;login:

AUGUST 2013 VOL. 38, NO. 4

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Security

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

FOR A COMPLETE LIST OF USENIX AND USENIX CO-SPONSORED EVENTS, SEE WWW.USENIX.ORG/CONFERENCES

USENIX Security '13: 22nd USENIX Security Symposium
August 14–16, 2013, Washington, D.C., USA
www.usenix.org/conference/usenixsecurity13

Workshops Co-located with USENIX Security '13

EVT/WOTE '13: 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections
August 12–13, 2013
www.usenix.org/conference/evtwote13

USENIX Journal of Election Technology and Systems (JETS)
Published in conjunction with EVT/WOTE
www.usenix.org/jets

CSET '13: 6th Workshop on Cyber Security Experimentation and Test
August 12, 2013
www.usenix.org/conference/cset13

HealthTech '13: 2013 USENIX Workshop on Health Information Technologies
Safety, Security, Privacy, and Interoperability of Health Information Technologies
August 12, 2013
www.usenix.org/conference/healthtech13

LEET '13: 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats
August 12, 2013
www.usenix.org/conference/leet13

FOCI '13: 3rd USENIX Workshop on Free and Open Communications on the Internet
August 13, 2013
www.usenix.org/conference/foci13

HotSec '13: 2013 USENIX Summit on Hot Topics in Security
August 13, 2013
www.usenix.org/conference/hotsec13

WOOT '13: 7th USENIX Workshop on Offensive Technologies
August 13, 2013
www.usenix.org/conference/woot13

LISA '13: 27th Large Installation System Administration Conference
November 3–8, 2013, Washington, D.C., USA
www.usenix.org/conference/lisa13

SESA '13: 2013 USENIX Summit for Educators in System Administration
CO-LOCATED WITH LISA '13
November 5, 2013, Washington, D.C., USA
www.usenix.org/conference/sesa13

FAST '14: 12th USENIX Conference on File and Storage Technologies
February 17–20, 2014, Santa Clara, CA, USA
www.usenix.org/conference/fast14
Submissions due: September 26, 2013

2014 USENIX Research in Linux File and Storage Technologies Summit
CO-LOCATED WITH FAST '14
February 17, 2014, Santa Clara, CA, USA

NSDI '14: 11th USENIX Symposium on Network Systems Design and Implementation
April 2–4, 2014, Seattle, WA, USA
www.usenix.org/conference/nsdi14
Abstracts due: September 20, 2013

2014 USENIX Federated Conferences Week
June 17–20, 2014, Philadelphia, PA, USA

USENIX ATC '14: 2014 USENIX Annual Technical Conference

HotCloud '14: 6th USENIX Workshop on Hot Topics in Cloud Computing

WiAC '14: 2014 USENIX Women in Advanced Computing Summit

HotStorage '14: 6th USENIX Workshop on Hot Topics in Storage and File Systems

UCMS '14: 2014 USENIX Configuration Management Summit

USENIX Security '14: 23rd USENIX Security Symposium
August 20–22, 2014, San Diego, CA, USA

OSDI '14: 11th USENIX Symposium on Operating Systems Design and Implementation
October 6–8, 2014, Broomfield, CO, USA

Diversity '14: 2014 Workshop on Diversity in Systems Research
CO-LOCATED WITH OSDI '14
October 4, 2014, Broomfield, CO, USA

LISA '14: 28th Large Installation System Administration Conference
November 9–14, 2014, Seattle, WA, USA

Stay Connected...

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

;login:

AUGUST 2013 VOL. 38, NO. 4

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Rikki Endsley
rikki@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Arnold Gatilao
Casey Henderson
Michele Nelson

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 bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

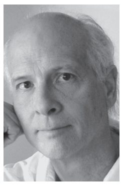
$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for non-members are $90 per year. Periodicals postage paid at Berkeley, CA, and additional offices.

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

There are dark clouds on the horizon, and an ominous deep rumble shaking the windows. What we have carefully ignored for so many years may soon be destroying all we hold dear. Of course, I am not writing about the weather…

On May 27, a US Department of Defense committee reported [1]:

> …that the cyber threat is serious and that the United States cannot be confident that our critical Information Technology (IT) systems will work under attack from a sophisticated and well-resourced opponent utilizing cyber capabilities in combination with all of their military and intelligence capabilities (a "full spectrum" adversary).

Most news reports focused on the loss of technology to the Chinese [2], but this was not the top finding of this committee. Their point was that US defense capabilities rely on networks of systems, and these systems are not, and cannot easily be, made secure.

Security Debt

In this issue, Dan Geer and Chris Wysopal write about security debt. Their concept is that all software has flaws when released, and as software is patched and upgraded over time, the number of flaws tends to grow. These flaws can be thought of as a debt that needs to be repaid over time, as the software matures.

Security debt refers specifically to those flaws that will be exploited maliciously. If you look at the graphs in Figure 2 on page 63, you can see that five of the most popular applications have security debt that is building over time, with the sum of these debts appearing to approach exponential growth.

Bilge and Dumitras, using a database of binary signatures collected from millions of Windows systems, show that beyond the visible debt of disclosed vulnerabilities, there is also the undisclosed threat of zero-days. Zero-days are vulnerabilities that are actively being exploited but have not been announced, and the average amount of time that zero-days were exploited as reported in this research is astounding. And keep in mind that the technique they used to detect binary exploits is prone to false negatives, so the numbers are likely much worse.

Some of my friends in the security business, as well as the US Congress, had been talking about legal measures that might be used to improve the state of security. Mike Scher, a friend, security geek, and a lawyer, agreed to write an article accessible to non-lawyers about the current state of software liability (page 26). My take based on Scher's cogent analysis is that we had better work on other approaches. Radical approaches.

Arrakis

Simon Peter and Tom Anderson write about Arrakis, an operating system designed to provide application stacks with direct access to devices designed to be multiplexed among VMs. But Arrakis isn't a hypervisor, and the application stacks are not the same as VMs. Instead, Arrakis sets up containers for application stacks as well as arranging access to devices, such as NICs that have multiple queues that can be allocated to specific applications. Peter expects that we will soon see block devices, first SSDs and later hard disks, that work similarly.

You might wonder what Arrakis has to do with security, but Arrakis reduces the software stack between applications and devices. Arrakis also makes use of features such as extended page tables so that different parts of an application can share the same address space, but also be better isolated: for example, Web browser tabs. Arrakis also puts applications in charge of sharing information that they own, as they both control and manage that information.

Jon Howell and fellow researchers have been taking a different approach. Although Jon wasn't able to write for this issue, he has plans to do so in the future. In the meantime, you may want to look at "Embassies," his NSDI paper [3] (summaries in this issue and online) and "How to Run POSIX Apps in a Minimal Picoprocess" [4], his ATC '13 paper. Jon has taken the work that started with Xax [5], designed to run desktop applications within a Web browser, and taken this notion much further. In Xax, libraries and applications required extensive rewriting before they could be run securely. In their latest work, Jon and his co-authors have created POSIX libraries and an emulator that allows desktop applications, such as the Gimp, Inkscape, and Abiword, to run within a picoprocess that requires only minimal support from an operating system.

This work takes away the ability of applications to make most system calls from libc (which does this for most applications), replacing many calls with emulations or simply stubs. They emulate most operating system services and rely on only a few actual system calls: allocating memory, network communications, getting time, and the ability to copy bits to a framebuffer and receive input events. This tremendously narrows the interface to the operating system and most system resources. For example, there is no direct access to the file system. This system relies on reading or writing files over the network interface. In Embassies, a picoprocess running a Web browser tab is not given full network access, but is only allowed access to the origin of the tab. These limitations, although imposed by software, do a lot to limit the access of exploited or malicious applications to both the local system and other systems.

I really like the idea behind this research: that applications can be run with minimal operating system support, in isolation. Instead of the 330 plus Linux system calls, only a handful are needed. Instead of running a LAMP stack within a VM on top of a hypervisor, you just run LAMP in a container, as Peter describes in his Arrakis article.

The Lineup

Of course, there is a lot more in this issue than what I've already mentioned. Abe Singer and Warren Anderson have spent years researching both the usability of passwords and methods for measuring the entropy of passwords that can be created when limited by password policies. Abe and Warren (page 14) take a close look at how current password policies actually make security worse, and suggest ways to improve password security. Along the way, they examine the myths that have lead to the password polices most commonly used today.

I interviewed Bill Cheswick, focusing on his work in security. Ches was already running a firewall for AT&T when the Morris worm appeared (November 1988), making his the earliest known firewall. He later went on to co-author the first book about firewalls, as well as found a company to map out intranets. Getting back to the concerns of the US DoD that I began this article with, I imagine that they could really use some of the technology that Ches helped develop for spotting "leaks" in networks.

David Lang begins a series about logging with an article about setting up logging for enterprises. Even if you don't have multiple datacenters, David presents a logging architecture that makes sense for organizations that have more than a handful of systems to monitor. I like his tiered system approach and the way it separates groups of systems by task, allowing upgrading or changes to the logging infrastructure without having to rebuild from scratch.

I met Bin Fan during the first NSDI '13 poster session. Fan was enthusiastically explaining cuckoo filters, which he used in his accepted paper at NSDI. Cuckoo filters have properties that make them more suitable than the more familiar Bloom filters for applications that require the ability to delete items in a set and need a low probability of false positives. Fan and his colleagues explain the advantages of cuckoo filters, as well as sharing the software testbed he used to generate his data.

Wyatt Lloyd and co-authors explain causal consistency in their article. Wyatt had presented Eiger, a noSQL database, at NSDI '13, and I was interested in having the authors tell us more about why they think that causal consistency might be better than some of the many other consistency models for distributed and replicated systems.

David Blank-Edelman, taking a page from Ruby, suggests that readers of his column "git smart." David explores various Perl modules that may increase your enjoyment of the Git source code management tool, through graphs, reports, and managing multiple Git repositories.

Dave Beazley explains how a bug that appeared in Python 3.3, in the Python package he supports, the PLY parser generator, came from a security-related change in the Python dictionary implementation. Dave explains how dictionaries work under the covers, and suggests ways to reduce the size of data structures that either use dictionaries or optionally use dictionaries. The bottom line is that if you use Python dictionaries, objects, or even modules, you will want to read his column.

Dave Josephsen has been playing with inotify(7), the group of Linux system calls used for monitoring file system events. Whereas you might have used dnotify in the past, Dave explains why you want to use inotify instead, and provides an online example C program to show you how.

I've already mentioned Dan Geer and Chris Wysopal's For Good Measure column about the danger of accumulating software debt.

Robert Ferrell has written about a number of topics this time, including the modern job interview, loadable cranial modules, tiny devices, and software updating. As always, Robert is looking toward a future that is both more secure and more sane.

Elizabeth Zwicky has written five book reviews this month. She begins with a book on building real systems that deals with the interface between marketing and programmers. Speaking of real, she then reviews an insightful collection of essays that deal with "bad data," i.e., data analysis in the real world. Next, she looks at a book that promises Data Insights but doesn't really deliver. She then turns her eye on Generation Blend, and finishes with a book on nighttime digital photography.

Mark Lamourine starts out with a book called *Hacker's Delight,* which is for hackers in the original sense of the word. He then takes a look at a book on Dart, a JavaScript replacement, and finishes with a book on Arduino-based distributed network sensors.

I reviewed Mike Lucas' second edition of *Absolute OpenBSD,* a good book to have if you plan on trying OpenBSD. BSDs in general are quite different from the Linux systems most people are familiar with, and Mike's detailed presentation really helps bridge that gap.

This issue also includes summaries of NSDI '13 presentations.

For many years, I've been writing about how poorly our current architectures function when it comes to building secure systems. There are many good reasons for this, the top ones being ease of programming and ease of use. Until we have system architectures that provide security by default, and that support much of the software and programming environments that people are already familiar with, we will continue to use insecure systems. I've mentioned some ideas in this column that I believe will lead us closer to the goal of secure systems, but right now, the storm is almost upon us.

References

[1] DoD Defense Science Board, "Resilient Military Systems and the Advanced Cyber Threat": http://www.acq.osd.mil/dsb/reports/ResilientMilitarySystems.CyberThreat.pdf.

[2] "Confidential Report Lists U.S. Weapons System Designs Compromised by Chinese Cyberspies," *Washington Post,* May 27, 2013: http://articles.washingtonpost.com/2013-05-27/world/39554997_1_u-s-missile-defenses-weapons-combat-aircraft.

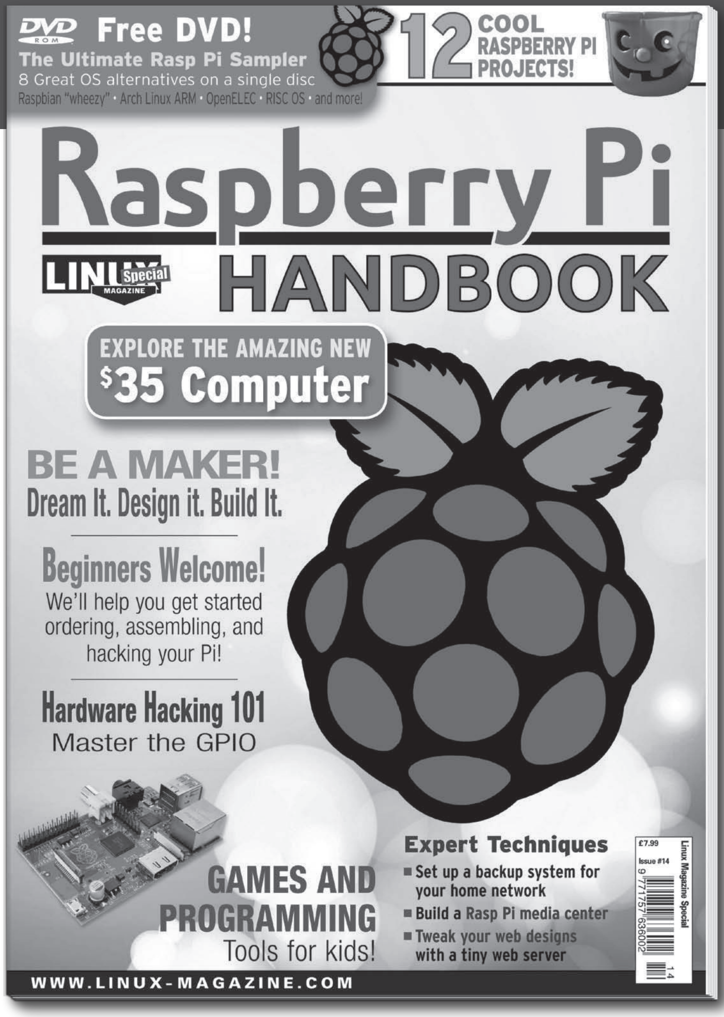
[3] J. Howell, B. Parno, and J.R. Douceur, "Embassies: Radically Refactoring the Web," NSDI '13: https://www.usenix.org/conference/nsdi13/embassies-radically-refactoring-web.

[4] J. Howell, B. Parno, and J.R. Douceur, "How to Run POSIX Apps in a Minimal Picoprocess," ATC '13: https://www.usenix.org/conference/atc13/how-run-posix-apps-minimal-picoprocess.

[5] J. Howell, J.R. Douceur, J. Elson, and J.R. Lorch, "Leveraging Legacy Code to Deploy Desktop Applications on the Web," OSDI '08: http://static.usenix.org/events/osdi08/tech/full_papers/douceur/douceur.pdf.

Shop the Shop shop.linuxnewmedia.com

Raspberry Pi HANDBOOK


Free DVD!
The Ultimate Rasp Pi Sampler
8 Great OS alternatives on a single disc
Raspbian "wheezy" · Arch Linux ARM · OpenELEC · RISC OS · and more!
12 COOL RASPBERRY PI PROJECTS!
Raspberry Pi HANDBOOK
LINUX MAGAZINE Special
EXPLORE THE AMAZING NEW $35 Computer
BE A MAKER!
Dream It. Design it. Build It.
Beginners Welcome!
We'll help you get started ordering, assembling, and hacking your Pi!
Hardware Hacking 101
Master the GPIO
GAMES AND PROGRAMMING
Tools for kids!
Expert Techniques
Set up a backup system for your home network
Build a Rasp Pi media center
Tweak your web designs with a tiny web server
WWW.LINUX-MAGAZINE.COM

Your companion for a strange and wonderful adventure...

You ordered your Raspberry Pi... You got it to boot...what now?

The Raspberry Pi Handbook takes you through an inspiring collection of projects. Put your Pi to work as a:

- media center
- photo server
- game server
- hardware controller
- and much more!

Discover Raspberry Pi's special tools for teaching kids about programming and electronics, and explore advanced techniques for controlling Arduino systems and coding GPIO interrupts.

WATCH YOUR NEWSSTANDS FOR THE ONLY RASPBERRY PI REFERENCE YOU'LL EVER NEED!

RASPBERRY PI ON NEWSSTANDS NOW OR ORDER ONLINE AT:

shop.linuxnewmedia.com/rpi

SECURITY

Investigating Zero-Day Attacks

LEYLA BILGE AND TUDOR DUMITRAS

Leyla Bilge became a Senior Research Engineer at Symantec Research Labs in February 2012 after obtaining her Ph.D. from EURECOM, based in the south of France, with work focusing on network-based botnet detection. In her thesis, she proposed three different network-based botnet detection schemes, one of which is Exposure.
Leylya_Yumer@symantec.com

Tudor Dumitras is an Assistant Professor in the Electrical and Computer Engineering Department at the University of Maryland, College Park. His research focuses on Big Data approaches to problems in system security and dependability. In his previous role at Symantec Research Labs he built the Worldwide Intelligence Network Environment (WINE). He has received the 2011 A. G. Jordan Award, from the ECE Department at Carnegie Mellon University, for an outstanding Ph.D. thesis and for service to the community; the 2009 John Vlissides Award, from ACM SIGPLAN, for showing significant promise in applied software research; and the Best Paper Award at ASP-DAC '03.
tudor.dumitras@gmail.com

We conducted a systematic study on data available through Symantec's Worldwide Intelligence Network Environment to help us to understand the duration and prevalence of zero-day attacks. Based on what we learned, we developed a methodology that automatically identifies zero-day attacks that have affected a large number of real hosts worldwide. Our methodology was not only able to detect already known zero-day attacks but also some that were previously unknown. Moreover, we discovered that the majority of zero-day attacks were able to stay undercover for a surprisingly long time.

A *zero-day attack* is a cyberattack exploiting a vulnerability that has not been disclosed publicly. There is almost no defense against a zero-day attack: while the vulnerability remains unknown, the software affected cannot be patched and antivirus products cannot detect the attack through signature-based scanning. For cybercriminals, unpatched vulnerabilities in popular software, such as Microsoft Office or Adobe Flash, represent a free pass to any target they might want to attack, from Fortune 500 companies to millions of consumer PCs around the world. For this reason, the market value of a new vulnerability ranges between $5,000 and $500,000 [7]. Examples of notable zero-day attacks include the 2010 Hydraq trojan, also known as the "Aurora" attack, which aimed to steal information from several companies; the 2010 Stuxnet worm, which combined four zero-day vulnerabilities to target industrial control systems; and the 2011 attack against RSA. Unfortunately, very little is known about zero-day attacks because, in general, data is not available until *after* the attacks are discovered. Prior studies rely on indirect measurements (e.g., analyzing patches and exploits) or the post-mortem analysis of isolated incidents, and they do not shed light on the duration, prevalence, and characteristics of zero-day attacks.

We conducted a systematic study of zero-day attacks from 2008 to 2011 and developed a technique for identifying and analyzing zero-day attacks from the data available through the Worldwide Intelligence Network Environment (WINE), a platform for data-intensive experiments in cybersecurity [8]. WINE includes field data collected by Symantec on 11 million hosts around the world. These hosts do not represent honeypots or machines in an artificial lab environment; they are real computers that are targeted by cyberattacks. For example, the *binary reputation* data set includes information on binary executables downloaded by users who opt in for Symantec's reputation-based security program, which assigns a reputation score to binaries that are not known to be either benign or malicious. The *antivirus telemetry* data set includes reports about host-based threats (e.g., viruses, worms, trojans) detected by Symantec's antivirus products.

The key idea behind our technique is to identify executable files that are linked to exploits of known vulnerabilities. We start from the public information about disclosed vulnerabilities (i.e., vulnerabilities that have been assigned a CVE identifier), available from vulnerability databases and vendor advisories. We use the public Threat Explorer Web site to determine threats identified by Symantec that are linked to these vulnerabilities, and then we query the antivirus telemetry data set in WINE for the hashes of all the distinct files (malware variants) that are detected by these signatures. Finally, we search the history of binary reputation submissions for these malicious files, which allows us to estimate *when* and *where* they appeared on the Internet.

Using this method, we identified and analyzed 18 vulnerabilities exploited in the real world before disclosure. Our findings include the following:

- Out of these 18 zero-day vulnerabilities, 11 were not previously known to have been employed in zero-day attacks, which suggests that zero-day attacks are more common than previously thought.
- A typical zero-day attack lasts 312 days on average and hits multiple targets around the world; however, some of these attacks remain unknown for up to 2.5 years.
- After these vulnerabilities are disclosed, the volume of attacks exploiting them increases by up to five orders of magnitude.

These findings have important technology and policy implications. The challenges for identifying and analyzing elusive threats, such as zero-day attacks, emphasize that experiments and empirical studies in cybersecurity must be conducted at scale by taking advantage of the resources that are available for this purpose, such as the WINE platform. This will allow researchers and practitioners to investigate mitigation techniques for these threats based on empirical data rather than on anecdotes and back-of-the-envelope calculations. For example, the fact that zero-day attacks are rare events, but that the new exploits are reused for multiple targeted attacks, suggests that techniques for assigning reputation based on the prevalence of files [3] can reduce the effectiveness of the exploit. Furthermore, because we quantify the increase in the volume of attacks after vulnerability disclosures, we provide new data for assessing the overall benefit to society of the *full disclosure* policy, which calls for disclosing new vulnerabilities publicly, even if patches are not yet available.

What Is a Zero-Day Attack?

In general, a zero-day attack is an attack that exploits vulnerabilities not yet disclosed to the public. The origins of this term are unclear. Accounts of events from World War II often use "zero day " when referring to the date set for a planned attack or the day when the attack actually occurred:

> October 20 [1943] was fixed as zero day for [V2] rocket attacks [on London] to begin.
>
> —Winston Churchill, *Closing the Ring,* 1951.

In the computer world, the term is used in the warez community when referring to any copyrighted work (e.g., software, movies, music albums) that is cracked, copied, and re-released on the same day as the official release. Additionally, naming a folder

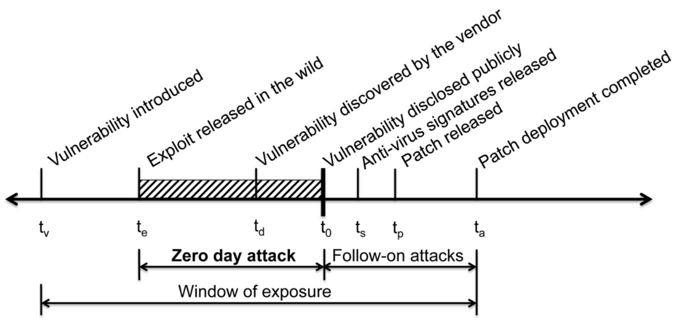

Vulnerability introduced
Exploit released in the wild
Vulnerability discovered by the vendor
Vulnerability disclosed publicly
Anti-virus signatures released
Patch released
Patch deployment completed
t_v
t_e
t_d
t_0
t_s
t_p
t_a
Zero day attack
Follow-on attacks
Window of exposure

Figure 1: Attack timeline. These events do not always occur in this order, but $t_a > t_p \geq t_d > t_v$ and $t_0 \geq t_d$. The relation between t_d and t_e cannot be determined in most cases. For a zero-day attack, $t_0 > t_e$.

"0day" placed it at the top of the list on a file sharing server as an attempt to draw attention to the item, because zero-day warez is usually sought after by downloaders.

When the "zero-day" qualifier was first applied to software vulnerabilities and what the original meaning was is unclear. Today, a "zero-day attack" usually is understood to be a cyberattack that exploits a vulnerability before the vulnerability's public disclosure date (rather than on the same day as the disclosure). These exploits correspond to vulnerabilities that the security community is generally unaware of, and such vulnerabilities are called "zero-day vulnerabilities." This is illustrated in the vulnerability timeline from Figure 1: a vulnerability is created when an exploitable programming bug is introduced in a popular software product (t_v), the vulnerability is then discovered by attackers and is exploited in the wild for conducting stealthy attacks (t_e), the vulnerability is discovered by the vendor (t_d) and disclosed to the public (t_0), leading to the release of countermeasures such as antivirus signatures for the exploit (t_s) and patches for the vulnerability (t_p). The vulnerability ceases to present a threat only when the patch deployment is completed (t_a). The disclosure dates (t_0) of vulnerabilities are tracked and recorded in several public databases, such as Common Vulnerabilities and Exposures (CVE). A zero-day attack is characterized by a vulnerability that is exploited in the wild before it is disclosed, i.e., $t_0 > t_e$. Our goals are to estimate te for, to measure the prevalence and duration of zero-day attacks, and to compare the impact of zero-day vulnerabilities before and after t_0.

Software vendors fix bugs and patch vulnerabilities in all their product releases, and as a result some vulnerabilities are never exploited or disclosed. We only consider vulnerabilities that have a CVE identifier. Similarly, in some cases vendors learn about a vulnerability before it is exploited, but consider it low priority, and cybercriminals may also delay the release of exploits until they identify a suitable target, to prevent the discovery of the vulnerability. Although the CVE database sometimes indicates when vulnerabilities were reported to the vendors, generally, determining the exact date when the vendor or the cybercriminals discovered the vulnerability or even which discovery came first is impossible. Moreover, some exploits are not employed for malicious activities before the disclosure date and are disseminated as proofs-of-concept, to help the software vendor understand the vulnerability and the antivirus vendors to update their signatures. We therefore focus on exploits that have been used in real-world attacks before the disclosure of the corresponding vulnerabilities.

