the corresponding post-processing script in the version control Web interface (e.g., GitHub).

Automated Validation

Validation of experiments can be classified in two categories. In the first one, the integrity of the experimentation logic is checked using existing continuous-integration (CI) services such as TravisCI, which expects a `.travis.yml` file in the root folder specifying what tests to execute. These checks can verify that the paper is always in a state that can be built; that the syntax of orchestration files is correct so that if changes occur (e.g., addition of a new variable), it can be executed without any issues; or that the post-processing routines can be executed without problems.

The second category of validations is related to the integrity of the experimental results. These domain-specific tests ensure that the claims made in the paper are valid for every re-execution of the experiment, analogous to performance regression tests done in software projects. Alternatively, claims can also be corroborated as part of the analysis code. When experiments are not sensitive to the effects of virtualized platforms, these assertions can be executed on public/free CI platforms (e.g., TravisCI runs tests in VMs). However, when results are sensitive to the underlying hardware, it is preferable to leave this out of the CI pipeline and make them part of the post-processing routines of the experiment. In the example above, a `validations.aver` file contains validations in the Aver [7] language that check the integrity of runtime performance metrics. Examples of these type of assertions are: "the runtime of our algorithm is 10x better than the baseline when the level of parallelism exceeds four concurrent threads"; or "for dataset A, our model predicts the outcome with an error of 5%."

Toolchain Agnosticism

We designed Popper as a general convention, applicable to a wide variety of environments, from cloud to high-performance computing. In general, Popper can be applied in any scenario where a component (data, code, infrastructure, hardware, etc.) can be referenced by an identifier, and where there is an underlying tool that consumes these identifiers so that they can be acted upon (install, run, store, visualize, etc.). We say that a tool is Popper-compliant if it has the following two properties:

1. Assets (code, packages, configurations, data, results, etc.) can all be associated with, and referenced using, unique identifiers.
2. The tool is scriptable (e.g., can be invoked from the command line) and can act upon given asset IDs.

In general, tools that are hard to script—e.g., because they don't provide a command-line interface (can only interact via GUI) or they only have a programmatic API for a non-interpreted language—are beyond the scope of Popper.

Experiment Templates

Researchers that decide to follow Popper are faced with a steep learning curve, especially if they have only used a couple of tools from the DevOps toolkit. To lower the entry barrier, we have developed a command-line interface (CLI) tool that provides a list of experiment templates and helps to bootstrap a paper repository that follows the Popper Convention (available at https://github.com/systemslab/popper).

Use Case

We now illustrate how to follow the Popper Convention when conducting an experiment. For detailed documentation, visit our wiki at https://github.com/systemslab/popper/wiki.

Initializing a Popper repository: Our Popper-CLI tool assumes a Git repository exists (Listing 2). Given a Git repository, we invoke the Popper-CLI tool and initialize Popper by issuing a `popper init` command in the root of the Git repository. This creates a `.popper.yml` file that contains configuration options for the CLI tool. This file is committed to the paper (Git) repository. After the Popper repository has been initialized, we can either create a new experiment from scratch or obtain an existing one by pulling an experiment template.

```
$ cd mypaper-repo

$ popper init
-- Initialized Popper repo

$ popper experiment list
-- available templates ---------------
ceph-rados  proteustm mpip
spark-bench gassyfs   zlog
malacology  torpor    blis

$ popper add gassyfs myexp
```

Listing 2: Initialization of a Popper repo

Adding a new experiment: Assume the code of the system under study has already been packaged. In order to add an experiment that evaluates a particular aspect of the system, we first start by stating, in either a language such as Aver [7] or in plaintext, the result validation criteria. We then proceed with the implementation of the logic of the experiment, mainly orchestration code: configuration, deployment, analysis and visualization of performance metrics, and validation of results. All these files are placed in the paper repository in order to make the experiment Popper-compliant (self-contained).

Obtaining an existing experiment: As mentioned before, we maintain a list of experiment templates that have been "Popperized." For this example, assume we select the `gassyfs` template from the list. GassyFS [8] is a new prototype in-memory file system that stores data in distributed remote memory. Although GassyFS is simple in design, it is relatively complex to set up. The combinatorial space of possible ways in which the system can be compiled, packaged, and configured is large. Having all this information in a Git repository simplifies the setup since one doesn't need to speculate on which things where done by the original authors; all the information is available. In Figure 4 we show results of an experiment that validates the scalability of GassyFS. We note that while the obtained performance is relevant, it is not our main focus. Instead, we put more emphasis on the goals of the experiment, how we can reproduce results on multiple environments with minimal effort, and how we can validate them. Re-executing this experiment on a new platform only requires us to have host nodes to run Docker and to modify the list of hosts given to Ansible (a list of hostnames); everything else, including validation, is fully automated. The Aver [7] assertion in Listing 3 is used to check the integrity of this result and expresses our expectation of GassyFS performing sublinearly with respect to the number of nodes. After the experiment runs, Aver is invoked to test the above statement against the experiment results obtained.

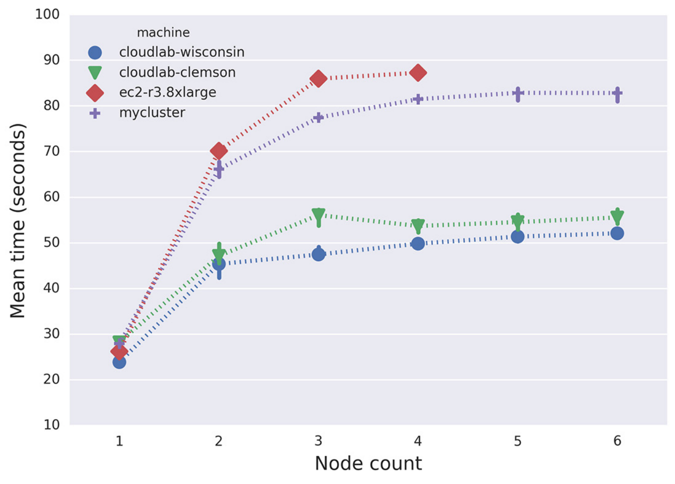


Figure 4: Scalability of GassyFS as the number of nodes in the GASNet cluster increases. The workload in question compiles Git. (*source:* https://github.com/systemslab/popper-paper/blob/login/experiments/gassyfs/visualize.ipynb)

```
when
  workload=* and machine=*
expect
  sublinear(nodes,time)
```

Listing 3: Assertion to check scalability behavior

Documenting the experiment: After we are done with an experiment, we might want to document it by creating a report or article. The Popper-CLI also provides us with manuscript templates. We can use the generic `article` template or other more domain-specific ones. To display the available templates we do `popper paper list`. In our example we use the template for USENIX articles by issuing a `popper paper add usenix`, which creates a `paper/` folder in the project's root folder, with a sample LaTeX file. We then can make reference to figures that have been generated as part of an experiment and reference them from the LaTeX file. We then generate the article (all paper templates have a `build.sh` command inside the paper folder) and see the new images added to the resulting PDF file.

The Case for Popper

We Did Well for 50 Years. Why Fix It?

Shared infrastructures "in the cloud" are becoming the norm and enable new kinds of sharing, such as experiments, that were not practical before. Thus, the opportunity of these services goes beyond just economies of scale: by using conventions and tools to enable reproducibility, we can dramatically increase the value of scientific experiments for education and for research. The Popper Convention makes available not only the result of a systems experiment but the entire experiment as well, and it allows researchers to study and reuse all aspects of it, making it practical to "stand on the shoulders of giants" by building upon the work of the community to improve the state-of-the-art without having to start from scratch every time.

The Power of "Crystallization Points"

Docker images, Ansible playbooks, CI unit tests, Git repositories, and Jupyter notebooks are all examples of artifacts around which broad-based efforts can be organized. Crystallization points are pieces of technology and are intended to be easily shareable, have the ability to grow and improve over time, and ensure buy-in from researchers and students. Examples of very successful crystallization points are the Linux kernel, Wikipedia, and the Apache Project. Crystallization points encode community knowledge and are therefore useful for leveraging past research efforts for ongoing research as well as education and training. They help people to form abstractions and common understanding that enables them to more effectively communicate reproducible science. By having popular tools such as Docker/Ansible as a lingua franca for researchers, and Popper to guide them in how to structure their paper repositories, we can expedite collaboration and at the same time benefit from all the new advances done in the DevOps world.

Perfect Is the Enemy of Good

No matter how hard we try, there will always be something that goes wrong. The context of systems experiments is often very complex, and that complexity is likely to increase in the future. Perfect repeatability will be very difficult to achieve. Recent empirical studies in computer systems [3, 9] have brought attention to the main issues that permeate the current practice of our research communities, where scenarios like the lack of information on how a particular package was compiled, or which statistical functions were used make it difficult to reproduce or even interpret results. Rather than aiming at perfect repeatability, we seek to minimize issues we currently face and to use a common language while collaborating to fix all these reproducibility issues. Additionally, following Popper quickly pays off at the individual level by improving productivity, e.g., when a researcher goes back to experiments to consider new scenarios.

DevOps Skills Are Highly Valued by Industry

While the learning curve for the DevOps toolkit is steep, having these as part of the skill set of students or researchers-in-training can only improve their curriculum. Since industry and many industrial/national laboratories have embraced (or are in the process of embracing) a DevOps approach, making use of these tools improves the prospects of future employment. These are skills that will hardly represent wasted time investments. On the contrary, this might be motivation enough for students to learn at least one tool from each of the stages of the experimentation workflow.

Conclusion

We named the convention Popper as a reference to Karl Popper, the philosopher of science who famously argued that *falsifiability* should be used as the demarcation criterion when determining whether a theory is scientific or pseudo-scientific. The OSS development model and the DevOps practice have proven to be an extraordinary way for people around the world to collaborate on software projects. As the use case presented here shows, by writing articles following the Popper Convention, authors can improve their personal workflows while at the same time generating research that is easier to validate and replicate. We are currently working with researchers from distinct scientific domains to help them "Popperize" their experiments and add new templates to our repository.

References

[1] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, "Popper: Making Reproducible Systems Performance Evaluation Practical," UC Santa Cruz School of Engineering, Technical Report UCSC-SOE-16-10, 2016: https://www.soe.ucsc.edu/research/technical-reports/UCSC-SOE-16-10.

[2] M. Httermann, *DevOps for Developers* (Apress, 2012).

[3] C. Collberg and T. A. Proebsting, "Repeatability in Computer Systems Research," *Communications of the ACM,* vol. 59, no. 3 (February 2016), pp. 62–69.

[4] V. Stodden, S. Miguez, and J. Seiler, "ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science," *Computing in Science & Engineering,* vol. 17, no. 1 (January 2015), pp. 12–19.

[5] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews, "Xen and the Art of Repeated Research," in *Proceedings of the 2004 USENIX Annual Technical Conference,* 2004.

[6] A. Wiggins, "The Twelve-Factor App": http://12factor.net.

[7] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "I Aver: Providing Declarative Experiment Specifications Facilitates the Evaluation of Computer Systems Research," *TinyToCS,* vol. 4 (2016): http://tinytocs.ece.utexas.edu/papers/tinytocs4_paper_jimenez.pdf.

[8] N. Watkins, M. Sevilla, and C. Maltzahn, "GassyFS: An In-Memory File System That Embraces Volatility," UC Santa Cruz School of Engineering, Technical Report UCSC-SOE-16-08, 2016: https://www.soe.ucsc.edu/research/technical-reports/UCSC-SOE-16-08.

[9] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses When Reporting Performance Results," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15):* http://htor.inf.ethz.ch/publications/img/hoefler-scientific-benchmarking_wide_HLRS.pdf.

Submit Your Work

26TH USENIX SECURITY SYMPOSIUM

August 16-18, 2017 • Vancouver, BC, Canada

The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security and privacy of computer systems and networks.

All researchers are encouraged to submit papers covering novel and scientifically significant practical works in computer security. Submissions are due on Thursday, February 16, 2017. The Symposium will span three days, with a technical program including refereed papers, invited talks, posters, panel discussions, and Birds-of-a-Feather sessions. Workshops will precede the Symposium on August 14 and 15.

Important Dates

- Paper submissions due: **Thursday, February 16, 2017, 5:00 p.m. EST**
- Invited talk and panel proposals due: **Thursday, February 18, 2017**
- Early-reject notification: **March 21, 2017**
- Notification to authors: **May 12, 2017**
- Final papers due: **June 29, 2017**
- Poster proposals due: **July 6, 2017**
- Notification to poster presenters: **July 13, 2017**
- Work-in-Progress submissions due: **August 16, 2017, noon CDT**

Program Co-Chairs

Engin Kirda, *Northeastern University*
Thomas Ristenpart, *Cornell Tech*

www.usenix.org/sec17

SECURITY

BeyondCorp Part III
The Access Proxy

LUCA CITTADINI, BATZ SPEAR, BETSY BEYER, AND MAX SALTONSTALL

Luca Cittadini is a Site Reliability Engineer at Google in Dublin. He previously worked as a Network Engineer at the Italian Central Bank in Rome. He holds a PhD in computer science from Roma Tre University. lucacittadini@google.com

Batz Spear is a Software Engineer at Google in Mountain View. He holds a PhD in computer science from UC Davis. batz@google.com

Betsy Beyer is a Technical Writer for Google Site Reliability Engineering in NYC. She has previously provided documentation for Google Data Center and Hardware Operations teams. Before moving to New York, Betsy was a lecturer in technical writing at Stanford University. She holds degrees from Stanford and Tulane. bbeyer@google.com

Max Saltonstall is a Program Manager for Google Corporate Engineering in New York. Since joining Google in 2011 he has worked on video products, internal change management, IT externalization, and coding puzzles. He has a degree in computer science and psychology from Yale. maxsaltonstall@google.com

This article details the implementation of BeyondCorp's front-end infrastructure. It focuses on the Access Proxy, the challenges we encountered in its implementation and the lessons we learned in its design and rollout. We also touch on some of the projects we're currently undertaking to improve the overall user experience for employees accessing internal applications.

In migrating to the BeyondCorp model (previously discussed in "BeyondCorp: A New Approach to Enterprise Security" [1] and "BeyondCorp: Design to Deployment at Google" [2]), Google had to solve a number of problems. Figuring out how to enforce company policy across all our internal-only services was one notable challenge. A conventional approach might integrate each back end with the device Trust Inferer in order to evaluate applicable policies; however, this approach would significantly slow the rate at which we're able to launch and change products.

To address this challenge, Google implemented a centralized policy enforcement front-end Access Proxy (AP) to handle coarse-grained company policies. Our implementation of the AP is generic enough to let us implement logically different gateways using the same AP codebase. At the moment, the Access Proxy implements both the Web Proxy and the SSH gateway components discussed in [2]. As the AP was the only mechanism that allowed employees to access internal HTTP services, we required all internal services to migrate behind the AP.

Unsurprisingly, initial attempts that dealt only with HTTP requests proved inadequate, so we had to provide solutions for additional protocols, many of which required end-to-end encryption (e.g., SSH). These additional protocols necessitated a number of client-side changes to ensure that the device was properly identified to the AP.

The combination of the AP and an Access Control Engine (a shared ACL evaluator) for all entry points provided two main benefits. By supplying a common logging point for all requests, it allowed us to perform forensic analysis more effectively. We were also able to make changes to enforcement policies much more quickly and consistently than before.

BeyondCorp's Front-End Infrastructure

Any modern Web application deployed at scale employs front-end infrastructure, which is typically a combination of load balancers and/or reverse HTTP proxies. Enterprise Web applications are no exception, and the front-end infrastructure provides the ideal place to deploy policy enforcement points. As such, Google's front-end infrastructure occupies a critical position in BeyondCorp's enforcement of access policies.

The main components of Google's front-end infrastructure are a fleet of HTTP/HTTPS reverse proxies called Google Front Ends (GFEs [3]). GFEs provide a number of benefits, such as load balancing and TLS handling "as a service." As a result, Web application back ends can focus on serving requests and largely ignore the details of how requests are routed.

BeyondCorp leverages the GFE as a logically centralized point of access policy enforcement. Funneling requests in this manner led us to naturally extend the GFE to provide other features, including self-service provisioning, authentication, authorization, and centralized logging. The resulting extended GFE is called the ***Access Proxy*** (AP). The following section details the specifics of the services the Access Proxy offers.

Features of the Extended GFE: Product Requirements

The GFE provides some built-in benefits that weren't designed specifically for BeyondCorp: it both provides load balancing for the back ends and addresses TLS handling by delegating TLS management to the GFE. The AP extends the GFE by introducing authentication and authorization policies.

Authentication

In order to properly authorize a request, the AP needs to identify the user and the device making the request. Authenticating the device poses a number of challenges in a multi-platform context, which we address in a later section, "Challenges with Multi-Platform Authentication." This section focuses on user authentication.

The AP verifies user identities by integrating with Google's Identity Provider (IdP). Because it isn't scalable to require back-end services to change their authentication mechanisms in order to use the AP mechanism, the AP needs to support a range of authentication options: OpenID Connect, OAuth, and some custom protocols.

The AP also needs to handle requests made without user credentials, e.g., a software management system attempting to download the latest security updates. In these cases, the AP can disable user authentication.

When the AP authenticates the user, it strips the credential before sending the request to the back end. Doing so is essential for two reasons:

- The back end can't replay the request (or the credential) through the Access Proxy.
- The proxy is transparent to the back ends. As a result, the back ends can implement their own authentication flows on top of the Access Proxy's flow, and won't observe any unexpected cookies or credentials.

Authorization

Two design choices drove our implementation of the authorization mechanism in a BeyondCorp world:

- A centralized Access Control List (ACL) Engine queryable via Remote Procedure Calls (RPCs)
- A domain-specific language to express the ACLs that is both readable and extensible

Providing ACL evaluation as a service enables us to guarantee consistency across multiple front-end gateways (e.g., the RADIUS network access control infrastructure, the AP, and SSH proxies).

Providing centralized authorization has both benefits and drawbacks. On the one hand, an authorizing front end frees back-end developers from dealing with the details of authorization by promoting consistency and a centralized point of policy enforcement. On the other hand, the proxy might not be able to enforce fine-grained policies that are better handled at the back end (e.g., "user A is authorized to modify resource B").

In our experience, combining coarse-grained, centralized authorization at the AP with fine-grained authorization at the back end provides the best of both worlds. This approach doesn't result in much duplication of effort, since the application-specific fine-grained policies tend to be largely orthogonal to the enterprise-wide policies enforced by the front-end infrastructure.

Mutual authentication between the proxy and the back end

Because the back end delegates access control to the front end, it's imperative that the back end can trust that the traffic it receives has been authenticated and authorized by the front end. This is especially important since the AP terminates the TLS handshake, and the back end receives an HTTP request over an encrypted channel.

Meeting this condition requires a mutual authentication scheme capable of establishing encrypted channels—for example, you might implement mutual TLS authentication and a corporate public key infrastructure. Our solution is an internally developed authentication and encryption framework called LOAS (Low Overhead Authentication System) that bi-directionally authenticates and encrypts all communication from the proxy to the back ends.

One benefit of mutual authentication and encryption between the front end and back end is that the back end can trust any additional metadata inserted by the AP (usually in the form of extra HTTP headers). While adding metadata and using a custom protocol between the reverse proxy and the back ends isn't a novel approach (for example, see Apache JServe Protocol [4]), the mutual authentication scheme between the AP ensures that the metadata is not spoofable.

As an added benefit, we can also incrementally deploy new features at the AP, which means that consenting back ends can opt in by simply parsing the corresponding headers. We use this functionality to propagate the device trust level to the back ends, which can then adjust the level of detail served in the response.

The ACL language

Implementing a domain-specific language for ACLs was key in tackling challenges of centralized authorization. The language both allows us to compile ACLs statically (which aids performance and testability) and helps reduce the logical gap between the policy and its implementation. This strategy promotes separation of duties among the following parties:

- **The team that owns the security policy:** Responsible for the abstract and statically compiled specification of access decisions
- **The team that owns the inventory pipeline:** Responsible for the concrete instantiation of a decision about granting access to a resource based on the specific device and user requesting access (see [2] for more details about the inventory pipeline)
- **The team that owns the Access Control Engine:** Responsible for evaluating and executing the security policy

The ACL language works using first-match semantics, which is similar to traditional firewall rules. While this model creates well-studied corner cases (for example, rules which shadow each other), we've found that the security team can reason about this model relatively easily. The structure of the ACLs we currently enforce consists of two macro-sections:

- **Global rules:** Usually coarse-grained and affect all services and resources. For example, "Devices at a low tier are not allowed to submit source code."
- **Service-specific rules:** Specific to each service or hostname; usually involve assertions about the user. For example, "Vendors in group G are allowed access to Web application A."

The above assumes that service owners can identify the sections of their URL space that need policies. Service owners should almost always be able to identify these sections, except for some cases in which the differentiation occurs in the request body (although the AP could be modified to handle this scenario). The portion of the ACL dealing with service-specific rules inevitably grows in size as the Access Proxy accounts for more and more services with a business need for a specialized ACL.

The set of global rules is very handy during security escalations (e.g., employee exit) and incident response (e.g., browser exploits or stolen devices). For example, these rules helped us successfully mitigate a zero-day vulnerability in third-party plugins shipped with our Chrome browser. We created a new high-priority rule that redirected out-of-date Chrome versions to a page with update instructions, which was deployed and enforced across the entire company within 30 minutes. As a result, the observed population of vulnerable browsers dropped very quickly.

Centralized Logging

In order to conduct proper incident response and forensic analysis, it's essential that all requests are logged to persistent storage. The AP provides an ideal logging point. We log a subset of the request headers, the HTTP response code, and metadata relevant to debugging or reconstructing the access decision and the ACL evaluation process. This metadata includes the device identifier and the user identity associated with the request.

Features of the Access Proxy: Operational Scalability

Self-Service Provisioning

Once the Access Proxy infrastructure is in place, developers and owners of enterprise applications have an incentive to configure their services to be accessible via the proxy.

When Google began gradually limiting users' network-level access into corporate resources, most internal application owners looked to the Access Proxy as the fastest solution to keep their service available as the migration proceeded. It was immediately clear that a single team couldn't scale to handle all changes to the AP's configuration, so we structured the AP's configuration to facilitate self-service additions. Users retain ownership of their fragment of the configuration, while the team that owns the AP owns the build system that collates, tests, canaries, and rolls out configurations.

This setup has a few main benefits:

- Frees the AP owners from continuously modifying the configuration per user requests
- Encourages service owners to own their configuration fragment (and write tests for it)
- Ensures a reasonable compromise between development velocity and system stability

The time it takes to set up a service behind the AP has effectively been reduced to minutes, while users are also able to iterate on their configuration fragments without requesting support from the team that owns the AP.

Challenges with Multi-Platform Authentication

Now that we've described the server side of BeyondCorp's front end—its implementation and the resulting challenges and complications—we'll take a similar view into the client side of this model.

At minimum, performing proper device identification requires two components:

- Some form of device identifier
- An inventory database tracking the latest known state of any given device

One goal of BeyondCorp is to replace trust in the network with an appropriate level of trust in the device. Each device must have a consistent, non-clonable identifier, while information about the software, users, and location of the device must be integrated in the inventory database. As discussed in the previous BeyondCorp papers, building and maintaining a device inventory can be quite challenging. The following subsections describe the challenges and solutions related to device identification in more detail.

Desktops and Laptops
Desktops and laptops use an X.509 machine certificate and a corresponding private key stored in the system certificate store. Key storage, a standard feature of modern operating systems, ensures that command-line tools (and daemons) that communicate with servers via the AP can be consistently matched against the correct device identifier. Since TLS requires the client to present a cryptographic proof of private key possession, this implementation makes the identifier non-spoofable and non-clonable, assuming it's stored in secure hardware such as a Trusted Platform Module (TPM).

This implementation has one main drawback: certificate prompts tend to frustrate users. Thankfully, most browsers support automatic certificate submission via policy or extension. Users might also be frustrated if the server rejects the TLS handshake when the client presents an invalid certificate. A failed TLS handshake results in a browser-specific error message that can't be customized. To mitigate this user experience issue, the AP accepts TLS sessions that don't have valid client certificates, and presents an HTML deny page when required.

Mobile Devices
The policies to suppress certificate prompts discussed above don't exist on major mobile platforms. Instead of relying on certificates, we use a strong device identifier natively provided by the mobile operating systems. For iOS devices, we use the identifier ForVendor, while Android devices use the device ID reported by the Enterprise Mobility Management application.

Special Cases and Exceptions
While we've been able to transition the vast majority of Web applications to the Access Proxy over the past few years, some special use cases either don't naturally fit the model or need some sort of special handling.

Non-HTTP Protocols
A number of enterprise applications at Google employ non-HTTP protocols that require end-to-end encryption. In order to serve these protocols through the AP, we wrap them in HTTP requests.

Wrapping SSH traffic in HTTP over TLS is easy thanks to the existing ProxyCommand facility. We developed a local proxy which looks a lot like Corkscrew, except the bytes are wrapped into WebSockets. While both WebSockets and HTTP CONNECT requests allow the AP to apply the ACLs, we opted to use WebSockets over CONNECT because WebSockets natively inherit user and device credentials from the browser.

In the cases of gRPC and TLS traffic, we wrapped the bytes in an HTTP CONNECT request. Wrapping has the obvious downside of imposing a (negligible) performance penalty on the transported protocol. However, it has the important advantage of separating device identification and user identification at different layers of the protocol stack. Inventory-based access control is a relatively new concept, so we frequently find that existing protocols have native support for user authentication (e.g., both LOAS and SSH provide this), but extending them with device credentials is non-trivial.

Because we perform device identification on the TLS layer in the wrapping CONNECT request, we don't need to rewrite applications to make them aware of the device certificate. Consider the SSH use case: the client and server can use SSH certificates to perform user authentication, but SSH doesn't natively support device authentication. Furthermore, it would be impossible to modify the SSH certificate to also convey device identification, because SSH client certificates are portable by design: they are expected to be used on multiple devices. Similar to how we handle HTTP, the CONNECT wrapping ensures we properly separate user and device authentication. While we use the TLS client certificate to authenticate the device, we might use the username and password to authenticate the user.

Remote Desktop
Chrome Remote Desktop, which is publicly available in the Chrome code base [5], is the predominant remote desktop solution at Google. While wrapping protocols in HTTP works in many cases, some protocols, like those powering remote desktop, are especially sensitive to the additional latency imposed by being routed through the AP.

In order to ensure that requests are properly authorized, Chrome Remote Desktop introduces an HTTP-based authorization server into the connection establishment flow. The server acts as an authorizing third party between the Chromoting client and the Chromoting host, while also helping the two entities share a secret, operating similarly to Kerberos.

We implemented the authorization server as a simple back end of the AP with a custom ACL. This solution has proven to work very well: the extra latency of going through the AP is only paid once per remote desktop session, and the Access Proxy can apply the ACLs on each session creation request.

Third-Party Software

Third-party software has frequently proved troublesome, as sometimes it can't present TLS certificates, and sometimes it assumes direct connectivity. In order to support these tools, we developed a solution to automatically establish encrypted point-to-point tunnels (using a TUN device). The software is unaware of the tunnel, and behaves as if it's directly connected to the server. The tunnel setup mechanism is conceptually similar to the solution for remote desktop:

- The client runs a helper to set up the tunnel.
- The server also runs a helper that acts as a back end of the AP.
- The AP enforces access control policies and facilitates the exchange of session information and encryption keys between the client and server helpers.

Lessons Learned

ACLs Are Complicated

We recommend the following best practices to mitigate the difficulties associated with ACLs:

- **Ensure the language is generic.** The AP's ACL has changed numerous times, and we've had to add new feeds (e.g., user and group sources). Expect that you'll need to regularly change the available features, and ensure that the language itself won't hamper these changes.
- **Launch ACLs as early as possible.** The reasons for doing so are twofold:
    - Ensures that users become trained on the ACLs and potential reasons for denial sooner rather than later.
    - Ensures that developers begin to adjust their code to meet the requirements of the AP. For example, we had to implement a cURL replacement to handle user and device authentication.
- **Make modifications self-service.** As previously mentioned, a single team that manages service-specific configuration doesn't scale to support multiple teams.
- **Create a mechanism to pass data from the AP to the back ends.** As mentioned above, the AP can securely pass additional data to the back end to allow it to perform fine-grained access controls. Plan for this required functionality early.

Emergencies Happen

Have well-tested plans in place to handle inevitable emergencies. Be sure to consider two major categories of emergencies:

- **Production emergencies:** Caused by outages or malfunctions of critical components in the request serving path
- **Security emergencies:** Caused by urgent needs to grant/revoke access to specific users and/or resources

Production Emergencies

In order to ensure the AP survives most outages, design and operate it according to SRE best practices [3]. To survive potential data source outages, all of our data is periodically snapshotted and available locally. We also designed AP repair paths that don't depend on the AP.

Security Emergencies

Security emergencies are more subtle than production emergencies, as they're easy to overlook when designing the access infrastructure. Be sure to factor ACL push frequency and TLS issues into user/device/session revocation.

User revocation is relatively straightforward: revoked users are automatically added to a special group as part of the revocation process, and one of the early global rules (see "The ACL language," above) in the ACL guarantees that these users are denied access to any resource. Similarly, session tokens (e.g., OAuth and OpenID Connect tokens) and certificates are sometimes leaked or lost and therefore need to be revoked.

As discussed in the first BeyondCorp article [1], device identifiers are untrusted until the device inventory pipeline reports otherwise. This means that even losing the certificate authority (CA) key (which implies inability to revoke certificates) doesn't imply losing control, because new certificates aren't trusted until they are properly catalogued in the inventory pipeline.

Given this ability, we decided to ignore certificate revocation altogether: instead of publishing a certificate revocation list (CRL), we treat certificates as immutable and simply downgrade the inventory trust tier if we suspect the corresponding private key is lost or leaked. Essentially, the inventory acts as a whitelist of the accepted device identifiers, and there is no live dependency on the CRL. The major downside of this approach is that it might introduce additional delay. However, this delay is relatively easy to solve by engineering fast-track propagation between the inventory and the Access Proxy.

You need a standard, rapid-push process for ACLs in order to ensure timely policy enforcement. Beyond a certain scale, you must delegate at least part of the ACL definition process to service owners, which leads to inevitable mistakes. While unit tests and smoke tests can usually catch obvious mistakes, logic errors will trickle through safeguards and make their way to production. It's important that engineers can quickly roll back ACL changes to restore lost access or to lock down unintended broad access. To cite our earlier zero-day vulnerability plugin example, our ability to push ACLs rapidly was key to our incident response team, as we could quickly create a custom ACL to force users to update.