What Do We Know About Zero-Day Attacks?

Most prior work has focused on the entire window of exposure to a vulnerability (see Figure 1), first defined by Schneier [9]. Arbaugh et al. evaluated the number of intrusions observed during each phase of the vulnerability lifecycle and showed that a significant number of vulnerabilities continue to be exploited even after patches become available [1]. Frei compared how fast Microsoft and Apple react to newly disclosed vulnerabilities and, although significant differences exist between the two vendors, both have some vulnerabilities with no patch available 180 days after disclosure [4]. A Secunia study showed that 50% of Windows users were exposed to 297 vulnerabilities in a year and that patches for only 65% of these vulnerabilities were available at the time of their public disclosure [5].

Although the market for zero-day vulnerabilities has not been studied as thoroughly as other aspects of the underground economy, the development of exploits for such vulnerabilities is certainly a profitable activity. For example, several security firms run programs, such as HP's Zero Day Initiative and Verisign's iDefense Vulnerability Contributor Program, that pay developers up to $10,000 for their exploits [7], with the purpose of developing intrusion-protection filters against these exploits. Between 2000 and 2007, 10% of vulnerabilities were disclosed through these programs [4]. Similarly, software vendors often reward the discovery of new vulnerabilities in their products, offering prizes up to $60,000 for exploits against targets that are difficult to attack, such as Google's Chrome browser [6]. Moreover, certain firms and developers specialize in selling exploits to confidential clients on the secretive, but legal, market for zero-day vulnerabilities. Sources from the intelligence community suggest that the market value of such vulnerabilities can reach $500,000 [7]. In particular, the price of exploits against popular platforms, such as Windows, iOS, or the major Web browsers, may exceed $100,000, depending on the complexity of the exploit and on how long the vulnerability remains undisclosed.

Identifying Zero-Day Attacks Automatically

To identify zero-day attacks automatically, we analyzed the historical information provided by multiple data sets. We conducted our study on the Worldwide Intelligence Network Environment (WINE), a platform for data-intensive experiments in cybersecurity [8]. WINE was developed at Symantec Research Labs for sharing comprehensive field data with the research community.

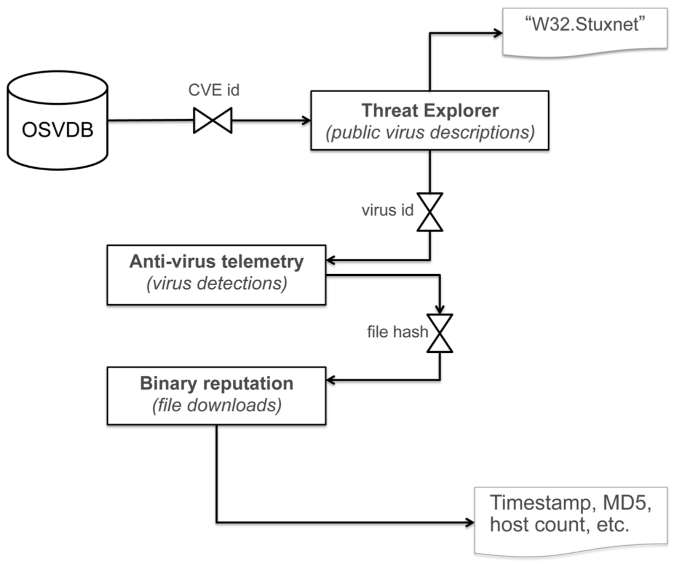


Figure 2: Overview of our method for identifying zero-day attacks systematically

WINE samples and aggregates multiple terabyte-size data sets, which Symantec uses in its day-to-day operations, with the aim of supporting open-ended experiments at scale. The data included in WINE is collected on a representative subset of the hosts running Symantec products, such as Norton Antivirus. These hosts do not represent honeypots or machines in an artificial lab environment; they are real computers, in active use around the world, that are targeted by cyberattacks. WINE also enables the reproduction of prior experimental results by archiving the reference data sets that researchers use and by recording information on the data collection process and on the experimental procedures employed.

We analyzed two WINE data sets: *antivirus telemetry* and *binary reputation*. The antivirus telemetry data records detections of known threats for which Symantec generated a signature that was subsequently deployed in an antivirus product. The antivirus telemetry data was collected between December 2009 and August 2011, and it includes 225 million detections that occurred on 9 million hosts. The binary reputation data reports all the binary executables—whether benign or malicious—that have been downloaded on end-hosts around the world. The binary reputation data has been collected since February 2008, and it includes 32 billion reports about approximately 300 million distinct files, which were downloaded on 11 million hosts. These files may include malicious binaries that were not detected at the time of their download because the threat was unknown. We note that this data is collected only from the Symantec customers who gave their consent to share it.

We correlated the WINE data sets with information from two additional sources: the Open Source Vulnerability Database

(OSVDB) and Symantec's Threat Explorer. OSVDB is a public vulnerability database, similar to CVE, providing information on the discovery, disclosure, and exploit release date of the vulnerabilities. Threat Explorer is a public Web site with historical information about most threats for which Symantec has generated antivirus signatures—including signatures for exploits of vulnerabilities with known CVE identifiers. Because the Microsoft Windows platform has been the main target for cyberattacks over the past decade, we focus on vulnerabilities in Windows or in software developed for Windows.

Analysis Method

Figure 2 illustrates our four-step analysis method. We start from the known vulnerabilities recorded in OSVDB, and we search Symantec's Threat Explorer for the CVE numbers of these vulnerabilities in order to identify the names of the viruses or worms that exploit them. We manually filter out the generic virus detections (e.g., "Trojan horse") listed on Threat Explorer, to compile a list of threat names that identify vulnerability exploits. We then search for these threat names in the antivirus telemetry, and we record the MD5 and SHA2 hashes of the exploits detected in the field. Having identified which executables exploit known CVE vulnerabilities, we search for each executable in the binary reputation data to estimate when they first appeared on the Internet. If at least one of these executables was downloaded before the disclosure date of the corresponding vulnerability, we conclude that we have identified a zero-day attack. More information about this analysis method is available in the conference version of this article [2].

Threats to Validity

As WINE does not include data from hosts without Symantec's antivirus products, our results may not be representative of the general population of platforms in the world. Although we cannot rule out the possibility of selection bias, the large size of the population in our study (11 million hosts and 300 million files) and the number of zero-day vulnerabilities we identified using our automated method (18, which is on the same order of magnitude as the 43 reported by Symantec analysts during the same period) suggest that our results have a broad applicability. Moreover, for the zero-day vulnerabilities detected toward the beginning of our data collection period, we may underestimate the duration of the attacks. We therefore caution the reader that our results for the duration of zero-day attack are best interpreted as *lower bounds.*

Analysis Results and Findings

Using this method, we identified 18 zero-day vulnerabilities: 3 disclosed in 2008, 7 in 2009, 6 in 2010, and 2 in 2011. From the annual vulnerability trends reports produced by Symantec and the SANS Institute, as well as blog posts on the topic of zero-day

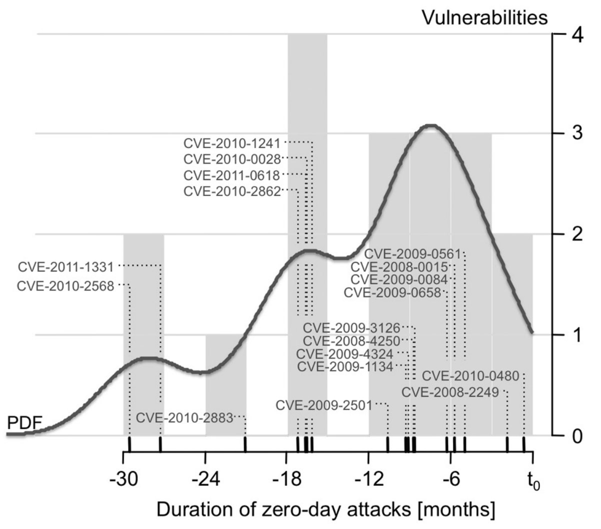


Figure 3: Duration of zero-day attacks. The histograms group attack durations in three-month increments, before disclosure, and the red rug indicates the attack duration for each zero-day vulnerability.

vulnerabilities, we found out that seven of our vulnerabilities are generally accepted to be zero-day vulnerabilities (see Table 1). For example, CVE-2010-2568 is one of the four zero-day vulnerabilities exploited by Stuxnet, and it is known to have also been employed by another threat for more than two years before the disclosure date (July 17, 2010). As shown in Table 1, most of these vulnerabilities affected Microsoft and Adobe products. More information about the zero-day vulnerabilities identified is available in [2].

Figure 3 illustrates the duration of zero-day attacks and their distribution: they lasted between 19 days (CVE-2010-0480) and 30 months (CVE-2010-2568), and the average duration of a zero-day attack was 312 days. Figure 3 illustrates this distribution. Fifteen of the zero-day vulnerabilities targeted fewer than 1,000 hosts, out of the 11 million hosts in our data set. On the other hand, three vulnerabilities were employed in attacks that infected thousands or even millions of Internet users. For example, Conficker exploiting the vulnerability CVE-2008-4250 managed to infect approximately 370,000 machines without being detected for more than two months. This example illustrates the effectiveness of zero-day vulnerabilities for conducting stealth cyberattacks.

Zero-Day Vulnerabilities After Disclosure

We also analyzed the increase in the number of malware variants exploiting these vulnerabilities over time. Figure 4 shows that, after the vulnerability disclosure, 183–85,000 more variants are recorded each day. One reason for observing the large number of new different files that exploit the zero-day vulnerabilities might be that they are repacked versions of the same exploits; however, it is doubtful that repacking alone can account

0-day vulnerability	Unknown	Description
CVE-2008-0015		Microsoft ATL Remote Code Execution Vulnerability (RCEV)
CVE-2008-2249	Yes	Microsoft Windows GDI WMF Integer Overflow Vulnerability
CVE-2008-4250	Yes	Windows Server Service NetPathCanonicalize() Vulnerability
CVE-2009-0084	Yes	Microsoft DirectX DirectShow MJPEG Video Decompression RCEV
CVE-2009-0561	Yes	Microsoft Excel Malformed Record Object Integer Overflow
CVE-2009-0658		Adobe Acrobat and Reader PDF File Handling JBIG2 Image RCEV
CVE-2009-1134	Yes	Microsoft Office Excel QSIR Record Pointer Corruption Vulnerability
CVE-2009-2501		Microsoft GDI+ PNG File Processing RCEV
CVE-2009-3126	Yes	Microsoft GDI+ PNG File Integer Overflow RCEV
CVE-2009-4324		Adobe Reader and Acrobat newplayer() JavaScript Method RCEV
CVE-2010-0028	Yes	Microsoft Paint JPEG Image Processing Integer Overflow
CVE-2010-0480	Yes	Microsoft Windows MPEG Layer-3 Audio Decoder Buffer Overflow Vulnerability
CVE-2010-1241	Yes	NITRO Web Gallery 'PictureId' Parameter SQL Injection Vulnerability
CVE-2010-2568		Microsoft Windows Shortcut 'LNK/PIF' Files Automatic File Execution Vulnerability
CVE-2010-2862	Yes	Adobe Acrobat and Reader Font Parsing RCEV
CVE-2010-2883		Adobe Reader 'CoolType.dll' TTF Font RCEV
CVE-2011-0618	Yes	Adobe Flash Player ActionScript VM Remote Integer Overflow Vulnerability
CVE-2011-1331		JustSystems Ichitaro Remote Heap Buffer Overflow Vulnerability

Table 1: New zero-day vulnerabilities discovered and their descriptions

for an increase by up to five orders of magnitude. More likely, this increase is the result of the extensive reuse of field-proven exploits in other malware.

Figure 5 shows the time elapsed until all the vulnerabilities disclosed between 2008 and 2011 started being exploited in the wild. Exploits for 42% of these vulnerabilities appear in the field data within 30 days after the disclosure date. This illustrates that the cybercriminals watch closely the disclosure of new vulnerabilities, in order to start exploiting them, which causes a significant risk for end-users.

Other Zero-Day Vulnerabilities

Every year, Symantec analysts prepare an "Internet Security Threats Report" (ISTR) in which new threats, vulnerabilities, and malware trends are reported. This report includes information about the zero-day vulnerabilities identified during the previous year. These reports identify 43 between 2008 and 2011: 9 in 2008, 12 in 2009, 14 in 2010, and 8 in 2011. For each year, our automated method discovers on average three zero-day vulnerabilities that were not known before and on average two zero-day vulnerabilities from the list reported by Symantec; however, we were not able to identify on average eight known zero-day vulnerabilities per year; these vulnerabilities are linked to Web attacks, polymorphic malware, non-executable exploits, or targeted attacks [2], which illustrates the current limitations of our method and suggests interesting directions for future research.

Discussion

Zero-day attacks are difficult to prevent because they exploit unknown vulnerabilities, for which there are no patches and no antivirus or intrusion-detection signatures. As long as software will have bugs and the development of exploits for new vulnerabilities will be a profitable activity, we will be exposed to zero-day attacks, it seems. In fact, 60% of the zero-day vulnerabilities we identify in our study were not known before, which suggests that there are many more zero-day attacks than previously thought—perhaps more than twice as many; however, reputation-based technologies, which assign a score to each file based on its prevalence in the wild and on a number of other inputs [3], single out rare events such as zero-day attacks and can reduce the effectiveness of the exploits.

The large fraction of new zero-day vulnerabilities we identify also emphasizes that zero-day attacks are difficult to detect through manual analysis, given the current volume of cyberattacks. Automated methods for finding zero-day attacks in field data, such as the method we propose in this paper, facilitate

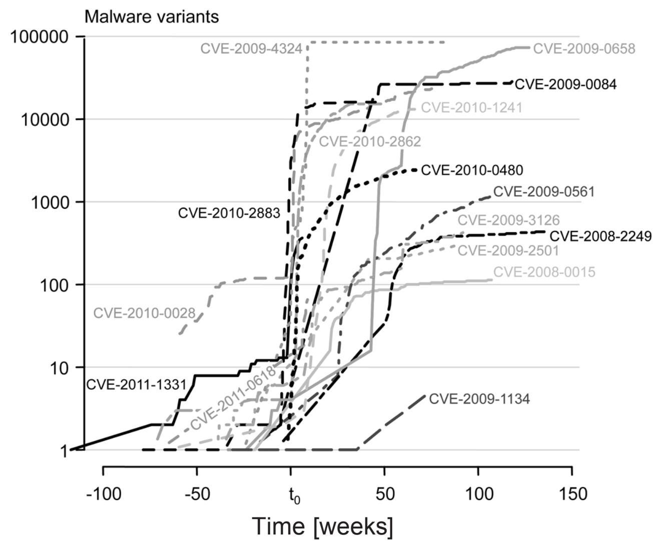


Figure 4: Increase in the number of malware variants exploiting zero-day vulnerabilities after they are disclosed (at time = t_0)

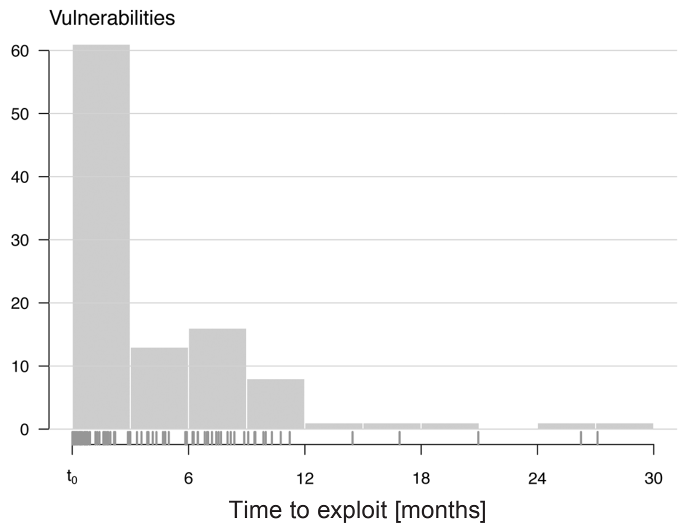


Figure 5: Time before vulnerabilities disclosed between 2008–2011 started being exploited in the field. The histograms group the exploitation lag in three-month increments, after disclosure, and the red rug indicates the lag for each exploited vulnerability. The zero-day attacks are excluded from this figure.

the systematic study of these threats. For example, our method allows us to measure the duration of zero-day attacks (Figure 3). While the average duration is approximately 10 months, the fact that all but one of the vulnerabilities disclosed after 2010 remained unknown for more than 16 months suggests that we may be underestimating the duration of zero-day attacks, as the data we analyze goes back only to February 2008. In the future, such automated techniques will allow analysts to detect zero-day attacks faster, e.g., when a new exploit is reused in multiple targeted attacks; however, this will require establishing mechanisms for organizations to share information about suspected targeted attacks with the security community.

Our findings also provide new data for the debate on the benefits of the full disclosure policy. This policy is based on the premise that disclosing vulnerabilities to the public, rather than to the vendor, is the best way to fix them because this provides an incentive for vendors to patch faster, rather than to rely on security-through-obscurity [9]. This debate is ongoing, but most participants agree that disclosing vulnerabilities causes an increase in the volume of attacks. Indeed, this is what the supporters of full disclosure are counting on, to provide a meaningful incentive for patching; however, the participants to the debate disagree about whether trading off a high volume of attacks for faster patching provides an overall benefit to the society.

The root cause of these disagreements lies in the difficulty of quantifying the real-world impact of vulnerability disclosures and of patch releases without analyzing comprehensive field data. We took a first step toward this goal by showing that the disclosure of zero-day vulnerabilities causes a significant risk for end-users, as the volume of attacks increases by up to five orders of magnitude; however, vendors prioritize which vulnerabilities they patch, giving more urgency to vulnerabilities that are disclosed or about to be disclosed. For example, 80% of the vulnerabilities in 2007 were discovered more than 30 days before the disclosure date [4]. At the same time, anecdotal evidence suggests that attackers also adapt their strategies to the expected disclosure of zero-day vulnerabilities. Because early disclosure reduces the value of zero-day vulnerabilities, the fees for new exploits are sometimes paid in installments, with each subsequent payment depending on the lack of a patch [7]. Additional research is needed for quantifying these aspects of the full disclosure tradeoff, e.g., by measuring how quickly vulnerable hosts are patched in the field, following vulnerability disclosures. Like our study of zero-day attacks, answering these additional research questions will require empirical studies conducted at scale, using comprehensive field data.

Conclusion

Zero-day attacks have been discussed for decades, but no study has yet measured the duration and prevalence of these attacks in the real world *before the disclosure of the corresponding vulnerabilities.* We take a first step in this direction by analyzing field data collected on 11 million Windows hosts over a period of four years. The key idea in our study is to identify executable files that are linked to exploits of known vulnerabilities. By searching for these files in a data set with historical records of files downloaded on end-hosts around the world, we systematically identify zero-day attacks and we analyze their evolution in time.

We identified 18 vulnerabilities exploited in the wild before their disclosure, of which 11were not previously known to have been employed in zero-day attacks. Zero-day attacks last on average 312 days, and up to 30 months, and they typically affect few hosts; however, there are some exceptions for high profile attacks, such as Conficker and Stuxnet, which we respectively detected on hundreds of thousands and millions of the hosts in our study, before the vulnerability disclosure. After the disclosure of zero-day vulnerabilities, the volume of attacks exploiting them increases by up to five orders of magnitude. These findings have important implications for future security technologies and for public policy.

Acknowledgments
We thank Jonathan McCune and Michael Hicks for stimulating discussions on the topic of zero-day attacks. We also thank Marc Dacier for his early feedback on our results, and Sam Perl from CERT for sharing his research into the origins of the term "zero-day attack." Finally, this research would not have been possible without the WINE platform, built and made available to the research community by Symantec. Our results can be reproduced by utilizing the reference data set WINE 2012-003, archived in the WINE infrastructure.

References

[1] W.A. Arbaugh, W.L. Fithen, and J. McHugh, "Windows of Vulnerability: A Case Study Analysis," *IEEE Computer,* vol. 33, no. 12, December 2000.

[2] L. Bilge and T. Dumitras, "Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World," ACM Conference on Computer and Communications Security, Raleigh, NC, October 2012.

[3] D.H.P. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium : Tera-Scale Graph Mining for Malware Detection," SIAM International Conference on Data Mining (SDM), Mesa, AZ, April 2011.

[4] S. Frei, "Security Econometrics: The Dynamics of (In) Security," Ph.D. thesis, ETH Zürich, 2009.

[5] S. Frei, "End-Point Security Failures, Insight Gained from Secunia PSI Scans," Predict Workshop, February 2011.

[6] Google Inc., "Pwnium: Rewards for Exploits," February 2012: http://blog.chromium.org/2012/02/pwnium-rewards-for-exploits.html.

[7] A. Greenberg, "Shopping for Zero-Days: A Price List for Hackers' Secret Software Exploits," Forbes, March 23, 2012: http://www.forbes.com/sites/andygreenberg/2012/03/23/shopping-for-zero-days-an-price-list-for-hackers-secret-software-exploits/.

[8] T. Dumitras and D. Shou, "Toward a Standard Benchmark for Computer Security Research: The Worldwide Intelligence Network Environment (WINE)," EuroSys BADGERS Workshop, Salzburg, Austria, April 2011.

[9] B. Schneier, "Full Disclosure and the Window of Exposure," September 2000: http://www.schneier.com/crypto-gram-0009.html.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Buy the Box Set!

Whether you had to miss a conference, or just didn't make it to all of the sessions, here's your chance to watch (and re-watch) the videos from your favorite USENIX events. Purchase the "Box Set," a USB drive containing the high-resolution videos from the te chnical sessions. This is perfect for folks on the go or those without consistent Internet access.

Box Sets are available for:

- **UCMS '13:** 2013 USENIX Configuration Mangement Summit
- **HotStorage '13:** 5th USENIX Workshop on Hot Topics in Storage and File Systems
- **HotCloud '13:** 5th USENIX Workshop on Hot Topics in Cloud Computing
- **WiAC '13:** 2013 USENIX Women in Advanced Computing Summit
- **NSDI '13:** 10th USENIX Symposium on Networked Systems Design and Implementation
- **FAST '13:** 11th USENIX Conference on File and Storage Technologies
- **LISA '12:** 26th Large Installation System Administration Conference

Learn more at:
www.usenix.org/boxsets

Rethinking Password Policies

ABE SINGER AND WARREN ANDERSON

Abe Singer is the Chief Security Officer for the Laser Interferometer Gravitational Wave Observatory at the California Institute of Technology. He has been a programmer, system administrator, security geek, occasional consultant, and expert witness. His areas of interests are in security that actually works.
abe@ligo.caltech.edu

Warren Anderson is a Visiting Assistant Professor in the Department of Physics at the University of Wisconsin-Milwaukee and a member of the LIGO Scientific Collaboration. His publications are primarily on black holes and gravitational waves. This is his first foray into the fascinating world of computer security.
warren.anderson@ligo.org

We are all familiar with having "rules" for passwords: they must have characters from various character sets, have a minimum length, get changed regularly, not be written down, etc. These rules are supposed to make passwords "secure," but there's little to no research to support that argument. In fact, they can even weaken security. We argue that it's time for a radical change of password policy. In the blog post "Security Myths and Passwords," Gene Spafford also made the case for questioning the conventional wisdom on security:

> In the practice of security we have accumulated a number of "rules of thumb" that many people accept without careful consideration. Some of these get included in policies, and thus may get propagated to environments they were not meant to address. It is also the case that as technology changes, the underlying (and unstated) assumptions underlying these bits of conventional wisdom also change. The result is a stale policy that may no longer be effective…or possibly even dangerous.

Even the US government "standards" on password strength appear to be based on nothing more than then-current default settings on a particular operating system. Most of the "best practices" in use today are based largely on folklore or, in some cases, on severely outdated theories of password strength. These password best practices have several usability problems. Some believe that security and usability are mutually exclusive, and so security has to make things difficult. We argue that security depends on usability.

Passwords have to be strong enough to defeat cracking attempts, yet usable. This requires both an understanding of usability, and quantitative measurements of password strength. We provide an overview here and propose a solution (see [8] for more detail).

Why Do We Have Password Rules?

Users, left to their own devices, tend to choose passwords using real words. Which is understandable—users want to have a password that's easy to remember. Attackers, knowing this, use dictionaries of real words for dictionary attacks: cracking.

Password-strength rules ostensibly force the user to choose a password that's not in the attacker's dictionary. More formally, the rules attempt to prevent successful dictionary attacks by ensuring that users choose passwords with sufficient entropy to render the attack infeasible. Entropy is the measure of the probability distribution of the passwords across the keyspace—a measure of the relative randomness of each password to all the other passwords. Note that password strength rules provide no protection from brute-force attack: an exhaustive attack against the entire keyspace. The defense against a brute force attack is an immense keyspace.

Standards for Password Rules

What few standards exist are based on research that is at best inconsistent and, in most cases, appear to be pulled out of thin air. For example, NASA's password requirements claim to be in compliance with the Federal Desktop Core Configuration and are representative of these "best practices." The Core Configuration itself may contain the settings that NASA uses, but no document within the FDCC provides any description or justification of password complexity requirements.

Here's a summary of what we could find about password rules among the various NIST and FIPS documents regarding passwords and computer security:

- Passwords shorter than 10 characters are usually considered to be weak. Passphrases shorter than 20 characters are usually considered weak (two different documents).
- Users are bad at choosing passwords; passwords should be automatically generated.
- It's difficult to measure the entropy of user-chosen passwords and there's not much data.
- Password composition is a factor in password requirements, but the specific requirements are up to the organization.
- Users should be trained on good password practices, and systems could restrict password choices based on password composition.
- Choose good passwords by using the first character of each word of a well known phrase, etc.
- When determining policies for password length and complexity, organizations should consider maximum and likely actual keyspace.
- Totally alphabetic password composition should be discouraged.

So whence the "best practices" in the NASA/FDCC requirements? It appears to come from Microsoft Windows NT Service Pack 2. NT SP-2 introduced hard-coded password strength requirements with a minimum length of six characters, and the password had to contain at least one character from the four character sets. Windows 2000 allowed for changing the settings, with an eight-character default password length. Microsoft gives no justification or citations for any of those requirements.

Additionally, the NSA [1] recommends passwords be at least 10 characters, contain at least one of each of the four character sets, and get changed every 60 days. They too provide no justification for those values.

Password Aging

There's no there there.

—Gertrude Stein, *Everybody's Autobiography*

Aging passwords—requiring users to change passwords at regular intervals—originated due to the use of hashing algorithms which were weak enough to be subject to a brute force attack. Password aging is a defense against brute force attacks, not dictionary attacks.

The NSA's Green Book details the relationship between password length and password lifetime, and includes formulae for calculating minimum password length. Note that at the time that the Green Book was written, brute-force attacks against the hash algorithms in use were considered within reach of government funded agencies.

For Windows 2000, Microsoft stated, "Where security is a concern, good values [for password lifetimes] are 30, 60, or 90 days. Where security is less important, good values are 120, 150, or 180 days." But they do not provide any definition for what "important" and "less important" are, nor how they calculated those numbers. The default password lifetime in Windows 2000 was 42 days.

None of these recommendations provide any analysis as to how much, if any, password aging reduces the risk of dictionary attacks. For any given password aging interval n, assuming some unknown attack on the passwords has equal probability of discovery at any point over n, the mean exposure time for a compromised password is $n/2$. It would seem that for any reasonable value of n, the exposure time would be unacceptable.

Passwords and Usability

This belief of the fundamental conflict between strong computer security mechanisms and usable computer systems pervades much of modern computing. According to this belief, in order to be secure, a computer system must employ security mechanisms that are sophisticated and complex—and therefore difficult to use.

—Matt Bishop, "Psychological Acceptability Revisited," *Security and Usability*

Computing professionals have long held onto the belief of an inherent tension between security and usability, that each works against the other, which has often led to a disregard of usability for the sake of securing systems. But that belief turns out to be a misconception based largely on a lack of understanding of the meaning of usability.

So what do we mean by "usability" in the context of security? Usability is often associated with perceived ease of use—the less effort required, the more usable the system. More fundamental properties of usability are [2, 3]:

- Is the user able to understand what is required of her? Can the user understand how to use the security mechanism properly, recognize when she's failed, and understand why?
- Is the user capable of using the mechanism properly?
- Does the user understand the goal of the security mechanism?
- Is the user motivated to follow the security requirements?
- Do the requirements and interface match the user's understanding of the security goals?

The study of "human factors" separates tasks into "production tasks" and "supporting tasks" (sometimes called "enabling tasks") [4]. Production tasks are the actual end goal of the user, the desired output. Supporting tasks are those that enable the user to work on the production tasks. For example, a user authenticating himself to the system enables that user to access data on the system. Accessing the data is the production task, and authentication is the supporting task. Users don't want to spend time on supporting tasks—those that have too much of an impact on production tasks affect the usability of the system and the productivity of the users.

"Ease of use" is an important property but is not completely equivalent to the "work factor"; the work factor of supporting tasks can involve not only physical time and effort, but cognitive load, the measure of the ability of people to learn [5]. The amount of mental effort a user has to expend on understanding security requirements and complying with them are all a cognitive load that affects the size of the supporting task.

In order for a security mechanism to be used properly, the user must be able to understand both how and why to use it, and be able to use it efficiently and effectively. Security depends on usability.

The Usability of Common Password Requirements

The following are common password requirements that have a negative impact on the usability of passwords:

- Rules for password complexity
- Requirements to change passwords on a periodic basis (password aging)
- Requirements not to reuse old passwords
- Prohibitions against writing down passwords

As we noted above, some of these rules were originally devised in a context that often does not apply today.

Password Complexity and Aging

Password complexity rules make the user expend time and effort to devise an acceptable password, and then memorize it. This imposes a cognitive load on the user and increases the supporting task work factor.

Password aging rules further increase the cognitive load and work factor, by forcing the user to repeat the process of devising and remembering passwords repeatedly.

The negative impact of this combination of rules has been noted in several places.

A study on password usage [6] within the FAA quantified the direct cost in staff time in changing passwords, noting that the costs were greatly magnified by the fact that users had numerous (up to 20!) passwords for different systems, all with different password rules and aging policies. Users were essentially in a steady-state of changing passwords.

This same study noted that due to the burden of remembering passwords, coupled with the impact of forgetting passwords on production tasks, users adopted numerous coping strategies, which were in turn violations of other security policies: leaving sessions logged in, sharing passwords with coworkers, writing passwords down, etc.

Even the federal government acknowledges that password changing can cause problems: "The FIPS guidelines actually acknowledge that the load on users created by frequent password changes creates its own risks, which in many contexts outweigh those created by changing a password less frequently." [4]

And here's the fun part: there is absolutely no risk justification for any of the time intervals (42 days, 3 months, 6 months, 1 year) seen in current "best practices." As far as we can tell, all of those numbers have been pulled out of thin air (or less well-lit regions).

Usability of Pro-Active Password Checking

Pro-active password checking [7] seemed like an effective approach to strong passwords at the time that it was proposed. Avoiding dictionary attacks was best solved by preventing users from entering passwords that were in the dictionary. That approach assumes, of course, that one can check against a dictionary that's at least as good as any attacker would use.

Computation power in 1992 was such that a reasonably modest dictionary of 100,000 words or so, plus common substitutions, was sufficient to deter attacks. But current computational power, combined with easy online access to comprehensive wordlists, has changed the landscape.

We made an attempt at implementing pro-active checking by doing what an attacker would do: creating the biggest dictionary we possibly could. Using 1-grams from the Google Book project. We started with a list of ~4,000,000 words, and after applying the Crack substitution algorithms, ended up with a dictionary of about 90,000,000 passwords.

Having users change their passwords while checking against the dictionary was a colossal usability failure. There were so many unacceptable words that users became frustrated trying to come up with an acceptable password, and ended up choosing randomly until one was accepted by the system.

Pro-active password checking fails usability because it's impossible for the user to understand how to comply with the rules without guessing, and ends up increasing both the work factor and cognitive load of choosing a new password.

Risks of Writing Down Passwords

The prohibition against writing down passwords is an assumed mandatory requirement [2]. So the user is forced to devise a difficult-to-remember password, and then immediately remember it, further exacerbating the cognitive load on the user [4]. Add to this the oftentimes useless feedback provided to the user while attempting to create an acceptable password.

But that risk from writing down passwords is very context dependent. Prohibition against writing passwords hails from the military, where the threat of a malicious insider (a spy) looking for written down passwords was substantial, and the liability of that risk, astronomical. That threat may be substantially lower in other contexts, where the threat of password guessing from a remote anonymous attacker is much higher.

And, as mentioned above, the burden of having to remember passwords causes users to take other measures that can impose equal or greater risks. Writing down passwords reduces the cognitive load for users, especially for passwords that get used infrequently.

Writing down passwords is also perceived as being very insecure because the passwords may get left someplace they are easily discovered. That risk can be easily mitigated with some simple rules for keeping the written password in a reasonably secure location (e.g., wallet, locked desk, etc.). Note that even the Green Book recommends that users memorize their passwords, but allows for writing down passwords as long as the written password is sufficiently secured.

In many environments, the risk of dictionary attacks against passwords greatly outweighs the risk of writing down passwords; strong passwords are more important than easily memorable passwords.

Single Sign On

The FAA study noted that many subjects had numerous passwords to remember. Reducing the number of passwords that users have to remember greatly reduces cognitive load. A single-sign-on system, where the user has to remember and use one at a given interval (once a shift, for example), has a profound effect on usability [2].

Passwords and Entropy

People often speak of password entropy as a measurement of password strength, and attempt to measure the entropy of a given password. But as stated above, entropy is the measurement of the relative randomness of all the passwords together—you can't measure the entropy of a single password.

The only way to guarantee high entropy of user-chosen passwords is to require users to enter passwords that are significantly different from other passwords. But the only way to achieve that is to reject the user's new password as being too similar to another password, which in turn provides hints about the composition of another password on the system.

Password character class rules fail to provide any guarantees of entropy because they do nothing to prevent users from choosing the same or similar passwords.

Improving the Usability and Security of Passwords at the Same Time

Here's a modest proposal to make password management more usable for users and improve the entropy of the passwords at the same time.

Provide single/common sign on to minimize the number of passwords the user must remember (reducing cognitive load) and the number of times the user has to authenticate (minimize supporting tasks).

Allow the user to write down her password, as long as it's done in a reasonably secure manner, reducing cognitive load, and reducing the need for users to adopt insecure coping strategies.

Eliminate password aging, minimizing work factor and cognitive load for devising and remembering new passwords. Only require password change when the password may have been compromised. To minimize compromise, prohibit (or at least discourage) the users from using the same password at sites out of your control.

Eliminating aging means that you need sufficient password entropy to prevent a dictionary attack. Even if you don't eliminate aging, you still need to be able to quantify the entropy in order to determine an aging interval that has acceptable risk.

So you need to implement password rules that guarantee sufficient entropy across the set of user passwords. But here's the rub: when you let users choose their own passwords, you can't devise password rules that are both usable and have enough entropy. We are publishing a paper in the near future that demonstrates this.

One answer to that dilemma is to not let users choose their passwords, but to generate passwords for them using a random algorithm. It's the easy, perhaps only, way to ensure entropy, and when done right, can be usable. At least, if the random passwords are sufficiently memorable (and typeable), they can be more usable than requiring the user to choose a complicated, difficult-to-remember password that he can't write down and has to change often.

While the cognitive load of learning the new password may be greater (and we're not sure that's true), it doesn't have to be much greater, and can be offset by allowing the user to write it down.

The above combined approach creates a "grand bargain" with the user: in return for not being able to choose her own password, the user will only have to learn the one assigned, can write it down to aid with memorization, and will never (normally) have to change it.

Reasonably Memorable Random Passwords

There is a standing assertion that random passwords are difficult to remember and therefore fundamentally unusable [3, 4]. However, these assertions turn on assumptions as to how those passwords get formed: e.g., random strings of characters. We argue that if done properly, they can be reasonably usable and memorable.

In order to randomly generate usable passwords, consider that not all users are the same; their criteria for acceptable passwords can vary:

- A short, complicated password requires less typing.
- A longer alpha-only password is easy to enter via iPhone/tablet.A very long password is easy to remember—e.g., a passphrase.

Random generation of passwords can be acceptable when the user is given a set of choices within the constraints of the password entropy requirements. Giving the user a limited set of choices also gives the user the opportunity to select a password he finds more memorable, reducing cognitive load.

To demonstrate, consider the following set of choices:

Passphrases generated from a word list	opinion parting theological infrastructure lecture vividly
Lowercase alphabetic passwords	vukizocylqhxzxiexq qgmblqmtngtiurtybj
Alpha-numeric passwords	khjd2gjact31koo7 ntrv5xbrvdbt6d05
Mixed-case alphabetic passwords	ywcgyRwIdUbBsL zmbLwdAFvQuIPQ
Random passwords	im&c<Z+I)<t^ XvG[9Hm8klpN

Our experience with this system found that the passphrases are reasonably easy to remember.

Generating memorable random passphrases requires drawing from a dictionary of words that are already well familiar to the user. The average English-speaking adult vocabulary is 20,000–50,000 words, but that list includes words the user will recognize but not know well enough to spell or remember. Using a dictionary of the 10,000 (or fewer) most frequent words seems to provide passphrases that are sufficiently memorable to the user.

Conclusion

Password rules shouldn't be used unless they're actually effective. Our proposed approach results in measurably strong passwords that we think are quite usable. But our experience to date is anecdotal; usability studies to validate our hypothesis would be a good area for future research.

References

[1] National Security Agency, "Guide to the Secure Configuration of Red Hat Enterprise Linux 5," Revision 4.2., August 26, 2011.

[2] Anne Adams and Martina Angela Sasse, "Users Are Not the Enemy," *Communications of the ACM,* vol. 42, no. 12 (December 1999), pp. 40-46, doi: 10.1145/322796.322806.

[3] J.H. Saltzer, M.D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE,* vol. 63, no. 9 (September 1975), pp. 1278, 1308, doi: 10.1109/PROC.1975.9939.

[4] Lorrie Cranor and Simson Garfinkel, *Security and Usability* (O'Reilly Media, Inc., 2005).

[5] A.-M. Horcher, G.P. Tejay, "Building a Better Password: The Role of Cognitive Load in Information Security Training," in Proceedings of the *IEEE International Conference on Intelligence and Security Informatics* (2009), pp.113, 118.

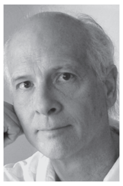
[6] K. Allendoerfer and S. Pai, "Human Factors Considerations for Passwords and Other User Identification Techniques part 2: Field Study, Results and Analysis" (DOT/FAA/TC-06/09).

[7] M. Bishop and D. Klein, "Improving System Security Through Proactive Password Checking," *Computers and Security,* vol. 14, no. 3 (May/June 1995), pp. 233–249.

[8] A. Singer and W. Anderson, "Rethinking Password Policies" (uncut): https://www.usenix.org/publications/login/august-2013-volume-38-number-4.

Bill Cheswick on Firewalls
An Interview

RIK FARROW



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

Ches is an early innovator in Internet security. He is known for his work in firewalls, proxies, and Internet mapping at Bell Labs and Lumeta Corp. He is best known for the book he co-authored with Steve Bellovin and now Avi Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker.* Ches is now looking for the next interesting thing to do, and is open to suggestions.
ches@cheswick.com

Like many USENIX members, I first met Bill Cheswick at a conference. Ches tended to stand out from the crowd, whether he was encouraging people to learn juggling using beanbags or making a presentation.

Ches later became famous after he co-authored the first book about firewalls with Steve Bellovin in 1994. This was not Ches's first adventure in the world of security by far, as he had been working with a firewall as early as 1987 while at Bell Labs.

Rik: Tell us about how you got involved with firewalls.

Ches: My first day at the Labs turned out to be the Christmas party in 1987. I walked up to Dave Presotto, the author of upas, and told him I wanted to be postmaster. I told him that networks were the wave of the future, and I wanted to learn a bit more about them.

He was delighted, and showed me the ropes. He also showed me around his application-level firewall, a VAX 11/750 running 4.3 BSD. Sendmail was replaced by upas, and IP forwarding was disabled. Insiders could use some software he and Howard Trickey had written to access the Internet from inside the company. I later modified this software, changed its name to "proxyd" because "gated" was already taken. I realized a couple decades later that this was the first use of the word "proxy" as it is now used. I need to drop a note to the editor of the OED.

In many ways postmaster was a much more interesting job at the time. There was no spam, which casts a gloomy gray pall over today's email. What we did have were a sea of networks and email addresses. In those days, my two main addresses were research!ches and ches@att.arpa.

Rik: How did you learn about TCP/IP? Back in the late '80s, this was a difficult topic to learn, with no books out yet.

Ches: Steve Bellovin and Dave Presotto had taught me the basics of TCP/IP, and I had *plenty* of lab work. I was using research UNIX, soon to be called the tenth (and last?) edition, and messing around with Plan 9, a lovely and cleaner rethink of UNIX designed from the ground up.

Then, one morning in November 1988, Marion Harris called, saying there was something bad happening on the ARPANET. NPR confirmed this.

I had a sinking feeling: would our firewall hold up to whatever was happening? That feeling has dominated my thoughts about security ever since. Besides, there would be no end to the complaints, ribbing, and whining if the attack got through. Working in the UNIX room toughened the skin. I rushed into work, and the whole place was abuzz.

The short answer was yes, the firewall had held. Peter Weinberger was on the phone basically saying "neener neener" to a variety of sites, especially Bellcore. Those folks had completely rejected the idea of a firewall, and were completely bogged down with the Morris worm. Exponential growth is very hard to control, and Robert Morris had gotten a bit of the worm wrong. This is actually an enduring lesson: the pros behind Stuxnet had a similar problem.

Back to that sinking feeling. How did we really do?

There was no direct IP connectivity to the intranet through our gateway, so the inside was safe there. But if the worm captured the gateway machine, the kernel allowed incoming connections to SMTP servers, most of which were running Sendmail. I ran a scan of the burgeoning Bell Labs/AT&T intranet and found more than 1,300 susceptible hosts.

Rik: What did you use to scan for port 25 in November 1988?

Ches: I probably used sweep, a shell script written by Mike Muuse. It tested machines for five different vulnerabilities.

One of the worm's attacks had been through Sendmail, a common source of security problems at the time. Dave had replaced it, so it wasn't an issue on the gateway.

"Best block is no be there" —Mr. Miyagi, *Karate Kid 2*.

We did not run any of the r* services on the gateway. No trust, no access, so that worked. The worm also broke in via a hole in the finger daemon. About six months before, toward the end of a day's hard work, I felt lazy and decided to wander around looking at the various hosts I administered.

I checked things out on our gateway, and noticed a number of services in /etc/inetd.conf that I was not familiar with. They ran as root, and that's not a good thing. Instead of diligently checking each one out, I simply disabled the lot of them, including fingerd. There were no users on the machine besides administrators. If somebody didn't like it, I could revisit the decision.

Though there were solid principles involved, to this day I consider this to be Security by Luck. Luck is a handy but unreliable tool, and does not help mitigate that sinking feeling I mentioned before.

There was one other stroke of luck. Steve Bellovin had an unprotected 56 Kb link from our intranet to Bellcore's network, where the worm was seething. The worm spread by using /etc/hosts as a list of target machines. Steve's machine at the Labs was at the end of the list and the worm always started attacking targets from the top of /etc/hosts. Any infected machine bogged down completely before reaching the last entries in the /etc/hosts, so it never reached us.

Security by Luck indeed! I had to fix this. I want security without that sinking feeling (there's a book title). Not confidence in security due to hubris, but due to "no being there." This has guided my approach since then.

Rik: I can see how that incident could inspire you. What did you do next?

Ches: Fresh from the uneasy victory over the Morris worm, I decided it was time to create a new firewall. The old one was hopelessly overloaded, and I wanted a design I could rely on. I created a belt-and-suspenders two machine solution out of a couple of MIPS machines. This was a very robust and high-performance design. Steve pointed out that both machines ran upas, which gave it the possibility of a common mode failure, but I had high confidence in upas. It never did let me down.

At about this time (late 1980s) Mark Horton obtained a class A address for AT&T from the powers-that-be by simply asking. That was a different era. The address block lay fallow for a while. We wanted to use it inside the growing AT&T intranet, but the routers of the day had subnetting problems and we couldn't deploy it. Oddly enough, our Cray computer seemed to *require* a class A network: I never did figure that one out.

So I took 12.0.0.0/8 and announced it to the Net, feeding the packets to a non-existent Ethernet address and running tcpdump on the traffic, which came to about 12 to 25 MB/day. Steve analyzed that traffic and wrote a fine paper. Basically, we were watching the death screams of attacked hosts that used IP address-based authentication. One of the steps was to flood a client machine with traffic so it couldn't complain about the attack on the associated server machine. Apparently the author of the code thought it would be fun to use AT&T's network address for the spoofed packets.

This is the first packet telescope I can remember, and I think I might even have coined the term "packet telescope," but my memory is fuzzy on that. I do know that monitoring unused IP addresses remains a very useful tool, and kc [Claffy] from CAIDA [1] gave a nice talk on current uses of the technology just recently.

Rik: I seem to recall that you started publishing about this time.

Ches: There were a lot of PhDs writing papers at the Labs. I had done some new work for the fancy firewall, so I wrote one and showed it to Fred Grampp. I was delighted when he nodded his head and said, "This is a nice paper." So I submitted it and gave the talk at Winter Anaheim USENIX in 1990. It was my first paper, my zeroth being a Permuted Index for TeX and LaTeX commands, something I still use occasionally.

Rik: How about your paper analyzing interactions with a honeypot you designed?

Ches: Chasing down the "attacks" was interesting. Was the attack casual, accidental, or evil? Discerning intent is important in security, which is where those little sticks that the border guards used to use in *Mission Impossible* came from. Who was willing to crash through a stick? The guys with the real barrier down the road wanted to know, and with just a little lead time.

I wanted to catch an actual bad guy and watch what he did. We eventually did, using an early honeypot, and wrote it up in An Evening With Berferd [2]. I almost didn't write it; it was like an optional English paper. I don't like to write, but I do love to have written.

By 1993, it was clear that firewalls were important, but the only real coverage was in a chapter in a book from Gene Spafford and Simson Garfinkel. I mentioned to Steve that we could probably find a dozen papers to staple together into a decent book. He suggested this to John Wait at Addison-Wesley, who had been bugging Steve for a book for about a decade. John said the book was a great idea, but we had to write it from scratch.

Thirteen English papers assigned! Steve and I worked great together; I had never been allowed to write an English paper with a co-author. We settled on a table of contents pretty quickly. Chapters would bounce back and forth for a couple of days and be nearly completed. Some sections demanded information I hadn't thought about. This provided incentive to fill in the knowledge gaps.

The book came out in time for the Spring 1994 Interop. John had estimated that we would sell 8,000 to 12,000 copies. The first printing was 10,000 copies, and sold out in a week. They rushed the second printing, not even waiting for us to correct three errors. The corrections finally made it to the third printing, not long afterward. The first edition sold more than 100,000 copies in at least a dozen languages. As Steve once said, it was a 320-page business card. For me, it was certainly the most important thing I have done in my career. A decade later I'd go to a meeting with some sharp techies in a big company, and they would come up to me later and say the book got them started in network security.

Despite a number of incentives and entreaties, we didn't come out with the second edition, mostly a rewrite, until 2003, with Avi Rubin helping out.

Rik: I imagine that, with the book completed, you started working on other things, such as dealing with "split" DNS.

Ches: After the book was published, Steve and I worked to merge DNS processing for inside and outside queries at the firewall. It ended up in two patents (switching and filtering).

A couple things of note happened in 1996. One was the Panix SYN packet attack. It started me thinking about how to trace anonymous packets back through the Internet. Hal Burch joined me the next summer and we worked on an idea: applying little denial-of-service attacks on possible incoming packet paths, and seeing if they perturb the packet flow. Then repeat, attempting to trace back to the source of the packets. The DoS attacks would be done by locating packet amplifiers on the Internet and carefully applying bursts of pain. We tried this on Lucent's intranet, and it usually worked.

The USENIX paper we wrote came back with two classes of referee comments: 1) we can't accept this paper until the technique is proven on the Internet, and 2) don't you dare try this on the Internet. This approach was certainly out-of-the-box, and much better suggestions were made by others. Today, we don't seem to much care anymore: packets come from everywhere.

Around this time, my role was changing. I remember one day my boss asked if I would do some modifications to upas to make it respond to Sendmail switches. I also received a request to come review the security of AT&T Worldnet, which was going beta in six weeks. I mentioned that I came to the Labs because I didn't like Sendmail and besides, wouldn't the AT&T consulting be a more useful pursuit for the company?

Rik: What other possibilities did the success of your book create for you?

Ches: The book's publication opened many doors. I was invited to speak at numerous conferences, public and less so. A recent count showed that I have spent non-trivial amounts of time in more than 30 countries. I had the opportunity to help a number of law enforcement groups to start coming up to speed on cybersecurity. There were a few CIO breakfasts. Insurance companies wanted to write hacking insurance, and were keen to understand the worst-case scenario (the "hurricane Andrew") of a cyberattack.

Steve and I got an invite from the folks at Renaissance Weekend [3]. Steve thought it was junk mail and discarded it. My wife called and verified that yes, it is the annual get-together that the President goes to and yes, we were invited. I have gone for most years since then, and it has been a rich source of family interactions, ideas, and new friends. Heck, I shook the President's hand the first year I was there. This has also been a wonderful place to answer my science guy questions since leaving Bell Labs. And yes, Bill Nye and I had a fine walk on the beach discussing some of his work on the avionics on the 747.

The other thing in 1996 was my first Highlands forum. I met people like Esther Dyson and Fred Cohen. We did a "day after scenario" developed by the Rand corporation. It went like this:

- Imagine it is ten years from now (2006) and a series of <buzz-word compliant> attacks seem to be happening. Evaluate the attacks and advise the President.
- Now it is a week later and a series of <very nasty buzzwork compliant> attacks are seen. Analyze and advise the President.
- And, finally, we are back in 1996. What should we be doing to prepare for all this?

I had remembered the early days of the MILNET (the military's ARPANET), which was connected to the ARPANET through three 56 Kb lines at one point. When bad things happened on the

ARPANET, the military folks cut the links—the turtle pulled in his head.

If we wanted to do this in 2006, would we know where the links were? Besides, maps are cool, and someone should watch and monitor connections and their changes. Who could do that?

If the Air Force pinged Finland, is that an act of war? What about traceroute? Who could collect such data? I asked if data provided by a corporate research project would be useful. The answer was an emphatic yes: of course we want your free data!

Rik: I can see where this is heading: Lumeta and networking mapping.

Ches: 1996 had brought on the "trivestiture" of AT&T into Lucent/Bell Labs, AT&T, and NCR. I had to choose where I wanted to go, so I made a spreadsheet using pluses and minuses, sort of the way a teenager might choose a steady date. The score came out 59 to 60, clearly a draw within the margin of error. Most of my security friends, a lot of mathematicians, etc., went to AT&T Shannon Lab. I chose to remain at Lucent with the systems folk and scientists.

Hal Burch and I started on Internet mapping in 1997. Hal was a crackerjack Olympic programmer who now works for Google. I had the idea to collect connectivity data by sending traceroute-style probes to a zillion networks, and graphing and analyzing the results.

Graphing was going to be a problem: a typical scan would hit 100,000 nodes. I decided to use brute force and lay out the data using a spring force algorithm. Ace programmer Hal managed to hack together some clever optimizations to get a layout algorithm that would produce a nice display overnight. The Lucent intranet took much less time. The Internet layouts were amazing, if not entirely useful. *Wired* published one in December 1998 [5].