Engineers Need Support

Transitioning to the BeyondCorp world does not happen overnight and requires coordination and interaction among multiple teams. At large enterprise scale, it's impossible to delegate the entire transition to a single team. The migration will likely involve some backwards-incompatible changes that need sufficient management support.

The success of the transition largely depends on how easy it is for teams to successfully set up their service behind the Access Proxy. Making the lives of developers easier should be a primary goal, so keep the number of surprises to a minimum. Provide sane defaults, create walkthrough guides for the most common use cases, and invest in documentation. Provide sandboxes for the more advanced and complicated changes—for example, you can set up separate instances of the Access Proxy that the load balancer intentionally ignores but that developers can reach (e.g., temporarily overriding their DNS configuration). Sandboxes have proven extremely useful in numerous cases, like when we needed to make sure that clients would be able to handle TLS connections after major changes to the X.509 certificates or to the underlying TLS library.

Looking Forward

While our front-end implementation of BeyondCorp has been largely quite successful, we still have a few pain points. Perhaps most obvious, desktops and laptops use certificates to authenticate, while mobile devices use a different device identifier. Certificate rotations are still painful, as presenting a new certificate requires a browser restart to ensure that existing sockets are closed.

To address both of these issues, we're planning to migrate desktops and laptops to the mobile model, which will remove the need for certificates. To carry out the migration, we plan to build a desktop device manager, which will look quite similar to the mobile device manager. It will provide a common identifier in the form of a Device-User-Session-ID (DUSI) that's shared across all browsers and tools using a common OAuth token-granting daemon. Once the migration is complete, we'll no longer need to authenticate desktops and laptops via a certificate, and all controls can migrate to use the DUSI consistently across all OSes.

Conclusion

Google's implementation of the Access Proxy as a core component of BeyondCorp is specific to our infrastructure and use cases. The design we ultimately implemented is well aligned with common SRE best practices and has proven to be very stable and scalable—the AP has grown by a number of orders of magnitude over the course of its deployment.

Any organization seeking to implement a similar security model can apply the same fundamental principles of creating and deploying a solution similar to the AP. We hope that by sharing our solutions to challenges like multi-platform authentication and special cases and exceptions, and the lessons we learned during this project, our experience can help other organizations to undertake similar solutions with minimal pain.

References

[1] R. Ward and B. Beyer, "BeyondCorp: A New Approach to Enterprise Security," *;login:,* vol. 39, no. 6 (December 2014): https://www.usenix.org/system/files/login/articles/login_dec14_02_ward.pdf.

[2] B. Osborn, J. McWilliams, B. Beyer, and M. Saltonstall, "BeyondCorp: Design to Deployment at Google," *;login:,* vol. 41, no. 1 (Spring 2016): https://www.usenix.org/system/files/login/articles/login_spring16_06_osborn.pdf.

[3] B. Beyer, C. Jones, J. Petoff, and N. Murphy, eds., *Site Reliability Engineering* (O'Reilly Media, 2016).

[4] Apache JServer Protocol: https://tomcat.apache.org/connectors-doc/ajp/ajpv13ext.html.

[5] https://src.chromium.org/viewvc/chrome/trunk/src/remoting/.

Excavating Web Trackers Using Web Archaeology

ADAM LERNER, ANNA KORNFELD SIMPSON, TADAYOSHI KOHNO, AND FRANZISKA ROESNER

Adam (Ada) Lerner is a PhD candidate in the Department of Computer Science & Engineering at the University of Washington. They received a BA from Amherst College. Their research is broadly within computer security and privacy, including the intersection of technical, social, and legal concerns. lerner@cs.washington.edu

Anna Kornfeld Simpson is a third-year PhD student at the University of Washington. She is an NSF Graduate Fellow and earned a BSE in computer science from Princeton University in 2014. She is broadly interested in security, privacy, and building secure systems informed by tech policy. aksimpso@cs.washington.edu

Tadayoshi Kohno is the Short-Dooley Professor in the Department of Computer Science & Engineering at the University of Washington. He received his PhD from the University of California San Diego and his BS from the University of Colorado. His research focuses on computer security, broadly defined. yoshi@cs.washington.edu

Franziska Roesner is an Assistant Professor in the Department of Computer Science & Engineering at the University of Washington. She received her PhD from the University of Washington and her BS from The University of Texas at Austin. Her research focuses broadly on topics in computer security and privacy, with a particular interest in understanding and improving security and privacy for end users of existing and emerging technologies. franzi@cs.washington.edu

Third-party Web tracking has recently become a frequent source of privacy concerns among researchers, policymakers, and the public. But how long has tracking been a part of the Web, and how has it changed over the years? These questions led us to build a tool, Tracking Excavator, which time travels using the Wayback Machine's archive of the Web. We were able to collect data on Web tracking over nearly the whole history of the Web, back to 1996, showing that archive-based measurements of the history of the Web are not only possible but are a powerful technique for examining Web tracking and other trends in the Web over its history.

A common problem we face as security and privacy researchers is our recurring need for time travel. Since security is hard to retrofit, we want to know what technologies and threats will be important in the future, so we can start studying and securing things now. This is particularly true for the Web, which changes very rapidly. Time travel into the future would be valuable, but at this time, the only technique we have is to wait.

We also sometimes wish to time travel backwards. Researchers like to measure the trends of how a technology or attack rises to prominence, but by the time one is ubiquitous, rapid changes in the Web have swept away the evidence, making it too late to measure how it became so popular. In this article we'll tell you about a paper we wrote when we found out how to time travel into the past of the Web, and the measurements we made using this technique of third-party Web tracking over the past 20 years [1].

Why Web Tracking?

Third parties are domains that appear on multiple different Web sites in order to provide valuable services, such as analytics, content delivery, social media integrations, and advertising. Many third parties track Web site visitors across the sites they visit, building up a profile of the Web sites they visit. This third-party Web tracking has become a major feature of the Web's economy: tracking underlies targeted advertising, making it a linchpin of Web sites funded by advertising revenue and enabling many of the services we all use so often. However, its importance and ubiquity has made some ask what privacy we're giving up in return, and news media, scientific researchers, policymakers, and the public as a whole have taken interest. The earliest measurements of third party, cookie-based tracking we are aware of came from the FTC in 2000 [2], while academics seem to have first begun to publish on the topic in 2009 [3].

Diverse people use the Web in many sectors of their lives, personal and professional. Recreation and commerce are common, but the Web also influences and enables sensitive activities, both intimate and public. On the intimate side, people explore their gender and sexuality, study religious and spiritual beliefs, and research their physical and mental health. Meanwhile, on the public side, the Web is important to our society, our politics, and our democracy, to the way people consume news, debate politics, and influence policymakers by public discourse and public comment. Engaging with these topics on the Web can provide great

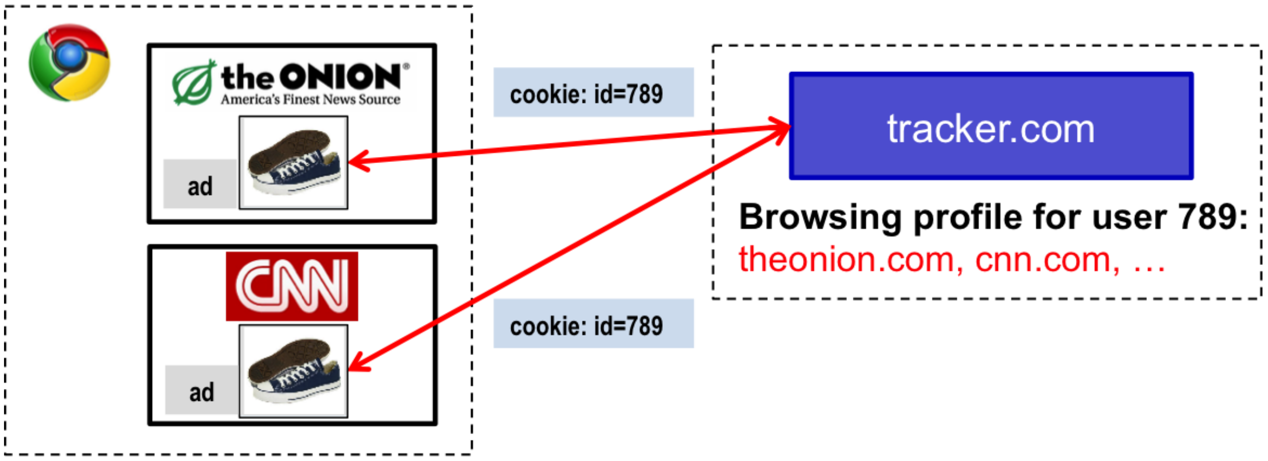


Figure 1 depicts a hypothetical tracker, tracker.com, which appears as a third party on both theonion.com and cnn.com. A cookie set by tracker.com when you visit The Onion is later sent to tracker.com when you visit CNN, allowing tracker.com to know which sites you visit.

benefits to individuals and society, but tracking people while they do so creates the possibility for privacy violations, chilling effects, and harm to those among us who may be most vulnerable. To balance these concerns, we need to understand how tracking happens so we can make informed choices.

Thinking of these issues, privacy researchers have been studying Web tracking for some time. Unfortunately, these measurements often use different methodologies, making them hard to compare or aggregate longitudinally. We wanted a longitudinal study of tracking in order to understand its trends and how it has responded to changes in the Web, such as in the way browsers are designed and in the technologies commonly used. To build this longitudinal picture, we needed a way to go back in time. Fortunately, the Wayback Machine (https://archive.org/web/) exists. It is a publicly available Web archive, an extensive collection of Web-page snapshots from the past 20 years, reaching back to 1996. The Wayback Machine serves the archived pages just like live Web sites; as an example, check out the *;login:* home page from 1997 at https://web.archive.org/web/19970606050039/http://www.usenix.org/publications/login/index.html.

The Wayback Machine contains both the contents and formatting of Web pages, as well as the JavaScript code and HTTP headers that were sent along with the page. This means that we can study not only the content of pages—the words and images—but also how pages were constructed technically and how they behaved dynamically, including the ways they may have used cookies and tracked visitors. Seeing this, we built a tool, Tracking Excavator, that uses the Wayback Machine to travel back in time and show us how Web tracking has evolved since its early days.

How Tracking Works

We measured third-party Web tracking over the past 20 years, focusing on cookie-based tracking. Cookie-based tracking occurs when a third party on the Web, such as an advertiser or social network provider, labels your browser with a unique identifier by asking your browser to store that identifier in a cookie. The third party then uses the identifier to recognize you in future, allowing it to build up a list of the places you go on the Web. We refer to this list of places you go as the browsing profile that a tracker learns about the people it tracks.

How does cookie-based tracking work technically? The following process is depicted in Figure 1. When you go to a Web site by typing its name (e.g., example.com) or clicking a link, we call that Web site a first party. First parties are the Web sites you visit intentionally, and you often have some direct relationship with them: you may purchase products there or sign up for an account. However, most first-party sites today also include one or more third parties. These third parties host some of the Web site's parts, such as scripts, images, style information (.css files), social media integrations, and advertisements. Your browser automatically contacts these third parties as part of its normal process of loading the first-party Web site. For example, when you visit example.com, your browser may load an advertisement from advertisements.com. Third parties can appear on many different first-party Web sites, so advertisements.com may also provide advertisements on other domains, like example2.com and example3.com. By appearing in these separate contexts, a third party may be able to track you across those sites, building up a browsing profile that describes where you've been on the Web.

When your browser makes requests to a particular third-party tracker for the first time, the tracker sends your browser a cookie that contains a unique identifier, to label your browser. Your browser stores cookies in a file and recalls these cookies, sending them whenever it makes future requests to that same third party, allowing the third party to recognize you when it serves your browser again. Since these requests to Web servers typically include information about the first-party site you're visiting, the third party learns about the sites you visit, and can associate that browsing profile with the unique identifier in your cookie. It's important to remember that they associate the browsing profile only with a unique identifier—often trackers don't know your name or who you are in real life, and are limited only to a (sometimes quite detailed!) picture of your interests and activities.

Tracking in All Its Glory

Some trackers are more complex or sophisticated than the simple description above. That said, when someone mentions "Web tracking," you should probably think of what we've described here. Our study shows that "vanilla" tracking—our name for the simple form described above—has been and still remains the most common type of cookie-based tracking. We studied more complicated types of tracking as well. For example, *referred trackers* are those trackers that share and exchange identifiers with one another, expanding their power to follow you to sites with different third parties. And we call trackers that you also sometimes visit as a first party, such as social network sites, *social trackers*, since your first-party relationship with them might allow them to link your browsing profile to your real life identity.

We classified the trackers we studied using a taxonomy we developed in earlier work [4]. That taxonomy classifies trackers by the way they behave and how those behaviors allow them to track people. The taxonomy we use also separates trackers according to the amount of extra information they have about the users they track (e.g., social trackers with whom users may have an account, such as facebook.com or google.com) and according to whether they share information with other trackers (referred trackers).

How to Time Travel (for Science!)

In this work, we developed archive-based, time-travel-capable measurement tools and used them to study the prevalence of Web tracking over the past 20 years. Our tool—Tracking Excavator—automatically browses the Wayback Machine's archive of the Web, collecting data about how the Web and its trackers used to behave. We analyzed this time-travel data to draw a picture of the history of tracking. In the future, our tool and our analysis techniques will allow us and other researchers to ask many other questions about topics, from security and privacy to software engineering and performance across the whole history of the Web. For example, recent work in 2014 and 2016 has provided two datapoints showing a downward trend of browser fingerprinting—how does that trend look over the entire lifespan of browser fingerprinting [5, 6]?

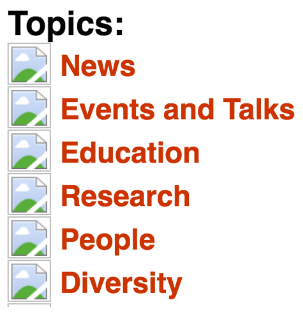

Topics:
News
Events and Talks
Education
Research
People
Diversity

Figure 2 shows an example of missing resources in the archive. Here images are missing from a 2003 snapshot of the University of Washington's CSE homepage (https://web.archive.org/web/20031001160742/http://www.cs.washington.edu/).

Getting here wasn't easy. We analyzed and quantified the limitations of the Wayback Machine and developed techniques for performing retrospective measurements in the presence of these limitations so that our measurements accurately reflected the Web of the past. The Wayback Machine is extensive, containing over 10 petabytes of Web site snapshots, but the world isn't perfect, and archives are no exception. The Wayback Machine is sometimes incomplete (as in Figure 2), with missing Web site data and cookies. Other times, the Wayback Machine makes mistakes that cause Web site snapshots to be *inconsistent*. Rather than reflecting the way a page looked at a single point in the past, it sometimes mashes up anachronistic resources from different points in time.

Other studies have relied on the Wayback Machine (for example, to predict whether Web sites will become malicious [7] and to study JavaScript inclusion [8]) and noted similar limitations. One of our goals was to systematically evaluate and develop mitigations for these limitations to enable future studies.

Given that our archival data is sometimes incomplete and sometimes anachronistic, we put a lot of work into ensuring that our results accurately reflected tracking that really happened in the past.

We crawled many archival Web pages to learn how and when these types of errors occur to ensure that our results accurately reflected the way tracking happened in the past. First, we measured the ways the Wayback Machine can have missing or anachronistic data, quantified them, and reasoned through the effects those errors would have on our analysis. Then we incorporated that understanding into Tracking Excavator and into our data analysis to winnow our observations to only what really existed. Finally, we think about the limitations of our data whenever we share or interpret our results. For example, since we must ignore many anachronistic requests, our measurements

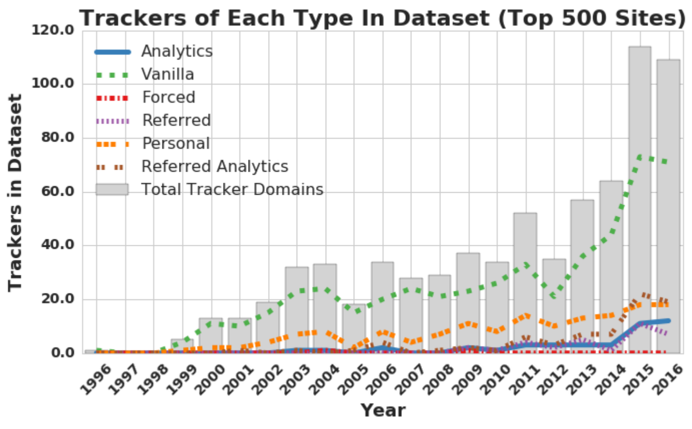


Figure 3: Number of tracking domains (gray bars) present on the 500 most popular sites of each year. The various line-styles represent different cookie-based tracking behaviors in our taxonomy; a single tracking domain may have multiple behaviors.

generally underestimate the amount of tracking that really happened.

We knew that our results undercounted the number and power of trackers, but by how much? Fortunately, we have been studying Web tracking for a few years, and we have data from the past five years. We used the same methods and taxonomy on live Web pages to measure Web tracking in 2011, 2013, 2015, and 2016. By comparing these past, ground-truth measurements to our archival results for the same periods, we were able to understand how our archival trends reflected real ones. The relationship between archival trackers and real trackers turns out to be quite consistent—see our paper for details of this validation.

Digging through the limitations of the Wayback Machine didn't just allow us to understand Web tracking. We hope that understanding and mitigating these limitations will allow other researchers to use these techniques in the future. We think that archive-based retrospective measurements are a powerful tool, and we hope our work will aid and inspire others to try measuring all sorts of aspects of security and privacy and other technical topics on the Web.

Our Results

Many people would believe intuitively that "Web tracking has increased over time" based on their experience of browsing the Web and reading the news. We shared this intuition too, but we found it valuable and informative to confirm this intuition scientifically. Quantifying this increase is a contribution to our understanding and debate over tracking. Using Tracking Excavator, we unveiled trends in the number of trackers, the extent of their coverage, and the prevalence and evolution of different tracking techniques. We discovered that the number of tracker domains present on the 500 most popular sites of each year has been increasing, that these domains are demonstrating increasingly complex tracking behaviors, and that the most prevalent domains are achieving greater coverage of popular Web sites and so can build a greater profile of users' browsing activity. We used a separate set of 500 popular sites for each year we studied, and those input datasets and our analysis code are available at our Web site, trackingexcavator.cs.washington.edu.

Thinking back to our challenges, we recall that the numbers which follow are generally undercounts, due to missing and inconsistent data in the archive. Additionally, we point out that our method measures only client-side, cookie-based behaviors that would enable tracking, but we cannot tell whether domains actually track users based on their cookies. We also don't look for alternative methods tracking, such as browser fingerprinting [9].

Quantifying Overall Tracking Behavior

The gray bars in Figure 3 show the overall number of tracker domains we saw for each year, while the different lines break those trackers down into the types of our taxonomy. Recall that these numbers were measured on the 500 Web sites that were most popular in that year. A single tracking domain can display multiple behaviors. The steadily increasing trend demonstrates that new players have continued to enter the game of tracking, developing profiles of users' browsing. Additionally, tracking domains are starting to use more complex techniques, such as sharing their data with additional third parties (as "referred" or "referred analytics" trackers) that allow further collection and dissemination of users' browsing histories. Refer to our paper [1] for more details about these types of trackers.

Figure 3 also shows an increase in personal trackers. These are tracking domains that users browse in their own right, as first parties, and that may have collected additional profiling information about a user. Some of the most popular sites on the Web, such as Facebook and Twitter, are included in this category. Personal trackers often appear today as social media buttons, such as the Facebook "Like" button or the Twitter "Tweet" button. Social trackers are particularly powerful because in addition to the browsing profile they build of you, they may also know a great deal about you through the profile, posts, and friendships you maintain on their social media site. By connecting their pseudonymous profile of browsing history to a social media profile, potentially across multiple machines, social trackers may be able to build deeper, longer lived, and more personal profiles of people.

The increase in the number of tracking domains corresponds with a general increase in the inclusion of third-party content over time. Figure 4 shows the distribution of the number of third parties on Web sites. Each line represents the data from one year, and the farther out from the axes the line lies, the larger the fraction of sites (y-axis) that had a larger number of third parties (x-axis). Although this distribution includes third parties such as content delivery networks (CDNs) that do not necessarily

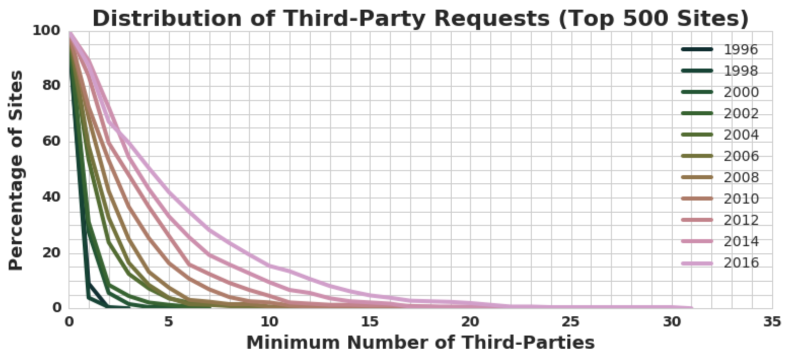


Figure 4: Distribution of included third parties for the top 500 most popular sites of each year. In 1996, less than 5% of sites included content from one or more third parties, while in 2016, 90% of sites included content from at least one third party, and 50% of sites included content from at least four third parties.

include any tracking behavior, it does capture sites whose cookies were not properly archived by the Wayback Machine that are missing from Figure 3. Of the 500 most popular sites in 1996, less than 5% of them included content from one or more third parties. By 2006, more than 50% included content from at least one third party, but few included content from more than five third parties. In archival data from 2016, 90% of the most popular sites include content from at least one third party, 50% of sites include content from at least four third parties, and 15% of sites include content from 10 or more third-party domains! Much as we found it valuable to confirm and quantify our intuitions about tracking, we also find it useful to quantify the increasing complexity of Web sites and the number of third parties that have the opportunity to observe people's browsing.

Quantifying the Most Powerful Trackers

While the number of trackers has grown over time, the above results don't tell us how large of a browsing profile each of those trackers can build. A third-party tracker can only track people who visit the sites where it appears, so we may be less concerned about the privacy implications of a large number of trackers, each of which appear on a small number of sites, and more concerned with a single tracker that appears on many sites. Therefore, we also examined the most prevalent trackers: the ones that show up on the greatest number of sites in the top 500. Figure 5 shows the most and second-most prevalent third parties each year. In the first decade of tracking that we measure, no tracker could directly track a person across more than 10% of the most popular sites. However, their reach increases in the second decade of our measurements: individual companies now have the ability to build significantly larger profiles of our browsing history.

One domain stands out in Figure 5: google-analytics.com (represented by the line at the top with stars through it) appears on nearly 200 of the top 500 most popular sites in 2011, and approximately a third of the sites in the years following.

We note that Google Analytics is designed as an on-site analytics script rather than a cross-site tracker, which means that its primary purpose is to provide analytics for a single domain rather than build cross-site browsing profiles. However, we observe that its high prevalence on popular sites gives it a large amount of power: its choices—and its transparency about its tracking policies and data sharing—can have a large effect on user privacy.

Changing Tracking Behavior

One encouraging anecdote comes from the mid-2000s. The archives of the early 2000s featured a number of pop-up advertisements, which we captured since our browser running Tracking Excavator allowed pop-ups. Pop-ups have a place in the Web tracking discussion because the browser treats the popped-up site as a first party (a site the user chose to go to) rather than a third party for the purposes of setting cookies. Therefore, tracking defenses that block third-party cookies are not effective against third-party pop-ups. From 2000–2004, we saw a significant increase in third-party pop-ups, but in 2005 the number of third-party pop-ups dropped dramatically. Some digging revealed that this change coincided with the decision by the developers of the Internet Explorer browser in 2004 (soon followed by other browsers) to block pop-ups by default. While it is likely that the browser manufacturers made this decision to improve user experience on the Web rather than for reasons related to tracking, it nevertheless also had positive effects for defending against tracking, because sites could not so easily evade third-party cookie blocking. As a result, the trackers were forced to implement other, more complex, techniques that we see later in the 2000s and 2010s.

Conclusion

In this study, we showed that it's possible to gather longitudinal data from the Wayback Machine in order to measure third-party Web tracking over nearly the whole history of the Web. In order to do so, we quantified and evaluated the challenges of using archival data in measurements, and developed techniques for mitigating those challenges, incorporating those techniques into our tool for retrospective measurements, Tracking Excavator. These techniques and this tool are not specific to Web tracking, and we hope that we've made a new type of measurement possible for others, enabling and inspiring them to go out and measure all kinds of properties of the Web retrospectively. If we're lucky, our minor form of Web-based time travel can support us in gathering the data we need as technologists, policymakers, and the public to make decisions about the way we can best make our technologies work for us while preserving our security and privacy.

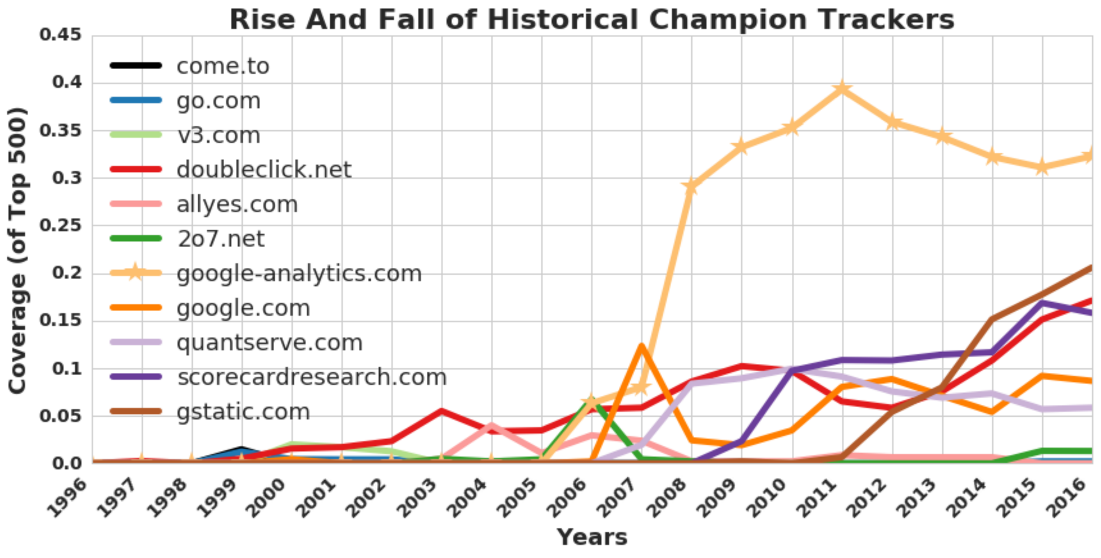


Figure 5: The third parties with the greatest or second-greatest presence on the top 500 most popular sites of each year. We refer to the number of sites a tracker appears on as its "coverage." Before 2007, we measured no domain on more than 10% of popular sites; now, several third parties appear on nearly 20% of sites.

References

[1] A. Lerner, A. Kornfeld Simpson, T. Kohno, and F. Roesner, "Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security '16)*: https://trackingexcavator.cs.washington.edu /InternetJonesAndTheRaidersOfTheLostTrackers.pdf.

[2] Federal Trade Commission, "Privacy Online: Fair Information Practices in the Electronic Marketplace: A Report to Congress," May 2000: http://www.ftc.gov/reports/privacy2000 /privacy2000.pdf.

[3] B. Krishnamurthy and C. Wills, "Privacy Diffusion on the Web: A Longitudinal Perspective," in *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*: http://web.cs.wpi.edu/~cew/papers/www09.pdf.

[4] F. Roesner, T. Kohno, and D. Wetheral, "Detecting and Defending Against Third-Party Tracking on the Web," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*: http://www.franziroesner.com /pdf/webtracking-NSDI2012.pdf.

[5] S. Englehardt and A. Narayanan, "Online Tracking: A 1-Million-Site Measurement and Analysis," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security (ACM CCS 2016)*: http://randomwalker.info/publications /OpenWPM_1_million_site_tracking_measurement.pdf.

[6] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (ACM CCS 2014)*: https://securehomes.esat.kuleuven.be/~gacar/persistent /the_web_never_forgets.pdf.

[7] K. Soska and N. Christin, "Automatically Detecting Vulnerable Websites Before They Turn Malicious," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security '14)*: https:// www.usenix.org/system/files/conference/usenixsecurity14 /sec14-paper-soska.pdf.

[8] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. "You Are What You Include: Large-Scale Evaluation of Remote JavaScript Inclusions," in *Proceedings of the 19th ACM Conference on Computer and Communications Security (ACM CCS 2012)*: https:// seclab.cs.ucsb.edu/media/uploads/papers/jsinclusions.pdf.

[9] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting," in *Proceedings of the IEEE Symposium on Security and Privacy (2013)*: https://seclab .cs.ucsb.edu/media/uploads/papers/sp2013_cookieless.pdf.

The Adblocking Tug-of-War

HAMED HADDADI, RISHAB NITHYANAND, SHEHARBANO KHATTAK, MOBIN JAVED, NARSEO VALLINA-RODRIGUEZ, MARJAN FALAHRASTEGAR, JULIA E. POWLES, EMILIANO DE CRISTOFARO, AND STEVEN J. MURDOCH

Hamed Haddadi is a Senior Lecturer in Digital Media at the School of Electronic Engineering and Computer Science, Queen Mary University of London. hamed.haddadi@qmul.ac.uk

Rishab Nithyanand is a PhD student at Stony Brook University. He is currently an OTF Senior Emerging Technology Fellow and a visitor at the International Computer Science Institute at Berkeley. rnithyanand@cs.stonybrook.edu

Sheharbano Khattak is a PhD student and Research Assistant in the Security and Networks and Operating Systems Groups at Computer Laboratory, University of Cambridge. Sheharbano.Khattak@cl.cam.ac.uk

Mobin Javed is a final year PhD student in computer science at UC Berkeley. Her research interests are in the areas of network security, privacy, and Internet measurement. mobin@cs.berkeley.edu

Narseo Vallina-Rodriguez is an Assistant Professor at IMDEA Networks, Madrid, and a Principal Investigator at the International Computer Science Institute in Berkeley, CA. narseo@icsi.berkeley.edu

Online advertising subsidizes a majority of the "free" services on the Web. Yet many find this approach intrusive and annoying, resorting to adblockers to get rid of ads chasing them all over the Web. A majority of those using an adblocker tool are familiar with messages asking them to either disable their adblocker or to consider supporting the host Web site via a donation or subscription. This is a recent development in the ongoing adblocking arms race which we have explored in our recent report, "Adblocking and Counter Blocking: A Slice of the Arms Race" [1]. For our study, we used popular adblockers, trawled the Web and analyzed some of the most popular sites to uncover how many are using anti-adblockers. Our preliminary analysis found that anti-adblockers come from a small number of providers, are widely used, and that adblockers also often block anti-adblockers.