I hadn't checked the graphing literature before trying this, and it was a good thing: at the time, the papers said that an 800-node graph was huge. Hal's cleverness, and Moore's Law, had made large layouts much more feasible. Still, I wish I had thought of the project in 1990.

We started collecting and saving daily traceroute data in late 1998, and continued almost uninterrupted until November 2011. I have all that data lying around, available for research use.

I also came up with a way to detect leaks in an intranet perimeter: you send a packet to an inside host, with the spoofed return address of an external "mitt" machine. Packets that make it outside may not have seen a firewall. There was a similar test for packets coming inside.

When someone from Lucent New Ventures came around in late 1999 asking if I had any ideas for businesses, I told him about the maps and the leaks. Research organizations should send such queries out on a regular basis. A company would pay money for this information.

Having a company like Lucent spin off a (hopefully) hotshot startup was a little like watching an old fat man giving birth: there is a lot of grunting, but one isn't really sure what's going to happen.

On October 1, 2000, seven of us armed with VC money spun off from Lucent to found Lumeta. The sell was difficult because we were in a new category of product. Is it a security product or a network management product? Well, both, but marketing and maybe even the customers didn't like that answer. We went to a lot of VC meetings and new product shows. I saw a lot of business ideas, many of them funded, and many of which I thought were not so hot.

Lumeta started collecting and processing intranet maps from a variety of the largest corporations. I would have loved to make the data available, but it was much too sensitive. We clearly had access to the most extensive graph data on intranets in the world. Most of the customers were quite happy with the results, and we *always* found something interesting in their networks.

The daily collection of the world's path data continued, under the name of the Internet Mapping Project. Before Lumeta, back in the spring of 1999 during the NATO/Serbian war, I had focused on collecting fine network data from the area. Steve Branigan produced a fine graph and movie [4] of the effects of aerial bombing. We also found the Yugoslavian embassy in the US.

The daily network probes did bring a slow stream of complaints, which faded away by the mid-2000s: by then there was simply too much "background radiation" of evil packets on the Internet. But after 9/11, I stopped caring. I focused some extra scans on the obvious suspects and started collecting data.

A couple years later we met with Richard Clarke in Washington. I had extracted every traceroute path collected over the previous six months that contained at least one Iranian address, and mapped it, coloring rarely used links in red. He looked it over and said he had been asking for this map for the past six months. He asked if it was classified? Well, we were just a few people from NJ, no clearance, etc. He called a three star general over in DoD, and we eventually sold our product to a number of branches of government. I am not cleared to know what they found. They told us that our product made the Republic a little bit safer. That's good enough for me.

For me, Lumeta was a fine outcome for a research project. It also was a lesson in business, and has helped me separate clever research projects that will never be used from those with business potential. I have used this skill a lot since then, particularly at AT&T Shannon Lab.

Rik: Tell us about your time at Shannon Lab.

Ches: I started at AT&T Shannon Lab in April 2007. These were the AT&T researchers spun out of Bell Labs back in 1996, processed through some bad times, and augmented by sharp new additions. The security research department had been disbanded a few years before, and most of my security colleagues had scattered to various university positions.

As with Bell Labs, it was an honor and pleasure to work there.

I did do some security work there. I tried a few experiments with passwords on smartphones. I recovered my stolen iPhone with the help of Steve Branigan and the NJ State police. We created a little useful case law concerning the use of WiFi localization. It turns out that the iPhone has five radios in it, and the phone company interacts with four of them. (Go ahead, Über geeks: count those radios carefully!) I did not use the phone company's resources (other than my time) to catch him, however.

But there was something in the water at Shannon. I was generating about four patent ideas a year, plus a number of business ideas. My patent count is about a dozen, with about six more crawling through the system. Alas, AT&T is not fertile ground for small new ideas, and only a couple of my efforts bore fruit.

Aside from some authentication and security patents, I created a new kind of movie (the slow movie) and a new way to see movies (movie thumbscapes). I think the latter would make terrific (and lucrative) wallpaper that would bring in some money and interest for a major movie. Unfortunately, I was unable to reach a leading filmmaker to show off the results. I also invented a new kind of extremely soft-core pornography, but I will skip that.

Yifan Hu added a terrific new layout algorithm to graphviz, much better than our efforts of a decade before, and laid out the entire AT&T corporate org chart. I added labels, colors, and other data to his positioning data to create what must be the world's largest org chart. We could color links by employee age, patent production, union membership, etc. I created one for the CEO's office. I believe this visualization would be a valuable tool for corporate consultants.

I stayed at Shannon for five years before I was laid off in April 2012.

Rik: What do you plan on doing in the future?

Ches: I am trying to figure out what to do next. It is clear that I don't fit into the usual employment slots in the usual corporate suspects. I am hanging out at Penn as a visiting scholar, and some projects are starting up. Teaching is a strong possibility: I have always enjoyed it.

I am working on iTeX, a tool for bringing LaTeX documents to the iPad, including arXiv and Project Gutenberg texts. I am translating a number of Project Gutenberg books into LaTeX.

I am always looking to work with sharp people on interesting projects. But I am not idle: I don't understand where I ever had time to go to work for 40 hours a week. I heard of one fellow who said if he retired a second time he would have to hire an assistant to get all the work done.

Resources

[1] The UCSD Network Telescope: http://www.caida.org/projects/network_telescope/.

[2] An Evening with Berferd: http://www.cheswick.com/ches/papers/berferd.pdf.

[3] Invitation-only retreats for preeminent authorities, emerging leaders, and their families: https://www.renaissanceweekend.org/home.htm.

[4] Effects of war on the Yugoslavian network: http://cheswick.com/ches/map/yu/.

[5] Internet map, circa 1998: http://www.cheswick.com/ches/map/gallery/wired.gif.

Professors, Campus Staff, and Students—
do you have a USENIX Representative on your 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 who directly interact with students. We fund one representative from a campus at a time. In return for service as a campus representative, we offer a complimentary membership and other benefits.

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, representatives receive a complimentary membership in USENIX with all membership benefits (except voting rights), and a free conference registration once a year
(after one full year of service as a campus rep).

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, contact
Julie Miller, Marketing Communications Manager, julie@usenix.org

www.usenix.org/students

Setting the Stage for a Software Liability Discussion

MICHAEL B. SCHER

Mike Scher is VP and General Counsel with the network security firm Nexum. An attorney and security technologist by trade, and an erstwhile legal anthropologist, his focus is on risk mitigation, from the legal to the social, and the technical to the procedural. Mike has been working where the policy tires meet the implementation pavement since 1993.
mscher@nexuminc.com

Software liability isn't what most people seem to think it is. It varies by jurisdiction, market, and more. The current state of affairs—nasty EULAs on the one hand, and dread of liability for ordinary bugs on the other—is probably less than optimal. A patchwork of state, federal, and "judge-made" law is inconsistent by its very nature, varying in complex ways for each situation. In this article, I try to equip the reader with some key concepts around product liability from the perspective of an attorney and security geek.

Among friends and coworkers (on the systems and security side of the industry), and among clients and partners, there is renewed interest in questions of liability for software in its many forms. A lot of the talk addresses controlling risk from a technical or legal perspective, and some addresses how things "ought to be" from a technical or legal perspective. They are frustrated and looking for change, but don't think new laws or regulations will do anything but make matters worse.

Many of them appear to have assumptions about the forms of liability operating today—assumptions that are at odds with how the various areas of liability in fact operate. Ultimately, it is critical that discussions about law build on an accurate sense of, generally, what the law is and how it operates. Bad policies, like bad arguments, are built on false premises. This article is an effort to help lay the conceptual groundwork for developers and sysadmins to engage effectively in discussions on future policy and law regarding software liability, security, safety, and responsibility. My take is US-centric, but the principles should stand one in good stead broadly, and some references here may help others explore the state of the law outside the US.

As with the 2006 ;login: article on negligence [1], "The content and positions contained in this article should not be taken as legal advice—the discussion is simply far too general and the subject matter too complex to safely use that way." The purpose is to make readers more conversant in the issues to apply that knowledge to policy discussions regarding their own areas of expertise.

Negligence Quick Review

Some seem to think we are headed for a negligence standard when we discuss the possibility of liability for flaws in software. A negligence standard is sort of the US default liability standard for anything not specifically and exclusively legislated, regulated, or otherwise under a different standard through common law.

In short, under a negligence standard, one need do the "reasonable" thing. Applied to software, "reasonable" has a lot to do with what a reasonable end-user should expect—which has a lot to do with industry standard practices, but (see again [1]) sometimes industry standards themselves aren't (legally speaking) "reasonable." Negligence is always "there," but licensing agreements, including shrink-wrap style agreements, can disclaim a lot of that liability, for almost anything short of reckless or willfully harmful conduct. Because shrink-wrap licenses vary in the extent to which they are enforceable (by content and by jurisdiction), only customers in a strong negotiation position relative to the licensor will consistently have the ability to shift risk back onto the licensor/reseller. The rest of us end-users might just have to take it as given or drive on. Due to the economics of combining contracts with torts litigation, practical liability from provider to end-user can disappear in a puff of EULA.

Warranties

With regard to negligence, the industry handles risk today through EULAs applying to licensed software. EULAs disclaim virtually all errors, as well as many of the warranties stemming from states' common law and statutes governing the sale of goods, such as the broadly adopted Uniform Commercial Code (UCC) Article 2. Most software isn't "sold" as a "good"—rather, it is licensed. Thus, as some scholars and practitioners will rapidly point out, UCC Article 2 doesn't even apply to many software transactions [2]. Readers may recall that the 1990s saw the controversial Uniform Computer Information Transactions Act (UCITA), which started as an outgrowth of UCC Article 2 warranties for goods. It proposed broad warranties for licensed software, but allowed virtually all warranties to be disclaimed. Only two states have adopted UCITA in any form [3, 4]. Some courts and a few states have declared all software to be a good subject to UCC Article 2, and sometimes software is delivered incorporated in a good that is subject to warranties. Still, EULAs and similar agreements disclaim many warranties, opening questions regarding general consumer protection law and contracts of adhesion, the answers to which, of course, vary by jurisdiction; however, warranty actions are not how most serious harms caused by product failure are handled.

Product Liability: Software and "the Market"

We're used to using free and commercial software to perform important functions—yet, when introduced, such software may be rife with functional problems that can corrupt data, cause halting, open a system to compromise, or bring about other significant issues. The discovery of security issues in software is a regular occurrence across most popular packages.

The market accepts file corruption and routine rebooting in early edition software, including some operating systems, suggesting that the marketplace has a period of adoption elasticity in which the benefit of inexpensive adoption outweighs the issues. At some point, in theory, mounting competition and pressure for stability and security influence the package producer. Even as we seem to expect the market to perform that function, the notion that critical-use software could fail as badly as the latest app we dropped on our smartphone is an alarming one—especially since consumer market-pressure correction comes after adoption. Still, we're not crazy for thinking people actively making choices can influence quality. We as a society click "Accept" to low standards for many reasons, some historical, some market structural, and some as part of the cost of doing business and keeping software prices low.

That last sounds like almost any competitive commercial goods-producing sphere: we want prices as low as possible, and are willing to accept some drop in quality in exchange, but we still want those goods to be without significant defect. The discussion around software liability hinges on that point: what form will liability take and where is the line that will permit bountiful software development while steering us away from a caveat emptor marketplace? We've discussed negligence for acts and omissions, and how warranty may apply to goods. The US (and much of the world) handles product liability for harms suffered from "defective goods" differently from other forms of liability, and quite differently from the way we handle most licensed software today.

Strict Liability and Products

Ordinary negligence can be a case-by-case, time-consuming, and not-always-predictable process, to say the least. Modern product liability ultimately posits we shouldn't have court cases looking at the micro-facts of each $100 buyer's case, that a buyer should be able to have a base-level confidence that products released into the marketplace are without "defect" to the extent of the product's "intended use." Due diligence and proximate causation are two key issues in negligence—did defendant's behavior fall below a reasonable standard and, if so, did that cause a foreseeable harm in a manner to which liability attaches?

In product liability, the causation question is often simpler. The complicated question is whether the root problem is a "defect." Product liability is inherently a strict liability regime, not a "due diligence" one. Once there is a harm, and a defect leading to the harm is identified, the product maker (and others in the chain of sale) are generally held liable. The definition of "defect" itself subsumes many issues similar to negligence. Because the defect affects many in similar fashion, and because each affected individual's contributory negligence need not be weighed on a case-by-case basis, product liability cases are generally brought as class actions (thus avoiding spending courts' time for each $100 case, permitting class-wide disposition of the matter, and allowing the company and those affected to move on).

Let's take a quick look at how the Restatement of the Law Third, Torts: Products Liability talks about the key term "defect" in goods. There are three forms of defect, broadly defined [5].

First, manufacturing defects, "when the product departs from its intended design, even if all possible care was exercised." Note that negligence isn't the issue with this form of defect; it focuses on the market and the good, not the maker's degree of care. It is possible to have a product be defective and its maker liable for harm even if all reasonable care was exercised. As a matter of public policy, one could say the sale of such a good is inherently unreasonable, but again, the negligence standard simply does not apply [6].

Second are design defects, "when the foreseeable risks of harm posed by the product could have been reduced or avoided by the adoption of a reasonable alternative design, and failure to use the alternative design renders the product not reasonably safe." Here the focus is on both the maker and the market. Liability for failing to use an alternative design hinges to a degree on the reasonable nature of the alternative design, and in that aspect is reminiscent of negligence questions, only to the extent of examining the availability and viability of alternatives.

Third are inadequate instructions or warnings defects, "when the foreseeable risks of harm posed by the product could have been reduced or avoided by reasonable instructions or warnings, and their omission renders the product not reasonably safe." We've all seen what we consider ridiculous instructions (e.g., "do not eat" on silica packets). Here the focus is on the maker interacting with the intended market—which market, from the news, will seem to many readers to have an ever-decreasing mentality. Many of the cases making news as if of the third type are actually of the second. The press tends to repeat these PR pitches uncritically. What, the press carry water for a PR firm? How unreasonable!

Software Liability Generally, Today

Depending on deal size, at the corporation level, an end-user company can push to have UCC Article 2 warranties explicitly apply, and go well beyond that, assuming the software company is eager for the business. That's a contractual engagement where sophisticated parties each with some degree of negotiating power negotiate a deal on price and license/liability terms.

Consumer-facing software is currently subject to a patchwork of liability standards, even at the federal level, with a negligence standard applying only to the extent EULAs can't disclaim it, which means most software won't see a negligence suit in some jurisdictions (but again, reckless or other egregious conduct generally can't be disclaimed). Warranties are a little harder to disclaim, again varying by jurisdiction and case specifics, but EULA language disclaims them broadly anyway.

When UCITA was proposed, a few states drafted "anti-UCITA" statutes that declared software a "good" subject to UCC Article 2, even if licensed, and some courts have also held software should be treated as a good. When software is licensed and treated not as a good, UCC Article 2 warranties don't apply (although when "sold" rather than licensed, it is a "good" in most jurisdictions). Even if and where UCC warranties for sold goods apply to licensed software, they may be subject to disclaimer in EULAs, subject to courts' interpretation of contracts of adhesion in the context of EULAs [7].

For example, some software licenses disclaim just about everything—even violation of intellectual property rights, which could see the end-user sued for patent violation and left to deal with it. Such EULAs essentially say, "this does more or less what we say it does; otherwise, use at own risk. Pay here." In some jurisdictions, software liability is today essentially under a contracts regime, subject to some consumer protection law related to contracts made between parties in unequal bargaining positions. Thus, with negligence and warranty generally disclaimed, subject perhaps to a complicated court battle, some consumers are left to pay for "your problem—deal with it" contract terms on software because they are in a significantly unequal bargaining position with the software producer or seller. Adding further complication, some jurisdictions treat such contracts as unenforceable.

Software liability can thus take the form of liability in negligence, in products liability, in contract (license terms providing a broad range of risk-shifting), and consumer or inter-business contracts for goods (warranty terms, explicit and implied). One almost needs to apply multivariable differential equations to solve for any particular jurisdiction along three major axes, each containing subordinate axes [8, 9]:

1. Liability regime: negligence, products liability, contract, warranty
2. Sold as: license or good
3. Shrink/clickwraps: enforceable or not, and to what degree

All that, without even looking at the complexities of other areas of federal and constitutional law, let alone criminal law.

The Future Isn't What It Used to Be

The complexity in liability for software calls out for a considered standard, even if it is one with broad flexibility. Courts are slowly, but not broadly, rejecting the ability to disclaim warranty in consumer software. But court-considered law is going to be inconsistent by the nature of the market and jurisdiction.

If we push toward a model for software liability, what could it be? If modeled on "goods," would that just be UCC-type warranty plus negligence law, and how much effect should a shrink/clickwrap have? Should we select a products strict liability regime? Is it easier for the industry to measure its "reasonable" behavior or to determine whether a product has "defects" (under the definitions above)?

When software is incorporated in hardware, the combination is sold "as a good" subject to UCC Article 2, with failure due to "defect" likely subject to product liability law. The reasons for that liability include that courts are presented with a product that failed, not an app ("plaintiff's microwave burst into flames"), even when software failure is the root cause. As a matter of public policy, it makes sense because the end-user is several steps removed from the software maker, and thus can't measure risk (the product manufacturer does that) or evaluate license terms (which, between a manufacturer and software supplier, don't look like what you and I normally get in EULAs).

Pressure to control risk is thus between supplier and manufacturer. So governed, a market risk-allocation still takes place, backed by Errors and Omissions/cyber liability insurance on the one hand, and products liability insurance on the other.

Some of my colleagues posit product liability for software will harm the industry. Yet the dizzying matrix of liability on software hasn't stymied software development in the US, from FOSS to mega-commercial. Software makers for products aren't running scared despite contracts between them and the product maker shifting risk onto them, from patent infringement to bodily harm.

To those who create software, a key concern is that the public does not understand the complexity of software, the mathematical impossibility of proving a system, the problems of design versus manufacture. There is concern that the vibrant and effective free software movement will be constrained. After all, haven't we seen the industry forced to improve in a market with viable, quality competition? These are valid concerns and any solution should distinguish among the various forms of license and market model ("sold," licensed for fee, FOSS). All complex systems are subject to subtle defect. Perfection in any form is impossible, its approximation expensive, and we're back to a cost versus quality discussion. Markets are supposed to be good at handling that kind of balance, though they tend to do so after harms appear.

To those outside the industry, it can seem like software makers want a "have their cake and eat it too" liability regime where they can both claim their software is perfect (e.g., "unbreakable") and be virtually without liability should it break, causing harm. That is also a valid concern and sits at the crossroads of a broad range of consumer-protection law.

Should the industry be satisfied with the current patchwork liability? Certainly, end-users of software incorporated in antilock braking systems probably would prefer the system not require a critical patch to prevent catastrophe 3-4 times a year (I am being generous). Such issues as they relate to the end-user are governed by products liability today. Could a reasonable dividing line for the form liability takes be the incorporation of software in a hard good sold as product? Perhaps a "shipped-with" divider between sold-as-good and "licensed"?

Could a manufacturer, rather than selling a good that incorporates software it has licensed, force the end-user to download and "relicense" the braking and other software on first "key-up"? Imagine starting up a new car and clicking through 20 EULAs (or one egregious one), waiving—subject to each state's consumer protection law, subject to each circuit's take on licensing vs. purchasing—all disclaimable liability for anything but mechanical failure. Those who have purchased provider-tied, app-laden smartphones have probably had a whiff of this experience.

These are the discussions we should be having. I hope this surface treatment of negligence, warranty, and product liability has helped arm you with terms and tools to better shape discussion of what "ought" to be, and to understand the complexity of how it "is" today.

References

[1] Michael Scher, "On Doing 'Being Reasonable'," *;login:,* vol. 31, no. 6, December 2006.

[2] For a brief discussion of software and UCC Article 2, see http://technologylicensinglitigation.com/applying-the-ucc-to-software-license-agreements/.

[3] For an excellent history, discussion, and description of UCITA, see http://www.jamesshuggins.com/h/tek1/ucita.htm.

[4] For a contemporaneous response to the UCC 2B proposal, see http://www.badsoftware.com/uccsqa.htm.

[5] American Law Institute, summary of *Restatement of the Law Third, Torts: Products Liability*: http://www.ali.org/index.cfm?fuseaction=publications.ppage&node_id=54.

[6] See, for humorous effect, http://snltranscripts.jt.org/76/76jconsumerprobe.phtml.

[7] Complex issues regarding licenses, EULAs, contracts of adhesion, and unconscionability are at play. A good summary of the development of cases through 2008 can be found at http://www.bicklaw.com/Publications/Unconscionable TermsandE-contracts.htm, and a discussion of click-through/browse-through terms can be found at https://ilt.eff.org/index.php/Contracts:_Click_Wrap_Licenses.

[8] In 1999, Clark Turner and Debra Richardson wrote "Software Defect Classes and No-Fault Liability," presenting an early discussion of the complexity of applying products-style "defect" and liability to software: http://www.users.csc.calpoly.edu/~csturner/fulltechreport.pdf.`

[9] For a similar discussion of the complexity of determining such issues, see Lloyd Rich, "If You Use A Shrinkwrap License It May Not Be Enforceable": http://corporate.findlaw.com/business-operations/if-you-use-a-shrinkwrap-license-it-may-not-be-enforceable-mass.html.

Other Resources:

Legal Information Institute, Products Liability: http://www.law.cornell.edu/wex/Products_liability.

HG.org, Legal Resources, Product Liability Law: http://www.hg.org/product-liability.html.

Macrothink Institute, "A Managerial Guide to Products Liability": http://www.macrothink.org/journal/index.php/ijld/article/view/1773/1458.

Enterprise Logging

DAVID LANG

David Lang is a Staff IT Engineer at Intuit, where he has spent more than a decade working in the Security Department for the Banking Division. He was introduced to Linux in 1993 and has been making his living with Linux since 1996. He is an Amateur Extra Class Radio Operator and served on the communications staff of the Civil Air Patrol California Wing, where his duties included managing the statewide digital wireless network. He was awarded the 2012 Chuck Yerkes award for his participation on various open source mailing lists.
david@lang.hm

When the topic of logging comes up, logs are generally recognized to be useful and that having a centralized log system is "industry best practice," and it's even required by most regulatory oversight plans (PCI, HIPAA). But figuring out how to get started in setting up a good logging system is hard, especially if you are already a good size organization when the topic is raised. If you start off by talking to vendors, getting quotes in the seven figure range is easy. This article is an introduction to logging, outlining an inexpensive architecture that can scale up to large log volumes, and providing pointers to a few basic tools to quickly and cheaply get value out of the logs beyond just satisfying audit requirements. Future articles will dive deeper into specific aspects of logging.

Benefits of an Enterprise Logging Plan

Getting started with logging in an enterprise can be as much a political/management issue over the effort and equipment involved as it is a technical issue of deciding what to do, so it's worth starting the discussion by reviewing the basic business benefits.

Logs record what happened. This seems like a trivial statement, but it's easy to get confused and think that logs mean more. Many things can go wrong, and having logs available helps you figure out what so that you can decide how to recover and prevent it from happening again. Examples of problems that you may need to investigate are misbehaving software, outside attacks, insider tampering, hitting system performance limits, and hardware failures (disk/memory/network errors).

Logs let you figure out how frequently things have happened, and this information can be used for utilization reports and capacity planning. Logs can also be analyzed to produce reports that show user behavior , which can then be used for marketing, product development, and detecting "odd" behavior that may indicate attacks. Logs can satisfy audit requirements by indicating who did what and when they did it (for both internal and external users).

Logs are invaluable for monitoring. Nothing can replace what the apps report about their own operations. If an app logs a message at 3:02 a.m. "unable to create file X No space left on device," saying that there's no problem does you no good because Nagios reported lots of disk space available at 3:00 and 3:10.

Collect Log Messages in a Single, Centralized Infrastructure

You most care about logs when something is (or has gone) wrong on the box where they were generated. Having a copy of the logs elsewhere lets you still see the logs. With cloud computing, this is even more critical than in a normal datacenter because a system is far more likely to go down, and when it does, you may never be able to get at its file system again.

By combining all the logs, you gain the ability to see what's happening across systems, to offload the log analysis from the systems that are serving your users, and to implement your tools and policies consistently across the enterprise. Protecting the logs from tampering if they're in one place rather than on every system is far easier.

Most compliance programs (PCI, HIPAA, etc.) require that you collect your logs in some central location. They don't say why, but the underlying reasons boil down to the advantages mentioned above.

You Should Try to Gather ALL Possible Log Messages

Because your logs are a record of what happened on your systems at some time in the past, you are usually not going to have a chance to tweak the logs to support the current problem you are dealing with. Some logs are more important than others, but you can always throw away or ignore logs that you have gathered, whereas you cannot go back in time and collect something that you didn't gather.

You should start with the premise that you will gather every log generated by every device, system, and application and only trim back if you find that you cannot support this. You cannot and should not process every log message the same way. Today's system performance is such that everyone except the largest companies can gather their logs into a single feed at a surprisingly low cost. Analyzing logs can be very expensive, so you will want to filter the logs as they go into your analysis tools, but different types of analysis will want different logs, so start off planning to gather everything.

Getting Started

Once you have decided to build an Enterprise-wide centralized logging system, you must determine the requirements you need it to satisfy.

Suggested Requirements for Enterprise Logging System

Vendor-Neutral Infrastructure

A good logging system will end up being used by just about every part of your organization. Any system you deploy is going to need to be changed at some point. If you build your logging infrastructure around a single vendor, changing it will be extremely painful. If you build it around standards, you can switch out portions of it at a time. While you are in the middle of migrating, you may not be able to take advantage of some features that only exist in one software package, but this will just temporarily degrade the system, not split it into two parallel systems.

Gather/Deliver the Logs in Near-Real Time

Many uses of logs require that you act on the logs shortly after they are generated. Any scheme that gathers logs nightly or hourly will not work for those uses, but if you gather the logs in near-real time you can support all the uses that will work with the batched gathering.

Run All Systems on the Same Time Zone

Running all your systems on UTC time is best, but even if you just pick the time zone of your main datacenter or office and use that everywhere, you are far better off than if each datacenter has systems running in its local time zone.

In theory this isn't a problem because all timestamps should include time zone information, but in practice, time zone information is frequently dropped; having timestamps from different time zones will confuse analysis of logs (including manual analysis).

The reason UTC is better than local time is that when rolling logs, storing them with filenames that have the timestamp as part of the filename is common; backwards adjustments due to daylight savings will cause you to overwrite and lose log files.

Fix Malformed Logs

Many devices (for example, Cisco Routers) have errors in logs that they send out. Fixing these errors early in the logging infrastructure makes it much easier to make use of the logs.

Add/Correct Log Metadata

Examples of metadata that can be useful to add/fix in log messages are timestamps, sources, and the office a log comes from. If you are in an enterprise large enough to have hostnames and IP addresses reused in different areas (e.g., think of how many workers who are telecommuting from home offices will be using 192.168.1.x IP addresses), adding additional information to the log message to be able to differentiate the duplicates can be extremely valuable.

No Modification Is Possible on Network Equipment and Appliances

This is less a requirement than a recognition of the reality that you cannot change how some devices send logs, so any scheme that requires that you run specific software on the system that's generating the log message cannot work as an enterprise-wide approach, no matter how well it works in a narrower deployment.

Minimize Configuration, Non-Default Software, and Load on the Systems Generating the Logs

While logs are valuable, if logging or administration of logging interferes significantly with the primary purpose of a device, odds are that the logging is going to suffer. The more work you have to do on each system, the higher the odds that the work isn't going to happen consistently, and you will end up with a gap in your logs that you are not aware of. Knowing that you are not receiving something that you would like to get is hard.

"Best Effort" Delivery of Logs

When you first think of the question "under what conditions is it OK to lose a log message," the normal reaction is "never." The problem with this is that the alternative to losing a log message when something goes wrong is to have the system stop. So the real question you must ask is, "Is this log message so critical that I would rather have the process/system stop working than have any possibility of losing the message?"

The answer to this is almost always “No, but I really would like to avoid losing logs if I can”; the rest of this article assumes that this is the case. There are ways to use modern logging daemons to deal with the ultra-reliable logging requirement (what I call “audit grade” logs), but it complicates the system and has horrible effects on performance (I have run tests in which I have measured greater than 1000x difference between “audit grade” and “best effort” performance).

Syslog, the De Facto Standard for Log Processing

The traditional UNIX tool for logs is syslog. Any log processing tool that has any pretension of being a general purpose tool is able to handle syslog messages. This makes syslog an obvious starting point; however, syslog has a poor reputation as a serious log tool because the versions of syslog that were the default on UNIX systems for the first couple of decades of syslog’s existence have had a combination of ultra-safe and ultra-unsafe defaults that have limited log rates from tens to low hundreds of logs per second, truncated messages at 1k characters, and either blocked system operation or silently dropped logs beyond these rates. Additionally, filtering in traditional syslog was complex and dependent on the originator of the log messages properly tagging each message; however, current logging daemons bear about as much resemblance to the traditional syslog that Eric Allman created as a quick hack for dealing with Sendmail logs as the cars in a Barrett-Jackson auction have with the cars that were on a Ford dealer’s lot in the heyday of the Model T. Most people, including many who deal with logs, do not realize that this has changed. There are now several additional logging implementations available for use, all of which are drastic improvements in performance and capability compared to the traditional syslog software, while still retaining software and network compatibility with traditional syslog. Since 2007, most Linux distros have switched to rsyslog as their default syslog daemon, and the rate of change over the past five years is staggering. Red Hat Enterprise 5.x ships with rsyslog3.22.2, but rsyslog 7.4 rolled out in June 2013. Additionally, syslog-ng, nxlog, and logstash are all free tools to consider if you dislike rsyslog. (Commercial syslog daemons, including a commercial version of syslog-ng, are also available.) Both rsyslog and syslog-ng now can handle more than one million logs per second, and all of these tools support a wide range of filtering and communication options. Combined with the fact that these all support the traditional syslog protocols means that you can choose whichever one you want, and switch from one to the other on your core infrastructure without having to change anything on your systems that are generating the logs.

As a logging protocol, syslog has the (dis)advantage that historically it has been poorly defined. Syslog has been around since the ‘80s with an RFC written for it in 2001 (and a follow-up in 2009), but the reality remains that you can throw just about anything at syslog and it will handle it in some form. This leads to the natural result that a lot of equipment (including from top name vendors) and software is generating syslog messages that don’t comply with any RFC, but the modern logging daemons are all flexible enough to be able to deal with the messages in a (relatively) sane manner. This great flexibility means that syslog is easy to get data into, and can deal with just about anything.

A recent development in the syslog world is the support across the many different logging daemons for JSON-structured logs. While this is primarily being driven by people who dream that all logs will be formatted to some standard, making it trivial for any application to parse and understand the log contents, this capability is absolutely wonderful for enterprise logging even if no such standard ever emerges [1]. This is because it makes it possible to take the original unstructured syslog message, wrap it in JSON and then add additional fields to hold information to the log message that is sent upstream. This maintains the separation between the original message and the new fields, allowing you to hand the original message to an analysis tool that doesn’t know about the new fields or format. This added information can include, for example, the environment the log was generated in, so that you can alert differently depending on whether the log was generated from a development machine or a production machine without needing separate logging systems.

Architecture

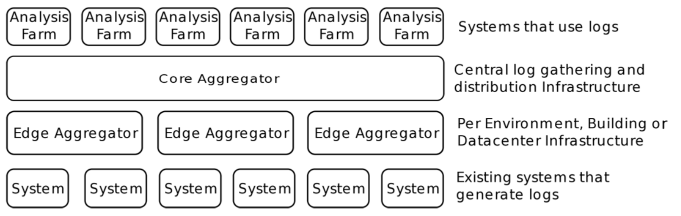


Figure 1: The best design for an enterprise logging infrastructure is divided into four main layers, which serve as clear boundaries between the logging responsibilities of the different systems in your enterprise.

The architecture should be able to handle a large enterprise with hundreds of thousands of systems across multiple datacenters. Smaller organizations can collapse the different layers in Figure 1 if appropriate. All of this infrastructure can be virtualized or cloud based, but performance or data sensitivity concerns may cause your organization to decide that parts of it should be kept in-house.

The Log Originators

Log Originators are all your normal servers, appliances, storage devices, switches, routers, firewalls, etc. These systems all send their logs to the closest Edge Aggregation systems, usually via UDP syslog.

For applications that cannot send their logs directly to syslog, you have several options to watch and scrape the logs from files to send them upstream. The syslog daemons mentioned above all have some capability to gather logs from local files, plus there are other, simpler tools that can do the job.

Noting that all the systems that you use for the rest of your logging infrastructure are also Log Originators and should be sending their locally generated log messages off to an Edge Aggregation system is important.

The Edge Aggregators

Edge Aggregation systems perform many different tasks: gather logs from local machines, fix malformed logs and add metadata, and queue logs as needed for reliable delivery to Core Aggregation systems.

Gather Logs From All Local Machines

There are many edge systems, distributed around the organization. The number of Edge Aggregation systems you deploy is a balancing act involving cost, complexity (the number of systems to manage), load on each system, and the reliability of delivering logs to the Edge Aggregators from your other systems.

- The closer the Edge systems are to the systems generating the logs, the more reliable your logs are going to be. While UDP syslog is extremely reliable over a local LAN switch, once you start sending it through links that can be bottlenecks (routers, firewalls, WANs, etc.), the chance of losing logs due to congestion, equipment failure, routing errors, ACL errors, etc. starts climbing rapidly.

In theory the answer is to use a more reliable transport than UDP syslog; however, many systems and appliances can only talk UDP syslog, so even if you change all your servers to a more reliable transport, you have only solved part of the problem. Deploying the Edge Aggregation systems close to the sources of the logs in HA pairs will let you survive system and network failures and congestion with the minimum loss of logs. The Heartbeat and Pacemaker projects [2] provide the tools to make implementing HA on a pair of Linux systems trivial.

At the very least, you should have Edge Aggregators before any WAN hops. I try hard to have a set of Edge Aggregation systems connected so that logs never have to go through a router or firewall before they will hit an Edge Aggregation system. In some extreme cases, I have used Edge Aggregation systems that have as many as 22 Ethernet ports on them (5x 4-port cards plus 2 on the motherboard) to allow me to connect directly to the different networks.

Fix Malformed Logs/Add Metadata

Fixing the logs and adding metadata should be done as close to the source of the logs as possible. There are several reasons for this:

- It limits the scope of one-off fixes that you may need to do for particular devices.
- Testing every message to see whether it needs to be fixed is expensive. Modifying a message is less expensive, but still not free. Doing this on the Edge Aggregators scales well.
- The Edge Aggregators know the actual source IP of the Log Originators, while systems further on only know what's in the message.
- The Edge Aggregators can have hard-coded values based on where they are in the network.

Queue Logs for Reliable Delivery to Core Aggregation Systems

Because there are relatively few Edge Aggregation systems, any effort you spend on them has a much higher cost-benefit ratio than work done on the Log Origination systems. Because these systems do not have to be used for anything else, you can replace your OS defaults with newer or different versions of logging software, and they can afford to expend more effort to deliver messages. If the messages are being delivered over a WAN link that goes down, these systems are perfectly positioned to queue messages to be delivered later. This is not mandating full "audit grade" reliability, but simply using one of the network protocols that will detect outages such as TCP syslog or Reliable Event Logging Protocol (RELP). Think about the safety of the links you are sending the logs over; you may want to encrypt the data before it is sent.

The Core Aggregation System

This farm of Edge Aggregator systems handles your entire log feed. Its purpose is to provide a single logical destination to which all the Edge Aggregation systems can deliver their messages, and a single logical source for distributing the logs out to the various Analysis Farms.

Logically, this is a single system (implemented as a load-balanced cluster of boxes as needed). If you only have one datacenter, you can easily collapse this functionality into your Edge Aggregators by multi-homing them with one leg on a network you use for your Analysis Farms. If you have multiple datacenters with a disaster recovery set of Analysis Farms, you will want to spread it across two datacenters that have Analysis Farms. The logs from each half of the Core Aggregator cluster should be sent to the other half so that both sets of Analysis Farms will see the full set of logs. The other option is to have multiple Core Aggregation clusters and have your Edge Aggregators send (and queue) logs to every Core cluster.

Because this farm of systems is handling the full log feed, a large enterprise will need to have these systems doing as little work as possible. Ideally, they should not be doing any processing of the log messages other than aggregating and delivering them.

When delivering a large volume of logs to many destinations (many different Analysis Farms), the resulting traffic can strain your network (as well as the systems doing the sending). One good way of dealing with this problem is to use Multicast MAC as I described in a 2012 LISA paper [3].

The Analysis Farms

Analysis Farms are the systems that do something with the logs, and because that includes lots of different things (especially in a large organization), doing this analysis can take many systems' worth of resources. So it's a good idea to think of the different sets of functionality as separate farms, even if you start off implementing multiple sets of functionality on one box (or an HA pair of boxes). This approach makes it much easier to split things apart as your needs grow.

In a large enterprise, it may not be reasonable for everyone who needs to see some logs to be able to do so. Depending on the capabilities of the tools that you use, you may opt to implement such restrictions in each tool, or you may choose to have multiple Analysis Farms of the same type, but filter the logs so that a given farm only contains the subset of logs that the users of that farm are allowed to have access to.

The following are a handful of basic functions that you need to have as part of your Analysis Farms.

Log Archiving

This can be as trivial as an HA pair of systems that just receives the logs and writes them to disk in simple gzip files for long-term data retention. In a more demanding environment, you could have these systems digitally sign the log files to make them tamper-resistant, encrypt the archives, and store the archives off-site.

Log Message Alerts

You can get started with Simple Event Correlator (SEC) on an HA pair of systems, and as your load climbs you can split the logs across different machines along the lines described in this LISA 2010 paper [4]. It's a very good idea to feed the alerts that are generated back into the system as new log messages that all other Analysis Farms can then see.

Reporting on Log Contents

Start by using rsyslog filtering to split logs into different files per application and then have simple scripts crunch these smaller files periodically. (I do hourly and daily reports this way.) SEC can also be useful for reporting. You can use a combination of Calendar rules and simple content matching rules to count occurrences of matches and output the counts at regular intervals.

Searching Logs

At low volumes, you can get by with zgrep, but as log volumes increase, this becomes unwieldy as a general purpose search tool; however, it's still great when looking at a small time window for specific data, especially when combined with rsyslog filtering to create files that only contain a given type of log. This is where Hadoop, Cassandra, ElasticSearch (all free), and Splunk (commercial) come into play.

Other Uses

Beyond the basic functions outlined above, Analysis Farms provide endless possibilities for other uses, among them:

- Artificial ignorance reporting
- Machine learning
- Predictive modeling
- Automated reactions

The really nice feature of this architecture is that you can add/remove Analysis Farms without having to reconfigure anything beyond the Core Aggregators (and if you use the Multicast MAC approach to distribute the data, you don't even have to reconfigure those). This lets you experiment freely with different tools without disrupting anything else. It also makes it hard for someone to generate a new set of logs and only send it to the analysis tool that they care about without it going to other groups (an app team forgetting to send the logs to the security team, for example).

Here's an example of a simple trick that you can implement to get a lot of value immediately. I like to add vmstat and iostat data to the logs. This both produces a tremendously dense set of performance related data with little impact to the systems and provides a heartbeat that you can use to detect if anything (including system failure) interrupts the logs. Doing this can be as simple as adding

```
nohup vmstat 60 |logger -t vmstat 2>&1 >/dev/null &
nohup iostat -xk 60 |logger -t iostat 2>&1 >/dev/null &
```

to your startup scripts. And a simple config to SEC similar to:

```
type=Single
ptype=perlfunc
pattern=sub {@test=split(' ', substr($_[0],16)); if ($test[1] =~
   /vmstat/ ) { return $test[0];} }
desc=vmstat_$1
action=create vmstat_heartbeat_$1 180 ( shellcmd sendmessage
   "$1" )
continue=takenext
```

Conclusion

The value that you get out of a logging system is related far more to the effort that you put into the system than the amount of money you spend on the system. You can get a lot of value quickly without spending a significant amount of money. I hope that this article helps provide a road map that you can use to get started dealing with your logs regardless of how much data you end up dealing with as your system grows.

References

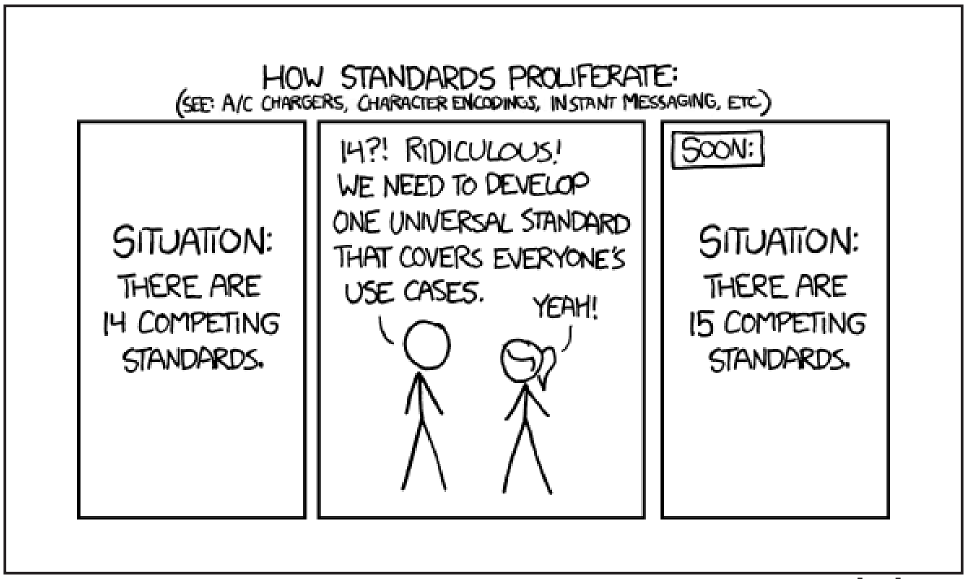
[1] http://json-ld.org/, http://cee.mitre.org/, https://fedorahosted.org/lumberjack/.

[2] http://linux-ha.org/wiki/Heartbeat and http://clusterlabs.org.

[3] David Lang, " Building a 100K log/sec Logging Infrastructure": https://www.usenix.org/conference/lisa12/building-100k-logsec-logging-infrastructure.

[4] Paul Krizak, "Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)": http://static.usenix.org/events/lisa10/tech/full_papers/Krizak.pdf.

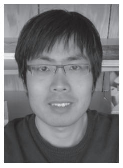
xkcd


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

xkcd.com

Cuckoo Filter: Better Than Bloom

BIN FAN, DAVID G. ANDERSEN, AND MICHAEL KAMINSKY



Bin Fan is a Ph.D. student in the Computer Science Department at Carnegie Mellon University. His research interests include networking systems, storage systems, and distributed systems. He is in the Parallel Data Lab (PDL) at CMU and also works closely with Intel Labs.
binfan@cs.cmu.edu

Michael Kaminsky is a Senior Research Scientist at Intel Labs and an adjunct faculty member in the Computer Science Department at Carnegie Mellon University. He is part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), based in Pittsburgh, PA. His research interests include distributed systems, operating systems, and networking.
michael.e.kaminsky@intel.com

David G. Andersen is an Associate Professor of Computer Science at Carnegie Mellon University. He completed his S.M. and Ph.D. degrees at MIT, and holds BS degrees in Biology and Computer Science from the University of Utah. In 1995, he co-founded an Internet Service Provider in Salt Lake City, Utah. dga@cs.cmu.edu

High-speed approximate set-membership tests are critical for many applications, and Bloom filters are used widely in practice, but do not support deletion. In this article, we describe a new data structure called the *cuckoo filter* that can replace Bloom filters for many approximate set-membership test applications. Cuckoo filters allow adding and removing items dynamically while achieving higher lookup performance, and also use less space than conventional, non-deletion-supporting Bloom filters for applications that require low false positive rates (ϵ< 3%).

Set-membership tests determine whether a given item is in a set or not. By allowing a small but tunable false positive probability, set-membership tests can be implemented by Bloom filters [1], which cost a constant number of bits per item. Bloom filters are efficient for representing large and static sets, and thus are widely used in many applications from caches and routers to databases; however, the existing items cannot be removed from the set without rebuilding the entire filter. In this article, we present a new, practical data structure that is better for applications that require low false positive probabilities, handle a mix of "yes" and "no" answers, or that need to delete items from the set.

Several proposals have extended classic Bloom filters to add support for deletion but with significant space overhead: *counting Bloom filters* [5] are four times larger and the recent *d-left counting Bloom filters* (dl-CBFs) [3, 2], which adopt a hash table-based approach, are still about twice as large as a space-optimized Bloom filter. This article shows that supporting deletion for approximate set-membership tests does not require higher space overhead than static data structures like Bloom filters. Our proposed cuckoo filter can replace both counting and traditional Bloom filters with three major advantages: (1) it supports adding and removing items dynamically; (2) it achieves higher lookup performance; and (3) it requires less space than a space-optimized Bloom filter when the target false positive rate ϵ is less than 3%. A cuckoo filter is a compact variant of a cuckoo hash table [7] that stores fingerprints (hash values) for each item inserted. Cuckoo hash tables can have more than 90% occupancy, which translates into high space efficiency when used for set membership.

Bloom Filter Background

Standard Bloom filters allow a tunable false positive rate ϵ so that a query returns either "definitely not" (with no error) or "probably yes" (with probability ϵ of being wrong). The lower ϵ is, the more space the filter requires. An empty Bloom filter is a bit array with all bits set to "0", and associates each item with k hash functions. To add an item, it hashes this item to k positions in the bit array, and then sets all k bits to "1". Lookup is processed similarly, except it reads k corresponding bits in the array: if all the bits are set, the query returns positive; otherwise it returns negative. Bloom filters do not support deletion, thus removing even a single item requires rebuilding the entire filter.

Counting Bloom filters support delete operations by extending the bit array to a counter array. An insert then increments the value of k counters instead of simply setting k bits, and lookup checks whether each of the required counters is non-zero. The delete operation decrements the values of the k counters. In practice the counter usually consists of four or more

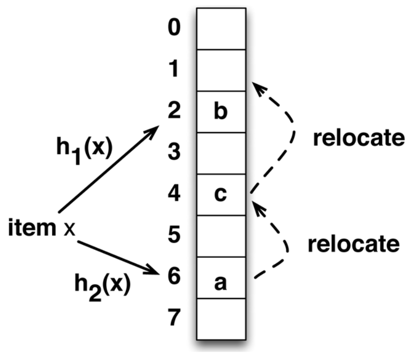


Figure 1: A cuckoo hash table with eight buckets

bits, and a counting Bloom filter therefore requires four times more space than a standard Bloom filter.

The work on d-left counting Bloom filters (dl-CBFs) [2, 3] is intellectually closest to our cuckoo filter. A dl-CBF constructs a hash table for all known items by *d-left hashing* [6], but replaces each item with a short fingerprint (i.e., a bit string derived from the item using a hash function). The dl-CBFs can reduce the space cost of counting Bloom filters, but still require twice the space of a space-optimized Bloom filter.

Cuckoo Filter

The cuckoo filter is a compact data structure for approximate set-membership queries where items can be added and removed dynamically in O(1) time. Essentially, it is a highly compact cuckoo hash table that stores fingerprints (i.e., short hash values) for each item.

Basic Cuckoo Hash Table

Cuckoo hashing is an open addressing hashing scheme to construct space-efficient hash tables [7]. A basic cuckoo hash table consists of an array of buckets where each item has two candidate buckets determined by hash functions $h_1(\cdot)$ and $h_2(\cdot)$ (see Figure 1). Looking up an item checks both buckets to see whether either contains this item. If either of its two buckets is empty, we can insert a new item into that free bucket; if neither bucket has space, it selects one of the candidate buckets (e.g., bucket 6), kicks out the existing item ("a"), and re-inserts this victim item to its own alternate location (bucket 4). Displacing the victim may also require kicking out another existing item ("c"), so this procedure may repeat until a vacant bucket is found, or until a maximum number of displacements is reached (e.g., 500 times in our implementation). If no vacant bucket is found, the hash table is considered too full to insert and an expansion process is scheduled. Though cuckoo hashing may execute a sequence of displacements, its amortized insertion time is still O(1). Cuckoo hashing ensures high space occupancy because it can refine earlier item-placement decisions when inserting new items.

Proper configuration of various cuckoo hash table parameters can ensure table occupancy more than 95%.

Dynamic Insert

When inserting new items, cuckoo hashing may relocate existing items to their alternate locations in order to make room for the new ones. Cuckoo filters, however, store only the items' fingerprints in the hash table and therefore have no way to read back and rehash the original items to find their alternate locations (as in traditional cuckoo hashing). We therefore propose *partial-key cuckoo hashing* to derive an item's alternate location using only its fingerprint. For an item x, our hashing scheme calculates the indexes of the two candidate buckets i_1 and i_2 as follows:

$$i_1 = \mathrm{HASH}(x),$$
$$i_2 = i_1 \oplus \mathrm{HASH}(x\text{'s fingerprint}).$$
Eq. (1)

The exclusive-or operation in Eq. (1) ensures an important property: i_1 can be computed using the same formula from i_2 and the fingerprint; therefore, to displace a key originally in bucket i (no matter whether i is i_1 or i_2), we can directly calculate its alternate bucket j from the current bucket index i and the fingerprint stored in this bucket by

$$j = i \oplus \mathrm{HASH}(\text{fingerprint}).$$
Eq. (2)

Hence, insertion can complete using only information in the table, and never has to retrieve the original item x.

Note that we hash the fingerprint before it is XOR-ed with the index of its current bucket, in order to help distribute the items uniformly in the table. If the alternate location is calculated by "i ⊕Fingerprint" without hashing the fingerprint, the items kicked out from nearby buckets will land close to each other in the table, assuming the size of the fingerprint is small compared to the table size. Hashing ensures that items kicked out can land in an entirely different part of the hash table.

Does Partial-Key Cuckoo Hashing Ensure High Occupancy?

The values of i_1 and i_2 calculated by Eq. (1) are uniformly distributed, individually. They are not, however, necessarily independent of each other (as required by standard cuckoo hashing). Given the value of i_1, the number of possible values of i_2 is at most 2^f where each fingerprint is f bits; when $f \le \log_2 r$ where r is the total number of buckets, the choice of i_2 is only a subset of all the r buckets of the entire hash table. For example, using one-byte fingerprints, given i_1 there are only up to 2^f=256 different possible values of i_2 across the entire table; thus i_1 and i_2 are dependent when the hash table contains more than 256 buckets. This situation is relatively common, for example, when the cuckoo

			# memory references lookup	
	Bits per item	**Load factor α**	Positive query	Negative query
Space-optimized Bloom filter	$1.44 \log_2(1/\epsilon)$	–	$\log_2(1/\epsilon)$	2
(2,4)-cuckoo filter	$(\log_2(\alpha/\epsilon)+3)/\alpha$	95.5%	2	2
(2,4)-cuckoo filter w/ semi-sort	$(\log_2(\alpha/\epsilon)+2)/\alpha$	95.5%	2	2

Table 1: Space and lookup cost of Bloom filters and two cuckoo filters

filter targets a large number of items but a moderately low false positive rate.

The table occupancy, though, can still be close to optimal in most cases (where optimal is when i_1 and i_2 are fully independent). We empirically show in the Evaluation section that this algorithm achieves close-to-optimal load when each fingerprint is sufficiently large.

Dynamic Delete

With partial-key cuckoo hashing, deletion is simple. Given an item to delete, we check both its candidate buckets; if there is a fingerprint match in either bucket, we just remove the fingerprint from that bucket. This deletion is safe even if two items stored in the same bucket happen to have the same fingerprint. For example, if item x and y have the same fingerprint, and both items can reside in bucket i_1, partial-key cuckoo hashing ensures that bucket $i_2 = i_1 \oplus$HASH(fingerprint) must be the other candidate bucket for both x and y. As a result, if we delete x, it does not matter if we remove the fingerprint added when inserting x or y; the membership of y will still return positive because there is one fingerprint left that must be reachable from either bucket i_1 and i_2.