The Perils of Targeted Advertising

The Internet economy today is largely driven by targeted advertising. Most "free" apps and Web services are bundled with third-party Web tracking [2] scripts and at times malicious code running at the user end, collecting and transmitting browsing history and personal data. Consumers often have no negotiating power in this ecosystem, despite clear evidence that such aggressive tracking and advertising often jeopardizes individuals' privacy, security, energy, and bandwidth [3]. This is particularly the case in the mobile advertising domain, where earnings are usually paid on ad impressions by the thousand, so ad brokers aim to maximize the number of ads and the frequency with which they get clicked on.

Adblockers vs. Anti-Adblockers—The Arms Race

Adblockers represent one of the ways in which consumers have retaliated against the targeted advertising industry. The main task of an adblocking software is to remove ads from users' Web pages, but some may even curb online tracking (also referred to as anti-trackers). The reasons for the rising popularity of adblockers include improved browsing experience, better privacy, and protection against malvertising. As a result, online advertising revenue is gravely threatened by adblockers, prompting publishers to actively detect adblock users, and subsequently block them or otherwise coerce the user to disable the adblocker—practices referred to as anti-adblocking (see Figure 1).

Example of Anti-Adblocking

An anti-adblocker detects adblockers by one of the following two approaches:

1. The anti-adblocker injects a bait advertisement container element (e.g., DIV), and then compares the values of properties representing dimensions (height and width) and/or visual status (display) of the container element with the expected values when properly loaded.

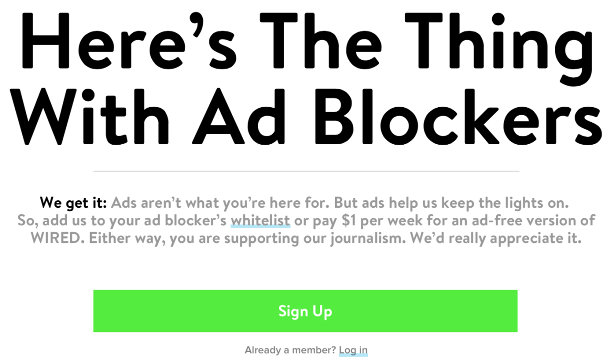
The Adblocking Tug-of-War

Marjan Falahrastegar is a PhD student at Computer Networks Group of Queen Mary University of London. marjan.falahrastegar@qmul.ac.uk

Julia Powles is a Postdoctoral Researcher in the Faculty of Law and Computer Laboratory at the University of Cambridge. jep50@cam.ac.uk

Emiliano De Cristofaro is a Senior Lecturer in the Information Security Group at University College London. e.decristofaro@ucl.ac.uk

Steven Murdoch is a Royal Society University Research Fellow in the Computer Science Department of University College London. s.murdoch@ucl.ac.uk


Here's The Thing With Ad Blockers

We get it: Ads aren't what you're here for. But ads help us keep the lights on. So, add us to your ad blocker's whitelist or pay $1 per week for an ad-free version of WIRED. Either way, you are supporting our journalism. We'd really appreciate it.

Sign Up

Already a member? Log in

Figure 1: An example of an anti-adblocking message. You very likely have seen pages like this from popular new sites. These are served to users when an anti-adblocking script has determined that the user has an adblocker installed.

2. The anti-adblocker loads a bait script that modifies the value of a variable, and then checks the value of this variable in the main anti-adblocking script to verify that the bait script was properly loaded. If the bait object is determined to be absent, the anti-adblocking script concludes that an adblocker is present.

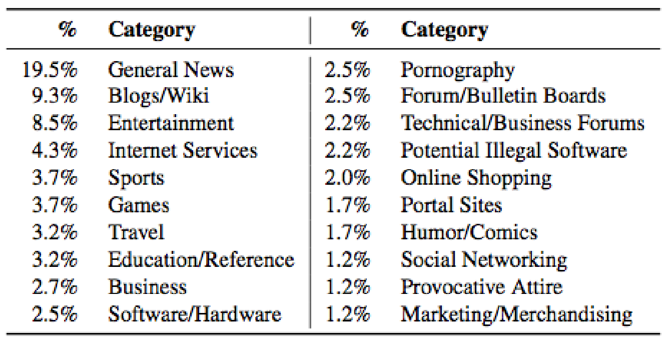
To track whether the user has turned off the adblocker after being prompted to do so, the anti-adblocker periodically runs the adblock check and stores the last recorded status in the user's browser using a cookie or local storage.

While incidents of anti-adblocking, and the legality of such practices, have received increasing attention, our current understanding is limited to online forums and user-generated reports. As a result, we lack quantifiable insights into the scale, mechanism, and dynamics of anti-adblocking. We have started to address these issues in our current research study, presented recently at USENIX FOCI '16 [1]. We did so by leveraging a novel approach for identifying third-party services shared across multiple Web sites to present a first characterization of anti-adblocking across the Alexa Top-5000 Web sites. Using a Web crawler to capture screenshots, HTML source code, and responses to all requests generated, we uncovered how anti-adblocking operates and mapped Web sites that perform anti-adblocking as well as the entities that provide anti-adblocking scripts.

Research Findings

Overall, we found that at least around 7% of Alexa Top-5000 Web sites employ anti-adblocking, with the practices finding adoption across a diverse mix of publishers, particularly publishers in the categories "general news," "blogs/wiki," and "entertainment." It turns out that these Web sites owe their anti-adblocking capabilities to 14 unique scripts pulled from 12 different domains. Surprisingly, anti-adblockers operate on a simple premise: if a bait object (i.e., an object that is expected to be blocked by adblockers—e.g., a JavaScript or DIV element named ads) on the publisher's Web site is missing when the page loads, the script concludes that the user has an adblocker installed. Figure 2 shows a summary of the types of Web sites deploying an anti-adblocking strategy.

Unsurprisingly, the most popular domains are those that have skin in the game—Google, Taboola, Outbrain, Ensighten, and Pagefair—the latter being a company that specializes in anti-adblocking services. Then there are in-house anti-adblocking solutions that are distributed by a domain to client Web sites belonging to the same organization: TripAdvisor distributes an anti-adblocking script to its eight Web sites with different country code top-level domains, while adult Web sites (all hosted by MindGeek) turn to DoublePimp. As a further element of the research, we visited a sample Web site for each anti-adblocking script



%	Category	%	Category
19.5%	General News	2.5%	Pornography
9.3%	Blogs/Wiki	2.5%	Forum/Bulletin Boards
8.5%	Entertainment	2.2%	Technical/Business Forums
4.3%	Internet Services	2.2%	Potential Illegal Software
3.7%	Sports	2.0%	Online Shopping
3.7%	Games	1.7%	Portal Sites
3.2%	Travel	1.7%	Humor/Comics
3.2%	Education/Reference	1.2%	Social Networking
2.7%	Business	1.2%	Provocative Attire
2.5%	Software/Hardware	1.2%	Marketing/Merchandising

Figure 2: Distribution of anti-adblocking Web sites by category according to McAfee's URL categorization

via AdBlock Plus, Ghostery, and Privacy Badger, and discovered that half of the 12 anti-adblocking suppliers are counter-blocked by at least one adblocker—suggesting that the arms race has already entered the next level.

Implications, Legality, and Ethics

The implications of the findings are manifold and complicated due to the involvement of a plethora of players: publishers, consumers, and a jostling array of intermediaries that compete to deliver ads, mostly supported by business models that involve taking a cut of the resultant advertising revenue. Advertising creates overhead for the users and telcos. In an extreme example, a mobile operator recently started to *block mobile ads altogether*. If such steps were to be widely adopted, they would severely limit the degree to which app developers could continue to innovate and create while maintaining the illusion of "free" apps and content for users. Arguably, handing control of the Web ecosystem to telecom companies or small yet powerful adblocking businesses that allow advertisers to *whitelist their ads* so that they are still seen by users is also an undesirable outcome for the freedom of the Web and Net Neutrality, and their effectiveness is debatable as this merely shifts control of which ads are displayed to users from one entity to another. Alternatively, efforts such as Brave (blog.brave.com) allow individuals to directly pay for the content of their favorite Web sites without being tracked.

The legality of adblocking is also potentially contestable under laws about anti-competitive business conduct and copyright infringement. To date, only Germany has tested these arguments in court, with adblockers winning most but not all of the cases. By contrast, anti-adblocking in the EU might in turn breach Article 5(3) of the Privacy and Electronic Communications Directive 2002/58/EC, as it involves interrogating an end-user's terminal equipment without consent.

Conclusion

Many consider adblocking to be an ethical choice for consumers and publishers to consider from both an individual and societal perspective. In reality, however, both sides have resorted to radical measures to achieve their goals. The Web has empowered publishers and advertisers to track, profile, and target users in a way that is unprecedented in the physical realm. In addition, publishers are inadvertently and increasingly serving up malicious ads. This has resulted in the rise of adblocking, which in turn has led publishers to employ anti-adblocking. The core issue is to get the balance right between ads and information: publishers turn to anti-adblocking to force consumers to reconsider the default blocking of ads for earnest publishers. But defaults are difficult to shift at scale. And, in any event, even worthy ad-supported publishers will fail if they do not redress in a fundamental way the reasons that brought consumers to adblockers in the first place. Regulation and proposals such as privacy-friendly advertising or mechanisms to give users more control over ads and trackers may provide a compromise in this space.

References

[1] R. Nithyanand, S. Khattak, M. Javed, N. Vallina-Rodriguez, M. Falahrastegar, J. E. Powles, E. De Cristofaro, H. Haddadi, and S. J. Murdoch, "Adblocking and Counter Blocking: A Slice of the Arms Race," 6th USENIX Workshop on Free and Open Communications on the Internet (FOCI '16), 2016: https://www.usenix.org/conference/foci16/workshop-program/presentation/nithyanand.

[2] M. Falahrastegar, H. Haddadi, S. Uhlig, R. Mortier, "Tracking Personal Identifiers Across the Web," Passive and Active Measurement Conference (PAM 2016), in *Lecture Notes in Computer Science*, vol. 9631, pp. 30–41: http://link.springer.com/chapter/10.1007%2F978-3-319-30505-9_3.

[3] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, J. Crowcroft, "Breaking for Commercials: Characterizing Mobile Advertising," in *Proceedings of the 2012 ACM Internet Measurement Conference*, pp. 343–356: http://dl.acm.org/citation.cfm?id=2398812.

[4] I. Thomson, "Ad-Blocker Blocking Web Sites Face Legal Peril at Hands of Privacy Bods": http://www.theregister.co.uk/2016/04/23/anti_ad_blockers_face_legal_challenges/.

Interview with Gordon Lyon

RIK FARROW

Fyodor Vaskovich (known to his family as Gordon Lyon) authored the open source Nmap Security Scanner in 1997 and continues to coordinate its development. He also maintains the seclists.org, insecure.org, sectools.org, secwiki.org, and nmap.org security resource sites and has authored seminal papers on remote operating system detection and stealth port scanning. He is a founding member of the Honeynet Project, former President of Computer Professionals for Social Responsibility (CPSR), and Technical Advisory Board member for Qualys and AlienVault. He also authored or co-authored the books *Nmap Network Scanning*, *Know Your Enemy: Honeynets*, and *Stealing the Network: How to Own a Continent.* fyodor@nmap.org

Rik Farrow is the editor of *;login:*. rik@usenix.org

I've known Gordon Lyon, as Fyodor Vaskovich, for over 15 years. Most of our meetings have been online, but we've also attended some meetups in between BlackHat and DefCon.

Fyodor's claim to fame is that he has been working on a network scanning tool, Nmap (nmap.org) longer than I've known him. What began as improvements on the handful of scanning tools available in the '90s has become an open source project as well as a successfully funded business. If you've never used Nmap, it is both a security tool and one useful to network and system admins: Nmap quickly scans networks in a long list of different ways.

There really isn't much on the Web about Fyodor, and we are fortunate that he has taken the time to tell us a bit more about his background and how various parts of Nmap came to be.

Rik Farrow: Can you tell us when you began programming?

Fyodor Vaskovich: I was lucky enough to grow up in a household with computers, since my father had previously worked at IBM. We had systems like the Commodore VIC-20 and the Apple IIe around 1980. And then later we upgraded to an IBM XT clone. In many ways these systems were all less powerful than what we now use in thermostats and watches. But interacting with them and exploring how they worked really taught me a lot.

Eventually I bought my own computer, a 286 running at a whopping 12 MHz! But the best part was the 2400 bps modem! It opened a whole new world of communicating with other computer users on BBSes. Not all over the world, unfortunately, since I couldn't afford long distance telephone calls. But all over the Phoenix metro area, where I lived in those days, was a good start!

Interacting with BBS software was interesting and fun, but I was blown away when I was introduced to my first UNIX shell. It was just so powerful, and the system running this shell had such a fast connection to the fledgling Internet! My high school friend had an account there, too, and raiding each other's accounts taught us a lot about UNIX permissions and security.

I started college studying molecular and cellular biology, but it didn't take me long to decide that computers (especially networking and security) were my real passions, and I switched to computer science. So the tl;dr answer to your question is that I was a self-taught programmer at first, but my university CS education took me to a much higher level.

RF: Until Nmap came out, there were three scanners that I know of: SATAN, very limited and more of a vuln scanner; the one written by Prof (better known as Julian Assange); and part of Hobbit's Netcat (nc). Were any of these early, primitive programs an influence on the first version of Nmap?

FV: Good memory! Yes, I had a full directory of port scanners at that time including the ones you mentioned, plus a very simple one named `reflscan`. Assange's `strobe` was definitely the fastest and most advanced at the time, and I learned a lot from it about nonblocking sockets, rate limiting, and congestion control. But it didn't support options such as the more stealthy

SYN scan or UDP protocol scanning. So I ended up hacking most of these scanners to add new options and features, but in the end I decided it was better to just write my own scanner containing everything I wanted in one program.

I wrote Nmap for my own network discovery use, but then decided to publish it in *Phrack* just in case anyone else found it useful. Apparently they did, and I was inundated with bug reports, fixes, and ideas. So I decided to release one more version. It continued to snowball, but I never could have imagined that I'd still be at it 19 years later as my full-time job!

RF: OS identification is an even earlier feature of Nmap than scripting. I'd become vaguely aware of some obvious differences between operating systems: for example, time-to-live (TTL) values falling into three well-defined buckets. But Nmap looks at a lot more when asked to identify an operating system. Can you tell us how OS identification got started and a bit about the features measured?

FV: Great question! I released Nmap in 1997 as a relatively simple tool for host discovery and port scanning, but then I became increasingly fascinated by the use of TCP/IP stack fingerprinting for the purposes of detecting the operating system running on a remote host. The idea is to send a series of probes to the host's IP address and study the results very carefully for patterns that indicate a certain OS. A tool named Queso really inspired me in this regard, and I tried to take it to the next level with Nmap. I first released Nmap IPv4 OS detection in 1998 and since then it has grown to test dozens of characteristics, from the way initial sequence numbers are generated to the TCP flags used in responses to unexpected packets. TCP options also disclose a wealth of information. Not just whether the remote system supports an option, but also what values it selects for options such as window scale or maximum segment size and the order in which it chooses to list the options in the TCP header.

Our traditional IPv4 OS detection system requires experts to generate signatures of system behaviors based on thousands of fingerprint submissions by Nmap users. This manual system has served us well, but I'm particularly excited about our new IPv6 OS detection system, which uses a very different technique. Instead of requiring experts to write signatures based on specific tests, our new system just feeds all the header data to machine learning systems and lets the computer do the hard work of finding patterns and characteristics of different operating systems in order to match a test subject system with our reference database of known OS responses. Right now we use a machine-learning classification technique called linear regression, but we had an intern this summer named Prabhjyot Singh who worked with a couple of great mentors in the Nmap Project to convert that to a random forest classifier. The results have so far been quite promising, and he continues to improve and test the system even though his Google-sponsored internship is over. We're hoping to integrate it into an Nmap release this year.

I tried to describe this as well as I could in two paragraphs, but folks wanting the full details can refer to our extensive online documentation (https://nmap.org/book/osdetect.html).

RF: Can you tell us about adding scripting capabilities to Nmap?

FV: Sometimes it seems like every program grows and grows until it has its own scripting language and maybe its own text editor. Well, we have no plans to integrate Emacs into Nmap, but we decided that a scripting engine was essential. As Nmap grew, there were more and more neat ideas that we couldn't implement because adding them all would bloat the core of Nmap too much. Also, as Nmap became larger and more complex, it became harder for casual contributors to make improvements because they had to learn so many subsystems. And it was difficult to review all their patches because any mistake in the code could crash all of Nmap. With the scripting engine, people write to a relatively simple API and don't have to know anything about Nmap internals. Their code stays separate rather than bloating Nmap itself, and mistakes should only cause that particular script to fail.

We decided to use the Lua scripting language (https://www.lua.org/) because it is simple and small and easy to embed in a way that enables extremely fast network performance. It's mostly known in the gaming world, but some versions of other networking tools like Wireshark and Snort use it as well.

The scripting engine has been a great success! We now have more than 500 scripts, all documented at https://nmap.org/nsedoc/.

RF: From reading the Nmap 7.2 announcement, it sounds like there is an actual community involved with Nmap. Some open source project authors say "we" when really they are the only person writing code. Can you tell us about your code community?

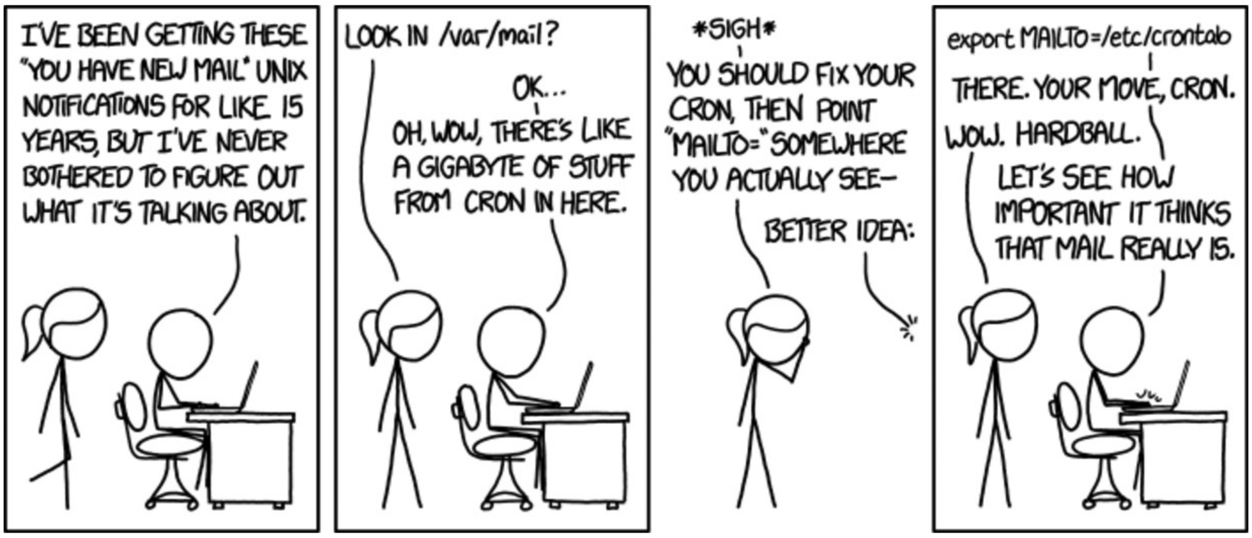
FV: Yes, we are fortunate to have a community which helps in myriad ways. Not everyone is a programmer, but we also receive contributions in the form of people submitting unknown operating system or version fingerprinting results, helping to translate our documentation, creating art, or just answering questions on the mailing list. A good bug report is a gift, too.

Contributing code has become harder as Nmap has grown more complex. NSE scripts, as discussed above, are a happy exception to that rule. We currently have more than 20 core code committers (https://svn.nmap.org/nmap/docs/committers.txt), although only a fraction are active at a given time. Every summer for the last 12 years Google has helped us fund many full-time interns (college or grad students) from all over the world. We've had 78 of them so far, and the program has been hugely successful both in developing new features and in mentoring new contributors who sometimes help out for years afterward. Many of our former interns have actually become mentors for their own Nmap interns years later!

RF: I am guessing that you are an independent consultant and that the licensing of Nmap and pen-testing are how you support yourself. Is that true?

FV: That used to be the case, but Nmap grew so much and its development became so complex that I quit consulting and devoted myself to Nmap in 2002. I'm really happy with our dual-license program. It allows open source use of Nmap for free, but companies who want to distribute it with their commercial software have to pay a license fee. The fee also includes support, indemnification, and warranties that companies expect anyway. Not only does this fund the project, but it means that Nmap has many more users who may not realize it is even being used to perform network discovery under the covers of their proprietary software. We have roughly 100 licensees, from some of the largest companies in the world to tiny new startups. I'm always excited to see what innovative uses they can find for Nmap results. For example, exploitation frameworks can use Nmap to determine the remote OS so that they send the right shellcode for exploiting a vulnerability. They may only get one chance because the wrong payload could crash the service. Another neat usage is enterprise software installers that use Nmap to find all the systems compatible with their agent or management software.

XKCD


I'VE BEEN GETTING THESE "YOU HAVE NEW MAIL" UNIX NOTIFICATIONS FOR LIKE 15 YEARS, BUT I'VE NEVER BOTHERED TO FIGURE OUT WHAT IT'S TALKING ABOUT.
LOOK IN /var/mail?
OK...
OH, WOW, THERE'S LIKE A GIGABYTE OF STUFF FROM CRON IN HERE.
SIGH
YOU SHOULD FIX YOUR CRON, THEN POINT "MAILTO=" SOMEWHERE YOU ACTUALLY SEE–
BETTER IDEA:
export MAILTO=/etc/crontab
THERE. YOUR MOVE, CRON.
WOW. HARDBALL.
LET'S SEE HOW IMPORTANT IT THINKS THAT MAIL REALLY IS.

xkcd.com

Statement of Ownership, Management, and Circulation, 9/30/16

Title: *;login:* Pub. No. 0008-334. Frequency: Quarterly. Number of issues published annually: 4. Subscription price $90.

Office of publication: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Headquarters of General Business Office of Publisher: Same. Publisher: Same.

Editor: Rik Farrow; Managing Editor: Michele Nelson, located at office of publication.

Owner: USENIX Association. Mailing address: As above.

Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: None.

The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes have not changed during the preceding 12 months.

Extent and Nature of Circulation			Average No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue (Fall 2016) Published Nearest to Filing Date
a. Total Number of Copies			2935	2800
b. Paid Circulation	(1)	Outside-County Mail Subscriptions	1278	1261
	(2)	In-County Subscriptions	0	0
	(3)	Other Non-USPS Paid Distribution	882	868
	(4)	Other Classes	0	0
c. Total Paid Distribution			2160	2129
d. Free Distribution By Mail	(1)	Outside-County	77	76
	(2)	In-County	0	0
	(3)	Other Classes Mailed Through the USPS	19	23
	(4)	Free Distribution Outside the Mail	512	288
e. Total Free Distribution			608	387
f. Total Distribution			2768	2516
g. Copies Not Distributed			167	284
h. Total			2935	2800
i. Percent Paid			78%	85%
Paid Electronic Copies			401	440
Total Paid Print Copies			2561	2569
Total Print Distribution			3169	2956
Percent Paid (Both Print and Electronic Copies)			81%	87%

Tuning OpenZFS

ALLAN JUDE AND MICHAEL W. LUCAS

Allan Jude is VP of Operations at ScaleEngine Inc., a global HTTP and Video Streaming CDN, where he makes extensive use of ZFS on FreeBSD. He is also the host of the video podcasts BSDNow.tv (with Kris Moore) and TechSNAP.tv. He is a FreeBSD src and doc committer, and was elected to the FreeBSD Core team in the summer of 2016. allanjude@freebsd.org

Michael W. Lucas has used UNIX since the late '80s and started his sysadmin career in 1995. He's the author of over 20 technology books, including *Absolute OpenBSD, PAM Mastery,* and *SSH Mastery.* Lucas lives with his wife in Detroit, Michigan, has pet rats, and practices martial arts. mwlucas@michaelwlucas.com

OpenZFS is such a powerful file system that it has found its way into illumos, Linux, FreeBSD, and other operating systems. Its flexibility requires whole new ways of thinking, however. If you're using OpenZFS for a special purpose, such as database storage or retaining particular sizes of files, you'll want to tune the file system for those purposes.

This article uses FreeBSD as a reference platform, as it's one of the biggest OpenZFS consumers. You will need to change paths and such for other operating systems, but all the ZFS information is consistent across platforms.

Recordsize

While many ZFS properties impact performance, start with `recordsize`.

The `recordsize` property gives the maximum size of a logical block in a ZFS dataset. The default `recordsize` is 128 KB, which comes to 32 sectors on a disk with 4 KB sectors, or 256 sectors on a disk with 512-byte sectors. The maximum `recordsize` was increased to 1 MB with the introduction of the `large_blocks` feature flag in 2015. Many database engines prefer smaller blocks, such as 4 KB or 8 KB. It makes sense to change the `recordsize` on datasets dedicated to such files. Even if you don't change the `recordsize`, ZFS automatically sizes records as needed. Writing a 16 KB file should take up only 16 KB of space (plus metadata and redundancy space), not waste an entire 128 KB record.

The most important tuning you can perform for an application is the dataset block size. If an application consistently writes blocks of a certain size, `recordsize` should match the block size used by the application. This becomes really important with databases.

Databases and ZFS

Many ZFS features are highly advantageous for databases. Every DBA wants fast, easy, and efficient replication, snapshots, clones, tunable caches, and pooled storage. While ZFS is designed as a general-purpose file system, you can tune it to make your databases fly.

Databases usually consist of more than one type of file, and since each has different characteristics and usage patterns, each requires different tuning. We'll discuss MySQL and PostgreSQL in particular, but the principles apply to any database software.

Tuning the block size avoids write amplification. Write amplification happens when changing a small amount of data requires writing a large amount of data. Suppose you must change 8 KB in the middle of a 128 KB block. ZFS must read the 128 KB, modify 8 KB somewhere in it, calculate a new checksum, and write the new 128 KB block. ZFS is a copy-on-write file system, so it would wind up writing a whole new 128 KB block just to change that 8 KB. You don't want that. Now multiply this by the number of writes your database makes. Write amplification eviscerates performance.

Low-load databases might not need this sort of optimization, but on a high-performance system it is invaluable. Write amplification reduces the life of SSDs and other flash-based storage that can handle a limited volume of writes over their lifetime.

The different database engines don't make `recordsize` tuning easy. Each database server has different needs. Journals, binary replication logs, error and query logs, and other miscellaneous files also require different tuning.

Before creating a dataset with a small `recordsize`, be sure you understand the interaction between VDEV type and space utilization. In some situations, disks with the smaller 512-byte sector size can provide better storage efficiency. It is entirely possible you may be better off with a separate pool specifically for your database, with the main pool for your other files.

For high-performance systems, use mirrors rather than any type of RAID-Z. Yes, for resiliency you probably want RAID-Z. Hard choices are what makes system administration fun!

All Databases

Enabling lz4 compression on a database can, unintuitively, decrease latency. Compressed data can be read more quickly from the physical media, as there is less to read, which can result in shorter transfer times. With lz4's early abort feature, the worst case is only a few milliseconds slower than opting out of compression, but the benefits are usually quite significant. This is why ZFS uses lz4 compression for all of its own metadata and for the L2ARC (level 2 adaptive replacement cache).

The Compressed ARC feature recently landed in OpenZFS and is slowly trickling out to OpenZFS consumers. Enabling cache compression on the dataset allows more data to be kept in the ARC, the fastest ZFS cache. In a production case study done by Delphix, a database server with 768 GB of RAM went from using more than 90 percent of its memory to cache a database to using only 446 GB to cache 1.2 TB of compressed data. Compressing the in-memory cache resulted in a significant performance improvement. As the machine could not support any more RAM, compression was the only way to improve. When your operating system gets compressed ARC, definitely check it out.

ZFS metadata can also affect databases. When a database is rapidly changing, writing out two or three copies of the metadata for each change can take up a significant number of the available IOPS of the backing storage. Normally, the quantity of metadata is relatively small compared to the default 128 KB record size. Databases work better with small record sizes, though. Keeping three copies of the metadata can cause as much disk activity, or more, than writing actual data to the pool.

Newer versions of OpenZFS also contain a `redundant_metadata` property, which defaults to *all*. This is the original behavior from previous versions of ZFS. However, this property can also be set to *most*, which causes ZFS to reduce the number of copies of some types of metadata that it keeps.

Depending on your needs and workload, allowing the database engine to manage caching might be better. ZFS defaults to caching much or all of the data from your database in the ARC, while the database engine keeps its own cache, resulting in wasteful double caching. Setting the `primarycache` property to *metadata* rather than the default *all* tells ZFS to avoid caching actual data in the ARC. The `secondarycache` property similarly controls the L2ARC.

Depending on the access pattern and the database engine, ZFS may already be more efficient. Use a tool like `zfsmon` from the `zfs-tools` package to monitor the ARC cache hit ratio, and compare it to that of the database's internal cache.

Once the Compressed ARC feature is available, it might be wise to consider reducing the size of the database's internal cache and let ZFS handle the caching instead. The ARC might be able to fit significantly more data in the same amount of RAM than your database can.

Now let's talk about some specific databases.

MySQL—InnoDB/XtraDB

InnoDB became the default storage engine in MySQL 5.5 and has significantly different characteristics than the previously used MyISAM engine. Percona's XtraDB, also used by MariaDB, is similar to InnoDB. Both InnoDB and XtraDB use a 16 KB block size, so the ZFS dataset that contains the actual data files should have its `recordsize` property set to match. We also recommend using MySQL's `innodb_one_file_per_table` setting to keep the InnoDB data for each table in a separate file, rather than grouping it all into a single `ibdata` file. This makes snapshots more useful and allows more selective restoration or rollback.

Store different types of files on different datasets. The data files need 16 KB block size, lz4 compression, and reduced metadata. You might see performance gains from caching only metadata, but this also disables prefetch. Experiment and see how your environment behaves.

```
# zfs create -o recordsize=16k -o compress=lz4 -o redundant_
metadata=most -o primarycache=metadata mypool/var/db/mysql
```

The primary MySQL logs compress best with gzip, and don't need caching in memory.

```
# zfs create -o compress=gzip1 -o primarycache=none mysql/var/
log/mysql
```

The replication log works best with lz4 compression.

```
# zfs create -o compress=lz4 mypool/var/log/mysql/replication
```

Tell MySQL to use these datasets with these my.cnf settings.

```
data_path=/var/db/mysql
log_path=/var/log/mysql
binlog_path=/var/log/mysql/replication
```

You can now initialize your database and start loading data.