Optimizing Space Efficiency

Set-Associativity: Increasing bucket capacity (i.e., each bucket may contain multiple fingerprints) can significantly improve the occupancy of a cuckoo hash table [4]; meanwhile, comparing more fingerprints on looking up each bucket also requires longer fingerprints to retain the same false positive rate (leading to larger tables). We explored different configuration settings and found that having four fingerprints per bucket achieves a sweet point in terms of the space overhead per item. In the following, we focus on the (2,4)-cuckoo filters that use two hash functions and four fingerprints per bucket.

Semi-Sorting: During lookup, the fingerprints (i.e., hashes) in a single bucket are compared against the item being tested; their relative order within this bucket does not affect query results. Based on this observation, we can compress each bucket to save one bit per item, by "semi-sorting" the fingerprints and encoding the sorted fingerprints. This compression scheme is similar to the "semi-sorting buckets" optimization used in [2]. Let us use the following example to illustrate how the compression works.

When each bucket contains four fingerprints and each fingerprint is four bits, an uncompressed bucket occupies 16 bits; however, if we sort all four four-bit fingerprints in this bucket, there are only 3,876 possible outcomes. If we precompute and store all of these 3,876 16-bit buckets in an extra table, and replace the original bucket with an index into the precomputed table, each bucket can be encoded by 12 bits rather than 16 bits, saving one bit per fingerprint (but requiring extra encoding/decoding tables).

Comparison with Bloom Filter

When is our proposed cuckoo filter better than Bloom filters? The answer depends on the goals of the applications. This section compares Bloom filters and cuckoo filters side-by-side using the metrics shown in Table 1 and several additional factors.

Space efficiency: Table 1 compares space-optimized Bloom filters and (2,4)-cuckoo filters with and without semi-sorting. Figure 2 further shows the trend of these schemes when varies from 0.001% to 10%. The information theoretical bound requires $\log_2(1/\epsilon)$ bits for each item, and an optimal Bloom filter uses 1.44 $\log_2(1/\epsilon)$ bits per item, or 44% overhead. (2,4)-cuckoo filters with semi-sorting are more space efficient than Bloom filters when < 3%.

Number of memory accesses: For Bloom filters with k hash functions, a positive query must read k bits from the bit array. For space-optimized Bloom filters that require $k=\log_2(1/\epsilon)$, when ϵ gets smaller, positive queries must probe more bits and are likely to have more cache line misses when reading each bit. For example, k equals 2 when $\epsilon = 25\%$, but the value quickly grows to 7 when $\epsilon = 1\%$, which is more commonly seen in practice. A negative query to a space optimized Bloom filter reads 2 bits on average before it returns, because half of the bits are set [8]. In contrast, any query to a cuckoo filter, positive or negative, always reads a fixed number of buckets, resulting in two cache line misses.

Static maximum capacity: The maximum number of entries a cuckoo filter can contain is limited. After reaching the maxi-

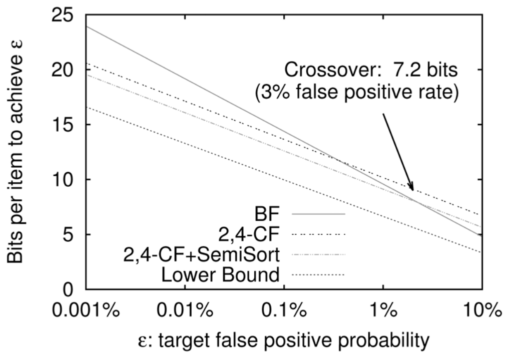


Figure 2: False positive rate vs. space cost per element. For low false positive rates (< 3%), cuckoo filters (CF) require fewer bits per element than the space-optimized Bloom filters (BF). The load factors to calculate space cost of cuckoo filters are obtained empirically.

mum load factor, insertions are likely to fail and the hash table must expand in order to store more items. In contrast, one can keep inserting new items into a Bloom filter at the cost of an increasing false positive rate. To maintain the same target false positive rate, the Bloom filter must also expand.

Limited duplicate insertion: If the cuckoo filter supports deletion, it must store multiple copies of the same item. Inserting the same item kb+1 times will cause the insertion to fail. This is similar to counting Bloom filters where duplicate insertion causes counter overflow. In contrast, there is no effect from inserting identical items multiple times into Bloom filters, or a non-deletable cuckoo filter.

Evaluation

We implemented a cuckoo filter in approximately 500 lines of C++ (https://github.com/efficient/cuckoofilter). To evaluate its space efficiency and lookup performance, we ran micro-benchmarks on a machine with Intel Xeon processors (L5640@2.27 GHz, 12 MB L3 cache) and 16 GB DRAM.

Load factor: As discussed above, partial-key cuckoo hashing relies on the fingerprint to calculate each item's alternate buckets. To show that the hash table still achieves high occupancy even when the hash functions are not fully independent,

f (bits)	mean of α	(gap to optimal)	variance of α
2	17.53%,	(-78.27%)	1.39%
4	67.67%,	(-28.13%)	8.06%
6	95.39%,	(-0.41%)	0.10%
8	95.62%,	(-0.18%)	0.18%
12	95.77%,	(-0.03%)	0.11%
16	95.80%,	(0.00%)	0.11%

Table 2: Load factor achieved by different f with (2,4)-cuckoo filter. Each point is the average of 10 runs.

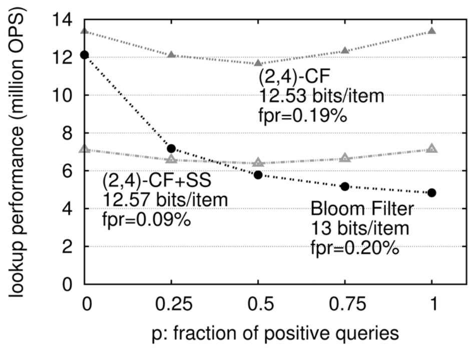


Figure 3: Lookup performance for a space-optimized Bloom filter and a (2,4)-cuckoo filter with a single thread. Each point is the average of 10 runs.

we built (2,4)-cuckoo filters using fingerprints of different sizes and measured the maximum load factor. We varied the fingerprint size from 2 bits to 16 bits, and each filter consists of 2^{25} (32 million) buckets. Keys are inserted to an empty filter until a single insertion relocates existing fingerprints more than 500 times (our "full" condition); then we stop and measure the mean and variance of achieved load factor α. As shown in Table 2, when the fingerprint is smaller than six bits, the table utilization is low, because the limited number of alternate buckets causes insertions to fail frequently. Once fingerprints exceed six bits, α approaches the optimal (i.e., that achieved using two fully independent hash functions).

Space efficiency: We measured the achieved false positive rates of Bloom filters and (2,4)-cuckoo filters with and without the semi-sorting optimization. When the Bloom filter uses 13 bits per item, it can achieve its lowest false positive rate of 0.20% with nine hash functions. With 12-bit fingerprints, the (2,4)-cuckoo filter uses slightly less space (12.53 bits/item), and its achieved false positive rate is 0.19%. When semi-sorting is used, a (2,4)-cuckoo filter can encode one more bit for each item and thus halve the false positive rate to 0.09%, using the same amount of space (12.57 bits/item).

Lookup Performance: After creating these filters, we also investigated the lookup performance for both positive and negative queries. We varied the fraction p of positive queries in the input workload from p=0% to 100%, shown in Figure 3. Each filter occupies about 200 MB (much larger than the L3 cache). The Bloom filter performs well when all queries are negative, because each lookup can return immediately after fetching the first "0" bit; however, its performance declines quickly when more queries are positive, because it incurs additional cache misses as it reads additional bits as part of the lookup. In contrast, a (2,4)-cuckoo filter always fetches two buckets in parallel, and thus achieves about the same, high performance for 100% positive queries and 100% negative queries. The performance drops slightly when p=50% because the CPU's branch prediction is least accurate (the probability of matching or not matching is exactly 1/2). A (2,4)-cuckoo filter with semi-sorting has a similar trend, but it is slower due to the extra encoding/decoding overhead when reading each bucket. In return for the performance penalty, the semi-sorting version reduces the false positive rate by half compared to the standard (2,4)-cuckoo filter. However, the cuckoo filter with semi-sorting still outperforms Bloom filters when more than 50% queries are positive.

References

[1] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Communications of the ACM,* vol. 13, no. 7 (1970), pp.422-426.

[2] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Bloom Filters via D-Left Hashing and Dynamic Bit Reassignment," in Proceedings of the Allerton Conference on Communication, Control and Computing, 2006.

[3] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An Improved Construction for Counting Bloom Filters," *14th Annual European Symposium on Algorithms,* 2006, pp. 684-695.

[4] U. Erlingsson, M. Manasse, and F. McSherry, "A Cool and Practical Alternative to Traditional Hash Tables, *Seventh Workshop on Distributed Data and Structures* (WDAS 2006), CA, USA, pp. 1-6.

[5] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking,* vol. 8, no. 3 (June 2000), pp. 281-293, doi: 10.1109/90.851975.

[6] M. Mitzenmacher and B. Vocking, "The Asymptotics of Selecting the Shortest of Two, Improved," *Proceedings of the Annual Allerton Conference on Communication Control and Computing* (1999), vol. 37, pp. 326-327.

[7] R. Pagh and F. Rodler, "Cuckoo Hashing," *Journal of Algorithms,* vol. 51, no. 2 (May 2004), pp.122-144.

[8] F. Putze, P. Sanders, and S. Johannes, "Cache-, Hash- and Space-Efficient Bloom Filters," *Experimental Algorithms* (Springer Berlin / Heidelberg, 2007), pp. 108-121.

A Short Primer on Causal Consistency

WYATT LLOYD, MICHAEL J. FREEDMAN, MICHAEL KAMINSKY, AND DAVID G. ANDERSEN

Wyatt Lloyd is a Postdoctoral Researcher at Facebook and will begin a position as an Assistant Professor at the University of Southern California in 2014. His research interests include the distributed systems and networking problems that underlie the architecture of large-scale Web sites, cloud computing, and big data. He received his Ph.D. from Princeton University in 2013 and his BS from Penn State University in 2007, both in Computer Science. Wyatt.Lloyd@gmail.com

Michael J. Freedman is an Associate Professor of Computer Science at Princeton University, with a research focus on distributed systems, networking, and security. Recent honors include a Presidential Early Career Award (PECASE), as well as early investigator awards through the NSF and ONR, a Sloan Fellowship, and DARPA CSSG membership. mfreed@cs.princeton.edu

Michael Kaminsky is a Senior Research Scientist at Intel Labs and is an adjunct faculty member of the Computer Science Department at Carnegie Mellon University. He is part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), based in Pittsburgh, PA. His research interests include distributed systems, operating systems, and networking. michael.e.kaminsky@intel.com

David G. Andersen is an Associate Professor of Computer Science at Carnegie Mellon University. He completed his S.M. and Ph.D. degrees at MIT, and holds BS degrees in Biology and Computer Science from the University of Utah. In 1995, he co-founded an Internet Service Provider in Salt Lake City, Utah. dga@cs.cmu.edu

The growing prevalence of geo-distributed services that span multiple geographically separate locations has triggered a resurgence of research on consistency for distributed storage. The CAP theorem and other earlier results prove that no distributed storage system can simultaneously provide all desirable properties—e.g., CAP shows this for strong Consistency, Availability, and Partition tolerance—and some must be sacrificed to enable others. In this article, we suggest causal consistency represents an excellent point in this tradeoff space; it is compatible with strong performance and liveness properties while being far easier to reason about than the previously-settled-for choice: "eventual" consistency.

Geo-distributed services are growing in popularity because they can survive datacenter failures and because they move services closer to end users, which lowers page load time and in turn drives up user engagement. For example, companies such as Facebook distribute their service across datacenters on the West Coast, East Coast, and Europe. The recent work in this space includes systems such as PNUTS [2], Walter [11], Gemini [6], Spanner [3], MDCC [5], and Bolt-on [1], as well as our own work on COPS [7] and Eiger [8].

So why does the increasing number of geo-distributed services make consistency a hot topic? Because there is a fundamental, unavoidable tradeoff between having guaranteed low-latency access (which we define as not having to send packets back-and-forth across the country) and making sure that every client sees a single ordering of all operations in the system (strong consistency) [7]. Guaranteed low latency is important because it keeps page load times low. Consistency is important because it makes systems easier to program. In our first work on this subject, COPS, we coined a term for low-latency-favoring systems: ALPS ("Availability, Low-latency, Partition tolerance, and Scalability"). This tradeoff is unavoidable as readers familiar with the famous CAP theorem might remember. Here's an example:

Consider concurrent writes and reads at two different datacenters. If you want both the write to have low latency and the read to have low latency, then you must satisfy them faster than the information can propagate to the other datacenter. In some circumstances, for example, a client might write data to the West Coast datacenter just before another client reads that object from the East Coast datacenter. The East Coast read will return stale information (i.e., it won't reflect that write that actually happened first) because, although the write completed on the West Coast, it hasn't propagated to the other datacenter. You could avoid this behavior and make the write take longer (wait for it to propagate to the East Coast) or the read take longer (fetch the data from the West Coast), but you cannot have both.

This tradeoff is pretty well understood, and is one of the several reasons behind the increasing prevalence of "eventual consistency," popularized by Amazon's Dynamo [4]. The other, of course, is availability: in this example, if the two datacenters cannot communicate, at least one of them must stop processing requests. Eventual consistency allows the datacenters to each return local results, rapidly, even if the other one is down. What it sacrifices, of course, is consistency: queries at different datacenters may see different results, in different order.

This is where causality comes in: you can provide something better than "eventual" consistency without sacrificing availability or low latency. That something is causal consistency, and it has been proved that no stronger form of consistency exists that can also guarantee low latency [9].

What Is Causal Consistency?

Causal consistency means that two events that are "causally" related (or potentially so) must appear in the same order. In other words, if action B occurred after action A (either because a user did A and then B, or because a different user saw A and then did B), then B must appear after A. As a concrete example, consider replying to a snarky comment on someone's Facebook post: your reply should be causally ordered after the snark. And, indeed, this is exactly what causally consistent replication can provide: your reply will never appear to have happened before the snark that triggered it.

Causal Consistency Is Good for Users

Causal consistency improves user experience because with it actions appear to everyone in the correct order. A common scenario where causality is important, but often isn't provided, is comments on social network posts, which sometimes appear out of order.

Consider this stream of posts:

> Oh no! My cat just jumped out the window.
> [a few minutes later] Whew, the catnip plant broke her fall.
> [reply from a friend] I love when that happens to cats!

It looks a little weird if what shows up on someone else's screen is:

> Oh no! My cat just jumped out the window.
> [reply from a friend] I love when that happens to cats!

There are even better examples, widely used, when talking about access control:

> [Removes boss from friends list]
> [Posts]: "My boss is the worst, I need a new job!"

If these two actions occur in the wrong order, then my post will not have been hidden from my boss as intended. Bad news bears.

Causal Consistency Is Good for Programmers

A stronger consistency model restricts the potential orderings of events that can show up at a remote datacenter. This simplifies the reasoning required of a programmer. Imagine two causally related events: Creating a new photo album and then uploading an image to it. If those events are replicated out-of-order, your code might have to try to cope with the idea of an image being uploaded to a nonexistent photo album. Or crash, because you never expected it to happen. In contrast, in a causally consistent system, you might never see the photo upload (or it could be delayed), but it will always occur after the creation of the album. This is the big win from causal consistency for programmers: They do not need to reason about out-of-order actions. Easier code, happier programmers.

What Are the Limitations of Causal Consistency?

Causal consistency is achievable with low latency, and it benefits users and programmers. But it has three drawbacks that practitioners should be aware of.

Drawback #1: Can only capture causality it sees. Actions that take place outside of the system are not seen and, unfortunately, not ordered by the system. A common example of this is a phone call: if I do action A, call my friend on another continent to tell her about A, and then she does action B, we will not capture the causal link between A and B.

Drawback #2: Cannot always enforce global invariants. Each datacenter in a causally consistent system is optimistic in that writes return once they are accepted in the local datacenter. This optimism makes it impossible to allow writes at every datacenter and guarantees global invariants, such as enforcing the rule that bank accounts never drop below 0 dollars.

True global invariants, however, may be rarer than you think. E-commerce is an often cited example, but online stores often handle stock that falls below 0 by issuing back orders for any sales that cannot be filled immediately. And readers familiar with the recent string of concurrent withdrawal attacks where bandits withdrew $40 million from 12 accounts [10] will recognize that even banks rarely enforce global invariants.

Drawback #3: Programmers must reason about concurrent writes. The optimism inherent in causality (when accepting writes at all datacenters) that prevents causal systems from enforcing global invariants also allows there to be concurrent writes to the same data. For instance, a person on the West Coast could update a data item while a person on the East Coast is simultaneously updating that same data item. What should a datacenter do when it has both updates? One common strategy—called the last-writer-wins rule or Thomas's write rule—is to pick one update arbitrarily and have it overwrite the other. This simple procedure is often sufficient: e.g., a social network user can only have one hometown.

There are situations, however, where a more complicated procedure is necessary. For instance, consider a friend request on the East Coast being accepted concurrently with a friend request on the West Coast. Each accepted friend request should increase the count of a user's friends by one (for a total of +2), but if we use the last-writer-wins rule, one update will overwrite the other (for only +1). Instead, we need programmers to write special functions to merge the concurrent updates together (that add the +1s together).

Reasoning about concurrent writes is the main difficulty with using causal consistency for programmers. Specifically, they must ask "are overwrite semantics sufficient?" and if they are not, they must write special functions that preserve the semantics they need.

Conclusion

Causal consistency is a better-than-eventual consistency model that still allows guaranteed low latency operations. It captures the causal relationships between operations and ensures that everyone sees operations in that order. This makes Web sites more intuitive for users, because their actions appear, and are applied, in the order they intended. Causal consistency also makes programming simpler by eliminating the need for programmers to reason about out-of-order operations.

References

[1] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "Bolt-on Causal Consistency," SIGMOD, June 2013.

[2] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni, "PNUTS: Yahoo!'s Hosted Data Serving Platform," VLDB, August 2008.

[3] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J.J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford, "Spanner: Google's Globally Distributed Database," OSDI, October 2012.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," SOSP, October 2007.

[5] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete, "MDCC: Multi-Datacenter Consistency," EuroSys, April 2013.

[6] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues, "Making Geo-Replicated Systems Fast as Possible, Consistent When Necessary," OSDI, October 2012.

[7] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen, "Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS," SOSP, October 2011.

[8] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen, "Stronger Semantics for Low-Latency Geo-Replicated Storage," NSDI, April 2013.

[9] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin, "Consistency, Availability, and Convergence," Technical Report TR-11-22, University of Texas at Austin, Department of Computer Science, 2011.

[10] Marc Santora, "In Hours, Thieves Took $45 Million in A.T.M. Scheme," *New York Times,* May 5, 2013.

[11] Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li, "Transactional Storage for Geo-Replicated Systems," SOSP, October 2011.

Arrakis: The Operating System as Control Plane

SIMON PETER AND THOMAS ANDERSON

Simon Peter is a post-doctoral research associate at the University of Washington, where his research focus is on operating systems and networks. Simon holds a Ph.D. in Computer Science from ETH Zurich, Switzerland, and is a founding member of the Barrelfish multi-core operating system research project. Simon has worked on many OS-related topics, including multi-core scheduling, distributed file systems, and distributed tracing, and contributes to various OS projects, including the Linux kernel, the GRUB2 boot loader, and the Debian distribution. simpeter@cs.washington.edu

Thomas Anderson is the Robert E. Dinning Professor of Computer Science and Engineering at the University of Washington. Professor Anderson is an ACM Fellow, and he has won the IEEE Koji Kobayashi Computers and Communications Award, the ACM SIGOPS Mark Weiser Award, the IEEE Communications Society William R. Bennett Prize, the NSF Presidential Faculty Fellowship, and the Alfred P. Sloan Research Fellowship. tom@cs.washington.edu

The recent trend toward hardware virtualization enables a new approach to the design of operating systems: instead of the operating system mediating access to the hardware, applications run directly on top of virtualized I/O devices, where the OS kernel provides only control plane services. This new division of labor is transparent to the application developer, but allows applications to offer better performance, security, and extensibility than was previously possible. After explaining the need for such an operating system design, we discuss the hardware and software challenges to realizing it and propose an implementation—Arrakis.

Consider a Web application, where one part executes within a Web service and another runs on the machine of an end user. On the service side it is important for operations to happen as efficiently as possible. Short response times are important to keeping users happy with the provided service, and if the application is executing in the cloud, the operator pays for the resources consumed. Users, on the other end, want to be as safe as possible from potentially buggy or malicious code that is now downloaded simply when they go to a Web page.

Unfortunately, today's operating systems are not designed to handle either of these cases efficiently. On the server side, the Web application might be created using multiple components, such as a MySQL database, an Apache Web server, and a Python language runtime, executing on top of an operating system like Linux. Figure 1 shows such an architecture. For every packet we handle on the network or database entry we read from the disk, we must invoke the Linux kernel and go through the various mechanisms it provides. This involves checking access permissions on system calls, data copies between user and kernel space, synchronization delays in shared OS services, and queues in device drivers to facilitate hardware multiplexing. Furthermore, hardware is typically virtualized in the cloud, and virtualization often requires another layer of multiplexing using another set of device drivers in the virtual machine monitor (VMM). Only after that is the I/O operation forwarded to the real hardware. As I/O performance keeps accelerating at a faster pace than single-core CPU speeds, this kind of interposition skews the I/O bottleneck to the operating system, which is mediating each application I/O operation in order to safely multiplex the hardware.

On the end-user side, we want fine-grained sandboxes to protect us from potentially harmful surprises from remote code of untrusted vendors, such as bugs and security holes. Systems such as Native Client (NaCl [6]) go to great lengths to provide a secure execution environment, while allowing the use of shared browser services, like the JavaScript runtime. Their task would be much simpler with the right level of hardware and OS support.

Driven by the commercial importance of cloud computing, hardware vendors have started to offer devices with virtualization features that can bypass the virtual machine monitor for many common guest OS operations. Among these are CPU virtualization, which has been around for several years, and I/O virtualization, which has entered the market recently. For example, an IOMMU makes device programming from a guest operating system safe, while Single-Root I/O Virtualization (SR-IOV) allows devices to do their own multiplexing and virtualization. Which hardware features are needed to improve the performance of our Web application beyond just bypassing the VMM?

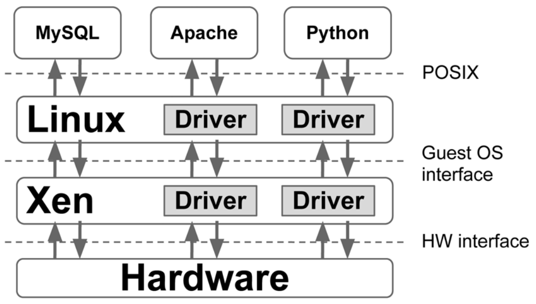


Figure 1: Application I/O paths for a virtualized Web service.

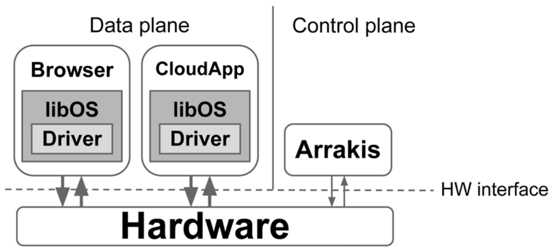


Figure 2: Arrakis I/O architecture

Hardware Support for User-Level Operating Systems

An inspiration for this work is the recent development of virtualizable network interfaces, such as the Intel X520 10 Gigabit Ethernet controller. These interfaces provide a separate pool of packet buffer descriptors for each virtual machine. The network interface demultiplexes incoming packets and delivers them into the appropriate virtual memory location based on the buffer descriptors set up by the guest operating system. Of course, the VMM still specifies which guest VMs are assigned to which virtual network device. Once the setup is done, however, the data path never touches the VMM. We would like to be able to demultiplex packets directly to applications, based on IP addresses and port numbers. For this to work, the network device needs to be more sophisticated, but Moore's Law favors hardware complexity that delivers better application performance, so such features are likely to be added in the future.

Entering the market now are hard disk controllers that allow hard disk partitions to be imported directly as virtual disks to guest operating systems. What we need is something more: the ability to give any application direct access to its own virtual disk blocks from user space. Unlike a fixed disk partition, applications could request the kernel to extend or shrink their allocation, as they are able to do for main memory today. The disk device maps the virtual disk block number to the physical location. Flash wear leveling and bad block remapping already support this type of virtualization. As with the network interface, the disk hardware would then read and write disk data directly to application memory.

An interesting research question we are investigating is whether we can efficiently simulate this model on top of existing hardware. The idea is to create a large number of disk partitions, which are then allocated as needed to different applications. Application data is spread across different partitions, but the application library synthesizes these partitions into a logical whole seen by the higher level code.

Power management can also be virtualized [4]. At the application level, knowing which devices need to be powered on and which can be put into low-power mode is easier. Applications are likely to know more about their present and future usage of a device, and therefore are capable of smarter power management than a device driver running within a traditional kernel.

Finally, Intel now supports multiple levels of (multi-level) page translation (Extended Page Tables [5]). The intent of this is to support direct read-write access by a guest operating system to its own page tables, without needing to trap into the hypervisor to reflect every change into the host kernel shadow page table seen by hardware. While useful for operating system virtualization, page translation hardware can also be used for a raft of application-level services, such as transparent, incremental checkpointing, external paging, user-level page allocation, and so forth.