MySQL—MyISAM

Many MySQL applications still use the older MyISAM storage engine, either because of its simplicity or just because they have not been converted to using InnoDB.

MyISAM uses an 8 KB block size. The dataset record size should be set to match. The dataset layout should otherwise be the same as for InnoDB.

PostgreSQL

ZFS can support very large and fast PostgreSQL systems, if tuned properly. Don't initialize your database until you've created the needed datasets.

PostgreSQL defaults to using 8 KB storage blocks for everything. If you change PostgreSQL's block size, you must change the dataset size to match.

The examples here use FreeBSD. Other operating systems will use different paths and have their own database initialization scripts. Substitute your preferred operating system commands and paths as needed.

PostgreSQL data goes in `/usr/local/pgsql/data`. For a big install, you probably have a separate pool for that data. Here I'm using the pool `pgsql` for PostgreSQL.

```
# zfs set mountpoint=/usr/local/pgsql pgsql
# zfs create pgsql/data
```

Now we have a chicken-and-egg problem. PostgreSQL's database initialization routine expects to create its own directory tree, but we want particular subdirectories to have their own datasets. The easiest way to do this is to let PostgreSQL initialize, and then create datasets and move the files. Here's how FreeBSD initializes a PostgreSQL database.

```
# /usr/local/etc/rc.d/postgresql oneinitdb
```

The initialization routine creates databases, views, schemas, configuration files, and all the other components of a high-end database. Now you can create datasets for the special parts.

Our test system's PostgreSQL install stores databases in /usr/local/pgsql/data/base. The Write Ahead Log, or WAL, lives in `/usr/local/pgsql/data/pg_xlog`. Move both of these out of the way.

```
# cd /usr/local/pgsql/data
# mv base base-old
# mv pg_xlog pg_xlog-old
```

Both of these parts of PostgreSQL use an 8 KB block size, and you would want to snapshot them separately, so create a dataset for each. As with MySQL, tell the ARC to cache only the metadata. Also tell these datasets to bias throughput over latency with the `logbias` property.

```
# zfs create -o recordsize=8k -o redundant_metadata=most -o
primarycache=metadata logbias=throughput pgsql/data/pg_xlog
# zfs create -o recordsize=8k -o redundant_metadata=most -o
primarycache=metadata logbias=throughput pgsql/data/base
```

Copy the contents of the original directories into the new datasets.

```
# cp -Rp base-old/* base
# cp -Rp pg_xlog-old/* pg_xlog
```

You can now start PostgreSQL.

Tuning for File Size

ZFS is designed to be a good general-purpose file system. If you have a ZFS system serving as file server for a typical office, you don't really have to tune for file size. If you know what size of files you're going to have, though, you can make changes to improve performance.

Small Files

When creating many small files at high speed in a system without a SLOG (Separate (ZFS-Intent) Log), ZFS spends a significant amount of time waiting for the files and metadata to finish flushing to stable storage.

If you are willing to risk the loss of any new files created in the last five seconds (or more if your vfs.zfs.txg.timeout is higher), setting the sync property to disabled tells ZFS to treat all writes as asynchronous. Even if an application asks that it not be told that the write is complete until the file is safe, ZFS returns immediately and writes the file along with the next regularly scheduled txg.

A high-speed SLOG lets you store those tiny files both synchronously and quickly.

Big Files

ZFS recently added support for blocks larger than 128 KB via the `large_block` feature. If you're storing many large files, certainly consider this. The default maximum block size is 1 MB.

Theoretically, you can use block sizes larger than 1 MB. Very few systems have extensively tested this, however, and the interaction with the kernel memory allocation subsystem has not been tested under prolonged use. You can try really large record sizes, but be sure to file a bug report when everything goes sideways.

On FreeBSD, the sysctl `vfs.zfs.max_recordsize` controls the maximum block size. On Linux, `zfs_max_recordsize` is a module parameter.

Once you activate `large_blocks` (or any other feature), the pool can no longer be used by hosts that do not support the feature. Deactivate the feature by destroying any datasets that have ever had their `recordsize` set to larger than 128 KB.

Storage systems struggle to balance latency and throughput. ZFS uses the `logbias` property to decide which way it should lean. ZFS uses a `logbias` of *latency* by default, so that data is quickly synched to disk, allowing databases and other applications to continue working. When dealing with large files, changing the `logbias` property to *throughput* might result in better performance. You must do your own testing and decide which setting is right for your workload.

With a few adjustments, you can make your database's file system fly…leaving you capacity to cope with your next headache.

This article was adapted from Allan Jude and Michael W Lucas, *FreeBSD Mastery: Advanced ZFS* (Tilted Windmill Press, 2016).

Do you have a USENIX Representative on your university or college campus? If not, USENIX is interested in having one!

The USENIX Campus Rep Program is a network of representatives at campuses around the world who provide Association information to students, and encourage student involvement in USENIX. This is a volunteer program, for which USENIX is always looking for academics to participate. The program is designed for faculty or staff who directly interact with students. We fund one representative from a campus at a time.

A campus rep's responsibilities include:

- Maintaining a library (online and in print) of USENIX publications at your university for student use
- Distributing calls for papers and upcoming event brochures, and re-distributing informational emails from USENIX
- Encouraging students to apply for travel grants to conferences
- Providing students who wish to join USENIX with information and applications
- Helping students to submit research papers to relevant USENIX conferences
- Providing USENIX with feedback and suggestions on how the organization can better serve students

In return for being our "eyes and ears" on campus, the Campus Representative receives access to the members-only areas of the USENIX Web site, free conference registration once a year (after one full year of service as a Campus Representative), and electronic conference proceedings for downloading onto your campus server so that all students, staff, and faculty have access.

To qualify as a campus representative, you must:

- Be full-time faculty or staff at a four-year accredited university
- Have been a dues-paying member of USENIX for at least one full year in the past

For more information about our Student Programs, please contact office@usenix.org

www.usenix.org/students

Lessons from Iraq for Building and Running SRE Teams

From the Assembly Line to the Web

KURT ANDERSEN

Kurt Andersen has been active in the anti-abuse community for over 15 years and is currently the Senior IC for the Consumer Services SRE team at LinkedIn. He also works as one of the Program Committee Chairs for the Messaging, Malware, and Mobile Anti-Abuse Working Group (M3AAWG.org). He has spoken at M3AAWG, Velocity, SREcon, and SANOG on various aspects of reliability, authentication, and security. He is one of the co-chairs for SREcon17 Americas, to be held in March 2017. kurta@linkedin.com

General Stanley McChrystal led the Joint Special Operations Task Force in Iraq in the mid to late 2000s. While in command of the Task Force, he was responsible for transforming an organization that was dominated by Taylorist reductionism into an agile, responsive network that could dynamically adapt and win in the threat landscape around them. In his book *Team of Teams: New Rules of Engagement for a Complex World* [1], McChrystal outlines the key lessons that emerged from that process. The same issues and challenges face Site Reliability Engineers and managers for SRE teams as we cope with the complexity of our own and partner ecosystems.

The Challenges of Growth in a Complex Environment

As one of the Senior Individual Contributors in the SRE (Site Reliability Engineering) group at LinkedIn, I and the other leaders in the organization are always looking to improve the capabilities and effectiveness of our SRE teams. As our membership, service offerings, and engineering teams have grown over the last few years, we have had to confront new challenges. Figure 1 shows the nearly seven-fold growth in the SRE organization over the last three years. The number of internal services that we support has grown even faster. This rate of growth makes it difficult to keep up with the influx of new personnel, which in turn makes it important to become even more conscious about communication, but some of our key difficulties have arisen because of changes in our physical presence.

In 2013, we were located on a single campus where one could walk from one end to the other in a few minutes. Our SRE organization initially expanded to include a team in Bangalore, India, operating almost completely opposite to Pacific Coast daytime hours, but that was less of a challenge than having our single campus fragmented with the opening of new offices 40 miles away in San Francisco, 10 miles away in Sunnyvale, as well as 3000 miles away in New York City. Suddenly, simple chats with colleagues on other teams required new logistical coordination which, in many cases, ended up choking off the conversations.

Matching the growth in individual engineers, what had been a very simple, limited management framework in 2013 had burgeoned by 2016 into multiple layers that matched the physical separation of the different embedded SRE teams. (LinkedIn's SRE teams are co-located with the development teams that they support, hence the term "embedded.") The separation and fragmentation between the different teams was leading to duplicated effort as well as reduced cohesion of the overall SRE organization.

In this context, and seeking strategies to address these challenges, I ran across McChrystal's book. The approach that he employed in transforming the bureaucratic environment of the US Armed Forces Joint Special Operations Command (JSOC) into an adaptable, agile fighting system mirrored many of the challenges that we were facing with the SRE organization, although on a fairly different scale—for instance, his version of the daily "standup" meeting involved up to 7000 people from around the globe and took two hours, six days a week!

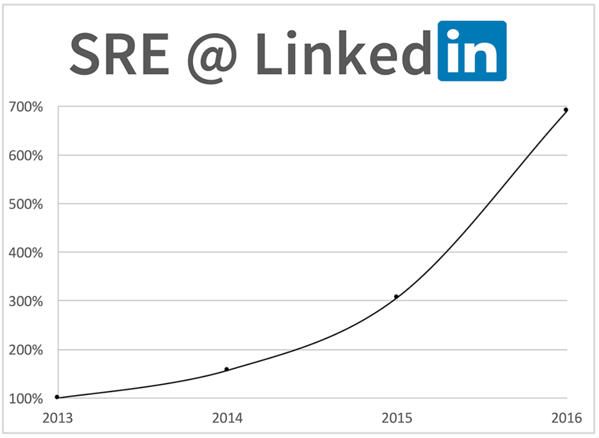


Figure 1: Growth in headcount in the SRE teams at LinkedIn

Be Careful What You Optimize For

Dating back to the midpoint of the last millennium, people became fixated on the idea of the "clockwork universe." This principle arose from a mechanical concept related to Newtonian physics and posits that if one knows all of the inputs to a particular system, it should be possible to exactly predict the outputs from that system. Fantastically complex mechanical clocks can be found in town squares across Europe as a testament to this perspective and the importance that it achieved as an organizing principle. Mechanical calculation led to the difference engines of Charles Babbage (circa 1822), which were intended to create the logarithm tables that were used for calculations of many different phenomena. While Babbage's difference engine was not actually constructed during his lifetime, the plans he drew up served as the basis for a project to build the engine, using metallurgy and capabilities that would have been feasible during his lifetime, which was completed in 2008. The Jacquard "programmable" loom for pattern weaving (1801) was another instance of similar thought toward automating repetitious tasks.

Around the early 1900s, Frederick Winslow Taylor extended the idea of a "clockwork universe" to create the principles of "scientific management." The key principles of this school of thought are:

- That absolute, inflexible standards be maintained throughout your establishment.
- That each employee of your establishment should receive every day clear-cut, definite instructions as to just what he is to do and how he is to do it, and these instructions should be exactly carried out, whether they are right or wrong.

Taylor strictly separated the roles of worker and manager. The manager had five functions: planning, organizing, commanding, coordinating, and controlling. The ultimate incentive for the manager was to gather and centralize more information in order to push more and more efficient directives to the organization, while the Taylorist worker's role was two-fold: provide information and await commands. All effort was to be expended in the pursuit of maximum efficiency, and all individuals were to be treated as fungible resources, no different from a bolt or a nail.

"Scientific management's" rigidly hierarchical and extremely siloed model of operations took the repetition of Babbage's difference engine and applied it to people. Sadly, the effects of treating people as expendable parts can be seen in the inhumane conditions that characterized the trench warfare of World War I. The scars of that warfare can still be seen across the landscape of Europe today, and the scars of inhumane management practices still plague many companies and individuals.

In the field of computers, the traditional sysadmin role was siloed, often undervalued, and only noticed when something went wrong. It led to the caricature of the BOFH (bastard operator from hell [6]).

From Complicated to Complex

The problems and limitations of the clockwork universe became more evident within the scientific community even as Taylor was making his mark on the management world. Relativity, quantum mechanics, and eventually the atomic bomb revolutionized physics, and the development of chaos theory in the 1970s showed that there is a phase-change difference between the complicated, mechanistic "clockwork" to the turbulent chaos of the modern world, where critical differences arise from imperceptible origins or interactions.

In his book, McChrystal points out that complicated systems can be made "robust," meaning less prone to failure, by building them up with more of what is already there: for instance, the Pyramids of Giza have survived millennia by being made of masses of stone. To make them stronger, you would just add more stone.

Complex environments, on the other hand, are "resilient" by the nature of the interconnections between their constituent parts. A coral reef is an example of a complex, resilient ecosystem that thrives based on the variety of organisms which share the space. Another example of complex interactions would be the way that a successful soccer team works together. If every team member is only aware of what is happening within their own local area of the field, and, even worse, if "their area of the field" is rigidly defined, then the team has little or no chance of being successful. Whether you are the goalie or a forward, you should have one "job" on the team: to win.

McChrystal pointed out that "to each unit [within the military bureaucracy that made up JSOC], the piece of the war that really mattered was the piece inside their box on the org chart." To succeed in their mission in Iraq, they needed to overcome the silos. They already had a model of highly effective small squads of operators in the elite military units such as the SEALS and

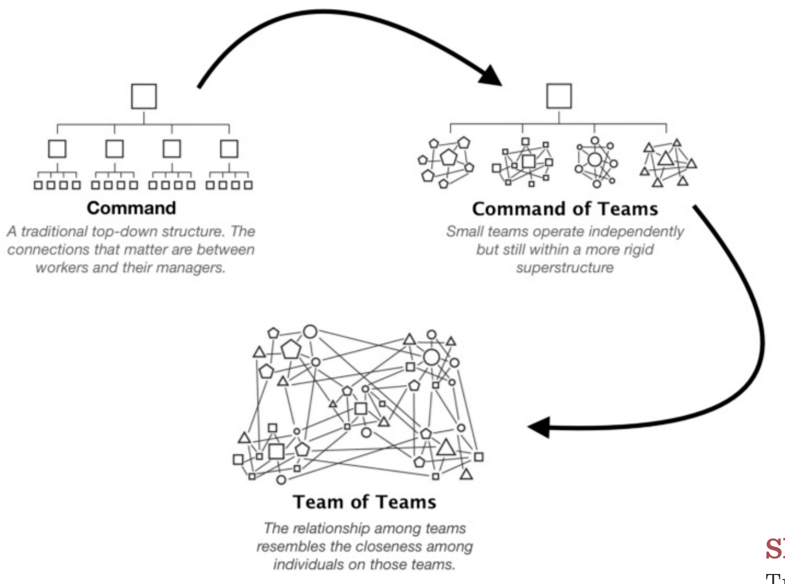

Command
A traditional top-down structure. The connections that matter are between workers and their managers.
Command of Teams
Small teams operate independently but still within a more rigid superstructure
Team of Teams
The relationship among teams resembles the closeness among individuals on those teams.

Figure 2: This diagram from *Team of Teams* illustrates various ways of organizing a command arrangement.

Rangers. The challenge was how to take that "small team effectiveness" and extend it across the organization of JSOC, moving from a "command of teams" to a "team of teams," and bringing in the adaptability and resilience of a network dynamic at all levels of the organization (see Figure 2).

McChrystal worked to build relationships and trust between disparate services by intentionally seeding key operators into other teams.

At LinkedIn, as our SRE organization rapidly grew, we have had to work on ensuring that we did not end up in silos. The division of our teams across multiple locations has made it somewhat natural for people to focus on those who are physically nearby, so we have instituted a number of initiatives, many organically driven, to break down the barriers that could otherwise occur.

The first strategy that we have found helpful, particularly for bootstrapping new technologies and driving adoption across our constantly growing engineering group, is what we call "virtual teams." There is a whole life cycle to the process for virtual teams, but generally they consist of like-minded folks from multiple management teams who share a particular interest or passion. As an example, our ELK (elasticsearch-logstash-kibana) environment at LinkedIn is run in a highly distributed manner, but the virtual team helps to guide by staying up to date on new versions and features while also adapting new versions to our internal deployment frameworks.

Another strategy that we use is based on our cultural value of transformation. As part of transformation, we think that learning is a never-ending aspect of work and life. Besides multiple tech talks that are provided by different teams and individuals every week, within the SRE organization we organize a full day every month of deep dive sessions into different technologies that SREs use. While not everyone in the organization is able to participate every month, this provides a common forum for people across the organization to meet together and improve their skills.

We have grown to the point where there are enough postmortems every week that we need to start drawing out higher level patterns, so a weekly "postmortem roll-up" provides this opportunity. We are also starting a program of inter-team rotations, where an engineer will spend half a quarter (six weeks) working on a different team. These different pieces are all helping us to avoid becoming a "command of teams" and maintain a "team of teams" dynamic.

Shared Awareness

Traditionally, and especially in the formulations of "scientific management," only particular people "need to know" information. Compartmentalizing information is a very effective way to create divisions and separations within an organization based on the simple difference of who is "in" and who is "out." Predicting exactly *who* needs to know a particular piece of information turns out to be a huge problem. In the war in Iraq, McChrystal found that opening up the flow of information was critical to the effectiveness of the teams of intelligence analysts and operators on the ground. He cites the problems that GM faced in dealing with their famous ignition switch recall because of compartmentalized information within the company. Simply no one had enough of the picture to recognize and take action quickly enough to avoid the deaths and subsequent public outcry leading to Congressional hearings that finally shone light on the underlying problem.

Most practitioners of DevOps and Site Reliability Engineering have recognized that measuring and monitoring is critical, but making the data available to everyone who can benefit from it is equally critical. It's important to take as broad a view of "everyone who can benefit" as possible because you never know how two seemingly independent issues may have a common contributing cause.

In *Social Physics* [2], Alex Pentland points out, "It is the idea flow within a community that builds the intelligence that makes it successful." Pentland expands upon that to identify both exploration of new ideas as well as engagement with concepts as critical determinants for effective idea flow. Engagement with the community, both learning from and giving back to, is one of our (LinkedIn SRE) keys to a robust organization. The company's strong tradition and culture of transparency at all levels also reinforces the practice of sharing information widely and freely.

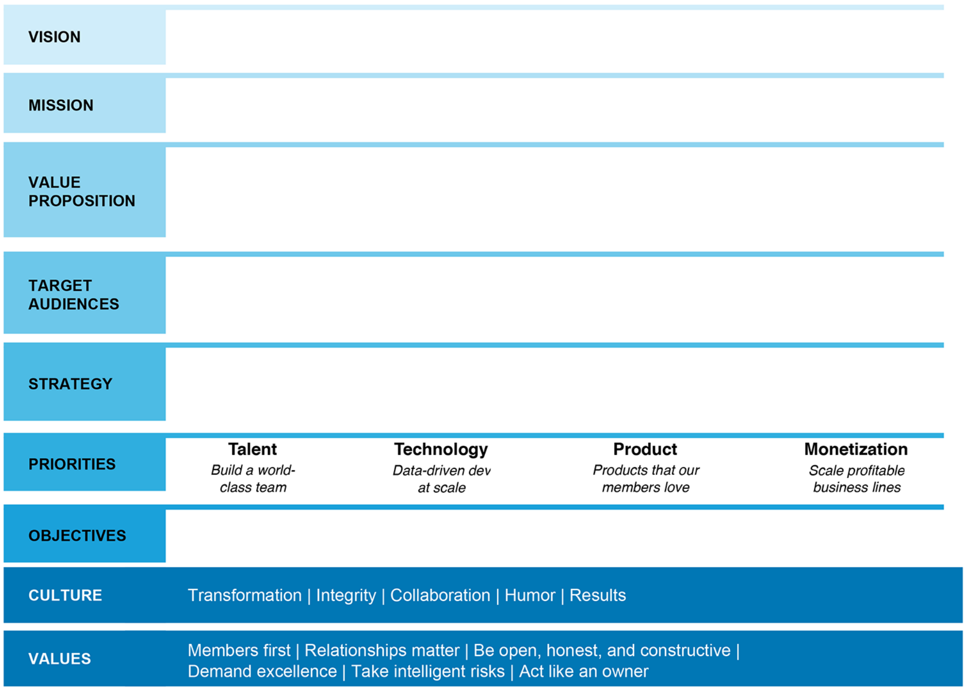

VISION
MISSION
VALUE PROPOSITION
TARGET AUDIENCES
STRATEGY
PRIORITIES
Talent
Build a world-class team
Technology
Data-driven dev at scale
Product
Products that our members love
Monetization
Scale profitable business lines
OBJECTIVES
CULTURE
Transformation | Integrity | Collaboration | Humor | Results
VALUES
Members first | Relationships matter | Be open, honest, and constructive | Demand excellence | Take intelligent risks | Act like an owner

Figure 3: LinkedIn vision-to-values framework

Empowered Execution

Teams of teams who have the shared understanding of the challenges and opportunities that face them are like a soccer team whose members are all paying attention and reacting to the situation across the entire field. But that team can still fail to be successful if the players have to check with the coach before they ever kick the ball. The example is a bit ludicrous, but many companies operate that way. Teams must be empowered to take action and make decisions on their own. It is critical to push decision-making as close to the action as possible.

A great example of a team responding on their own without the need for external guidance is illustrated in the *Site Reliability Engineering* chapter on incident response [3]. In the positive portrayal of effective incident response ("A Managed Incident"), the team of people know how and when to engage, have a good understanding of the different roles that need to be covered, and carry out the incident response empowered by that scaffolding.

Sidney Dekker, who is one of the foremost authorities on accident causation and human error, points out: "When we find that things go right under difficult circumstances, it's mostly because of people's adaptive capacity; their ability to recognize, adapt to, and absorb changes and disruptions, some of which might fall outside of what the system is designed or trained to handle." It is key, when considering automation and process design, to keep in mind the importance of people's adaptive capabilities. This applies to SRE teams building automation as well as to software developers building externally facing products.

In the example of the JSOC in Iraq, McChrystal cites the demonstrated value of these principles, which he terms "eyes on, hands off," as leading to an increase in both quality and speed of execution. When they had previously attempted via a "robustness" strategy to just "work harder," the team was able to raise their raids per month from 10 to 18, but by changing to an empowered team of teams, they increased the number of raids by a factor of 17, accomplishing 300 raids per month with only minor increases in personnel.

Peter Seibel, in his gigamonkeys blog [4], makes the case for investing in engineering effectiveness in proportion to the size of an engineering organization by modeling the multiplying effect of the right investments. We have found that SRE teams can catalyze improvements in site benefits that outstrip the investment of talent in the SRE teams themselves.

Leadership

The last major theme that McChrystal covers in his book has to do with what he learned about leading an organization which was made up of teams of teams. The metaphor that he adopted was to "lead like a gardener," which means that each touchpoint is an opportunity to encourage, guide, and strengthen the culture of the teams. The role of leaders became focused on providing the appropriate "good ground" for the groups.

At LinkedIn we have the benefit of having the core cultural tenets consistently reinforced throughout the organization. They provide touchpoints when making decisions. In proposing any new initiative, people are expected to go through what we call a "vision to values" framework, illustrated by Figure 3.

Every initiative is evaluated through the lens of the organization's values and culture, and tested against the priorities. This applies to external, member-facing services and features, and also to every internal initiative.

Another part of leadership is planning for the future. Phil Libin, founder of Evernote, points out the "lesson of 3 and 10" [5] that he learned applies to technology companies, especially when growth is prolific: roughly every time something triples, the systems need to be refactored in order to continue working well. In our case, the SRE organization has grown by a factor of seven over three years, and many pieces have had to be adjusted. We have had to become much more intentional about collaboration and maintaining relationships that used to be organic. Our site traffic and membership have increased significantly. Our deployment velocity has gone from deploying 500 services once a month (each) to over 5000 deployments per month. We've expanded the number of locations with engineering presence and the number of datacenters that we actively serve traffic from.

As leaders, whether in the management ranks or among individual contributors, it is still critical to be inspiring people to work together to accomplish a shared vision. As a reader of *;login:*, just like the attendees at SREcon16 Europe where this talk was initially given, you are a leader in your profession. Accept the challenge and consider what effect you have among the people you interact with: *What are you inspiring others toward?*

This article is derived from a talk given at SREcon16 Europe in July 2016.

References

[1] S. McChrystal (with T. Collins, D. Silverman, C. Fussell), *Team of Teams: New Rules of Engagement for a Complex World* (Portfolio / Penguin, 2015).

[2] A. Pentland, *Social Physics: How Social Networks Can Make Us Smarter* (Penguin, 2015).

[3] B. Beyer, C. Jones, J. Petoff, and N. Murphy, "Managing Incidents," in *Site Reliability Engineering* (O'Reilly Media, 2016).

[4] Peter Seibel's blog: gigamonkeys.com.

[5] Rule of 3 & 10: https://www.sequoiacap.com/article/the-rule-of-3-and-10/.

[6] Wikipedia, "Bastard Operator from Hell," last modified on Aug. 22, 2016: https://en.wikipedia.org/wiki/Bastard_Operator_From_Hell.

Interrupt Reduction Projects

BETSY BEYER, JOHN TOBIN, AND LIZ FONG-JONES

Betsy Beyer is a Technical Writer for Google Site Reliability Engineering in NYC. She has previously provided documentation for Google Data Center and Hardware Operations teams. Before moving to New York, Betsy was a Lecturer in technical writing at Stanford University. She holds degrees from Stanford and Tulane. bbeyer@google.com

John Tobin is a Site Reliability Engineering Manager at Google Dublin. He manages Bigtable, Cloud Bigtable, and a cross-storage SRE automation project, and has worked on several of Google's storage systems. He holds an MSc from Trinity College Dublin, where he also worked before joining Google in 2010. johntobin@google.com

Liz Fong-Jones is a Senior Site Reliability Engineering Manager at Google and manages a team of SREs responsible for Google's storage systems. She lives with her wife, metamour, and two Samoyeds in Brooklyn. In her spare time, she plays classical piano, leads an EVE Online alliance, and advocates for transgender rights. lizf@google.com

Interrupts are a fact of life for any team that's responsible for maintaining a service or software. However, this type of work doesn't have to be a constant drain on your team's bandwidth or resources.

This article begins by describing the landscape of work faced by Site Reliability Engineering (SRE) teams at Google: the types of work we undertake, the logistics of how SRE teams are organized across sites, and the inevitable toil we incur. Within this discussion, we focus on interrupts: how teams initially approached tickets, and why and how we implemented a better strategy. After providing a case study of how the ticket funnel was one such successful initiative, we offer practical advice about mapping what we learned to other organizations.

Cognitive Flow State and Interrupts

Types of Work

Teams that write and maintain software must decide how to allocate people's time between the main types of work they undertake: planned development, immediate response to outages, and customer requests or lower-urgency production issues.

This article classifies work using the following conventions:

- **On-call/pager response:** Immediate response to outages
- **Tickets and interrupts:** Medium-urgency production issues and customer issues
- **Project work:** Proactive development and systems/network engineering work

In order of most to least urgent, we can make generalizations about how to handle each kind of work.

On-call/pager response is critical to the immediate health of the service, and requires a response with an urgency of minutes. Resolving each on-call incident takes between minutes and hours, and our response requires two components: time-sensitive mitigation followed by in-depth investigation and changes to prevent recurrence.

Tickets and other interrupts typically have an urgency of days to weeks and usually take between minutes and hours to resolve. These issues frequently prevent the team from achieving reliability goals or are blocking to either internal or external customers. Most teams at Google use a bug or ticket-tracking tool to manage tickets. For simplicity's sake, this article focuses specifically on tickets, the most common form of interrupts handled by our SRE teams.

Project work has an urgency ranging from weeks to the long backlog of wishlist ideas every team maintains. This type of work requires multiple days of sustained concentration in order to facilitate cognitive flow state [3]; co-scheduling interrupts or pages with project work will disrupt even the most diligent engineer's focus and prevent them from making meaningful progress.

Toil and Operational Load

Google categorizes pager response and tickets/interrupts as ***toil***, or reactive work. For a more in-depth discussion of toil, and why and how we seek to minimize it, see Chapter 5 of *Site Reliability Engineering*, "Eliminating Toil" [1], and the follow-up article in the Fall 2016 issue of *;login:*, "Invent More, Toil Less" [2]. Although dealing with toil can provide insight into which properties of the system ought to be improved in the long term, toil itself does not directly provide long-lasting value to a team or service. In a best-case scenario, toil merely allows a team to run in place; in a worst-case scenario, toil consumes enough engineering effort that a service eventually deteriorates. We cap toil at a maximum of 50% of a team's total engineering time, with the expectation that most teams will instead spend 60–70% of their time on project work.

In order to improve a service and reduce the human effort required in maintenance over time, teams must actively work on projects to reduce operational load. As toil decreases, teams can expand their scope to scalably support more services and undertake more interesting project work. Here we focus on one specific category of toil—tickets—and how we successfully reduced their drain on more meaningful project work.

Context: Google's SRE Team Setup

Most Google SRE teams are spread across two continents, with six to eight people in each of two sites, which together form one team responsible for a given set of services. We assign a primary and secondary on-call in each site, with each site handling 12 hours per day. The primary on-call typically handles most pages. Although your context will be different from ours, the principles we articulate in this article should translate to your organization.

Initial Approaches to Tickets

The Naïve Approach

Originally, many teams at Google approached tickets by assigning a primary on-call to handle pager duty, while round-robin assigning tickets across the team. This setup frequently led to undesirable outcomes, as engineers couldn't successfully undertake project work and ticket duty simultaneously. Handling random interruptions from tickets prevented engineers from entering a cognitive flow state, so they were unable to achieve meaningful traction on project work. On the other hand, engineers working heads-down on a project missed ticket response expectations because they weren't actively checking for tickets.

Some teams moved in the direction of centralization by assigning tickets to the expert with specialized knowledge or recent experience with a given component. However, this strategy resulted in uneven load and still disrupted people's attention, making project delivery unpredictable. Delegating the less intellectually interesting work to a team's newest, least experienced team members served only to burn out those team members. We clearly needed a way to dig out of this detrimentally ticket-driven workflow.