Arrakis: The Operating System Is the Control Plane

What is required on the software side to allow applications direct hardware I/O? Ideally, we would like a world in which the operating system kernel is solely responsible for setting up and controlling data channels to hardware devices and memory. The hardware delivers data and enforces resource and protection boundaries on its own. Applications receive the full power of the unmediated hardware. To make this possible, we partition the operating system into a data plane and a control plane. This is in analogy to network routing, where the router OS is responsible for setting up data flows through the router that can occur without any software mediation.

Figure 2 shows this division in the Arrakis operating system. In Arrakis, the operating system (control plane) is only responsible for setting up hardware data channels and providing an interface to applications to allow them to request and relinquish access to I/O hardware, CPUs, and memory. Applications are able to operate directly on the unmediated hardware (data plane).

Direct hardware access may be made transparent to the application developer, as needed. We can link library operating systems into applications that can provide familiar abstractions and

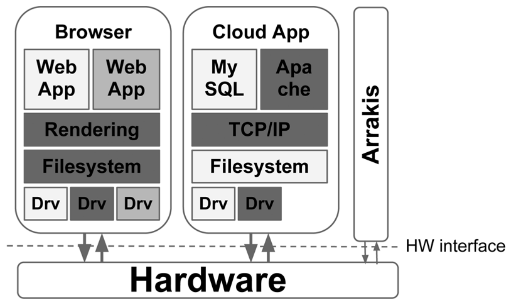


Figure 3: Example application containers containing a browser and a cloud application

mechanisms, such as the POSIX system call interface, thread scheduling, inter-processor communication, virtual memory management, file systems, and network stacks. These lightweight library operating systems execute within the same protection domain as the application.

The most important abstraction we are providing in Arrakis is that of an application container. An application container is a protection domain that provides a small interface to the Arrakis kernel to request the setup and tear down of unmediated channels to I/O hardware and memory, but otherwise provides the hardware itself. Figure 3 shows two such application containers. The Arrakis kernel is solely responsible for providing the mechanisms to allow allocating hardware resources to these containers, and, to allow applications to communicate, an interface to share memory, as well as a mechanism for directed context switches, akin to lightweight remote procedure calls (LRPC [1]), which facilitates low latency communication on a single core.

Use Cases

A number of applications can benefit from Arrakis' design, among them Web applications, databases, cloud computing, and high-performance computing (HPC) applications. We look back at Figure 3 and discuss two concrete examples of Web browsers and cloud applications within this section.

High-Performance Cloud Infrastructure

In Arrakis, we are able to execute the TCP/IP stack and network card device driver all within the same application and eliminate any system call protection boundary crossings, packet demultiplexing code, and kernel copy in/out operations that are typically required in a monolithic operating system. What's more, we can customize the network stack to better match the requirements of the Web server, down to the device driver. For example, the device driver could map packet buffers into the application in such a way that TCP/IP headers can be pre-fabricated and just mapped in front of the payload. The application can simply write the payload into the mapped buffer space. If packet checksumming is required, it can be offloaded to the network interface card.

A more complex cloud application may include a MySQL database server in addition to the Web server. The database is a fully trusted component of the cloud application; however, both MySQL and Apache ship within their own set of processes. Typically, these are connected via UNIX domain or TCP/IP sockets that need to be traversed for every request and the operating system has to be invoked for each traversal. This introduces overhead due to the required context switch, copy and access code operations, as well as OS code to ensure that data passed from one application to the other does not violate security. Avoiding these overheads can further reduce round-trip request latencies.

Arrakis allows us to run processes of both servers within the same protection domain. This eliminates most of the aforementioned overheads. Data can simply be remapped between applications, without sanity checks, and a context switch would not involve a journey through the operating system.

Application-Level Sandboxing

Web browsers have evolved into running a myriad of complex, untrusted Web applications that consist of native and managed code, such as HTML5 and JavaScript. These applications have access to low-level hardware and OS features, such as file systems and devices. Sandboxing this code is important to protect against security flaws and bugs that threaten system integrity.

In Arrakis, we are able to leverage hardware support for Extended Page Tables (EPT) to set up different protection domains within the browser. Each sandbox occupies a different protected address space within the browser's application container, with shared code and data mapped into all of its address spaces. This allows for a simple sandboxing implementation that, consequently, has a smaller attack surface.

Device drivers may be sandboxed as well using this approach. Furthermore, requesting channels to multiple virtual functions of the same hardware device from the kernel is possible. This allows us to replicate device drivers within the Web browser and run each replica within its own protection domain directly on these virtual functions multiplexed by the hardware. For example, we can request a virtual function per Web application and run the driver replica within that Web application. If a buggy device driver fails, only the Web application instance that triggered the failure will have to be restarted. The failure will not impact the rest of the browser environment or, worse, the rest of the system.

Lightweight Sharing

Providing Arrakis would be relatively easy if applications were complete silos—we could just run each application in its own lightweight virtual machine and be done. Our interest is also in providing the same lightweight sharing between applications as in a traditional operating system, so the user sees one file system, not many partitions, and applications are able to share code and data segments between different processes. How might this be done?

In Arrakis, an application can directly read and write its file and directory data to disk, without kernel mediation. File layout and recovery semantics are up to the application; for example, a Web browser cache might use a write-anywhere format, since losing several seconds of data is not important, while others might use traditional write-ahead logging. In the common case, most files are used only by the applications that wrote them; however, we still need to be able to support transparent access by other applications and system utilities, such as system-wide keyword search and file backup. How do we design OS services that efficiently allow the same sharing among multiple applications as that offered by operating systems that mediate each I/O operation?

To achieve this, the format of files and directories is independent of name look up. In Arrakis, we insert a level of indirection, akin to NFS vnodes. When a file name look up reaches an application-specific directory or file, the kernel returns a capability associated with the application handling storage of the corresponding file. The capability is used to access the file's contents, by invoking a file memory mapping interface that is provided by the storage handling application's library operating system. This allows us to share files safely and efficiently among untrusted applications.

Related Work

The security/performance tradeoffs of monolithic operating system designs have been of concern several times in the past. Particularly relevant are Exokernel [3] and the SPIN operating system [2].

Exokernel tried to eliminate operating system abstractions, and thus allowed applications to implement their own. Applications can link library operating systems that contain the abstractions that fit best with an application's operation. Note that it was not possible to set up several protection domains within a library operating system and thus sandboxing was equally difficult as in today's operating systems. Furthermore, to be able to safely multiplex a single hardware device to multiple library operating systems, Exokernel had to resort to the use of domain-specific languages that had to be uploaded into the kernel for proper disk and network multiplexing.

SPIN allowed uploading application-specific extensions into the operating system kernel. This way, applications could access the hardware and OS services more directly and gain a speedup. To make this safe and protect the rest of the system from potentially buggy or malicious extensions that were executing in supervisor mode, SPIN required the use of a type safe programming language (Modula-3) for extension development. This allowed for an extension to be checked against all its accesses before executing it within the OS kernel, but required the implementation of all extensions within this language.

Conclusion

Now is the time to take a fresh look at the division of labor between the operating system, applications, and hardware. Recent hardware trends are enabling applications to become miniature operating systems, with direct I/O and virtual memory access, while safety and resource boundaries are enforced by the hardware.

We propose a division of the operating system into a control plane and a data plane that allows applications direct access to the hardware in the common case. Applications can provide their own storage, network, process, and memory management without mediation by the operating system kernel.

We are in the early stages of developing the Arrakis operating system. Our Web site, http://arrakis.cs.washington.edu/, provides further information and development status updates.

References

[1] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy, "Lightweight Remote Procedure Call," *Proceedings of the 12th ACM Symposium on Operating Systems Principles,* December 1989, pp. 102-113.

[2] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers, "Extensibility, Safety and Performance in the SPIN Operating System," *Proceedings of the 15th ACM Symposium on Operating Systems Principles,* December 1995, pp. 267-284.

[3] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr., "Exokernel: An Operating System Architecture for Application-Level Resource Management," *Proceedings of the 15th ACM Symposium on Operating Systems Principles,* December 1995, pp. 251-266.

[4] R. Nathuji and K. Schwan, "Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems," *Proceedings of the 21st ACM Symposium on Operating Systems Principles,* October 2007, pp. 265-278.

[5] Intel 64 and IA-32 Architectures Software Developer's Manual, August 2012.

[6] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native Client: A Sandbox for Portable, Untrusted X86 Native Code," *Communications of the ACM,* vol. 53, no. 1, January 2010, pp. 91-99.

Practical Perl Tools

Git Smart

DAVID N. BLANK-EDELMAN

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs. David is honored to have been the recipient of the 2009 SAGE Outstanding Achievement Award and to serve on the USENIX Board of Directors beginning in June of 2010. dnb@ccs.neu.edu

In the very first paragraph, this column will attempt to be both contrite and useful (even to people who don't like Perl). This issue, we'll explore how Perl can improve your Git (git-scm.com) experience. But first, I must confess: I stole the title of this column from a most excellent Ruby gem (the heresy of mentioning it here!) found at github.com/geelen/git-smart. This gem adds a new subcommand "smart-pull" that knows how to do things like automatically stash work in progress so a "git pull" can succeed. Do check it out. But enough Ruby dalliance, let's see what Perl can do for us.

Oh, okay, just a little more dalliance as a small introduction. In this column I'm not going to spend virtually any time talking about what Git is, why you would to use it (see git-scm.com), or even how to use it (see the many, many "Git's not so bad once you learn how it works . . . here's a bunch of lollipop diagrams" articles on the Net for that). I will say that I have been thoroughly enjoying (ever) learning and using Git over the past year and a half or so. There's definitely a similarity between Perl and Git. They both share a certain internally consistent obtuseness that yields a great deal of power and productivity upon greater study. Given this, I think it is interesting to take a look at what happens when the two worlds collide.

Me Git Pretty Someday

The first category of Perl-Git convergence comes in the form of adding more spiffy to some existing Git commands. For example, App::Git::Spark lets you type "git spark {arguments}" to see a visual representation of the commit activity of a particular contributor. It uses sparklines (a term coined by Edward Tufte: www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0001OR&topic_id=1—they are cool, tiny, inline charts) to show how many commits took place over a certain time period. Here's a quick example that shows the number of commits to the repository broken out by week for the last eight weeks:

```
$ git vspark -w 8 dnb
Commits by dnb over the last 8 weeks
total: 183   avg: 23   max:      45
        4  █
       36  █████████▏
       30  ███████▍
        7  █▊
       18  ████▌
       19  ████▊
       24  ██████
       45  ███████████▎
```

Another subcommand is added in a similar fashion by the App::gitfancy module. When installed (and put in your path) you can type "git fancy {arguments}" and it will print out "a more readable graph" than the standard "git log" command provides (so brag the docs). This graph is similar to the Git log output I've heard called "the train tracks" that attempts to show the way the different branches have diverged and merged into the master branch of a project. So instead of the output of

```
$ git log --graph --oneline
```

looking like this:

```
*   0a490db Merge branch 'devel' into production
|\
| * 729cfd5 removing cups from s_desktop
| * 3118ab4 everything but restricted gets cups
| * fcfaf8b No need for gdm in the server class
* | 4310280 adding dependency repo to puppet list
|/
* 8dd68f3 adding subversion to all managed machines
* 78349f7 fixing order of facts
```

showing how some work branched off of the master at 8dd68f3 later to be merged back in at 0a490db, we can use "git fancy" and see:

```
| M   *0a490db (h) prod (HEAD, origin/prod, prod) Merge
      branch 'devel' into prod
.-+
0 |   729cfd5 (r) origin/devel removing cups from s_desktop
0 |   3118ab4 (r) origin/devel everything but restricted gets
      cups
0 |   fcfaf8b (r) origin/devel No need for gdm in the server
      class
| 0   4310280 (h) prod adding dependency repo to puppet list
0-^   8dd68f3 (r) origin/devel adding subversion to all
      managed machines
0     78349f7 (r) origin/devel fixing order of facts
```

Besides the cute ASCII graphics and the color (which you can't see), it is doing a number of things to the output, such as using one column per each branch, displaying clearly where the Merge took place (the M character on the line), distinguishing the branches from each other, and so on.

One last subcommand in the same vein if perhaps only to prove it is possible to have too much of a good thing: the module Git::Glog claims to provide a "spicey [sic] git-log with a hint of gravatars, nutmeg and cinnamon."

If for some reason you've always dreamed of seeing a person's gravatar ("Your Gravatar is an image that follows you from site to site appearing beside your name when you do things like comment or post on a blog" according to www.gravatar.com) next to a person's name in the "git log" output, you may have to contain your excitement when I tell you your dream has come true. Hopefully, this excitement isn't too diminished when I mention that the picture you see when typing "git glog" is actually an ASCII down-rez'd version of your gravatar (think blocky, really blocky, and largely unrecognizable). I come right up to the edge of understanding why you might want to use this module but don't quite get there. I'm including it in this column less as a cautionary tale and more as a source of inspiration for the sorts of "out there" things you could implement.

Dancing Git

The next category of Perl-Git interactions isn't nearly as snazzy because it is fairly obvious and straightforward. At some point you may want to perform operations on a Git repository from Perl. There are two directions you can go when looking for a module for this purpose. The first, more experimental route is to find a module that makes use of the (again more experimental) libgit2 C library. As a small aside, I first heard of libgit2 in Vicent Marti's great talk called "My Mom Told Me That Git Doesn't Scale" (which you can watch at vimeo.com/53261709 as of the time of this writing). The reason why I'm repeating "more experimental" so many times is that these modules seem a bit less polished to me (and indeed libgit2 may also fall into that category though it has really come a long way). Modules in this category include Git::Raw and Git::XS.

The other kind of module calls the standard "git" binary directly. It is likely to be less efficient but more solid in the short term. We're going to look at one of the modules that works this way: Git::Repository. Working with Git::Repository is, as I mentioned before, fairly obvious and straightforward if you know which Git command lines you would normally execute by hand.

The first step is to create a Git::Repository object pointing either at the working directory:

```
use Git::Repository;
$repo = Git::Repository->new( work_tree => $directory );
```

or the bare repository (the something.git directory):

```
    $repo = Git::Repository->new( git_dir => $bare_repo_dir );
```

or both if need be:

```
$repo = Git::Repository->new( work_tree => $directory,
                              git_dir   => $bare_repo_dir  );
```

And from there we call run() with the Git command we'd like to perform. If by hand, you would type:

```
$ git add wp-content/plugins
$ git commit -m 'updating WP plugins'
```

The Perl version would be:

```
use Git::Repository;

$repo = Git::Repository->new( git_dir => $bare_repo_dir );
$repo->run( add    => 'wp-content/plugins' );
$repo->run( commit => 'updating WP plugins' );
```

Pretty simple, no? My especially eagle-eyed readers might notice that when you call Git on the command line, it sometimes provides (what it thinks is helpful) output in response to your commands. Anything sent to STDERR by the commands is just printed to STDERR by the code above. If you'd prefer to capture the STDERR output so your code can change its behavior accordingly, instead of calling run(), you would call the command() method. It essentially provides a handle that you read from:

```
my $output = $repo->command( commit => 'updating WP plugins' );
print $output->stderr->getlines(); # prints the STDERR output
print $output->stdout->getlines(); # prints the STDOUT output
$output->close;
```

Git::Repository has some other nice methods for working with the Git command line. See the Git::Repository::Tutorial and other documentation in the package. There are a number of other possible Perl modules that perform a similar function, including Git::Wrapper and VCI (a version control system-agnostic framework).

Git Me More

Given the number of modules that fall into this category, I would say that there is a burning need out in the larger community for a solution that helps you manage multiple Git repositories at the same time. Let's say you have a "build" directory that includes a bunch of working directories in it, each a clone of a different remote repository containing the components that knit together. You can easily imagine wanting to be able to perform a pull on all of the repositories so you have the latest version of all of the components included before beginning a build. Modules that help with this problem include Group::Git, App::Rgit, Git::Bunch, App::GitGot, GitMeta, mr (found at http://joeyh.name/code/mr/), and rgit. I'll demonstrate two of these but I recommend checking them all out to see which one most closely matches your particular needs and work style. They are pretty similar, though some have features that might scratch your specific itch (for example, mr knows how to handle "any combination [of] subversion, git, cvs, mercurial, bzr, darcs, cvs, vcsh, fossil and veracity repositories").

Most of these modules are not designed to be used directly by a programmer; they largely serve as the library behind a new command line script run to perform your actions. For example, App::GitGot provides a "got" command, App::Rgit provides "rgit", Group::Git provides "group-git" and so on. Given that, let me show you some command line examples from the first two I just mentioned.

For "got", we can type

```
$ cd working-directory-of-a-repo
$ got add # will prompt you for info about that repo
```

and "got" will add it to a list of repositories it is tracking for you (the list can be seen with "got list"). To run a command on all of those repositories, it is just something along the lines of

```
$ got status
```

to see something like this:

```
1) ldap-config   : OK
2) migration     : OK
3) puppet        : OK
```

To work on a single repository, you can ask for it by name, as in:

```
$ got status puppet
```

Even spiffier, you can also

```
$ got cd puppet
```

and it will spawn a shell right in that repo's working directory.

For a slightly less "sticky" experience (i.e., one that doesn't require you to explicitly track certain repositories), rget is lovely. It lets you perform operations on all of the Git repositories found in or below a certain directory (i.e., "recursive git"):

```
# show the status for all of the repositories in/below current dir
$ rgit status
```

One nice feature is it defines special tokens that are set based on the repository being worked on. For example: %n is the current repository name and %b becomes what it calls a "bareified relative path." The documentation shows these examples of token use:

```
# Tag all the repositories with their name
$ rgit tag %n

# Add a remote to all repositories in "/foo/bar" to their
# bare counterpart in qux on host
GIT_DIR="/foo/bar" rgit remote add host git://host/qux/%b
```

Captain Hook

If we want to move away from talking about command lines and into the backend of administering Git repositories, we should talk a bit about hooks. If you've used hooks with another version control system like Subversion, you've probably encountered the idea that the version control software could call scripts when certain actions like commits take place. I mentioned SVN intentionally because it ships with a Perl script called "commit-email.pl" (sometimes packaged in a separate subversion-tools package). This script is meant to be called after each commit has taken place so that the owner of the repo can receive email notification of actions on that repository.

Git has a similar hook system, and indeed there are Perl modules meant to help make use of it. For example, the Git::Hooks package offers a system for having a single script handle all of your hooks. In this script you define sections (from the doc):

```
PRE_COMMIT {
    my ($git) = @_;
    # ...
};

COMMIT_MSG {
    my ($git, $msg_file) = @_;
    # ...
};
```

The documentation shows how you can implement hooks that restrict commits to being over a certain size or matching certain Perl::Critic standards. It also provides a few plugins for further extending the system.

If you need something a little simpler, Git::Hook::PostReceive parses an incoming commit and makes it easy to work with its contents. Here's the example from the documentation:

```
# hooks/post-receive
use Git::Hook::PostReceive;
my $payload = Git::Hook::PostReceive->new->read_stdin( <STDIN> );


$payload->{new_head};
$payload->{delete};


$payload->{before};
$payload->{after};
$payload->{ref_type}; # tags or heads

for my $commit (@{ $payload->{commits} } ) {
    $commit->{id};
    $commit->{author}->{name};
    $commit->{author}->{email};
    $commit->{message};
    $commit->{date};
}
```

Do Something Interesting

As a way of ending this column, I wanted to show one last interesting intersection of the two worlds. We haven't seen all of the possible connections (e.g., there are a number of useful modules for interacting with the very popular GitHub service like Net::Github, Pithub, and Github::Fork::Parent), but this one deserves special mention.

The GitStore module lets you use a Git repository as a "versioned data store." To give credit where credit is due, this module was inspired by an article from 2008 called "Using Git as a Versioned Data Store in Python" (newartisans.com/2008/05/using-git-as-a-versioned-data-store-in-python/) and its subsequent reimplementation in Ruby. The main premise is you can point this module at a repo and then put "stuff" into that repo, creating versions as you desire. Here's the sample code from the documentation:

```
use GitStore;

my $gs = GitStore->new('/path/to/repo');
$gs->set( 'users/obj.txt', $obj );
$gs->set( ['config', 'wiki.txt'], { hash_ref => 1 } );
$gs->commit();
$gs->set( 'yyy/xxx.log', 'Log me' );
$gs->discard();

# later or in another pl
my $val = $gs->get( 'user/obj.txt' ); # $val is the same as $obj
# $val is { hashref => 1 } );
my $val = $gs->get( 'config/wiki.txt' );


# $val is undef since discard
my $val = $gs->get( ['yyy', 'xxx.log' ] );
```

I said "stuff" above because you can place all sorts of things into the datastore: objects, data structures, contents of variables, etc. In the first section, you can see that we are storing Perl objects under some name in the datastore. In the first set() line, the object is stored under the name "users/obj.txt" and is retrieved using this key in the get() example. The really cool part of this example can be found in the commit() call. With that call, we're doing a "git commit," and hence are committing that version of the datastore. This may not be the fastest datastore available, but for certain applications it is pretty darn cool.

Take care and I'll see you next time.

Building a Better Dictionary

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009) and *Python Cookbook* (3rd Edition, O'Reilly & Associates, 2013). He is also known as the creator of Swig (http://www.swig.org) and Python Lex-Yacc (http://www.dabeaz.com/ply/index.html). Beazley is based in Chicago, where he also teaches a variety of Python courses. dave@dabeaz.com

One of the software projects that I maintain is the PLY parser generator (http://www.dabeaz.com/ply). In a nutshell, PLY is a Python implementation of the classic lex and yacc tools used for writing parsers, compilers, and other related programs. It's also not the kind of program that tends to change often—to be sure, I'm not aware of any sort of space-race concerning the implementation of LALR(1) parser generators (although perhaps there's some startup company Lalrly.com just waiting to strike parsing gold).

As a stable piece of software, PLY only receives occasional bug reports, which are mostly in the form of minor feature requests; however, I recently received a report that PLY was randomly failing its unit tests on Python 3.3. Specifically, if you ran its unit test suite twice in succession, different sets of unit tests would fail each time. For a program involving no randomness or threads, this development was puzzling to say the least.

This problem of randomly failing unit tests was ultimately tracked down to a recent security-related change in Python's dictionary implementation. I'll describe this change a bit later, but this incident got me thinking about the bigger picture of Python dictionaries. If anything, it's safe to say that the dictionary is part of the bedrock that underlies the entire Python interpreter. Major parts of the Python language, such as modules and objects, use dictionaries extensively. Moreover, they are widely used as data structures in user applications. Last, but not least, the implementation of dictionaries is one of the most studied and tuned parts of the interpreter.

Given their importance, you might think that the dictionary implementation would be something that's set in stone. To be sure, Python's core developers are reluctant to make changes to something so important; however, in the past couple of years, the implementation of dictionaries has been evolving in interesting and unusual ways. In this article, I hope to peel back the covers a little bit and discuss how dictionaries work along with some notable recent changes.

Dictionaries as Data Structures

Most Python programmers are familiar with using a dictionary as a simple data structure. For example:

```
s = {
   'name': 'ACME',
   'shares': 100,
   'price': 123.45
}
```

A dictionary is simply a mapping of keys to values. To perform calculations, you simply access the key names:

```
>>> s['shares'] * s['price']
12345.0
>>> s['shares'] = 75
>>> s['name']
'ACME'
>>>
```

Dictionaries are unordered. Thus, if you look at the ordering of the keys, they're usually not in the same order as originally specified when the dictionary was created. For example:

```
>>> s
{'price': 123.45, 'name': 'ACME', 'shares': 75}
>>> s.keys()
['price', 'name', 'shares']
>>>
```

Although the lack of ordering sometimes surprises newcomers, it's not something that causes concern in most programs; it's just an artifact of the implementation.

From dictionaries to classes is only a small step. For example, suppose you have a class like this:

```
class Stock(object):
    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

If you make an instance, it's actually just a thin wrapper around a dictionary. For example:

```
>>> s = Stock('ACME', 100, 123.45)
>>> s.shares * s.price
12345.0
>>> s.__dict__
{'price': 123.45, 'name': 'ACME', 'shares': 100}
>>>
```

Naturally, most of this is old news to anyone who's been programming in Python for a while.

Dictionary Implementation

Under the covers, dictionaries are implemented as hash tables. Each entry in a dictionary is represented by a structure (hashval, key, value) where hashval is an integer hashing code, key is a pointer to the key value, and value is a pointer to the value. The special hash value used in this triple is not something you normally think about, but it's easily obtained using the built-in hash() function (note: to get examples that exactly match what's shown, use Python 2 compiled for a 64-bit platform):

```
>>> hash('name')
-4166578487145698715
>>> hash('shares')
-5046406209814648658
>>>
```

When an empty dictionary is first created, a small eight-element array of dictionary entry structures is allocated. Entries are inserted into this array at positions determined by bit-masking the above integer hash codes. For example:

```
>>> hash('name') & 7
5
>>> hash('shares') & 7
6
>>> hash('price') & 7
2
>>>
```

The numerical order of the above positions determine the order in which keys will appear when you look at a dictionary. For example:

```
>>> s.keys()
['price', 'name', 'shares']
>>>
```

If you add a new key to a dictionary, its insertion position is determined in the same way. For example:

```
>>> hash('time') & 7
7
>>> s['time'] = '9:45am'
>>> s
{'price': 123.45, 'name': 'ACME', 'shares': 75, 'time': '9:45am'}
>>>
```

If two keys map to the same index, a new position is found by repeatedly perturbing the index to a new value until a free slot is found. Without explaining the rationale for the mathematical details, the following session illustrates what happens if you add a new entry `s['account'] = 1` to the above dictionary:

```
>>> hval = hash('account')
>>> index = hval & 7
>>> index           # Collision with "price"
2
>>> perturb = hval
>>> index = (index << 2) + index + perturb + 1
>>> index & 7        # Collision with "name"
5
>>> perturb >>= 5
>>> index = (index << 2) + index + perturb + 1
>>> index & 7        # Collision with "name"
5
```

```
>>> perturb >>= 5
>>> index = (index << 2) + index + perturb + 1
>>> index & 7        # Free slot: position 0
0L
>>>
```

Indeed, if you try it, you'll find that the new entry appears first in the resulting dictionary:

```
>>> s['account'] = 1
>>> s
{'account': 1, 'price': 123.45, 'name': 'ACME', 'shares': 75, 'time':
    '9:45am'}
>>>
```

As dictionaries fill up, that collisions will occur and performance will degrade becomes increasingly more likely (for instance, notice that four different table positions were checked in the above example). Because of this, the size of the array used to hold the contents of a dictionary is increased by a factor of four whenever a dictionary becomes more than two-thirds full. This is a rather subtle implementation detail, but you can notice it if you carefully observe what happens if you add a sixth entry to the above dictionary:

```
>>> s
{'account': 1, 'price': 123.45, 'name': 'ACME', 'shares': 75, 'time':
    '9:45am'}
>>> s['date'] = '05/26/2013'
>>> s
{'account': 1, 'name': 'ACME', 'price': 123.45, 'shares': 75,
    'time':'9:45am', 'date': '05/26/2013'}
>>>
```

Notice how 'name' and 'price' swapped places when the next item was inserted. This is due to an expansion of the dictionary size from 8 to 32 entries and a recomputation of the hash table positions. In the new dictionary, the new positions for 'name' and 'price' are as follows:

```
>>> hash('name') & 31
5
>>> hash('price') & 31
10
>>>
```

To be fair, these kinds of details are not something that most programmers ever need to concern themselves with other than to realize that dictionaries involve some extra overhead both in computation and memory.

Digression: Dictionary Alternatives

If you're using dictionaries to store a lot of small data structures, it's probably worth noting that there are much more efficient alternatives available. For example, even a small dictionary has a memory footprint larger than you might expect:

```
>>> s = { 'name': 'ACME', 'shares': 100, 'price': 123.45}
>>> import sys
>>> sys.getsizeof(s)
280
>>>
```

Here you see that the dictionary is 280 bytes in size (actually, 296 bytes in Python 3.3). Keep in mind, that this size is just for the dictionary itself, not for the items stored inside. If this seems like a lot, you're right. The extra overhead can add up significantly if creating a large number of small data structures (e.g., imagine a program that's read a million line CSV file into a list of dictionaries representing each row).

Class instances are even more inefficient, adding an additional 64 bytes of overhead to the total size. In fact, a basic instance with no data at all requires 344 bytes of storage when one adds up all of the parts. For example:

```
>>> s = Stock('ACME', 100, 123.4)
>>> sys.getsizeof(s)
64
>>> sys.getsizeof(s.__dict__)
280
>>>
```

If you're working with data, there are some better choices. One such option is to create a named tuple:

```
>>> from collections import namedtuple
>>> Stock = namedtuple('Stock', ['name', 'shares', 'price'])
>>> s = Stock('ACME', 100, 123.45)
>>> s.name
'ACME'
>>> s.shares * s.price
12345.0
>>> sys.getsizeof(s)
80
>>>
```

A named tuple gives you the nice attribute access normally associated with a class and much more compact representation; however, as a tuple, the attributes are immutable. If you need mutability, consider defining a class with `__slots__` instead:

```
class Stock(object):
    __slots__ = ('name', 'shares', 'price')
    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

This produces an even more compact representation:

```
>>> s = Stock('ACME', 100, 123.45)
>>> s.name
'ACME'
>>> s.shares = 75
>>> sys.getsizeof(s)
72
>>> hasattr(s, '__dict__')    # No underlying __dict__
False
>>>
```

The use of `__slots__` on a class is actually the most compact representation of a data structure in Python without resorting to lower-level hacks such as binary encodings or C extensions. It's even smaller than using a tuple:

```
>>> s = ('ACME', 100, 123.45)
>>> sys.getsizeof(s)
80
>>>
```

Therefore, if you're working with a lot of data, and you're thinking about using dictionaries because of their programming convenience, consider some of these alternatives instead.

Randomized Key Ordering

In late 2011, a new kind of denial-of-service attack that exploited hash-table collisions was unveiled (see "Efficient Denial of Service Attacks on Web Application Platforms" at http://events.ccc.de/congress/2011/Fahrplan/events/4680.en.html). Without going into too many details, this attack involves sending carefully crafted requests to a Web server that push Python's hash-table collision handling algorithm into worst-case O(n**2) performance—the end result of which is that a clever hacker can make a server consume vast numbers of CPU cycles.

To combat this, Python now randomly salts the computation of hash values from run-to-run of the interpreter. This is something that is enabled by default in Python 3.3 or that can be enabled by the -R option to the interpreter in Python 2.7. For example:

```
bash % python -R
>>> s = {'name': 'ACME', 'shares':100, 'price':123.45 }
>>> s
{'shares': 100, 'name': 'ACME', 'price': 123.45}
>>>
```

```
bash-3.2$ python -R
>>> s = { 'name': 'ACME', 'shares':100, 'price':123.45 }
>>> s
{'name': 'ACME', 'shares': 100, 'price': 123.45}
>>>
```

The random salting makes it impractical for an attacker to construct requests that will work everywhere; however, the randomization can also cause funny things to happen in certain programs that use dictionaries.

In the case of PLY, randomness of dictionary order changed the numbering of states in a large automatically created state machine. This, in turn, caused a certain randomness in the ordering of output messages being checked by unit tests.

Although random ordering is harmless to the overall execution of the program, I had to fix a number of unit tests to take it into account. I also selectively introduced a few uses of OrderedDict instances (from the collections module) to force a predictable order on data structures of critical importance to the construction of state tables.

Split-Key Dictionaries

Python 3.3 introduces yet another improvement on dictionaries related to their use in class instances. In a class such as the Stock class presented earlier, observe that every instance is going to have exactly the same set of keys. Taking this observation into account, Python 3.3 dictionaries actually have two internal representations; a combined representation where keys and values are stored together and a split representation where the keys are only stored once and shared among many different dictionaries.

For instances, the more compact split representation is used. This is a bit hard to view directly, but here is a simple example that shows the impact on the memory footprint:

```
>>> s = Stock('ACME', 100, 123.45)
>>> sys.getsizeof(s)
64
>>> sys.getsizeof(s.__dict__)   # Note: Greatly reduced size
104
>>>
```

Indeed, if you try a further experiment in which you create one million identical instances, you'll find the total memory use to be about 169 MB. On the other hand, creating one million identical dictionaries requires almost 293 MB.

This change in implementation is interesting in that it now makes the use of a class a much better choice for storing data structures if you care about memory use. The only downside is that all benefits are lost if you perform any manipulation of instances that add attributes outside of the __init__() method. For example:

```
>>> sys.getsizeof(s.__dict__)
104
>>> s.date = '5/27/2013'
>>> sys.getsizeof(s.__dict__)  # Flips to combined dictionary
296
>>>
```

Final Words

If there's any take-away from this article, it might be that parts of Python often assumed to be frozen in time are still a target of active development. Dictionaries are no exception. If you make the move to Python 3.3, you'll find that they are used in a much more efficient way than before (especially for instances).

This is by no means the last word. At this time, Raymond Hettinger, one of Python's core developers, has been experimenting with yet another dictionary representation which is even more memory efficient. Some details about this can be found at http://code.activestate.com/recipes/578375-proof-of-concept-for-a-more-space-efficient-faster/.

usenix

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Why Join USENIX?

We support members' professional and technical development through many ongoing activities, including:

- Open access to research presented at our events
- Workshops on hot topics
- Conferences presenting the latest in research and practice
- LISA: The USENIX Special Interest Group for Sysadmins
- *;login:*, the magazine of USENIX
- Student outreach

Your membership dollars go towards programs including:

- Open access policy: All conference papers and videos are immediately free to everyone upon publication
- Student program, including grants for conference attendance
- Good Works program

Helping our many communities share, develop, and adopt ground-breaking ideas in advanced technology

Join us at www.usenix.org

COLUMNS

iVoyeur
inotify

DAVE JOSEPHSEN

Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project. dave-usenix@skeptech.org

The last time I changed jobs, the magnitude of the change didn't really sink in until the morning of my first day, when I took a different combination of freeways to work. The difference was accentuated by the fact that the new commute began the same as the old one, but on this morning, at a particular interchange, I would zig where before I zagged.

It was an unexpectedly emotional and profound metaphor for the change. My old place was off to the side, and down below, while my future was straight ahead, and literally under construction.

The fact that it was under construction was poetic but not surprising. Most of the roads I travel in the Dallas/Fort Worth area are under construction and have been for as long as anyone can remember. And I don't mean a lane closed here or there. Our roads drift and wander like leaves in the water—here today and tomorrow over there. The exits and entrances, neither a part of this road or that, seem unable to anticipate the movements of their brethren, and are constantly forced to react. They stretch and grasp, and struggle to keep pace, and sometimes they even manage to connect things, albeit usually not the things they claim to connect. The GPS units, having given up years ago, refuse to commit themselves, offering only vague, directionless suggestions that begin "continue to merge."

On that particular morning—and really every morning that one takes a rarely traveled road in DFW—that new zig was literally and figuratively the beginning of an adventure into a wholly unexplored country, despite my having traveled the route several times in the past. Often, because of the unfortunate, disembodied exits, these adventures involve the accidental continued merging onto another highway in an utterly unexpected direction (usually north). So it was that I arrived emotionally depleted and 15 minutes late on the first morning of my new job. They didn't seem to notice.

I can imagine neither the ultimate goal of the master plan under which our Department of Transportation labors nor whether its intent is whimsical or malevolent, but it certainly has provided ample food for reflection over the years. Many of my professional undertakings remind me of this or that aspect of our highways.

There are, for example, problems that seem to recur every so often that, like the exits in Grapevine TX, take me in a new and surprising direction every time I visit. File system notification seems to be one such problem. Every time I've had a need to come up with a foolproof way to monitor changes to a set of files or directories, my options seem to have changed radically.

With three in-kernel solutions—dnotify, inotify, and fanotify—kernel instrumentation like systemtap, several external libraries like libevent, and myriad security-focused tools like snoopy logger and samhain, there are now more ways to monitor changes to files than there are types of file in UNIX.

On this visit, after an afternoon of reading, I decided to try out the inotify API, which is built-in to Linux kernels >= 2.6.13. Created to address several shortcomings in dnotify, namely, a vastly simplified (signals free) interface to more precise events about more specific file-system objects (dnotify only works on directories), inotify makes dnotify obsolete. The consensus seems to be that unless you very specifically need an in-kernel solution for monitoring directories that will never be unmounted on Linux kernels < 2.6.13, dnotify should be ignored. I should also mention that there are a few wrapper libraries that provide a more portable and abstract interface to inotify and inotify-like functionality on systems other than Linux. Among these are inotify_tools, FAM, and Gamin.

Further, the inotify_tools package provides two programs that are suitable for use in shell scripts: inotifywatch, which collects and reports inotify events in a "tail -f" fashion, and inotifywait, which blocks waiting for user-specified inotify events.

Inotify's interface is pretty simple. After creating an inotify instance with inotify_init(), the application informs the kernel which files it's interested in watching with one or more calls to inotify_add_watch(). Each call to inotify_add_watch() is accompanied by a path name and a bitmask specifying the event types to watch for. The add_watch function returns a file descriptor, which can be poll()'d, select()'d, or simply read() by your application. When successfully read, the file descriptor returns one or more `inotify_event` structures, each of which contains the details of a single file system event. When the application has finished its monitoring duties, it closes the watch file descriptor. I've provided a small example program in Listing 1.

Adding and Removing Watches

The add_watch function is not recursive, and therefore must be called on each subdirectory in a given directory that you want to monitor. If you call it on a directory, it will monitor all files in that directory (but not files in subdirectories). The function is formally defined as follows:

```
int inotify_add_watch(int fd, const char *pathname, uint32_t mask);
```

Add_watch returns 0 on success or -1 on failure. The first argument is the file descriptor returned from inotify_init(), and is used as a means of referring to our inotify instance. The second argument is the path to the file or directory you want to monitor. The application needs to have read permission on this object for the call to succeed. The third argument is a collection of event bits OR'd together. There are 23 possible events, 12 of which represent file system actions, like `IN_CREATE` (a file or directory was created).

A few event constants are shorthand for combinations of other events. For example, the event `IN_MOVED_FROM` is set when a file is moved out of a monitored directory, while `IN_MOVED_TO` is set when a file is moved in to a monitored directory. The shorthand event `IN_MOVE` can be used in lieu of both MOVED events. The shorthand event `IN_ALL_EVENTS` can be used to subscribe to all event types.

When a successful read returns an `inotify_event` struct, the same event constants are used in the struct to communicate the type of event that has transpired. Several of the event types only occur as output from inotify in these structs. Examples include `IN_ISDIR`, which is set whenever an event describes a directory, or `IN_UNMOUNT`, which is set when a directory is unmounted.

Finally, a few event types can be set in the bitmask to specify options to the inotify subsystem, like `IN_DONT_FOLLOW`, which turns off dereferencing of symbolic links. The complete list of event types is available in the inotify(7) man page.

Calling inotify_add_watch() on a file or directory that is already being watched replaces the event mask for that file or directory, but specifying `IN_MASK_ADD` in the replacement mask modifies this behavior such that the new mask is OR'd with the old one.

A call to inotify_rm_watch() explicitly removes watches on named files or directories. Whenever you explicitly remove a watch, or a file is moved outside a watched directory structure, inotify generates an event with the `IN_IGNORED` bit set.

Reading Events

The FD returned by add_watch follows the universal I/O convention, and may be treated like any other file descriptor. If no events have occurred when your application attempts to read() the descriptor, it blocks until an event is available. Applications may use poll() or select() for nonblocking behavior. Successful reads yield a stream of bytes, which is composed of one or more serialized `inotify_event` structs. The struct is defined as follows:

```
struct  inotify_event {
int          wd;       //the FD on which the event occurred
uint32_t  mask;    //bitmask describing the event
uint32_t  cookie;  //cookie to detect related events
uint32_t   len;     //size of the name field in bytes
char     name[]; //null terminated filename (optional)
};
```

The `wd` descriptor is used by applications that are monitoring multiple files or directories via the same inotify file descriptor. To use it, your application needs to keep an internal map that relates the file descriptors returned by add_watch, to the files you passed into add_watch.

The mask is a bitmask that describes the event using the constants I've discussed above.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <error.h>
#include <sys/inotify.h>
#include <limits.h>

#define BUF_SIZE (10 * (sizeof(struct inotify_event) + NAME_MAX + 1))

void
printEvent(struct inotify_event *event)
{
    printf("wd::%d, ", event->wd);

    printf("mask :: ");
    if (event->mask & IN_ACCESS)        printf("IN_ACCESS ");
    if (event->mask & IN_ATTRIB)        printf("IN_ATTRIB ");
    if (event->mask & IN_CLOSE_NOWRITE) printf("IN_CLOSE_NOWRITE ");
    if (event->mask & IN_CLOSE_WRITE)   printf("IN_CLOSE_WRITE ");
    if (event->mask & IN_CREATE)        printf("IN_CREATE ");
    if (event->mask & IN_DELETE)        printf("IN_DELETE ");
    if (event->mask & IN_DELETE_SELF)   printf("IN_DELETE_SELF ");
    if (event->mask & IN_IGNORED)       printf("IN_IGNORED ");
    if (event->mask & IN_ISDIR)         printf("IN_ISDIR ");
    if (event->mask & IN_MODIFY)        printf("IN_MODIFY ");
    if (event->mask & IN_MOVE_SELF)     printf("IN_MOVE_SELF ");
    if (event->mask & IN_MOVED_FROM)    printf("IN_MOVED_FROM ");
    if (event->mask & IN_MOVED_TO)      printf("IN_MOVED_TO ");
    if (event->mask & IN_OPEN)          printf("IN_OPEN ");
    if (event->mask & IN_Q_OVERFLOW)    printf("IN_Q_OVERFLOW ");
    if (event->mask & IN_UNMOUNT)       printf("IN_UNMOUNT ");
    printf("\n");

    if (event->len > 0)
        printf("name :: %s\n", event->name);
}
int main (int argc, char *argv[]){
    int fd, wd, rr,i; //fd and wd are file descriptors
    char buf[BUF_SIZE]; //returned events come here
    char *p;
    struct inotify_event *event;

    fd = inotify_init(); //dear kernel, inotify plsthnx

    if (fd < 0)
        perror("inotify_init");

    for(i=1;i<=argc;i++){
        // setup a watch on all events for every file passed as an arg
        wd=inotify_add_watch(fd,argv[i],IN_ALL_EVENTS);
        if (fd < 0)
            perror("inotify_init");

        printf("watching %s using wd %i\n", argv[i], wd);
    }

    for(;;){ //event loop
        rr=read(fd,buf,BUF_SIZE); //any events?
        if(rr == 0){
            printf("uh-oh, read from inotify returned 0!\n");
            return 42;
      }
        if(rr<0) //something went wrong with the read syscall
            perror("read");

        printf("read %ld bytes from inotify fd\n",(long) rr);

        for(p=buf;p<buf+rr;){ //as long as there are more bytes in buf
            event = (struct inotify_event *) p; //cast the current addr as a struct
            printEvent(event); //decode it
            p += sizeof(struct inotify_event) + event->len; //move to the start of the next struct
        }
    }
}
```

Listing 1

Cookies are currently only used to associate MOVE events that are the result of renaming files. When a file is renamed, two events will be generated: an IN_MOVED_FROM event for the old file name and an IN_MOVED_TO event for the new file name. When this occurs, the value of the cookie field will be the same in both events, so that the application can associate the two.

If an event occurs to a file inside a monitored directory, the name field will be set to the name of the file, and len will indicate the number of bytes allocated for the name field. If, however, the event occurs to the directory itself, name will be NULL and len will be set to 0.

Because name is a dynamically allocated field, predicting the necessary size of the read buffer for the next event struct is impossible until you've read the struct and dereferenced the len field.; however, we can safely assume the size of the next struct will be smaller than:

```
(sizeof(struct inotify_event) + NAME_MAX + 1)
```

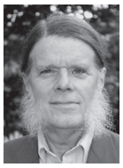
where NAME_MAX is the local OS constant that specifies the maximum size of a file name (usually set in limits.h). In Listing 1, I'm passing a buffer 10 times this size to read(). This will allow the application to retrieve at least 10 events with a single read() efficiently, and use pointer arithmetic to split them out. A read from an inotify file descriptor will yield the number of available events that will fit in the supplied buffer. In the event that you pass a buffer that is too small to hold the next single event struct, read() will fail with EINVAL.

Because add_watch is not recursive, for inotify applications to dynamically detect and add_watch newly created subdirectories in a currently watched directory is pretty common. To keep things simple, I didn't include an example of that in my sample code, but I hope given this article's Listing 1, that it's an obvious enough exercise for the reader.

As always I hope you've enjoyed the ride. Until next time, continue merging.

For Good Measure
Security Debt

DAN GEER AND CHRIS WYSOPAL


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

Chris Wysopal, Veracode's CTO and co-founder, is responsible for the company's software security analysis capabilities. In 2008 he was named one of InfoWorld's Top 25 CTOs and one of the 100 most influential people in IT by eWeek. One of the original vulnerability researchers and a member of L0pht Heavy Industries, he is an author of L0phtCrack and Netcat for Windows, and is the lead author of *The Art of Software Security Testing* published by Addison-Wesley.
cwysopal@gmail.com

Blessed are the young for they shall inherit the national debt.

— *Herbert Hoover*

When you start a company, you take on financial debt so that you can reach your market in time. When you release a product, you take on technical debt, and for the same reason. Ward Cunningham talked about this in 1992 [1]:

> [I]mmature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to extreme specialization of programmers and finally an inflexible product. Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite...The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation.

One of the present authors proposed [2] that cyberinsecurity is the critical form of technical debt, if for no other reason that a bug is exercised by an accident but a vulnerability is exercised by an enemy. Consider your security vulnerabilities to be a debt note that has been purchased by someone who is out to get you—not only are you in debt, but the debt can be called at the most inconvenient time calculable.

Every software release is debt issuance, and vulnerabilities are common in fresh code [3], but as Clark et al. point out [4], you can roll your code often enough that the attackers can't keep up. (Google appears to roll Chrome every 3–5 days.) Rolling over a financial debt cheaply means life is good, unless and until there is a rate shock.

The problem with rolling over your security debt, however, is that you can soon have no idea what is going on. If you are a supplier, then you may choose to buy outside testing, that is to say you may choose to get your debt rated, but with sub-week release cycles it is not possible to test within cycle—test results are always for a now previous version. If you are a consumer, your test might be the most trivial of all tests, viz., whitelisting the hash taken from the supplier's golden master, but propagation time for the whitelist may well not keep up with the rate of issuance, just like a rating agency that can't even rubber stamp what the mortgage lender is issuing as fast as they are issuing it.

Let's say you've been rolling over your cyberinsecurity debt for long enough that you have a considerable debt overhang built into your products, or into your enterprise deployment of everything from Aardvarks to Zebras. Well, you can pay it down. Microsoft showed us how when it declared cyberinsecurity debt bankruptcy and built IIS 6.0 from scratch. That rewrite brought an untenably rising incidence of reported vulns down to a dull roar [5], as seen in Figure 1.

Of course, there are substantial security debts building elsewhere; here, in Figure 2 we demonstrate this buildup with some obvious choices, all on the same timeline as Figure 1, and their sum, which is a lower bound on net security debt buildup.

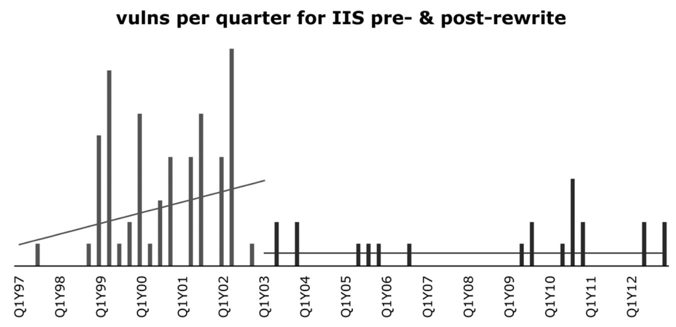


Figure 1: Rising incidence of reported vulnerabilities down

Financial bankruptcy is especially easy in the US, which is why the US economy rebounds from boneheaded financial mistakes faster than elsewhere—you just throw out the trash . . . unless the thing "you" need to bankrupt is too big to fail (TBTF). Now you can't shuck the debt. TBTF in finance is a bank whose failure would kill other firms. TBTF in cyber is an installed base too big to overwrite. As with Marsh Ray's TLS renegotiation attack [6], an installed base that is TBTF means that all that can be done is to add mitigating software on top of it. Adding software increases the attack surface. Happy New Day.

In the previous installment of this column [7], we proposed a market approach to dealing with cyberinsecurity risk by separating out severity from frequency of cyberinsecurity events. Severity is context dependent and a matter of taste. Frequency is an objective, mensurable fact, thus it can be the basis for a market. In synopsis, a futures market in the frequency of cyberinsecurity events (a trendline based on a counting function) dodges the question of severity (the maximum excursion of some unhappy cost curve). Trendlines are ordinal-scale statistics, i.e., good enough for decision support. Trendlines do not require the precision of definitions (what is "severity?") that frustrate the appearance of a hard science of cybersecurity. The key to the proposed market in cyberinsecurity event futures is an underlying debt pool from which the security events come, an underlying debt pool for which the security events provide an estimate. That underlying debt pool is, obviously, accumulated cyberinsecurity debt. A street cop cannot know how much heroin is for sale, but he can follow the price and adjust his policing based on which strategies raise the price of heroin. A cybersecurity cop cannot know how many vulnerabilities are present in the code on which he depends, but he can follow the price of cyberinsecurity event futures (and not the price of zero-days).

If cyberinsecurity insurance is written as a fixed dollar amount per cyberinsecurity event, then the predicted exposure of the insurer is simply the predicted frequency of cyberinsecurity events. And if cyberinsecurity events are, in turn, a linear function of cyberinsecurity debt load, then we have a third alternative (hedging in cyberinsecurity event futures) to what had been

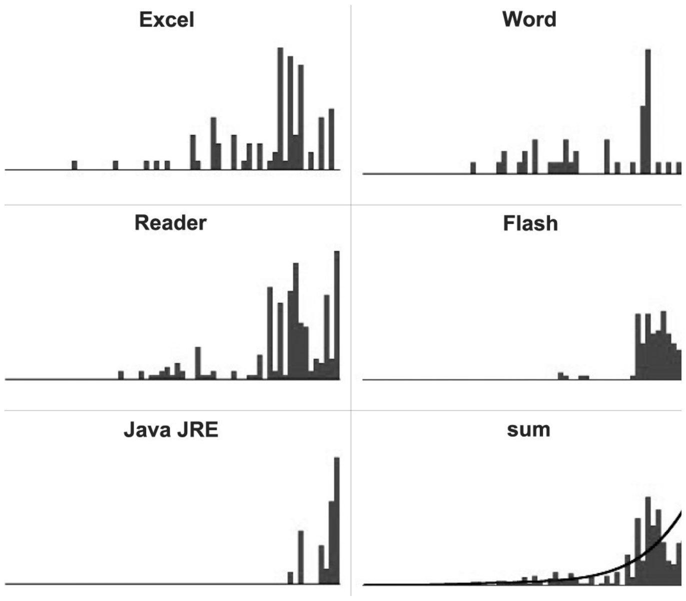


Figure 2: Security debts building

a choice of two less attractive alternatives: continuing to roll over the cyberinsecurity debt (of unknown size) or paying that debt down through codebase bankruptcy.

We consider Adobe's recent conversion to Software as a Service [8] to be an unacknowledged cyberinsecurity debt bankruptcy with Adobe remaining as a debtor in possession. Perhaps cyberinsecurity debt avoidance explains part of why the market capitalization of the top three SaaS vendors is growing five times as fast as the top three (product) sales vendors [9], as shown in Figure 3.

The collectivization of risk can be voluntary (you buy insurance) or involuntary (you are taxed to bail out TBTF). Insurance at industrial scale requires reinsurers—entities that sell insurance to insurers such as for linked-losses, viz., catastrophes where a single event (hurricane) causes large numbers of losses. The