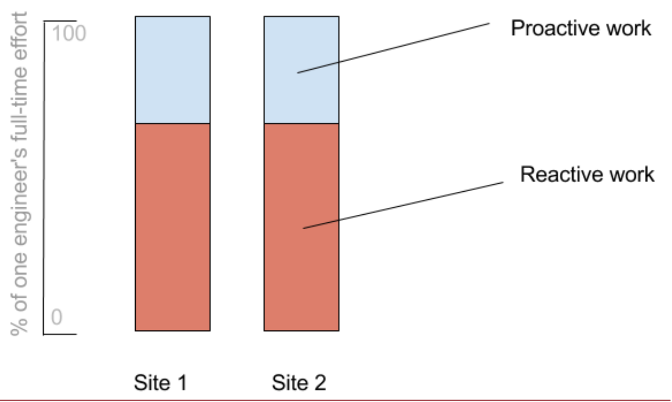


Figure 1: Splitting tickets between two people: the optimistic/naïve scenario

Centralizing Tickets

As discussed in the chapter "Dealing with Interrupts" in *Site Reliability Engineering* [1], spreading ticket load across an entire team causes context switches that impact valuable flow time. Once we articulated the need to preserve cognitive flow state [3], a better strategy became clear: we needed to staff a dedicated ticket rotation.

Most SRE teams naïvely implemented this strategy by tasking the secondary on-caller at each site with a somewhat vague and meandering directive:

- Work on tickets until the queue is empty, filing bugs for small improvements as you see ways to improve how specific tickets are handled, or to eliminate them entirely.
- See if you can find commonalities in the tickets you just solved, and do some proactive project-like work to prevent future tickets.

This strategy did at least acknowledge that proactive work is essential to keeping toil manageable as a service increases in size. However, it proved suboptimal: we were resolving tickets but not making small improvements.

In large part, inefficiencies resulted because overall ticket load doesn't necessarily come in whole-person increments. For instance, if your team fields enough tickets to occupy 1.2 people globally per week, you might decide to split the load between two people. While this split would ideally result in the work distribution shown in Figure 1, in actuality, the scenario shown in Figure 2 is much more likely.

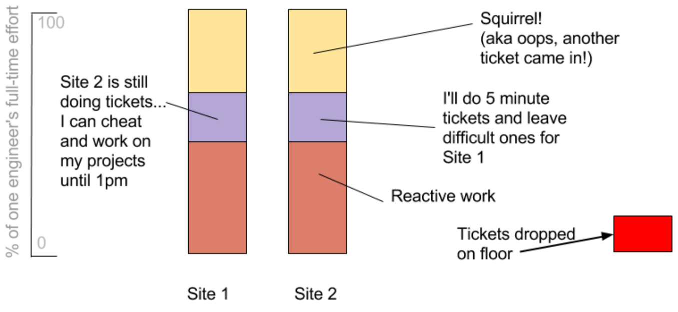


Figure 2: Splitting tickets between two people: the actual scenario

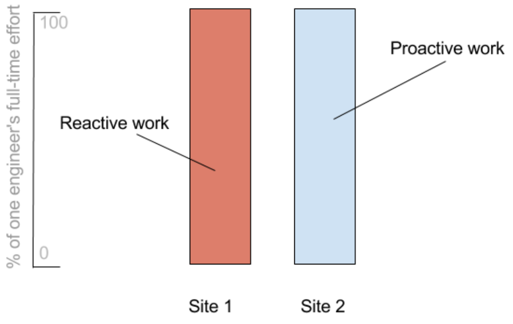


Figure 3: Work division through smarter interrupt handling

This setup is far from optimal in terms of cognitive flow, and time zone differences can lead team members to "cheat" on tickets—all too often, engineers at a site only start to work on tickets when coworkers at their partner site go home. At this point, it's tempting to leave frustrating tickets to SREs at your partner site, because you feel less personally connected to those teammates.

For the reasons described above, the ticket handling approach taken by Bigtable SRE (the team John and Liz manage) wasn't working well for a number of years.

A Better Alternative

We realized that while we initially focused on centralization and fairness/symmetry as a goal, we instead should have focused on maximizing cognitive flow state as a goal in and of itself. Accordingly, we readjusted our goal. We still tasked team members with identifying and solving commonalities in reactive/interrupt-driven work. However, we now explicitly allocated this job, which we'll refer to as "interrupt reduction project on duty," as a separate role from ticket work.

Why is this approach more effective? It hits the sweet spot of undertaking small to medium-sized projects to reduce operational load—projects that require more than 30 minutes of attention, but are too small to account for on a quarterly planning cycle. In our experience, we've identified many such projects that can be completed in less than a week. Assigning one person to work on interrupt reduction projects gives them enough uninterrupted cognitive flow time to complete those projects. Furthermore, assigning one dedicated person to ticket duty at a time ensures accountability for tickets: because that person is singly responsible for tickets, they can't divert responsibility for unresolved tickets onto the other site, or cherry-pick all the easy tickets.

Since instituting these changes, Bigtable SRE is meeting our ticket response expectations more often, and our incoming ticket volume has decreased, as shown in the graphs in Figure 4 (further explained in the case study).

Our division of ticket duty and interrupt reduction project work now looks like Figure 3.

In this new model, we rotate the ticket duty and interrupt reduction project roles between sites once per quarter. In order to ensure fairness, we rotate people into the ticketeer or interrupt reduction rotation according to on-call rotations (e.g., on a weekly basis).

The cost of handing off this new category of small to medium-sized interrupt reduction projects from shift to shift means that our new approach isn't a substitute for undertaking more substantial projects to reduce operational load. Substantive projects are still important—not only for reducing toil in the long term, but for career growth, as well. As such, they should still be accounted for using existing planning and project management processes. We also don't recommend tasking a single person with 12 one-week projects, as doing so would harvest low-hanging fruit, but at an unfair cost to that person's career growth.

Implementation Details

As we refine the details of our approach to interrupt reduction projects, we've found that the following tactics work well for us.

Project Ideas

Project ideas for the interrupt reduction project on duty come from two main sources:

- Current/past ticket handlers who file annoyances into a bug hotlist as they resolve tickets
- Technical Leads (TLs) who have a high-level view of the service

Project Assignment and Handoff

There are generally many more project ideas than engineering time to implement them, so the TL or someone with an overall view of the service should sort the project list by impact. In the interest of preserving autonomy among team members, we don't suggest assigning projects. Instead, let people choose from the

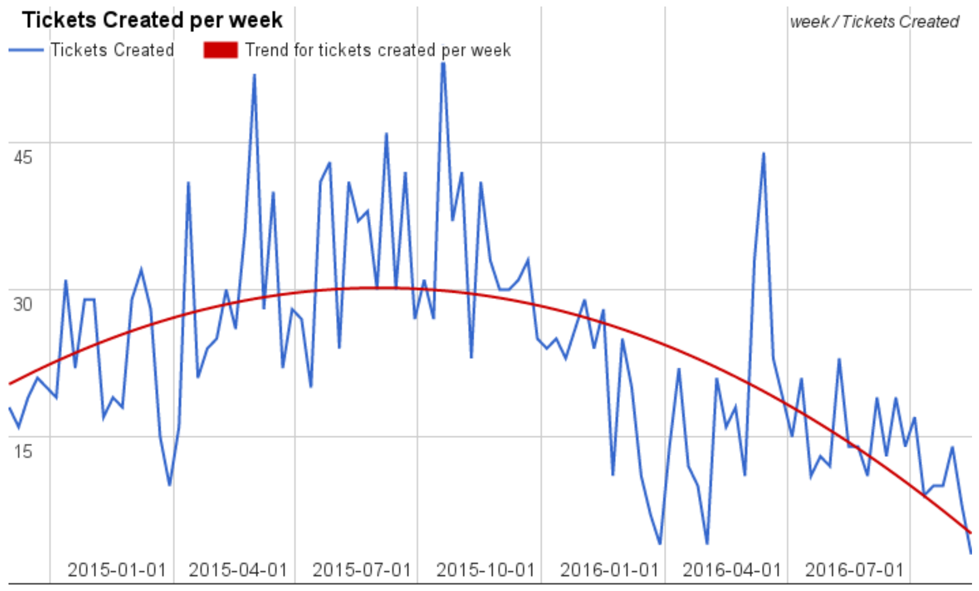


Figure 4: Tickets created per week

top 10 projects in the list. If there is any bleed over from last week's project, the interrupt reduction project on duty should finish that project first.

Handling Excess Ticket Load
If your team has more tickets than one person can handle, you have two options for dealing with the excess load:

- Task the interrupt reduction project on duty with tickets for one day per week.
- Decide to relax ticket response expectations for a period of time, until the work pays off.

Overall Effects

Measurable Effects
We've found that most of the time, one dedicated ticket handler can resolve all tickets, which frees up one person's time for interrupt reduction projects. This is one of the most significant results of our process change, as it proves that some people weren't properly focused on tickets during their ticket duty shifts.

We've reduced overall ticket volume, as shown in the case study that follows (see Figure 4). As a result, we're able to resolve the smaller number of incoming tickets more quickly, although velocity gains are somewhat countered by the increasing difficulty of the tickets we receive—by providing our customers with a better service, we've increased demand for help with complex problems like improving performance.

We complete approximately three of these small strategic interrupt reduction projects every four weeks.

Nooglers (new Googlers) training on the service spend less time on boring or repetitive tickets and more time on interesting tickets that actually create learning opportunities.

Less Measurable Effects
People complain less when they're doing tickets. It feels like we're spending more time on difficult and rewarding tickets and less time on simple or repetitive tickets.

Customers are happier about our ticket handling. Faster ticket resolution helps build goodwill with our customers.

The tension between sites about the effort put into tickets has disappeared, which has led to better overall cross-site relationships.

Example Case Study: Ticket Funnel
In mid-2015, the Bigtable SRE Team was dealing with a high load of customer tickets. The number of tickets opened per week had increased by roughly 50% over the previous year, from 20+ to 30+ (see Figure 4), and we frequently complained that many tickets could easily be handled by redirecting customers to automation or documentation. Building a simple ticket funnel system to guide customers to appropriate automation or documentation was a natural choice for our first interrupt reduction project.

Instead of directly creating a ticket, customers now work through a simple Web interface where they traverse a decision tree. Non-leaf nodes in the tree are represented as a list of questions linking to child nodes, and leaf nodes do one of the following:

- Link to the relevant self-service automation or documentation.
- Provide a form that generates a customer ticket.

By immediately pointing customers to relevant automation or documentation, we both reduced the number of incoming tickets and improved the quality of service for customers, who no longer have to wait for a human to answer easily resolvable questions. By asking for specific information based on the type of problem, we eliminated an unnecessary round trip of requesting information that could have been supplied up front. Now that this infrastructure is in place, it's easy to update the decision tree with more questions and leaf nodes as we identify more common requests and issues.

Although we'd been talking about undertaking this project for two years, it ended up only taking about two weeks of work. As such, the ticket funnel is a successful example of work that's enabled by the interrupt reduction projects approach: the project was too large to complete on the spur of the moment, but not large enough or important enough to be a standalone project tracked on a longer timescale.

Once our solution was in place, it took a bit of time to reeducate customers, who fully embraced the ticket funnel once they discovered its utility. While measuring tickets that never got filed is difficult, we do have some data that speaks to the ticket funnel's success:

- Figure 4 shows that the ticket creation rate dropped by roughly half after we implemented the ticket funnel, from 30+ to 15+ per week. While we'll never completely eliminate tickets, the overall trend has most definitely reversed.
- Quarterly customer satisfaction surveys reveal an overall happier customer base.
- Anecdotally, we see far fewer tickets that can be resolved by pointing customers at automation or documentation.

Applying This Strategy in Your Organization

Figuring Out a Strategy

In order to determine how to best apply a similar strategy at your organization, consider the following series of questions.

How much time does your team have to work on interrupt reduction projects?

- How much time do you allocate to tickets each week? e.g., *1 person? 4 people?*
- How much time do you actually spend working on tickets each week? e.g., *1 person? 2.5 people?*

Subtract bullet two from bullet one. If you're left with significantly less than 100% of one person's time, you probably won't be able to make meaningful progress on interrupt reduction projects using the slack from ticket duty. If this is the case, you have two possible solutions:

- Spend less time on large projects.
- Spend less time on tickets.

If you choose the second option, you need to think about implementation:

- Will you stop working on tickets entirely? Or will you postpone working on a class of tickets until the automation to deal with them is in place? How will this course of action affect your customers?
- Do you have an expected service level for ticket response time or resolution speed?

The nature of your customers (e.g., internal vs. external) greatly affects not only the answers to these questions, but the expected timeline and impact of the projects. The level of disruption that customers will accept is proportional to the benefit they can expect, so make sure to clearly communicate the motivations for your actions, expected disruptions, timeline, and expected benefits. Market realities will greatly constrain your tactics when it comes to externally visible products, so work with product management and marketing and sales to determine how to accomplish your goals without severely impacting business.

Who will work on the interrupt reduction projects?

Make interrupt reduction projects part of the normal ticket duty rotation, which we assume is fairly scheduled and distributed. If that's not the case, think carefully about who will work on these projects. It's important that the work be seen as valuable by the team. Choose people who are enthusiastic or particularly productive in order to create a good initial impression.

How will you convince your team to adopt this approach?

Here are some selling points you may be able to use or adapt:

- Each team member will spend 50% less time on tickets.
- Completing a small interrupt reduction project quickly and seeing immediate impact creates a good deal of satisfaction.
- Interrupt reduction projects will improve the systems your team uses on a daily basis.
- Eventually, your ticket load will decrease. The remaining tickets will be issues that actually merit investigation, and improvements to tooling will make some tickets easier to deal with than they were previously.

How will you safeguard the time allocated for interrupt reduction projects?

It's tempting for people to ignore interrupt reduction projects in favor of large projects, especially if those large projects have external commitments or interest. Therefore, creating accountability around interrupt reduction projects is important. You might accomplish this by publishing objectives around these projects, reporting on them regularly, tracking them, or announcing interrupt reduction project velocity in regular reports.

Suggested Interrupt Reduction Projects

The following generic suggestions for interrupt reduction projects should provide a substantial return on the time invested in them.

Identify the Sources of Your Toil

It may seem obvious, but before you can effectively reduce toil, you need to understand the sources of your toil. Consider adding metadata (e.g., cause, impact, time to fix) to tickets to help determine recurring issues and your biggest time sinks.

Improve Your Documentation

Many engineers are allergic to writing documentation, but documentation is a very low cost way to address customer needs and improve ticket handling. It's much easier to handle a ticket if the process is documented, and documentation is a good first step towards automating a process. If customers can find and use good documentation, they won't need to open a ticket.

Often, the blank page effect is the biggest impediment to writing documentation: someone doesn't know where to start, so they don't start. Provide a standard template for each type of documentation (customer facing, internal procedures, <your type here>) to make getting started easier.

Pick the 10 Most Annoying Small Bugs and Fix Them

Your team should be creating lists of bugs for the rough edges, shortcomings, and difficulties encountered in the course of everyday work—otherwise those problems will never be fixed. Pick the ten most annoying small bugs and fix them. Preferably, choose commonly encountered bugs, as people will notice when they're fixed. Consider choosing bugs related to one or two systems, rather than scattered small improvements, so that progress is significant and noticeable. Seeing improvements encourages team members to file bugs, providing a ready source of interrupt reduction projects.

Takeaways

If tickets/interrupts are an inevitable part of your team's workload, be thoughtful in formulating a strategy to handle them. If you don't implement some type of strategy to proactively reduce tickets, their volume is likely to spiral out of control and become unsustainable in the medium and long term. It's also important to ensure that handling tickets doesn't constantly disrupt the cognitive flow state of your engineers.

Our recommendations for approaching tickets/interrupts, which have been implemented by multiple storage-related services at Google, include four concrete components:

- Centralize your ticket load, either onto engineers who are already expecting interruptions (e.g., primary or secondary on-call) or to a dedicated ticket duty rotation.
- Track ideas for small interrupt reduction projects that will reduce toil.
- Put a framework in place that reserves time for small (20–30 hours) proactive projects.
- Treat tickets and small proactive interrupt reduction projects as separate rotations, distributed among team members and sites on a regular basis.

Acknowledgments

Many thanks to Vladimir Rusinov, who invested heroic effort in analyzing and fixing the root causes of customer tickets, and to Todd Underwood and Dave O'Connor for their input in writing this article.

References

[1] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems* (O'Reilly Media, 2016).

[2] B. Beyer, B. Gleason, D. O'Connor, and V. Rau, "Invent More, Toil Less," *;login:*, vol. 41, no. 3 (Fall 2016): https://www.usenix.org/publications/login/fall2016/beyer.

[3] Flow: https://en.wikipedia.org/wiki/Flow_(psychology).

SRE
CON®_

SREcon is a gathering of engineers who care deeply about site reliability, systems engineering, and working with complex distributed systems at scale. It strives to challenge both those new to the profession as well as those who have been involved in it for decades. The conference has a culture of critical thought, deep technical insights, continuous improvement, and innovation.

SRECON17 AMERICAS

MARCH 13–14, 2017 • SAN FRANCISCO, CA, USA

www.usenix.org/srecon17americas

The full program and registration will be available in January 2017.

SRECON17 ASIA/AUSTRALIA

MAY 22–24, 2017 • SINGAPORE

www.usenix.org/srecon17asia

SRECON17 EUROPE/MIDDLE EAST/AFRICA

AUGUST 30–SEPTEMBER 2, 2017 • DUBLIN, IRELAND

www.usenix.org/srecon17europe

usenix®
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

In Praise of Metaclasses!

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009). He is also known as the creator of Swig (http://www.swig.org) and Python Lex-Yacc (http://www.dabeaz.com/ply.html). Beazley is based in Chicago, where he also teaches a variety of Python courses. dave@dabeaz.com

Much maligned and misunderstood, metaclasses might be one of Python's most useful features. On the surface, it might not be clear why this would be the case. Just the name "metaclass" alone is enough to conjure up an image of swooping manta rays and stinging bats attacking your coworkers in a code review. I'm sure that there are also some downsides, but metaclasses really are a pretty useful thing to know about for all sorts of problems of practical interest to systems programmers. This includes simplifying the specification of network protocols, parsers, and more. In this installment, we'll explore the practical side of metaclasses and making Python do some things you never thought possible. Note: This article assumes the use of Python 3.

When I was first learning Python 20 years ago, I remember taking a trip to attend the Python conference. At that time, it was a small affair with just 50 or 60 enthusiastic programmers. I also remember one presentation in particular—the one that proposed the so-called "metaclass hack" for Python. There were a lot of frightened stares during that presentation and to be honest, it didn't make a whole lot of sense to me at the time. Some short time later, metaclasses became known as Python's "killer joke" in reference to a particular Monty Python sketch. Nobody was able to understand them without dying apparently.

Flash forward to the present and I find myself at home writing some Python code to interact with the game Minecraft. I'm buried in a sea of annoying low-level network protocol details. The solution? Metaclasses. In an unrelated project, I find myself modernizing some parsing tools I wrote about 15 years ago. Once again, I'm faced with a problem of managing lots of fiddly details. The solution? Metaclasses again. Needless to say, I'm thinking that metaclasses are actually kind of cool—maybe even awesome.

That said, metaclasses have never really been able to shake their "killer joke" quality in the Python community. They involve defining objects with the "class" statement, and inheritance is involved. Combine that with the word "meta" and surely it's just going to be some kind of icky object-oriented monstrosity birthed from the bowels of a Java framework or something. This is really too bad and misses the point.

In this article, I'm going to take a stab at rectifying that situation. We'll take a brief look at what happens when you define a class in Python, show what a metaclass is, and describe how you can use this newfound knowledge to practical advantage with an example.

Defining Classes

Most Python programmers are familiar with the idea of defining and using a class. One use of classes is to help you organize code by bundling data and functions together. For example, instead of having separate data structures and functions like this:

```
p = { 'x': 2, 'y': 3 }
def move(p, dx, dy):
    p['x'] += dx
    p['y'] += dy
```

a class lets you glue them together in a more coherent way:

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

Another use of classes is as a code-reuse tool. This is common in libraries and frameworks. For example, a library will provide a base set of code for you to use and then you extend it with your own functionality via inheritance. For example, here is some code using the `socketserver` module in the standard library:

```
from socketserver import TCPServer, BaseRequestHandler

class EchoHandler(BaseRequestHandler):
    def handle(self):
        while True:
            data = self.request.recv(1024)
            if not data:
                break
            self.request.sendall(data)

serv = TCPServer(('', 25000), EchoHandler)
serv.serve_forever()
```

There is a third use of classes, however, that is a bit more interesting. Step back for a moment and think about what's happening when you define a class. Essentially, a class serves as an enclosing environment for the statements that appear inside. Within this environment, you actually have a lot of control over how Python behaves—you can bend the rules and make Python do things that are not normally possible. For example, altering definitions, validating code, or building little domain-specific languages. A good example can be found in defining an enum in the standard library [1]. Here is an example:

```
from enum import Enum

class State(Enum):
    OPEN = 1
    CLOSED = 2
```

If you start using this class and start thinking about it, you'll find that it has some unusual behavior. For example, the class variables `OPEN` and `CLOSED` that were defined as integers no longer possess those types:

```
>>> State.OPEN
<State.OPEN: 1>
>>> type(State.OPEN)
<enum 'State'>
>>> isinstance(State.OPEN, int)
False
>>>
```

Something has implicitly altered the class body in some way. You'll also find that `Enum` classes don't allow duplicate definitions. For example, this produces an error:

```
class State(Enum):
    OPEN = 1
    CLOSED = 2
    OPEN = 3

Traceback (most recent call last):
...
TypeError: Attempted to reuse key: 'OPEN'
```

If you give different names to the same value, you get an alias.

```
class State(Enum):
    OPEN = 1
    CLOSED = 2
    SHUTDOWN = 2

>>> State.CLOSED
<State.CLOSED: 2>
>>> State.SHUTDOWN
<State.CLOSED: 2>
>>> State.CLOSED is State.SHUTDOWN
True
>>>
```

If you try to inherit from an enumeration, you'll find that it's not allowed:

```
class NewState(State):
    PENDING = 3
Traceback (most recent call last):
...
TypeError: Cannot extend enumerations
```

Finally, attempting to create instances of an Enum results in a kind of type-cast rather than the creation of a new object. For example:

```
>>> s = State(2)
>>> s
<State.CLOSED>
>>> s is State.CLOSED
True
>>>
```

So *something* is not only changing the body of the class, it's monitoring the definition process itself. It's bending the normal rules of assignment. It's looking for errors and enforcing rules. Even the rules of instance creation and memory allocation have apparently changed.

These unusual features of `Enum` are an example of a metaclass in action—metaclasses are about changing the very meaning of a class definition itself. A metaclass can make a class do interesting things all while hiding in the background.

Metaclasses

Now that we've seen an example of a metaclass in action, how do you plug into this machinery yourself? The key insight is that a class definition is itself an instance of an object called `type`. For example:

```
class Spam(object):
    def yow(self):
        print('Yow!')

>>> type(Spam)
<class 'type'>
>>>
```

The *type* of a class is its metaclass. So `type` is the metaclass of `Spam`. This means that `type` is responsible for everything associated with the definition of the `Spam` class.

Now suppose you wanted to alter what happens in class creation? Here are the neat, head-exploding tricks that you can use to hook into it. This is going to look rather frightening at first, but it will make much more sense once you try it afterwards. Official documentation on the process can be found at [2].

```
class mytype(type):

    @classmethod
    def __prepare__(meta, clsname, bases):
        print('Preparing class dictionary:', clsname, bases)
        return super().__prepare__(clsname, bases)

    @staticmethod
    def __new__(meta, clsname, bases, attrs):
        print('Creating class:', clsname)
        print('Bases:', bases)
        print('Attributes:', list(attrs))
        return super().__new__(meta, clsname, bases, attrs)

    def __init__(cls, clsname, bases, attrs):
        print('Initializing class:', cls)
        super().__init__(clsname, bases, attrs)

    def __call__(cls, *args, **kwargs):
        print('Creating an instance of', cls)
       return super().__call__(*args, **kwargs)
```

In this code, we've subclassed `type` and installed hooks onto a few important methods that will be described shortly. To use this new type as a metaclass, you need to define a new top-level object like this:

```
# Top-level class
class myobject(metaclass=mytype):
    pass
```

After you've done that, using this new metaclass requires you to inherit from `myobject` like this:

```
class Spam(myobject):
    print('—Starting:', locals())
    def yow(self):
        print('Yow!')
    print('—Ending:', locals())
```

When you do this, you're going to see output from the various methods:

```
Preparing class dictionary: Spam (<class '__main__.myobject'>,)
—Starting: {'__qualname__': 'Spam', '__module__': '__main__'}
—Ending: {'__qualname__': 'Spam', '__module__': '__main__',
'yow': <function Spam.yow at 0x10e6cc9d8>}
Creating class: Spam
Bases: (<class '__main__.myobject'>,)
Attributes: ['__qualname__', '__module__', 'yow']
Initializing class: <class '__main__.Spam'>
```

Keep in mind, you have not created an instance of `Spam`. All of this is triggered automatically merely by the *definition* of the `Spam` class. An end user will see that the class `Spam` is using inheritance, but the use of a metaclass is not apparent in the specification. Let's talk about the specifics.

Before anything happens at all, you will see the `__prepare__()` method fire. The purpose of this method is to create and prepare the dictionary that's going to hold class members. This is the same dictionary that `locals()` returns in the class body. But how does Python know to use the `__prepare__()` method of our custom type? This is determined by looking at the type of the parent of `Spam`. In this case `myobject` is the parent, so this is what happens:

```
>>> ty = type(myobject)
>>> ty
<class 'meta.mytype'>
>>> d = ty.__prepare__('Spam', (myobject,))
Preparing class dictionary: Spam (<class '__main__.myobject'>,)
>>> d
{}
>>>
```

Once the class dictionary has been created, it's populated with a few bits of name information, including the class name and enclosing module.

```
>>> d['__qualname__'] = 'Spam'
>>> d['__module__'] == __name__
>>>
```

Afterwards, the body of the `Spam` class executes in this dictionary. You will see new definitions being added. Upon conclusion, the dictionary is fully populated with definitions. The print statements in the top and bottom of the class are meant to show the state of the dictionary and how it changes.

After the class body has executed, the `__new__()` method of the metaclass is triggered. This method receives information about the class, including the name, bases, and populated class dictionary. If you wanted to write code that did anything with this data prior to creating the class, this is the place to do it.

After `__new__()` is complete, the `__init__()` method fires. This method is given the newly created class as an argument. Again, this is an opportunity to change parts of the class. The main difference between `__new__()` and `__init__()` is that `__new__()` executes prior to class creation, `__init__()` executes after class creation.

The `__call__()` method of a metaclass concerns instance creation. For example:

```
>>> s = Spam()
Creating an instance of <class '__main__.Spam'>
>>> s.yow()
Yow!
>>>
```

"Yow" is right! You have just entered a whole new realm of magic. The key idea is that you can put your fingers on the knobs of class definition and instance creation—and you can start turning the knobs. Let's do it.

Example: Building a Text Tokenizer

Let's say you were building a text parser or compiler. One of the first steps of parsing is to tokenize input. For example, suppose you had an input string like this:

```
text = 'a = 3 + 4 * 5'
```

And you wanted to tokenize it in a sequence of tuples like this:

```
[ ('NAME', 'a'), ('ASSIGN', '='), ('NUM', 3),
  ('PLUS', '+'), ('NUM', 4), ('TIMES', '*'), ('NUM', 5) ]
```

One way to do this is write low-level code using regular expressions and the `re` module. For example:

```
# tok.py

import re

# Patterns for the different tokens
NAME = r'(?P<NAME>[a-zA-Z_][a-zA-Z0-9_]*)'
NUM = r'(?P<NUM>\d+)'
ASSIGN = r'(?P<ASSIGN>=)'
PLUS = r'(?P<PLUS>\+)'
TIMES = r'(?P<TIMES>\*)'
ignore = r'(?P<ignore>\s+)'

# Master re pattern
pat = re.compile('|'.join([NAME, NUM, ASSIGN, PLUS, TIMES,
ignore]))

# Tokenization function
def tokenize(text):
    index = 0
    while index < len(text):
        m = pat.match(text, index)
        if m:
            tokname = m.lastgroup
            toktext = m.group()
            if tokname != 'ignore':
                yield (tokname, toktext)
            index = m.end()
        else:
            raise SyntaxError('Bad character %r' % text[index])

if __name__ == '__main__':
    text = 'a = 3 + 4 * 5'
    for tok in tokenize(text):
        print(tok)
```

Although there's not a lot of code, it's kind of low-level and nasty looking. For example, having to use named regex groups, forming the master pattern, and so forth. Let's look at a completely different formulation using metaclasses. Define the following metaclass:

```
from collections import OrderedDict
import re

class tokenizemeta(type):
    @classmethod
    def __prepare__(meta, name, bases):
        return OrderedDict()

    @staticmethod
    def __new__(meta, clsname, bases, attrs):
        # Make named regex groups for all strings in the class body
        patterns = [ '(?P<%s>%s)' % (key, val) for key, val in attrs
.items()
                     if isinstance(val, str) ]

        # Make the master regex pattern
        attrs['_pattern'] = re.compile('|'.join(patterns))
        return super().__new__(meta, clsname, bases, attrs)
```

This metaclass inspects the class body for strings, makes named regex groups out of them, and forms a master regular expression. The use of an `OrderedDict` is to capture definition order—something that matters for proper regular expression matching.

Now, define a base class with the general `tokenize()` method:

```
class Tokenizer(metaclass=tokenizemeta):
    def tokenize(self, text):
        index = 0
        while index < len(text):
            m = self._pattern.match(text, index)
            if m:
                tokname = m.lastgroup
                toktext = m.group()
                if not tokname.startswith('ignore'):
                    yield (tokname, toktext)
                index = m.end()
            else:
                raise SyntaxError('Bad character %r' % text[index])
```

Now why did we go through all of this trouble? It makes the specification of a tokenizer easy. Try this:

```
class Simple(Tokenizer):
    NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
    NUM = r'\d+'
    ASSIGN = r'='
    PLUS = r'\+'
    TIMES = r'\*'
    ignore = r'\s+'

# Use the tokenizer
text = 'a = 3 + 4 * 5'
tokenizer = Simple()
for tok in tokenizer.tokenize(text):
    print(tok)
```

That's pretty cool. Using metaclasses, you were able to make a little specification language for tokenizing. The user of the `Tokenizer` class just gives the token names and regular expressions. The metaclass machinery behind the scenes takes care of the rest.

Adding Class Dictionary Magic

You can do even more with your tokenizer class if you're willing to stretch the definition of a dictionary. Let's subclass `OrderedDict` and change assignment slightly so that it detects duplicates:

```
class TokDict(OrderedDict):
    def __setitem__(self, key, value):
        if key in self and isinstance(key, str):
            raise KeyError('Token %s already defined' % key)
        else:
            super().__setitem__(key, value)

class tokenizemeta(type):
    @classmethod
    def __prepare__(meta, name, bases):
        return TokDict()

    ...
```

In this new version, a specification with a duplicate pattern name creates an error:

```
class Simple(Tokenizer):
    NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
    NUM = r'\d+'
    ASSIGN = r'='
    PLUS = r'\+'
    TIMES = r'\*'
    NUM = r'\d+'
    ignore = r'\s+'

Traceback (most recent call last):
  ...
KeyError: 'Token NUM already defined'
```

You could stretch it a bit further, though. This version allows optional action methods to be defined for any of the tokens:

```
from collections import OrderedDict
import re

class TokDict(OrderedDict):
    def __init__(self):
        super().__init__()
        self.actions = {}

    def __setitem__(self, key, value):
        if key in self and isinstance(key, str):
            if callable(value):
                self.actions[key] = value
            else:
                raise KeyError('Token %s already defined' % key)
        else:
            super().__setitem__(key, value)

class tokenizemeta(type):
    @classmethod
    def __prepare__(meta, name, bases):
        return TokDict()

    @staticmethod
    def __new__(meta, clsname, bases, attrs):
        # Make named regex groups for all strings in the class body
        patterns = [ '(?P<%s>%s)' % (key, val) for key, val in
attrs.items() if isinstance(val, str) ]
        # Make the master regex pattern
        attrs['_pattern'] = re.compile('|'.join(patterns))

        # Record action functions (if any)
        attrs['_actions'] = attrs.actions

        return super().__new__(meta, clsname, bases, attrs)

class Tokenizer(metaclass=tokenizemeta):
    def tokenize(self, text):
        index = 0
        while index < len(text):
            m = self._pattern.match(text, index)
            if m:
                tokname = m.lastgroup
                toktext = m.group()
                if not tokname.startswith('ignore'):
                    if tokname in self._actions:
                        yield (tokname, self._actions[tokname](self,
toktext))
                    else:
                        yield (tokname, toktext)
                index = m.end()
            else:
                raise SyntaxError('Bad character %r' % text[index])
```

This last one might require a bit of study, but it allows you to write a tokenizer like this:

```
class Simple(Tokenizer):
    NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
    NUM = r'\d+'
    ASSIGN = r'='
    PLUS = r'\+'
    TIMES = r'\*'
    ignore = r'\s+'
    # Convert NUM tokens to ints
    def NUM(self, text):
        return int(text)

    # Uppercase all names (case-insensitivity)
    def NAME(self, text):
        return text.upper()

# Example
text = 'a = 3 + 4 * 5'
tokenizer = Simple()
for tok in tokenizer.tokenize(text):
    print(tok)
```

If it's working, the final output should appear like this:

```
('NAME', 'A')
('ASSIGN', '=')
('NUM', 3)
('PLUS', '+')
('NUM', 4)
('TIMES', '*')
('NUM', 5)
```

Notice how the names have been uppercased and numbers converted to integers.

The Big Picture

By now, you're either staring at amazement or in horror at what we've done. In the big picture, one of the great powers of metaclasses is that you can use them to turn class definitions into a kind of small domain-specific language (DSL). By doing this, you can often simplify the specification of complex problems. Tokenization is just one such example. However, it's motivated by a long history of DSLs being used for various facets of software development (e.g., lex, yacc, RPC, interface definition languages, database models, etc.).

If you've used more advanced libraries or frameworks, chances are you've encountered metaclasses without even knowing it. For example, if you've ever used the Django Web framework, you describe database models using classes like this [3]:

```
from django.db import models

class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models
.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

This involves metaclasses. It might not be obvious, but there is a whole set of code sitting behind the `models.Model` base class that is watching definitions and using that information to carry out various magic behind the scenes. A benefit of using a metaclass is that it can make it much easier for an end user to write specifications. They can write simple definitions and not worry so much about what's happening behind the scenes.

A Contrarian View

A common complaint lodged against metaclasses is that they introduce too much implicit magic into your program—violating the "Explicit is better than implicit" rule from the Zen of Python. To be sure, you don't actually need to use a metaclass to solve the problem presented here. For example, we possibly could have written a `Tokenizer` with more explicit data structures using a class definition like this:

```
class Simple(Tokenizer):
    tokens = [
        ('NAME', r'[a-zA-Z_][a-zA-Z0-9_]*'),
        ('NUM', r'\d+'),
        ('ASSIGN', r'='),
        ('PLUS', r'\+'),
        ('TIMES', '\*'),
        ('ignore', r'\s+')
    ]
    actions = {
        'NAME': lambda text: text.upper(),
        'NUM': lambda text: int(text)
    }
```

It's not much more code than the metaclass version, but it frankly forces me to squint my eyes more than usual. Of course, they also say that beauty is in the eye of the beholder. So your mileage might vary.

Final Words

In parting, be on the lookout for metaclass magic the next time you use an interesting library or framework—they're often out there hiding in plain sight. If you're writing your own code and faced with problems involving complex or domain-specific specifications, metaclasses can be a useful tool for simplifying it.

References

[1] enum module: https://docs.python.org/3/library/enum .html.

[2] Customizing class creation (official documentation): https://docs.python.org/3/reference/datamodel.html #customizing-class-creation.

[3] Django models: https://docs.djangoproject.com/en/1.10 /topics/db/models/.

Practical Perl Tools
The Whether Man

DAVID N. BLANK-EDELMAN

David Blank-Edelman is the Technical Evangelist at David Blank-Edelman is the Technical Evangelist at Apcera (the comments/views here are David's alone and do not represent Apcera /Ericsson). He has spent close to 30 years in the system administration/DevOps/SRE field in large multiplatform environments including Brandeis University, Cambridge Technology Group, MIT Media Laboratory, and Northeastern University. He is the author of the O'Reilly Otter book *Automating System Administration with Perl* and is a frequent invited speaker/organizer for conferences in the field. David is honored to serve on the USENIX Board of Directors. He prefers to pronounce Evangelist with a hard 'g'.
dnblankedelman@gmail.com

There's a character in *The Phantom Tollbooth* by Norton Juster (one of my favorite books) called the Whether Man. He introduces himself like this:

"I'm the Whether Man, not the Weather Man, for after all it's more important to know whether there will be weather than what the weather will be."

In this column we're going to see if we can indeed bring weather to our programs. To do so, we're going to make use of the API provided by The Dark Sky Company, the people behind the popular app of the same name (and also the people who provided the API service forecast.io, renamed as of this week to the Dark Sky API). The API service they provide is a commercial one, so if you plan to build something that will hit their API more than a thousand times a day, you will need to pay. But for the experiments in this column, we should be pretty safe from the $0.0001 per forecast fee.

Two Flavors of API

The Dark Sky service offers two kinds of API, Forecast and the Time Machine (really). The first kind provides just a single forecast for next week's weather, the second will give us the opportunity to query for the conditions at an arbitrary time in the past or the future.

For both of these APIs, the actual method of querying and working with the data will be the same. Before we do anything, we'll need to sign up for a secret key. This key has to be sent along with every request we make. It's "secret" in that it shouldn't be distributed with your code. It's the thing that ties usage of the API back to your account (so you can pay for the service as needed). I'll be X'ing it out in all of the same code in this column.

Once we have a key, we can make an HTTPS request to the Dark Sky API endpoint. It is going to respond with a blob of (well-formed) JSON that we'll have to parse. Nothing special compared to our previous columns. Though there's an existing module for the previous name of the API (Forecast::IO), the tasks are so easy we'll just use more generic modules for what we need. Let's do some sample querying with both API types.

Whoops, one small complication that is worth mentioning: when you query either API, you will need to be specific about where you want to know about the weather. Whether there is weather is key but so is where is that weather. The Dark Sky API needs you to specify the latitude and longitude of the place before it can tell you the weather information for that place. The API will not geocode for you (i.e., translate an address to a latitude/longitude pair). There are a number of Web sites that will do small amounts of geocoding for you, but if you require more than a handful of translations, you will want to use a separate service for this part. We've talked about geocoding using Perl in this column before so some past columns in the archives are likely to be helpful. For this column I'm just going to use the coordinates of Boston (42.3600825, -71.0588801).

Give Me a Forecast

According to the Dark Sky docs, the format of the Forecast API is just:

```
https://api.darksky.net/forecast/[key]/[latitude],[longitude]
```

Let's query that URL and dump the data we get back. For fun I'm using the client that ships with Mojolicious and Data::TreeDumper which has purdy data structure dumps:

```
# note IO::Socket::SSL has to be installed for
# Mojo::UserAgent TLS support. It will fail silently
# if you don't
use Mojo::UserAgent;
use Data::TreeDumper;

my $API_KEY  = 'XXXXXXXXXXXXXXXXXXXX';
my $location = '42.3600825,-71.0588801';
my $endpoint = 'https://api.darksky.net/forecast';

my $ua = Mojo::UserAgent->new;

my $forecast =
  $ua->get( $endpoint . '/' . $API_KEY . '/' . $location )->
         res->json;

print "Powered by Dark Sky (https://darksky.net/poweredby/)\n";

print DumpTree( $forecast, 'forecast', DISPLAY_ADDRESS => 0 );
```

The Mojo::UserAgent call performs the GET (as long as we have IO::Socket::SSL installed; see the comment). We then take the result (->res) and parse the JSON we get back (->json). The end result is we get back a Perl data structure we can then access or manipulate to our heart's content.

Here's a dump of that data structure. This is an excerpt of the dump where I have removed all but the first instance of each kind of info (for example, just the first hour, not all of them). We get back lots of data (though if we didn't want all of this, there is an optional "exclude" flag we could pass, but where's the fun in that?):

```
Powered by Dark Sky (https://darksky.net/poweredby/)
forecast
|- currently
|  |- apparentTemperature = 72.96
|  |- cloudCover = 0.18
|  |- dewPoint = 56.87
|  |- humidity = 0.57
|  |- icon = clear-day
|  |- nearestStormBearing = 229
|  |- nearestStormDistance = 99
|  |- ozone = 261.5
|  |- precipIntensity = 0
|  |- precipProbability = 0
|  |- pressure = 1018.92
|  |- summary = Clear
|  |- temperature = 72.96
|  |- time = 1474582497
|  |- visibility = 10
|  |- windBearing = 116
|  `- windSpeed = 4.8
|- daily
|  |- data
|  |  |- 0
|  |  |  |- apparentTemperatureMax = 77.14
|  |  |  |- apparentTemperatureMaxTime = 1474560000
|  |  |  |- apparentTemperatureMin = 61.63
|  |  |  |- apparentTemperatureMinTime = 1474542000
|  |  |  |- cloudCover = 0.13
|  |  |  |- dewPoint = 56.15
|  |  |  |- humidity = 0.64
|  |  |  |- icon = partly-cloudy-night
|  |  |  |- moonPhase = 0.72
|  |  |  |- ozone = 262.42
|  |  |  |- precipIntensity = 0
|  |  |  |- precipIntensityMax = 0
|  |  |  |- precipProbability = 0
|  |  |  |- pressure = 1020.31
|  |  |  |- summary = Partly cloudy in the evening.
|  |  |  |- sunriseTime = 1474540400
|  |  |  |- sunsetTime = 1474584172
|  |  |  |- temperatureMax = 77.14
|  |  |  |- temperatureMaxTime = 1474560000
|  |  |  |- temperatureMin = 61.63
|  |  |  |- temperatureMinTime = 1474542000
|  |  |  |- time = 1474516800
|  |  |  |- visibility = 9.9
|  |  |  |- windBearing = 127
|  |  |  `- windSpeed = 1.27
|  |- icon = rain
|  `- summary = Light rain tomorrow and Saturday, with
temperatures
falling to 63°F on Monday.
|- flags
|  |- darksky-stations
|  |  `- 0 = KBOX
|  |- isd-stations
|  |  |- 0 = 725090-14739
|  |- lamp-stations
|  |  |- 0 = KASH
|  |- madis-stations
|  |  |- 0 = AV085
|  |- sources
|  |  |- 0 = darksky
|  `- units = us
|- hourly
```

```
|  |- data
|  |  |- 0
|  |  |  |- apparentTemperature = 73.52
|  |  |  |- cloudCover = 0.16
|  |  |  |- dewPoint = 56.82
|  |  |  |- humidity = 0.56
|  |  |  |- icon = clear-day
|  |  |  |- ozone = 261.61
|  |  |  |- precipIntensity = 0
|  |  |  |- precipProbability = 0
|  |  |  |- pressure = 1018.98
|  |  |  |- summary = Clear
|  |  |  |- temperature = 73.52
|  |  |  |- time = 1474581600
|  |  |  |- visibility = 10
|  |  |  |- windBearing = 109
|  |  |  `- windSpeed = 4.54
|  |- icon = rain
|  `- summary = Drizzle tomorrow evening.
|- latitude = 42.3600825
|- longitude = -71.0588801
|- minutely
|  |- data
|  |  |- 0
|  |  |  |- precipIntensity = 0
|  |  |  |- precipProbability = 0
|  |  |  `- time = 1474582440
|  |- icon = clear-day
|  `- summary = Clear for the hour.
|- offset = -4
`- timezone = America/New_York
```

The initial "Powered by" header is just my attempt to keep to the letter of the terms of service at https://darksky.net/dev/docs/terms#attribution.

There are a few important details that might not be obvious just looking at the dump above. First, the values we are getting back are in US-standard notation (e.g., Fahrenheit). This is the default, although we can send in a query parameter requesting the response be in metric units (or even automatically in the units of the place). There's a similar choice available for the human-readable summary fields above. Second, the time information we get back is in UNIX-standard time (i.e., seconds from the epoch). To translate that to local time, we could "scalar localtime $value."

The docs at https://darksky.net/dev/docs have the full details of the data we dumped. To access just parts of it, we could use simple code like this:

```
print "On " . scalar localtime(
                $forecast->{currently}->{time} ) . " it was:\n";
print $forecast->{currently}->{temperature} . " degrees F\n";
print( ( $forecast->{currently}->{humidity} * 100 )
          . "% relative humidity\n" );
```

Hop into My Time Machine

Let's move on to the second API, the "Time Machine" (cue the theremin). This API's request format is very similar to the last one (same endpoint and everything), it just adds the field you'd expect:

```
https://api.darksky.net/forecast/[key]/
[latitude],[longitude],[time]
```

The time field is either in the UNIX-standard time format we mentioned before or you can use this format:

```
[YYYY]-[MM]-[DD]T[HH]:[MM]:[SS][timezone]
```

(where time zone is optional).

Our code looks very similar:

```
use Mojo::UserAgent;
use Data::TreeDumper;

my $API_KEY  = 'XXXXXXXXX';
my $location = '42.3600825,-71.0588801';
my $endpoint = 'https://api.darksky.net/forecast';

my $exclude  = 'currently,minutely,hourly,alerts,flags';

# tomorrow
my $time = time + (24 * 60 * 60);

my $ua = Mojo::UserAgent->new;

my $forecast =
  $ua->get( $endpoint . '/'
                         . $API_KEY
                               . '/'
                               . $location
                               . ',' . $time
                               . '?exclude=' . $exclude )->res->json;

print "Powered by Dark Sky (https://darksky.net/poweredby/)\n";

print DumpTree( $forecast, 'forecast', DISPLAY_ADDRESS => 0 );
```

This code shows the daily info for tomorrow's forecast (because we've excluded the other portions we didn't want to see):

```
Powered by Dark Sky (https://darksky.net/poweredby/)
forecast
|- daily
|  `- data
|     `- 0
|        |- apparentTemperatureMax = 67.33
|       |- apparentTemperatureMaxTime = 1474747200
|       |- apparentTemperatureMin = 55.72
|       |- apparentTemperatureMinTime = 1474772400
|       |- cloudCover = 0.33
|       |- dewPoint = 46.6
|       |- humidity = 0.6
|       |- icon = partly-cloudy-night
|       |- moonPhase = 0.79
|       |- ozone = 302.59
|       |- precipIntensity = 0.0074
|       |- precipIntensityMax = 0.0965
|       |- precipIntensityMaxTime = 1474693200
|       |- precipProbability = 0.64
|       |- precipType = rain
|       |- pressure = 1017.92
|       |- summary = Mostly cloudy in the morning.
|       |- sunriseTime = 1474713327
|       |- sunsetTime = 1474756758
|       |- temperatureMax = 67.33
|       |- temperatureMaxTime = 1474747200
|       |- temperatureMin = 55.72
|       |- temperatureMinTime = 1474772400
|       |- time = 1474689600
|       |- visibility = 9.76
|       |- windBearing = 335
|       `- windSpeed = 8.82
|- latitude = 42.3600825
|- longitude = -71.0588801
|- offset = -4
`- timezone = America/New_York
```

To wrap this up, let's do something fun with the time machine API. There's a character in the Flintstones cartoon that always had a cloud following him around. If you ever wondered if the same thing happened to you, we could query the API to find out if the weather has been consistent for each of your birthdays. Here's some code that does this:

```
use Mojo::UserAgent;
use Text::Graph;

my $API_KEY  = 'XXXXXXXXXXXXX';
my $location = '42.3600825,-71.0588801';
my $endpoint = 'https://api.darksky.net/forecast';

my $exclude = 'currently,minutely,hourly,alerts,flags';

my $birth_year = 1970;

my $birth_date = "08-08";
my $current_year = 1900 + (localtime)[5];

my $collection;

my $ua = Mojo::UserAgent->new;

print "Powered by Dark Sky (https://darksky.net/poweredby/)\n";
for my $year ( $birth_year .. $current_year ) {
    my $forecast =
      $ua->get( $endpoint . '/'
          . $API_KEY . '/'
          . $location . ','
          . "$year-${birth_date}T12:01:01"
          . '?exclude='
          . $exclude )->res->json;

     print "$year:
            $forecast->{daily}->{data}->[0]->{summary}\n";

    $collection{ $forecast->{daily}->{data}->[0]->{icon} }++;

}

my $graph = Text::Graph->new('Bar');
print "\n",$graph->to_string( \%collection );
```

In case you are curious, that's not actually my birthday. And yes, this isn't quite accurate because I haven't been at those coordinates my entire life. But if you can suspend disbelief for a moment, check out the output:

```
Powered by Dark Sky (https://darksky.net/poweredby/)
1970: Partly cloudy throughout the day.
1971: Mostly cloudy throughout the day.
1972: Partly cloudy until evening.
1973: Mostly cloudy throughout the day.
1974: Mostly cloudy throughout the day.
1975: Rain until afternoon, starting again in the evening.
1976: Heavy rain until evening, starting again overnight.
1977: Mostly cloudy throughout the day.
1978: Mostly cloudy throughout the day.
1979: Breezy in the morning and mostly cloudy throughout the
day.
1980: Mostly cloudy starting in the afternoon.
1981: Rain overnight.
1982: Mostly cloudy throughout the day.
1983: Mostly cloudy in the morning.
1984: Mostly cloudy throughout the day.
1985: Rain until afternoon.
1986: Mostly cloudy throughout the day.
1987: Light rain starting in the afternoon, continuing until
evening.
1988: Partly cloudy throughout the day.
1989: Partly cloudy until evening.
1990: Heavy rain until afternoon.
1991: Mostly cloudy throughout the day.
1992: Mostly cloudy throughout the day.
1993: Partly cloudy throughout the day.
1994: Partly cloudy until evening.
1995: Partly cloudy overnight.
```

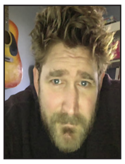
```
1996: Foggy in the morning.
1997: Rain overnight.
1998: Partly cloudy until evening.
1999: Mostly cloudy throughout the day.
2000: Mostly cloudy starting in the afternoon, continuing until
evening.
2001: Partly cloudy until afternoon.
2002: Partly cloudy starting in the afternoon.
2003: Rain in the morning and afternoon.
2004: Mostly cloudy starting in the afternoon.
2005: Partly cloudy in the morning.
2006: Partly cloudy throughout the day.
2007: Rain in the morning.
2008: Heavy rain in the evening.
2009: Partly cloudy starting in the afternoon, continuing until
evening.
2010: Partly cloudy throughout the day.
2011: Rain starting in the afternoon, continuing until evening.
2012: Partly cloudy throughout the day.
2013: Light rain in the morning.
2014: Partly cloudy starting in the afternoon, continuing until
evening.
2015: Partly cloudy in the afternoon.
2016: Partly cloudy until evening.

fog                 :
partly-cloudy-day   :*****************************
partly-cloudy-night :**
rain                :***********
wind                :
```

I threw in the last part for fun. It is a graph of the icons a weather program might show (another kind of summary, really) for the day. Looks like I've lived a pretty "partly cloudy" life. I'm curious to see what sort of fun things you can do with this API, so have at it.

Take care, and I'll see you next time.

iVoyeur
Dogfood and the Art of Self-Awareness

DAVE JOSEPHSEN



Dave Josephsen is the sometime book-authoring Developer Evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

Yesterday, I ate lunch in a bar in Northwest Montana. I munched on their fish-and-chips plate (which was way better than it had any business being) and tried to ignore the *Bones* re-run playing in 4k clarity while the old-timers drank and argued behind me.

The argument concerned a certain very old copper mine with a long and storied history of screwing over everyone but their board of directors. I won't get into the politics of it with you, or bore you with my opinions, but suffice to say I could tell who was in the right, and I think you could too if you were there.

It wasn't so much the logic of the arguments, nor the passion with which they were delivered by either side. It was *the tone* used by those in the wrong—a certain manner of speaking that belies a particular mode of thought; I'm sure you would recognize it. I did. It was that same tone we used in the Marine Corps in those tiny moments of uncertainty that always accompanied our preparations to do a bad thing in the name of some supposed greater good. Even before that, though, I recognized it from the days of my youth, when I knew I'd done a bad thing but I was trying to convince myself, or someone else (or both), that I had a good reason.

It was that guilty-as-sin *yeah...but* tone. We can all recognize it in others as long as we've recognized it in ourselves; and we all have.

"Know thyself" was one of the Delphic maxims, did you know that? Literally carved in stone into the temple of Apollo at Delphi. It's one of our oldest and best thoughts; one of those things we've been thinking since we've been capable of thinking about good and bad.

Sorry if I'm being a bit of a downer, but I actually find that a really comforting thought, that our self-awareness carries with it a certain, well, inescapable self-awareness. All we have to do is pay attention to ourselves.

Speaking of self-awareness, here's an interesting but not often answered question from my current day job:

What's monitoring the monitoring system?

I know, that's the kind of question asked by people who want to sell you something, but it's also one of those questions that triggers a certain degree of guilt within those of us who don't have a good answer for it. That's why the pre-sales engineers love asking it. They intuit our guilt because they've recognized it in themselves.

But how important a question is it really? I suppose it depends. There's something of a continuum of monitoring aptitude. The shops at the baseline competency level don't really distinguish between monitoring and alerting. Monitoring *is* the system that sends alerts, so for them, a monitoring outage is an alerting outage. Those stakes aren't very high honestly. They may not be alerted to a problem, but once they do find out, they'll SSH into that system and poke around manually. That kind of sucks but it's not the end of the world.

Moving up the continuum, however, you begin to encounter shops that use monitoring as a means of understanding system behavior. By that I mean, when an operations person wants to know if the app slowness they're experiencing is isolated to them or a widespread issue, they turn to the monitoring system to find the 95th percentile latency on HTTP requests. Then maybe they'll break out that data by node to find a misbehaving instance, and tell the chatbot to destroy that instance and replace it with a new one. In those sorts of shops, a monitoring outage directly affects our ability to reason about and fix problems with our systems.

Losing visibility at that level in the continuum sucks even more, but at Librato, we're in an even worse pickle. The monitoring system is not only our primary means of understanding the behavior of our systems; it *is* our systems. We are a SaaS monitoring shop, so a monitoring outage here equates to a catastrophic business interruption. I know, weird, right? So in this issue I thought it might be fun to explore the question of what monitors the monitoring system at Librato.

The tl;dr is, of course: Librato, but the story of *how* is pretty interesting and, I think, worth telling. In fact, Librato is the result of a pivot [1] from a product called *Silverline*. Silverline was designed to dynamically adjust the performance characteristics of a machine image in order to save money on hosting costs (nerf-your-CPU-so-you-spend-less-as-a-service). The engineers who built Silverline obviously needed a scalable means of measuring granular system performance, and so, like so many shops before them, they built a custom metrics solution. However, unlike so many shops before them, they did a *really* good job of it, and Librato was born.

When metrics became the operational focus of the company, the engineers were already quite accustomed to having unfettered access to an essentially free, high-quality metrics and monitoring tool. For them, building a thing and measuring its performance were the same undertaking, so they naturally relied on Librato to build and maintain everything. Put more succinctly: they used Librato to monitor the operational characteristics of Librato, thereby becoming their own biggest customer.

I cannot recommend this strategy for your monitoring endeavors, but it worked out pretty well for us in practice. In many ways it was even quite beneficial. It certainly brought us closer to our customers, since literally every employee at Librato could provide customer support because everyone was using the tool every day. It also gave us a far more solid baseline understanding of the technical limitations of the system than most startups have, since we were the ones who were stressing it the hardest.

It wasn't long, however, before a few very large engineering shops signed up, and UID1 (as we affectionately refer to ourselves) was no longer even close to the most voluminous metrics source. And as anyone who maintains an API will attest, along with more and larger customers comes a certain amount of API abuse. So it was with us. It really is remarkable that after 20 years in the field, you can still be surprised by end-user behavior. Humbling, but remarkable.

Unexpected patterns of end-user behavior are an inevitability for which no Chaos-Monkey can prepare you. We saw customers doing things that we'd never imagined, because honestly, they're kind of unimaginable. Can you imagine a scenario where you'd need to insert an epoch timestamp into the *source name* of a metric when you were going to plot in a line graph where *x* is time/day anyway? I mean why would you ever need to create a new, unique metric and source object for every single measurement you take?

That's just one of the many, *many* real-life things that we've seen real-life customers do in real life.

With time-series [2] datastores you make certain assumptions about the cardinality of measurement *sources*. We can handle high-cardinality measurements as long as we can make assumptions like those. I won't bore you with the details, but the accidental generation of high-cardinality *sources* can really wreak havoc on an optimized datastore like ours. Really any sort of behavior that causes us to create rows on the order of millions of unique IDs (or anything else that isn't proper time-series data) per minute or second is basically guaranteed to trigger pager duty to wake me up in the pre-dawn.

When problems like that happen, we need to be able to get to our own metrics to diagnose the issue, and we can't do that if the system is being effectively DOS'd by another end user.

Enter Dogfood

Our solution to this problem is an environment we named *Dogfood* (in reference to that somewhat gross Microsoft colloquialism [3], eating your own dog food).

Dogfood is pretty much a mini-Librato—a miniature re-creation of our production environment just for our use. It resides on AWS in US-West, on the opposite coast of the US from our Production and Staging environments. It is fed data by way of our production stream-processing tier, which is a custom-built stream processing system we've talked publicly about in the past [4].

Well-implemented stream processing is a lovely thing, and at Librato we rely very heavily on the combination of SuperChief (our own beloved Storm replacement) and Kafka [5], which we use as a cache between our various stream-processing entities. These components make it possible for us to quickly persist raw-resolution measurements as they arrive to our API while simultaneously processing them for alerting and copying them over to the Dogfood environment.

The pattern is simple. Worker threads from one service take measurements off the wire and write them to Kafka queues (*topics* in Kafka parlance). Workers from other services read them out of queue and process them in parallel. A single measurement, for example, that hits our API is immediately copied to several places by our ingestion pipeline:

- Directly to a fast-path service designed to persist it in Cassandra
- To a Kafka topic read by the Alerting service (for alert detection)
- To a Kafka topic for the service that processes 60-second roll-ups
- To a Kafka topic read by Dogfood workers
- Kafka topics for other stuff I can't talk about yet

The Dogfood path is triggered for every measurement that's submitted by user: UID1. We implemented Dogfood duplication as a stream-processing job like this so that any metrics we create in our day-to-day work will automatically be picked up and sent to Dogfood. Monitoring systems succeed when they're easy to use, so I feel pretty strongly that this is a critical component to Dogfood's success. It just wouldn't work if, as an engineer, you had to remember to create every metric twice: once in production and once in Dogfood.

But What About Ingestion Pipeline Problems?

The downside of using production streaming infrastructure to tee off metrics to Dogfood is the possibility that we will have a critical blocking outage in the production stream processing tier that will affect Dogfood metrics. Problems like this are actually relatively rare given both the simplicity of Dogfood processing (it's just a single write operation) and the parallel nature of stream processing with Kafka. In fact the most wonderful thing about our stream processing is how well it isolates workloads from each other. Given separate Kafka topics and dedicated services behind each, it's very rare in practice for us to experience an issue that crosses multiple topics, much less unrelated ones. Those sorts of issues are pretty much always going to be upstream of us at AWS (where Dogfood won't help us anyway).

Another downside is the possibility of a problem in the API ingestion pipeline upstream of the stream-processing tier. If the metrics can't make it into the stream-processing tier, then they won't make it to Dogfood either. This is a more likely failure mode, and one that we've experienced in the past. In practice, however, because of the nature of our architecture, the absence of Dogfood metrics is a pretty damning indicator of an ingestion problem, so when this happens we already know exactly where to look.

Most of us prefer to use live data from production day-to-day because it's the same system our customers use, but if we ever experience a service degradation, we can switch to Dogfood seamlessly to diagnose the problem and work toward a fix. Dogfood might be the most elaborate answer ever to the question of *What's monitoring the monitoring system,* but then who can put a price on self-awareness?

Take it easy.

References

[1] Wikipedia, "Pivot," last modified on Sept. 13, 2016: https://en.wikipedia.org/wiki/Lean_startup#Pivot.

[2] Dave Josephsen, "Sensical Summarization of Time-Series" (blog entry), August 11, 2014: http://blog.librato.com/posts/time-series-data.

[3] Wikipedia, "Eating your own dog food," last modified on Sept. 3, 2016: https://en.wikipedia.org/wiki/Eating_your_own_dog_food.

[4] SuperChief: http://www.heavybit.com/library/blog/streamlining-distributed-stream-processing-with-superchief/.

[5] J. Shekhar and A. Khurana, "Streaming Systems and Architectures," *;login:*, vol. 41, no. 1 (Spring 2016): https://www.usenix.org/system/files/login/articles/login_spring16_03_shekhar.pdf.

For Good Measure

Implications of the IoT

DAN GEER

Dan Geer is the CISO for In-Q-Tel and a security researcher with a quantitative bent. He has a long history with the USENIX Association, including officer positions, program committees, etc. dan@geer.org

Everybody is predicting great things, within varying interpretations of the word "great," for the Internet of Things. You are doubtless tired of hearing that. As good an answer as any to the question "When was the IoT born?" is when the number of connected devices exceeded the number of humans, while as good an answer as any to the question "What is a thing?" is any data-handling entity that cannot be found in contempt of court.

To remind ourselves of the basic numbers, Figure 1 is IoT size, Figure 2 is how many humans, and Figure 3 is thus the number of things per person.

The counts of human population are probably pretty close to correct. The counts for the IoT are surely arguable. The smooth curve in Figure 1 is simply Cisco's calculated exponential from their 1992 figure of one million devices to their estimate of 50.1 billion in 2020. In each of Figures 1, 2, and 3, the values for 2015 and for 2020 are highlighted. Last year might well have been the real birthdate of the IoT, in other words. Or maybe you don't want to compare all humans to the number of connected devices, only humans who are connected. Globally for 2015, 46% of humans were connected, and the year when there were more *connected* devices than *connected* humans was accordingly earlier, perhaps 2010. Regardless, the IoT is between infant and toddler.

There are two aspects to oncoming growth like this that are directly relevant to public policy, one germane to "For Good Measure" and one not. The "not" is the lifetime resource cost, including energy cost of manufacture, operation, and disposal, that an exponentially increasing number of powered devices necessarily represents. Regulators are looking at this with whatever passes for glee in such places—IT, broadly defined, already accounts for 5–10% of the developed world's energy use.

The other (and germane) aspect is that of attack surface. We obviously don't know what the attack surface of the IoT is—we can scarcely imagine what "attack surface" means in context or even if it means something unitary and evaluable, but whatever that attack surface is, given (genuinely) exponential growth in counts of devices, it is hard to imagine that there is no risk being added to the connected parts of the globe. Just to keep aggregate risk static requires that the risk per device not only fall faster than the curve of deployment rises, but faster still if it is to drown out the legacy risk of devices previously installed. That is a tall order. Ergo, we should doubtless assume for planning purposes that we will see a significant, ongoing increase in the aggregate attack surface.

Or not. Does the attack surface construct even make sense when we are contemplating 10^{10} devices? Clearly, redundancy can contribute real survivability value for a sensor deployment—one broken sensor just doesn't matter if there are others doing the same data capture. One is reminded that network layout can deliver resistance to random faults or resistance to targeted faults but not both. Could not the same be said of sensor data and its roll-up to decision support, that one's threat model has to make some tradeoffs between being invulnerable to random sensor failure and being invulnerable to targeted (intentional) sensor failure?

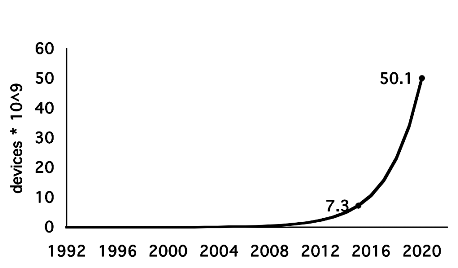


Figure 1: Billions of devices

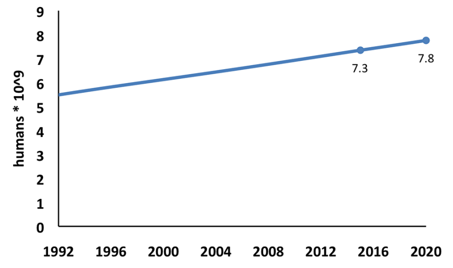


Figure 2: Billions of humans

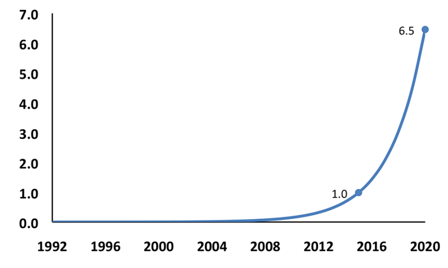


Figure 3: Devices per human

That said, the real question is what should we measure either for the numerator (risk) or the denominator (normalization to something)? *Hourly* data traffic today exceeds annual data traffic of only 10 years ago, and IoT devices are nothing if not traffickers in information. So is data volume the base proportionality constant for an "IoT attack surface"? Or is the attack surface proportional to the percentage of small devices that have Turing-complete remote management interfaces? Or is the attack surface proportional to the minimum practical latency between problem discovery and effective problem repair (thinking now of 1000 million devices with a common mode vulnerability just discovered).

As you know, there is much focus today in the security product market on behavioral security, on accumulating an idea of what routine operation looks like the better to detect badness early, but is anyone actually proposing watching 10^{10} data sources for actionable anomalies? Presumably not, but does that tell us of a latent need or does it tell us something else again? Something about redundancy or about minimizing dependence on singleton devices? Something about trading off the risks of single points of failure against the risks of common mode failure?

It is likely that there are no best answers, and that all answers will be context dependent—a threat model to rationalize attack surface measurement for, say, medical care will be something entirely different than for, say, shopping mall inventory control. Any context that actually matters will have to have an attack surface metric (or something like it) that scales well; Qualcomm's Swarm Lab at UC Berkeley has notably predicted 1000 radios per human by 2025, while Pete Diamandis' *Abundance* calls for 45×10^{12} networked sensors by 2035. These kinds of scale cannot be supervised, they can only be deployed and left to free-run. If any of this free-running is self-modifying, the concept of attack surface is just plain over. So, too, is the concept of trustworthy computing, at least as presently understood.

In any case, we are past the point of no return here. The IoT and its scale make most of our gross measures (like attack surface, say) into historical curiosities. The present author has long thought that the pinnacle goal of security engineering to be "No Silent Failure," and with the IoT at its predicted scale, perhaps that goal will now meet its most formidable challenge. It may be that for the IoT we security metricians will have to start over. It may be that our metrics for the IoT will be less observational and more analytic, such as "How much silent failure is tolerable?" Surely adding 10^{10} devices to the connected world increases its complexity, and more complexity means less system predictability, which conflicts with security goals. Distributions of events that we can detect and count today are looking more and more like power laws. If that is an emergent truth and not our confusion, then it is Nassim Taleb's prediction that matters most: "[We are] undergoing a switch between [continuous low grade volatility] to…the process moving by jumps, with less and less variations outside of jumps."

Yes, predictions around the IoT are a dime-a-dozen, many of them are non-falsifiable, and no forecaster ever got fired for adding an extra zero to some rosy daydream. What's your wager?

Extending Go Applications with Exec Plugins

KELSEY HIGHTOWER

Kelsey Hightower has worn every hat possible throughout his career in tech, and enjoys leadership roles focused on making things happen and shipping software. Kelsey is a strong open source advocate focused on building simple tools that make people smile. When he is not slinging Go code, you can catch him giving technical workshops covering everything from programming to system administration and distributed systems.
kelsey.hightower@gmail.com

Go has found a sweet spot among developers for building system tools ranging from Web services and distributed databases to command line tools. Most system tools of this nature tend to support multiple backends for providing application-specific functionality. For example, think about a command line tool that manages DNS records. Given the number of DNS providers available today, it would be nearly impossible to build support for every DNS API a user would want to interact with. This is where a plugin system can help. Plugin systems provide a common interface for extending applications with new functionality without major changes to the primary application.

In the Go ecosystem, the two most common ways of extending an application are by adopting an RPC plugin mechanism or by adding new code to a project that implements a plugin interface. Writing plugins using an RPC interface has the benefit of supporting plugins that live outside the core code base. This way, users can add and remove plugins without recompiling the main application. The major drawback to RPC plugins is the increased complexity that comes with running each plugin as a daemon and interacting with them over a network socket.

The practice of using RPC plugins is a bit of a hack; however, the method has become widespread because there is no other way to extend a Go application without adding new code and recompiling the main application. Go lacks the ability to load and execute external code at runtime, a feature that is common in languages like C or Java.

In the case of source plugins, each plugin lives in the main code base and typically implements a well-defined interface. Each plugin is only responsible for implementing the methods defined in the interface, which can greatly streamline plugin development.

Let's dive deeper into this topic and write some code that implements a source plugin mechanism. The application we are going to build is called translate, so named because it can translate a message from English to another language. It works like this:

```
$ translate -m "hello" -t spanish
hola
```

The translate application supports multiple languages through a simple plugin system. For each language we want to support we must add a new Go package for that language and implement the following interface:

```
type Translator interface {
  Translate(message string) (string, error)
}
```

Each translation plugin must provide a method named Translate that takes a message argument and returns the translated message and an error if the translation fails. The translate application allows users to select the translation plugin to use via the -t flag.

Create the following directories to hold the translate app source code:

```
$ mkdir -p $GOPATH/src/translate/plugins/spanish
```

Change into the translate source directory:

```
$ cd $GOPATH/src/translate
```

Save the following source code to a file named main.go:

```
package main

import (
   "flag"
   "fmt"
   "log"
   "translate/plugins"
   "translate/plugins/spanish"
)

func main() {
   var translator string
   var message string

   flag.StringVar(&translator, "t", "english", "Which translator
plugin to use")
   flag.StringVar(&message, "m", "", "The message to translate")
   flag.Parse()

   var t plugins.Translator

   switch translator {
   case "spanish":
      t = spanish.New()
   default:
      fmt.Printf("Plugin %s not found.", translator)
   }

   response, err := t.Translate(message)
   if err != nil {
      log.Fatal(err)
   }
   fmt.Println(response)
}
```

Save the following source code to a file named plugins/translator.go:

```
package plugins

type Translator interface {
   Translate(message string) (string, error)
}
```

```
Save the following source code to a file named plugins/spanish/
translator.go
```

```
package spanish

import (
   "errors"
)

type Translator struct{}

func New() Translator {
   return Translator{}
}

func (t Translator) Translate(message string) (string, error) {
   if message == "hello" {
      return "hola", nil
   }
   return "", errors.New("Translation error.")
}
```

At this point you can build and execute the translator application:

```
$ go build .
$ ./translate -t spanish -m "hello"
hola
```

As you can see, the -t flag allows us to select the language translator to use, and the -m flag sets the message to translate. The translate program will select the right plugin to process the translation based on the value of the -t flag. Let's review the code that makes this happens:

```
switch translator {
   case "spanish":
      t = spanish.New()
   default:
      fmt.Printf("Plugin %s not found.", translator)
}
```

Notice the problem here? We must know and implement every translator plugin before compiling the application. To extend the translate application, you'll have to modify its source tree, recompile, and reinstall the translate application. For many end users, source plugins set a barrier too high for contribution, which results in more work for project maintainers, who must either build or review every plugin users want to implement or use. A better solution to this problem would be to allow users to extend the translate application without modifying the code base. That's where exec plugins come in.

Exec Plugins

Exec plugins allow you to leverage external binaries as a plugin framework for extending applications. It's easy to think of exec plugins as similar to talking to an RPC endpoint. For each action, an executable can be invoked with a specific set of flags or environment variables to complete a task. Exec plugins can be written in any language (avoiding one drawback of source plugins) and have simple interface requirements.

Let's explore how an exec plugin system can improve the extensibility of our translate application. The goal is to keep the same top-level interface but make it possible to add new translation plugins without recompiling the translate application.

In the next part of this tutorial, you'll rewrite the translate application to provide extensibility through exec plugins. Start by deleting the current translate code base:

```
$ rm -rf $GOPATH/src/translate/*
```

Change to the translate source directory:

```
$ cd $GOPATH/src/translate
```

Save the following source code to a file named main.go:

```
package main

import (
    "bytes"
    "flag"
    "fmt"
    "log"
    "os"
    "os/exec"
    "path"
)

var (
    pluginsDir string
)

func main() {
    var translator string
    var message string

    flag.StringVar(&translator, "t", "english", "Which translator
plugin to use")
    flag.StringVar(&message, "m", "", "The message to translate")
    flag.StringVar(&pluginsDir, "p", "/tmp", "The plugin
directory")
    flag.Parse()

    t := path.Join(pluginsDir, translator)
    if _, err := os.Stat(t); os.IsNotExist(err) {
        fmt.Printf("Plugin %s not found.\n", translator)
        os.Exit(1)
    }

    cmd := exec.Command(t, "-m", message)
    var response bytes.Buffer
    cmd.Stdout = &response
    err := cmd.Run()
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(response.String())
}
```

The main difference from the original translate application is that we no longer need to know and implement each plugin before compile time. Instead, the translator plugin (identified by the -t flag with the path to an executable) will be called to translate the message:

```
t := path.Join(pluginsDir, translator)
if _, err := os.Stat(t); os.IsNotExist(err) {
   fmt.Printf("Plugin %s not found.", translator)
   os.Exit(1)
}
```

If the executable is not found on the file system, the translate application will print an error message and exit non-zero. This is a big improvement over the source plugin model, where the search for plugins is done only at runtime. Now plugins can be added by simply creating a binary and placing it in the translate plugins directory. If our app were a daemon, this would mean no restarts!

At this point you can build and execute the translate application:

```
$ go build .
$ ./translate -t spanish -m "hello"
Plugin spanish not found.
```

The updated translate application throws an error here because we have not written or installed any exec plugins. Let's write our first translator plugin and add support for a new translation; how about Japanese?

But just like the source plugin we need an interface to conform too. In the case of the exec plugin, we define the following interface:

- Translator plugins MUST accept a message to translate via an "-m" flag.
- Translator plugins MUST print the translation to standard out if the translation is successful.
- Translator plugins MUST exit non-zero if the translation fails.

With our interface defined, let's create a plugin.

First, create the the following directory to hold the Japanese translate plugin source code:

```
$ mkdir -p $GOPATH/src/japanese-translate-plugin/
```

Change into the Japanese translate plugin source directory:

```
$ cd $GOPATH/src/japanese-translate-plugin/
```

Save the following source code to a file named main.go:

```
package main

import (
   "flag"
   "fmt"
   "log"
   "os"
)

func main() {
   var message string
   flag.StringVar(&message, "m", "", "The message to translate")
   flag.Parse()

   if message == "hello" {
      fmt.Printf("こんにちは")
      os.Exit(0)
   }
   log.Fatal("Translation error.")
}
```

Compile the Japanese translator plugin:

```
$ go build -o japanese .
```

Before we copy the Japanese translator plugin to the translate application's plugin directory, we should test that it works. This is a clear benefit of exec plugins—we can test them outside of the main application:

```
$ ./japanese -m "hello"
こんにちは
```

Everything seems to be working. Now you need to copy the Japanese translator plugin to the translate plugin directory:

```
$ cp japanese /tmp/japanese
```

With the Japanese translator plugin in place, we can now rerun the translate application and use the Japanese plugin to handle the translation:

```
$ cd $GOPATH/src/translate
$ ./translate -m "hello" -t japanese
こんにちは
```

Following this pattern, adding support for new translations is easy:

```
$ cp norwegian /tmp/norwegian
$ ./translate -m "goodbye" -t norwegian
ha det
```

Exec plugins provide a simple mechanism for extending applications in a way that empowers end users and reduces maintenance overhead for project owners, and is a testament to the longevity of UNIX semantics and philosophy.

/dev/random

ROBERT G. FERRELL

Robert G. Ferrell is an award winning author of humor, fantasy, and science fiction, most recently *The Tol Chronicles* (www.thetolchronicles.com).
rgferrell@gmail.com

The other day I ran across an online tribute to the late Dan Blocker, "Hoss" Cartwright from the television series *Bonanza*. Even though that show played a significant role in my childhood during its original run, the modern age has had its insidious effect on my imagination, and I immediately began visualizing not the sprawling Ponderosa and Virginia City, but rather browser plugins that block posts from people named "Dan."

We have ad blockers, spam blockers, pornography blockers, opposing philosophy blockers, verbosity blockers like Twitter, and intelligence blockers (also known as the comments section of any news story or blog). Something about the anonymity and physical separation of the Web brings out the very, *very* worst in people. While it appears that a large segment of the population is responsible, maybe the same small group of malcontents is actually generating all of the commentary under different pseudonyms. Whatever the case, I wish they would take up some other hobby, like BASE jumping or free soloing.

If it's this bad on Earth, imagine what the Interplanetary or Interstellar Internets connecting human colonies scattered across the galaxy would look like. *You* imagine it, because I'm going to pass on that one. As this is an election year here in the US, my overall opinion of humanity's claim to sentience is already in the toilet.

Getting back to blockers, the one I want to see would be an idiocy blocker. While I already have a fairly effective version on my computer, known by the technical term "power button," it has the considerable drawback of also blocking the (admittedly increasingly rare) content I actually *want* to see. I suppose I could simply stop reading the news, since any given day can be summed up by mixing and matching from the following headlines: Ceasefire Violated, Terrorist Bombing, Trade Union Strikes, Politician Bumbles, Climate Changes, Vehicle Crashes, Species Faces Extinction, and Media-Manufactured Personality Engages in Behavior that Would not Make the News if Anyone Else Did It.

How would an idiocy blocker work? Probably not very well, since blocking idiocy would be tantamount to blocking online contributions *in toto*, but presuming idiocy could indeed be distinguished with any reliability from non-idiotic commentary, the ideal blocker would either blank idiocy out entirely or replace it with something the developers or end user have deemed non-idiotic. That substituted text—the First Amendment, perhaps—is likely to be wholly irrelevant to the topic at hand, but in this it differs little from most idiotic commentary.

Filtering out idiocy will require some robust algorithms, naturally, because while idiocy is seldom subtle, it can be somewhat cryptic. As an aid to coders who might be considering such an app, I have grouped some of the more common and egregious idiots into broad categories.

1. **Focus-Pocus:** This person has one or a small number of pet topics, commentary concerning which they will place following absolutely any online post, from major disaster to slapstick comic. Often they don't even attempt to tie it into the subject matter of the original item, spewing their idiotic irrelevancies with no segue whatever. Popular topics for these sorts include political candidates, climate change, puzzling interpretations of the US Constitution, misanthropy, and conspiracy theories. (I do not, incidentally, favor that last term because it degrades the noble noun *theory*.) The worst examples of this species of idiot can work all of the above into one mind-numbing run-on sentence that can, if viewed without proper safety precautions, lead to debilitating neural trauma.
2. **The Clever Dan:** This idiot has some form of self-induced brain damage that convinces him of his own wholly illusionary sparkling wit. Moreover, it convinces him that it is his solemn duty to share said wit in any and all conceivable fora, especially those where crass humor is antithetical to the situation. Look for CDs below testimonials to the deceased, reports of horrific tragedies, heartwarming accounts of nice things done for those with terminal illnesses, and any piece that reminds us all of the capricious nature of fate.
3. **The Nonsequiturian:** The issue here is not so much the logic or nature of the commentary itself, but the choice of venue. These particular idiots simply drop their mission statements in wherever opportunity presents, as a form of philosophical spam. They in no way pass the Turing Test and are probably merely bots that take vaguely human form while putting the "artificial" in Artificial Intelligence.
4. **The Recycler:** This is really a subcategory of Type 1, but instead of simply launching into idiot spew right away, recyclers will pretend to be engaged in debate with some other commenter(s) over points at least marginally relevant to the subject matter before inevitably working the rhetoric around to their pet topic. The most talented of these idiots can keep up the illusion of relevancy for several posts before it becomes obvious that they're really only here to push some totally different agenda.
5. **The Angler:** While the other four types of idiots are mostly harmless, albeit annoying, anglers are quite simply evil. They want to steal your identity and thus your money. They accomplish this by posting some seriously spammy content you would think absolutely no one would fall for—and yet people regularly do. Let's be clear about something: if you could really make $3,000 a week for 10 hours surfing the Internet, we'd all be doing it. Of course, they're probably telling the truth about how much they make, and that the Internet is involved. They just fib a little about the actual mechanism.

There are, of course, myriad other species and subspecies of idiot to be found on the Interwebs, but to filter them all out would be a Herculean task that would leave us with no one at all to talk to. If you truly wish to filter out idiots completely, I have found it quite effective to set one's firewall to disallow comments or email from any IP address that starts with "1" or "2."

Book Reviews

MARK LAMOURINE

SPA Design and Architecture
Emmit A. Scott, Jr.
Manning Publications, 2016, 288 pages
ISBN 978-1-61729-243-1

It's been a long time since I worked on the front end of a Web service. I'm passing familiar with JavaScript and completely comfortable with HTML and the DOM. The ideas of AJAX and REST services are clear. I thought I had a reasonable handle on how the client side of Web applications were built.

The "SPA" in the title stands for "Single Page Application," and you've almost certainly used one. If you've used almost any of Google's applications or searched for local movies you've seen an SPA.

SPAs have become prevalent in the last five years with the adoption of the HTML5 standard and a set of JavaScript frameworks which take care of much of the boilerplate and common behaviors. They are designed to move much of the application logic to the client side (your browser) and to minimize the delays involved in repeated page loads that characterized early Web applications.

Scott's goal in this book is to show you the internals of an SPA and then how to assemble them into an application.

After the mandatory chapter introducing SPAs in general, I was a little surprised that the next two didn't seem to speak directly to SPAs at all. In Chapter 2 Scott gives a survey of the MVC (model view controller) pattern and its derivatives. All of the JavaScript frameworks for SPAs are based on one of these models. Scott cites a number of the most popular ones as examples and notes some of the benefits, costs, and quirks of each one without picking a favorite.

Chapter 3 is a tutorial on modules in JavaScript. I would have thought the module construct was a well-known idiom, but I admit I learned a lot from Scott's description.

My confusion resolved as I continued to read.

It turns out that this kind of book is hard to write and not easy to read and understand at the first pass. Developing modern Web applications requires fluency in at least three "languages" as well as the behaviors and quirks of all of the major Web browser rendering systems (even when using a framework to abstract them). Designing and implementing the client side of an application requires the developer to have an intricate understanding of how the data flows from a server through the client application and how that is, in turn, presented by the browser. Scott guides the reader through these interactions from the browser back to the server as well as touching on testing and debugging. That it took me a couple of times through to digest it is a reflection of my own meager background in this area.

Scott starts with application "routing" and the idea that in an SPA there is only a single "Web page" but there are multiple views of the service. The view is selected through the "router," which takes advantage of the browser URL history and the ability of the browser to decompose a URL and respond as instructed by some loaded JavaScript.

In the next chapter, Scott reveals how to control the layout and presentation of the views using HTML, CSS, and various templating frameworks. This is where the application is given both a structure and a style that (it is hoped) presents the user with the information and behaviors they need to complete their tasks efficiently.

The JavaScript module pattern comes back to the fore now. Scott shows how this pattern can be used to map logic to each view in a clean, coherent manner. He also discusses how the data will be represented in the client-side model of the application. This leads nicely into the final active part of the SPA: communicating with the server.

In this chapter Scott examines how to generate and respond to asynchronous requests to the server both with several of the major SPA framework mechanisms and using the XMLHttpRequest method directly. He details asynchronous data exchange in both directions and shows how to build the service interactions into the modules that make up the client-side data model.

The final sections cover unit testing and client-side tasks. The latter are actions that the client may take which are not directly associated with any particular model or view. He presents them as a means to run repetitive tasks during development such as code CI and testing.

Scott doesn't try to create a single application in his narrative. Because he is reflecting on a number of different frameworks, using their contrasts to highlight behaviors and features, no single sample application would fit. He does include a short application example using Angular.js and Backbone.js in the appendices, but his presentation in the main body of the book is fairly agnostic to any framework selection.

Each chapter concludes with a set of questions and exercises that are meant to help the reader set the main concepts in memory and to give some active practice. These "challenges" are indeed a challenge, not something you can merely cut-and-paste from the text. A reader who follows through will get much more than a casual reader.

In the end I liked Scott's focus on concepts and options rather than advocating for one framework or another. I think I was not as prepared as I should have been to take this book on. My knowledge of JavaScript and DOM is rusty, and I have not kept up on current practice and idiom. This required me to go outside and brush up to be sure I'd understood and absorbed what was presented. This is a good book for someone who has gotten their hands dirty with browser programming and is ready to start learning how to design a fast modern Web service.

Single Page Applications: JavaScript End-to-End
Michael S. Mikowski and Josh C. Powell
Manning Publications, 2014, 408 pages
ISBN 978-1-617290-75-0

"You can do it all in JavaScript. Here's how" is the message that the authors offer in *Single Page Applications: JavaScript End-to-End.* Their aim is to build an SPA demonstrator, client, and server side completely in JavaScript. They go so far as to avoid even the popular SPA JavaScript frameworks, choosing instead to build the core functionality, the routing and view selection logic, even the HTML template resolution directly in JavaScript.

I'm not sure most people would be willing to take on the extra work that the SPA frameworks offload for you, but there's certainly a lot to be learned by looking at how one would do it.

Mikowski and Powell follow the tried-and-true narrative of building a simple demo app and enhancing it chapter by chapter. In their case it actually shows a good Agile style progression, although I'm not sure if that was their intent. Because they are building both the client and the server in JavaScript, and you don't get to the server part until more than half way through, you also learn a lot about mocking data and services in JavaScript.

They begin by building a skeleton for the application, which they call the "Shell." This is a kind of root module for the application. Features will be hung off this module and will add functionality as the development process progresses. From here they show how to add logic and presentation to each new feature in a clean, incremental way. They develop the data models and views in conjunction so the reader can see how the back and front are related and how data flows in and out.

On the server side, Mikowski and Powell use Node.js and MongoDB. They promote the idea that using a single language for both the client and server makes development easier. While in general I agree, I wouldn't normally have picked JavaScript as my one language, but since the browsers have chosen for us it will have to do. Certainly the use of JSON for data transfer and storage does remove lots of the hassle of encoding and decoding data both for communications and database storage.

The authors are very conscious of the development environment and developer tasks. While developing the components, they also lay out best practices for directory structure and file naming for consistency and maintainability. These seem to mirror other recommendations I've seen. They have an appendix devoted to a set of JavaScript coding styles. For someone overwhelmed with the actual design and implementation of a service, these nicely structured guidelines are actually a time-saver when learned and applied. There's no need to spend time rediscovering what others have already done (and likely as not having to refactor the mess to conform when you find out what they already knew).

The final area Mikowski and Powell talk about in the main section is actually something I hadn't considered part of the normal development process: design and adaptation to search engine and analytics services that crawl your site, and third-party caching services. This section was an eye opener for someone who's never worked with these except as a user or out of intellectual curiosity. In this section you learn some about how these services work and how to make your application friendly to them.

I don't think I'll be adopting this approach to application design, but what I learned here will certainly inform how I look at the systems I work on and how I build new ones.

Go in Practice
Matt Butcher and Matt Farina
Manning Publications, 2016, 288 pages
ISBN 978-1-63343-007-6

I'm familiar with Manning's "in Action" series and have actually reviewed *Go in Action* here. I was curious what would be different about *Go in Practice*. The cover notes that the book "includes 70 techniques." It turns out that "in practice" means this is a Go cookbook.

Most cookbooks I've read have spent most of the time on the shelf gathering dust. Either the recipes are for things that are either obvious or rare and obscure. I was pleasantly surprised by *Go in Practice*. Butcher and Farina have managed to create a reference for Go idiom and good practice. Given how quirky Go can be, especially for someone coming from a scripting language (I had some nostalgic flashbacks to my days coding C), a manual of good practice is a welcome find.

I think Butcher and Farina may have the same impression of the cookbook metaphor as I do. They avoid the term throughout the book, substituting "task" and "technique." The word "cookbook" is only used once, in what I suspect is an editorial description on the back cover. I'll use their terminology because I think what they present is better than a set of recipes.

Go in Practice is not a language reference. The authors do highlight some of the significant language features in the first section: multiple return values, dummy return values, goroutines, and channels. They also discuss package management, revision control, and Go's relationship to other popular languages.

The next section is one I particularly liked and will use often: a complete section on managing inputs and configuration for CLI programs. This has always seemed to me to be an overlooked part of most language teaching.

The full set of techniques covers things you'd expect—e.g., testing and debugging, Web service communications—but it also includes some things that I haven't seen in other places, such as aspects of coding for the cloud. This includes writing API interactions with cloud services which avoid lock in and how cloud-hosted programs can get VM information from the providers.

They close out the set of techniques with a section on code reflection and automatic code generation in Go. These are advanced techniques and probably shouldn't be used lightly. Most people will end up using annotations and tags for tasks like JSON or XML processing. However unlikely it is, they also show how to create new ones and then process them.

Each technique opens with a paragraph or two on the problem to be solved, then a brief description of the solution. The meat is in the discussion and code fragments that follow. The layout of the code is clean and contains clear annotations. I often try to read both the paper and ebook forms of the books I review. As much as I love to have bound paper on a shelf, the ebook has an edge in this case. The diagrams and code samples in the ebook have color graphics and highlighting which add an appeal that the black and white on paper can't match.

The ebook format is also well suited to handy access on tablets or browsers. For as long as I'm coding Go, I expect I'll keep *Go in Practice* close.

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

Free subscription to *;login:*, the Association's quarterly magazine, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks and operating systems, and book reviews

Access to *;login:* online from December 1997 to the current issue: www.usenix.org/publications/login/

Discounts on registration fees for all USENIX conferences

Special discounts on a variety of products, books, software, and periodicals: www.usenix.org/member-services/discount-instructions

The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers

For more information regarding membership or benefits, please see www.usenix.org/membership/or contact office@usenix.org. Phone: 510-528-8649.

USENIX Board of Directors

Communicate directly with the USENIX Board of Directors by writing to board@usenix.org.

USENIX: It's Not What You Might Think

Cat Allman, USENIX Board

When Casey Henderson, USENIX Executive Director, asked me to run for the board, I was surprised. (Wildly flattered because I think of board members as elite technologists, which I am not, but surprised.) I don't code. There—I said it. I don't write code. So what the heck am I doing here on the Board? Everyone knows that USENIX is an academic organization for CS professors and their students, right? Well, no.

I first got involved with USENIX back in the 1980s while setting up a computer service bureau for a design firm. I'd known about USENIX through my brother for some time and felt their resources for system administrators would be useful to me, both technically and for justifying my costs to my computer illiterate bosses. This last part turned out to be the most important reason for me. The most challenging part of building and running what turned into a decent-sized production IT department / profit center for management who thought "high-tech" was a style of interior design was getting them to understand the costs involved, managing their expectations, and getting them to pay me what I was worth. USENIX publications and community helped me with all that and gave me people to talk with who understood what I was doing.

After what seemed like a heck of a long time, I found myself working at Sendmail, Inc., sponsoring USENIX events to drive usage of Sendmail's products and services. USENIX events were the best source of high quality sales leads for us, plus the people were a pleasure. Booth duty—I actually enjoyed it at USENIX events, where the attendees are smart and—wait for it—engineers. Honest, practical, problem-solving engineers.

Next step in my USENIX evolution was joining the staff. In 2002 I jumped to the other side of the desk and spent 4.5 years working 80% time in the Berkeley office marketing the org to potential commercial sponsors. Cold-calling is not my favorite thing, but you develop a taste for the hunt, you get to talk with some terrific people, and helping USENIX, the organization that had done so much for me, support itself was deeply satisfying.

But I couldn't say no to Google's Open Source Programs Office. My time with Sendmail had drawn me deeply into the world of free and open source software, and once you've found your tribe it's hard to stay away. USENIX had roots in open source but at the time was increasingly focused on academic publishing, a worthy thing, but not "my thing" and not what drew me to USENIX all along. The chance to work full time promoting open source was way too good to pass up. An added benefit was being back on the buyer's side of the desk where I could sponsor USENIX again. (Most recently, Google sponsored USENIX Open Access!)

Speaking of Open Access; this effort is hugely important to the spread of CS education and technical innovation, and I am deeply proud to be associated with USENIX around this issue in particular.

Fast forward to three years ago when Casey approached me about running for the Board. I was initially reluctant since, as I said, me and academic publishing—not so much—but as we talked I came to see that the USENIX community is in a great position to serve the latest generation of SRE/DevOps through our continuing blend of cutting edge research and advanced professional practice. Call it "embracing our roots" or simply recognizing our strengths; we have so much to offer, and I want to be a part of this effort.

So there you have it: a USENIX Board member who was never an academic, can't code, and has worked in or with "industry" for the 30+ years she's been involved with USENIX. I agreed to run for the Board (and thank you everyone who voted for me!) to give back to an organization that has done so much for me, and to encourage USENIX to more fully embrace creating content for and by working practitioners. The Enigma conference this past January was a great example of USENIX's proud history of surfacing research that furthers advanced practice. I'm looking forward to more of these kinds of conferences in the future and hope to see you all there!

Impressive Results for Team USA at 2016 International Olympiad in Informatics

Brian C. Dean, Director, USA Computing Olympiad

I am thrilled to be able to report to the USENIX community another highly successful year for the USA Computing Olympiad and its participation in the International Olympiad in Informatics!

Those who follow the "sport" of competitive programming know the International Olympiad in Informatics (IOI) as the most prestigious algorithmic computer science competition at the high school level. The IOI is held in a different country each year, and the 28th annual IOI took place in August 2016 in Kazan, one of the largest cities in Russia and the capital of its Tatarstan Republic. Delegations from 80 countries attended, each bringing a team of their top four high-school computing students. It is a phenomenal opportunity for the students not only to be able to compete at such a high level, but also to meet and interact with peers from around the world with similar interests and talent.

The IOI takes place over an entire week, offering competitors a chance to experience local culture, food, and customs. Excursions from our home base at Kazan Federal University included a trip to the Kazan Kremlin, a tour of the picturesque island of Sviyazhsk, and a visit to Innopolis, an entire futuristic tech-centric city created from the ground up in just the past four years. I am told that incriminating video footage even exists of the USA delegation taking part in an exercise to learn traditional Russian styles of dance.

However, the main event is the competition. Two five-hour contests held on separate days each feature three challenging problems which the students, working individually, have to solve, most of them using C++. The problems are algorithmic in nature, so the key to getting high scores is to implement algorithms that are fast enough to solve the largest test cases within a certain time limit. Top students at the IOI are given gold, silver, and bronze medals.

Out of 308 contestants at the event, only 26 received the highly coveted gold medals. Team USA struggled on the first competition day but turned in stellar results on the second day to complete a dramatic comeback, yielding the following results:

- Daniel Chiu (gold medal), Catlin Gabel School, OR
- Lawrence Li (gold medal), The Harker School, CA
- Dhruv Rohatgi (gold medal), The Davidson Academy of Nevada, NV
- Calvin Lee (silver medal), Homeschooled, NY

Accompanying these students to Kazan were team leader Brian Dean (a computer science professor at Clemson University who has trained the USA IOI team now for 20 years) and deputy leader Travis Hance, a recent MIT graduate now working at Dropbox, who competed for team USA in 2009.

No team at IOI 2016 earned four gold medals, and the only others earning three gold medals were Russia and China, tying the USA for top country by medal count. Of the 25 years the USA has now competed in this event, it has only won three or more golds seven times, six of these happening in the past seven years, demonstrating how we have now reached a level of consistent excellence as one of the top competitors at the IOI.

High-school computing education in the USA is not known for its support of advanced programming and problem-solving concepts, so how has team USA reached this impressive level of performance? Fortunately, advanced students can find the resources they need to excel through the USA Computing Olympiad, a national nonprofit program that provides free online training and programming contests for students at all levels. Our contests were recently extended to four divisions—ranging from a bronze division that is accessible to students just learning to program, up to a platinum division that challenges the best students in the world—with problems that are IOI-level or even harder. Tens of thousands of students have participated in our online training site, and thousands compete in our online contests each year. This year we again set new records for participation, roughly double where we were five years ago.

Top students across the USA from the online USACO contests—roughly two dozen—are invited to a rigorous summer training camp in early June, where they learn advanced algorithmic techniques from our dedicated volunteer staff (many of them former IOI team members themselves). At the end of training camp, the top four students are named to the USA team to attend the IOI.

The USACO plays a vital role in computing education in the USA, helping to ensure a steady stream of highly talented computational problem-solvers moving through the pipeline from high school to university. The program has now been around long enough to see the amazing impact of our alums, from becoming superstar professors in academia to innovators in the tech startup arena. Many wonderful programs exist for helping with computing education and outreach, but very few address the needs of our most advanced students or the increasingly important topic of algorithmic problem solving. The USACO fills a vital niche in this respect, and I am constantly thanked by students from many countries at the IOI for the free resources we have made available for all to use (many other countries utilize these resources, and our contests routinely have participation from 70+ countries).

Our program would not exist if it were not for USENIX and our other dedicated sponsors, and high-school computing owes all of these generous organizations a debt of gratitude for their wide-reaching contributions. USENIX, in particular, has contributed hundreds of thousands of dollars over a span of 16 years! Directing the USACO has been one of the most fulfilling activities of my professional career, and I look forward with great enthusiasm to continuing our momentum into next season as we train for IOI 2017 in Tehran, Iran.

If you are interested in more information on USACO or wish to participate in any of our activities (they are free and open to all), please visit our Web site, usaco.org.

Thanks to Our Volunteers

by Casey Henderson, USENIX Executive Director

As many of our members know, USENIX's success is attributable to a large number of volunteers who lend their expertise and support for our conferences, publications, good works, and member services. They work closely with our staff in bringing you the best in the fields of systems research and system administration. Many of you have participated on program committees, steering committees, and subcommittees, as well as contributing to this magazine. The entire USENIX staff and I are most grateful to you all. Below, I would like to make special mention of some people who made particularly significant contributions in 2016.

Program Chairs

Enigma 2016
David Brumley and Parisa Tabriz

14th USENIX Conference on File and Storage Technologies (FAST '16)
Angela Demke Brown and Florentina Popovici

2016 USENIX Research in Linux File and Storage Technologies Summit (Linux FAST Summit '16)
Christoph Hellwig and Ric Wheeler

13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)
Katerina Argyraki and Rebecca Isaacs

2016 USENIX Workshop on Cool Topics in Sustainable Data Centers (CoolDC '16)
Weisong Shi and Thomas F. Wenisch

SREcon16
Liz Fong-Jones, Melita Mihaljevic, and Coburn Watson

2016 USENIX Annual Technical Conference (USENIX ATC '16)
Ajay Gulati and Hakim Weatherspoon

Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)
Sunny Consolvo and Matthew Smith

8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '16)
Nitin Agrawal and Sam H. Noh

8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)
Austin Clements and Tyson Condie

SREcon16 Europe
Narayan Desai and John Looney

25th USENIX Security Symposium (USENIX Security '16)
Thorsten Holz and Stefan Savage

2016 USENIX Summit on Hot Topics in Security (HotSec '16)
Damon McCoy and Franziska Roesner

2016 USENIX Workshop on Advances in Security Education (ASE '16)
Mark Gondree and Zachary N J Peterson

6th USENIX Workshop on Free and Open Communications on the Internet (FOCI '16)
Amir Houmansadr and Prateek Mittal

9th USENIX Workshop on Cyber Security Experimentation and Test (CSET '16)
Eric Eide and Mathias Payer

10th USENIX Workshop on Offensive Technologies (WOOT '16)
Natalie Silvanovich and Patrick Traynor

12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)
Kimberly Keeton and Timothy Roscoe

4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW '16)
Peter Desnoyers and Kaoutar El Maghraoui

30th Large Installation System Administration Conference (LISA16)
John Arrasjid and Matt Simmons

2016 Summit for Educators in Systems Administration (SESA '16)
Kyrre Begnum and Charles Border

Other Chairs and Major Contributors

FAST '16
Work-in-Progress/Posters Co-Chairs: Haryadi Gunawi and Daniel Peek
Tutorial Coordinator: John Strunk

NSDI '16
Poster Session Co-Chairs: Aruna Balasubramanian and Laurent Vanbever

SOUPS 2016
General Chair: Mary Ellen Zurko
Invited Talks Chair: Yang Wang
Lightning Talks and Demos Chair: Elizabeth Stobert
Panels Chair: Tim McKay
Posters Co-Chairs: Michelle Mazurek and Florian Schaub
Tutorials and Workshops Co-Chairs: Adam Aviv and Mohammad Khan
Publicity Chair: Patrick Gage Kelley

USENIX Security '16
Invited Talks Chair: Adrienne Porter Felt
Invited Talks Committee: Tyrone Grandison, Alex Halderman, Franziska Roesner, and Elaine Shi
Poster Session Chair: Raluca Popa
Poster Session Committee Members: Nikita Borisov and Mathias Payer
Work-in-Progress Reports (WiPs) Coordinator: Patrick Traynor

FOCI '16
Publicity Chair: Sandy Ordonez

OSDI '16
Poster Session Co-Chairs: George Porter and Chris Rossbach
Luncheon on Supporting Diversity in Systems Research Organizer: Dilma Da Silva

LISA16
Invited Talks Co-Chairs: Pat Cable and Ben Cotton
Tutorial Co-Chairs: Mike Ciavarella and Chris St. Pierre
Workshops Chair: Lee Damon
LISA Lab Coordinators: Christopher DeMarco and Andrew Mundy
LISA Build Coordinators: Branson Matheson and Brett Thorson

Storage Pavilion and Data Storage Day at LISA16
Organizer: Jacob Farmer of Cambridge Computer

2016 *USENIX Journal of Education in System Administration (JESA)*
Editors-in-Chief: Kyrre Begnum and Charles Border

USENIX Board of Directors
Cat Allman, John Arrasjid, Michael Bailey, David Blank-Edelman, Angela Demke Brown, Daniel V. Klein, Brian Noble, Kurt Opsahl, Carolyn Rowland, and Hakim Weatherspoon

Audit Committee
Eric Allman, John Arrasjid, and Niels Provos

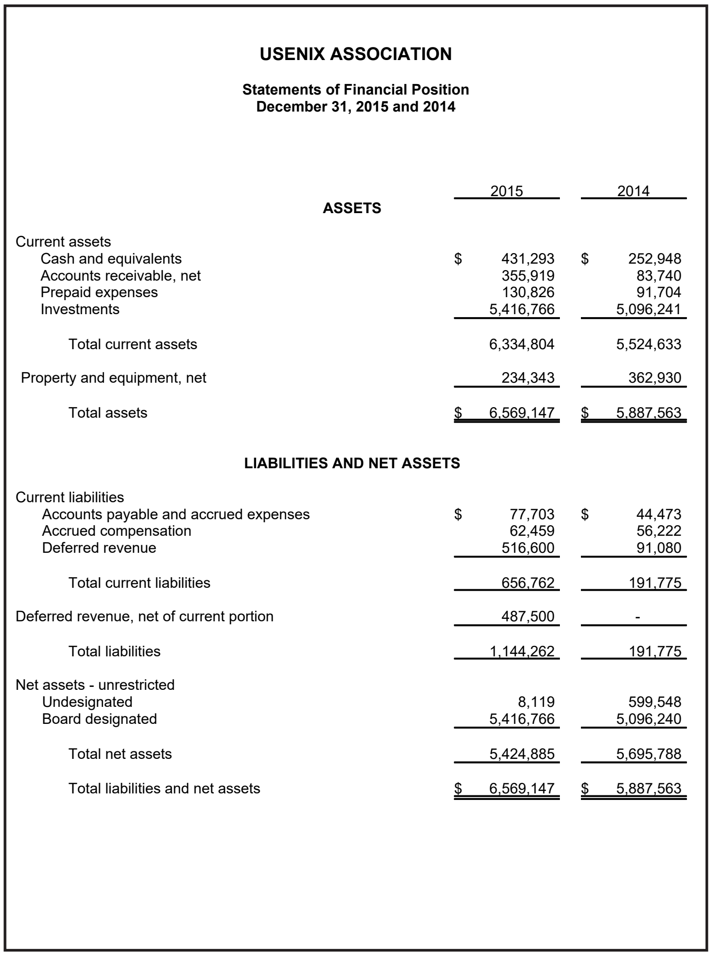
HotCRP Submissions and Reviewing System
Eddie Kohler

USA Computing Olympiad (co-sponsored by USENIX)
Team Leader: Brian Dean
Deputy Team Leader: Travis Hance

USENIX ASSOCIATION FINANCIAL STATEMENTS FOR 2015

The following information is provided as the annual report of the USENIX Association's finances. The accompanying statements have been reviewed by Michelle Suski, CPA, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public Accountants. The 2015 financial statements were also audited by Bong, Hillberg Lewis Fischesser LLP, CPAs. Accompanying the statements are charts that illustrate the breakdown of the following: operating expenses, program expenses, and general and administrative expenses. The operating expenses for the Association consist of the following: program expenses, management and general expenses, and fundraising expenses, as illustrated in Chart 1. The operating expenses include the general and administrative expenses allocated across the Association's activities. Chart 2 shows the breakdown of USENIX's general and administrative expenses. The program expenses, which are a subset of the operating expenses, consist of conferences and workshops; membership (including *;login:* magazine); projects, programs, and good works projects; their individual portions are illustrated in Chart 3. The Association's complete financial statements for the fiscal year ended December 31, 2015, are available on request.

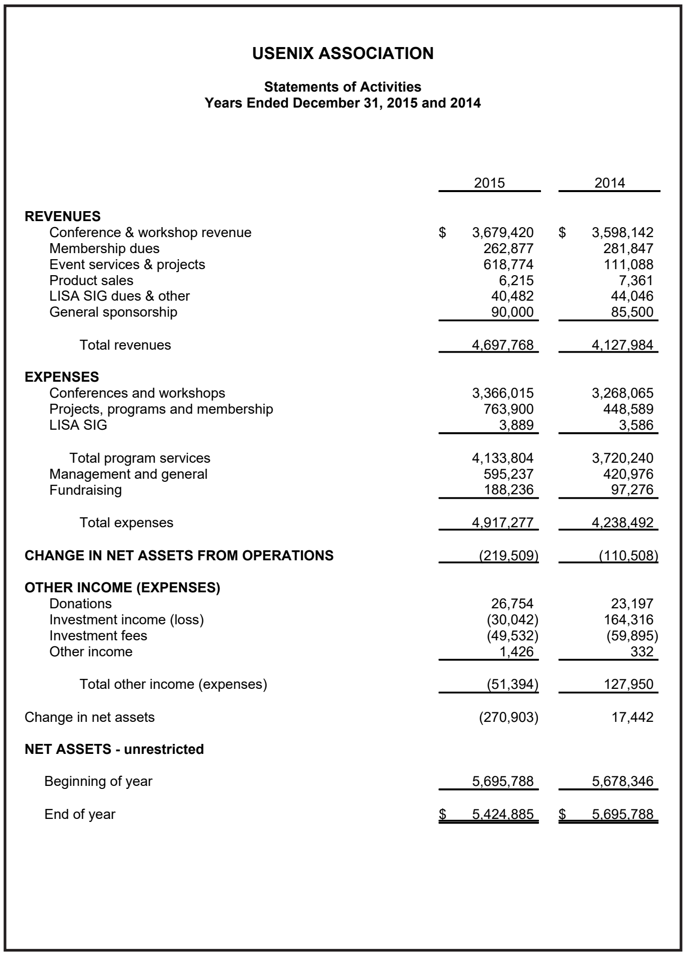
Casey Henderson, Executive Director



USENIX ASSOCIATION

Statements of Financial Position
December 31, 2015 and 2014

	2015	2014
ASSETS		
Current assets		
Cash and equivalents	$ 431,293	$ 252,948
Accounts receivable, net	355,919	83,740
Prepaid expenses	130,826	91,704
Investments	5,416,766	5,096,241
Total current assets	6,334,804	5,524,633
Property and equipment, net	234,343	362,930
Total assets	$ 6,569,147	$ 5,887,563
LIABILITIES AND NET ASSETS		
Current liabilities		
Accounts payable and accrued expenses	$ 77,703	$ 44,473
Accrued compensation	62,459	56,222
Deferred revenue	516,600	91,080
Total current liabilities	656,762	191,775
Deferred revenue, net of current portion	487,500	-
Total liabilities	1,144,262	191,775
Net assets - unrestricted		
Undesignated	8,119	599,548
Board designated	5,416,766	5,096,240
Total net assets	5,424,885	5,695,788
Total liabilities and net assets	$ 6,569,147	$ 5,887,563



USENIX ASSOCIATION

Statements of Activities
Years Ended December 31, 2015 and 2014

	2015	2014
REVENUES		
Conference & workshop revenue	$ 3,679,420	$ 3,598,142
Membership dues	262,877	281,847
Event services & projects	618,774	111,088
Product sales	6,215	7,361
LISA SIG dues & other	40,482	44,046
General sponsorship	90,000	85,500
Total revenues	4,697,768	4,127,984
EXPENSES		
Conferences and workshops	3,366,015	3,268,065
Projects, programs and membership	763,900	448,589
LISA SIG	3,889	3,586
Total program services	4,133,804	3,720,240
Management and general	595,237	420,976
Fundraising	188,236	97,276
Total expenses	4,917,277	4,238,492
CHANGE IN NET ASSETS FROM OPERATIONS	(219,509)	(110,508)
OTHER INCOME (EXPENSES)		
Donations	26,754	23,197
Investment income (loss)	(30,042)	164,316
Investment fees	(49,532)	(59,895)
Other income	1,426	332
Total other income (expenses)	(51,394)	127,950
Change in net assets	(270,903)	17,442
NET ASSETS - unrestricted		
Beginning of year	5,695,788	5,678,346
End of year	$ 5,424,885	$ 5,695,788

Continued on page 94

USENIX ASSOCIATION FINANCIAL STATEMENTS FOR 2015

Chart 1: USENIX 2015 Operating Expenses

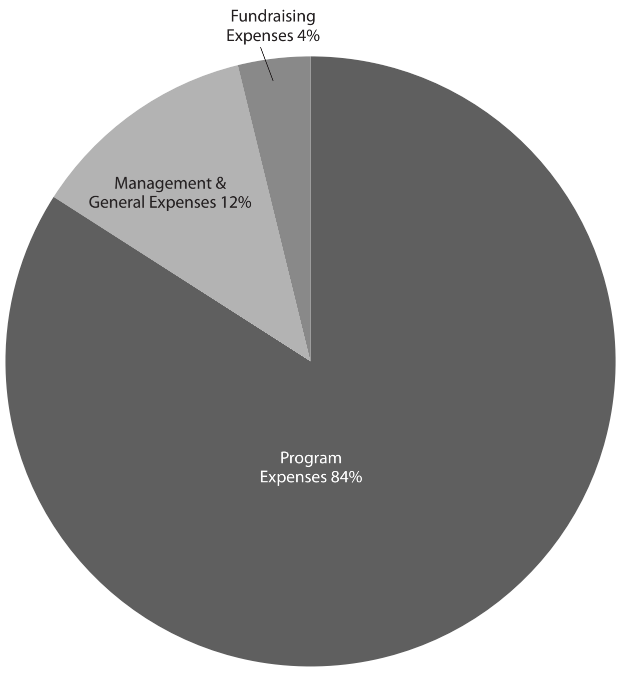


Chart 2: USENIX 2015 General & Administrative Expenses

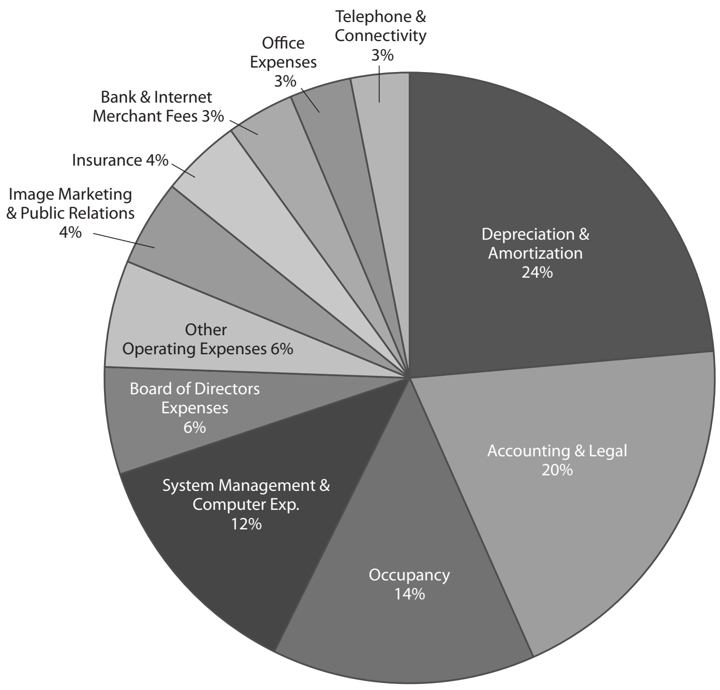
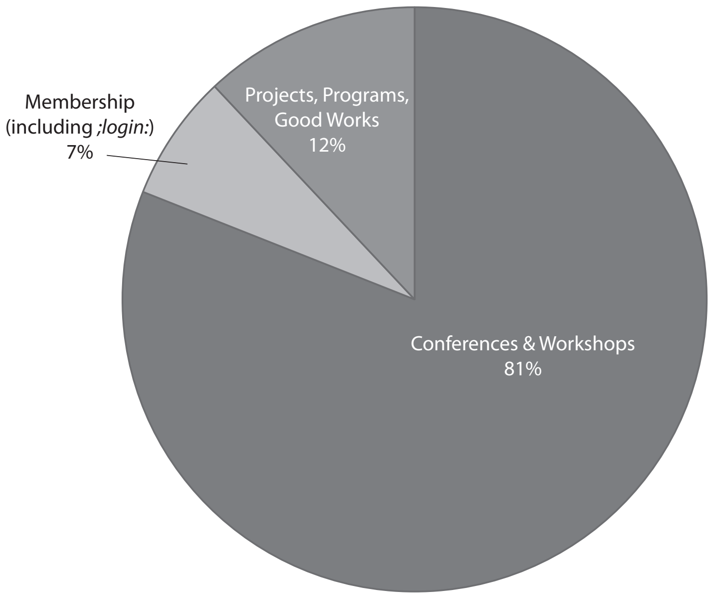


Chart 3: USENIX 2015 Program Expenses



2017 USENIX Annual Technical Conference

July 12–14, 2017 • Santa Clara, CA, USA

Sponsored by USENIX, the Advanced Computing Systems Association

Important Dates

- Paper submissions due: Tuesday, February 7, 2017, 11:59 p.m. GMT
- Notification to authors: Monday, April 24, 2017
- Final papers due: Wednesday, May 31, 2017

Conference Organizers

Program Co-Chairs

Dilma Da Silva, *Texas A&M University*
Bryan Ford, *École Polytechnique Fédérale de Lausanne (EPFL)*

Program Committee

To be announced

Overview

Authors are invited to submit original and innovative papers to the Refereed Papers Track of the 2017 USENIX Annual Technical Conference. We seek high-quality submissions that further the knowledge and understanding of modern computing systems with an emphasis on implementations and experimental results. We encourage papers that break new ground, present insightful results based on practical experience with computer systems, or are important, independent reproductions/refutations of the experimental results of prior work. USENIX ATC '17 has a broad scope, and specific topics of interest include (but are not limited to):

- Architectural interaction
- Big data infrastructure
- Cloud computing
- Datacenter networking
- Deployment experience
- Distributed and parallel systems
- Embedded systems
- Energy/power management
- File and storage systems
- Mobile and wireless
- Networking and network services
- Operating systems
- Reliability, availability, and scalability
- Security, privacy, and trust
- System and network management and troubleshooting
- Usage studies and workload characterization
- Virtualization

USENIX ATC '17 is especially interested in papers broadly focusing on practical techniques for building better software systems: ideas or approaches that provide practical solutions to significant issues facing practitioners. This includes all aspects of system development: techniques for developing systems software; analyzing programs and finding bugs; making systems more efficient, secure, and reliable; and deploying systems and auditing their security.

Experience reports and operations-oriented studies, as well as other work that studies software artifacts, introduces new data sets of practical interest, or impacts the implementation of software components in areas of active interest to the community are well-suited for the conference.

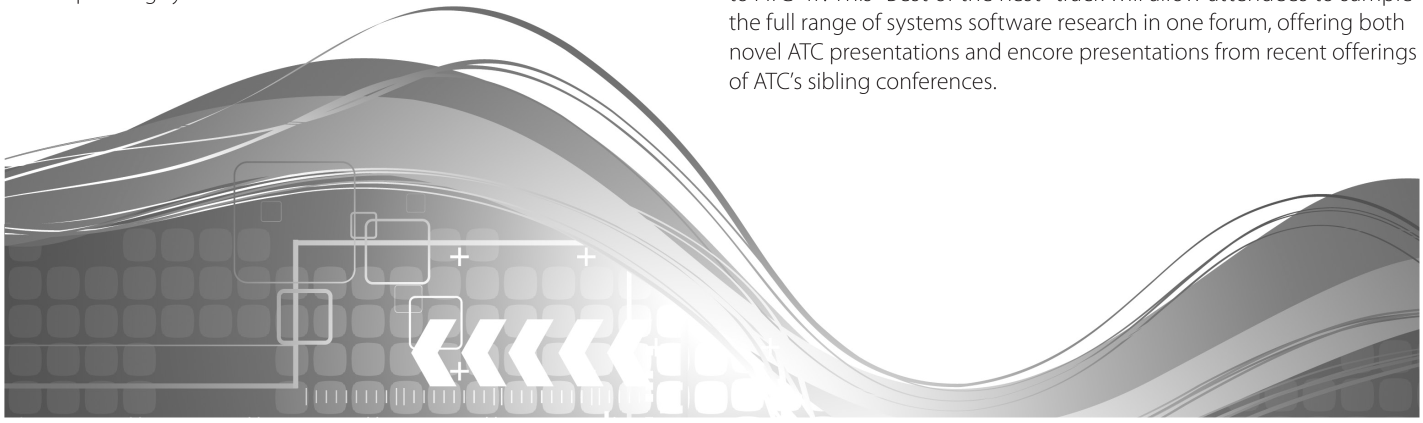
The conference seeks both long-format papers consisting of 11 pages and short-format papers of 5 pages, including footnotes, appendices, figures, and tables, but not including references. Short papers will be included in the proceedings and will be presented as normal but in sessions with slightly shorter time limits. For industrial practitioners, if you are interested in the Practitioner Talks Track, which accepts proposals for 20-minute or 40-minute talks, please refer to the USENIX ATC '17 Call for Talks Web page, which will be available soon.

Best Paper Awards

Cash prizes will be awarded to the best papers at the conference. Please see the USENIX proceedings library for Best Paper winners from previous years.

Best of the Rest Track

The USENIX Annual Technical Conference is the senior USENIX forum covering the full range of technical research in systems software. Over the past two decades, USENIX has added a range of more specialized conferences. ATC is proud of the content being published by its sibling USENIX conferences and will be bringing a track of encore presentations to ATC '17. This "Best of the Rest" track will allow attendees to sample the full range of systems software research in one forum, offering both novel ATC presentations and encore presentations from recent offerings of ATC's sibling conferences.



What to Submit

Authors are required to submit full papers by the paper submission deadline. *It is a hard deadline; no extensions will be given.* All submissions for USENIX ATC '17 will be electronic, in PDF format, via the Web submission form on the Call for Papers Web site, www.usenix.org/atc17/cfp.

USENIX ATC '17 will accept two types of papers:

Full papers: Submitted papers must be no longer than 11 single-spaced 8.5" x 11" pages, including figures and tables, but not including references. You may include any number of pages for references. Papers should be formatted in 2 columns, using 10-point type on 12-point leading, in a 6.5" x 9" text block. Figures and tables must be large enough to be legible when printed on 8.5" x 11" paper. Color may be used, but the paper should remain readable when printed in monochrome. The first page of the paper should include the paper title and author name(s); reviewing is single blind. Papers longer than 11 pages including appendices, *but excluding references,* or violating formatting specifications will not be reviewed. In a good paper, the authors will have:

- Addressed a significant problem
- Devised an interesting and practical solution or provided an important, independent, and experimental reproduction/refutation of prior solutions
- Clearly described what they have and have not implemented
- Demonstrated the benefits of their solution
- Articulated the advances beyond previous work
- Drawn appropriate conclusions

Short papers: Authors with a contribution for which a full paper is not appropriate may submit short papers of at most 5 pages, not including references, with the same formatting guidelines as full papers. You may include any number of pages for references. Examples of short paper contributions include:

- Original or unconventional ideas at a preliminary stage of development
- The presentation of interesting results that do not require a full-length paper, such as negative results or experimental validation
- Advocacy of a controversial position or fresh approach

For more details on the submission process and for templates to use with LaTeX and Word, authors should consult the detailed submission requirements linked from the Call for Papers Web site, www.usenix.org/atc17/cfp. Specific questions about submissions may be sent to atc17chairs@usenix.org.

By default, all papers will be made available online to registered attendees before the conference. If your accepted paper should not be published prior to the event, please notify production@usenix.org. In any case, the papers will be available online to everyone beginning on the first day of the conference, July 12, 2017.

Papers accompanied by nondisclosure agreement forms will not be considered. Accepted submissions will be treated as confidential prior to publication on the USENIX ATC '17 Web site; rejected submissions will be permanently treated as confidential.

Simultaneous submission of the same work to multiple venues, submission of previously published work, or plagiarism constitutes dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may take action against authors who have committed them. See the USENIX Conference Submissions Policy at www.usenix.org/conferences/submissions-policy for details.

Note that the above does not preclude the submission of a regular full paper that overlaps with a previous short paper or workshop paper. However, any submission that derives from an earlier paper must provide a significant new contribution (for example, by providing a more complete evaluation), and must explicitly mention the contributions of the submission over the earlier paper. If you have questions, contact your program co-chairs, atc17chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Authors will be notified of paper acceptance or rejection by April 24, 2017. Acceptance will typically be conditional, subject to shepherding by a program committee member.

Poster Session

The poster session is an excellent forum to discuss ideas and get useful feedback from the community. Posters and demos for the poster session will be selected from all the full paper and short paper submissions by the poster session chair. If you do not want your submissions to be considered for the poster session, please specify on the submission Web site.

Program and Registration Information

Complete program and registration information will be available in April 2017 on the conference Web site.

Rev. 10/27/16

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES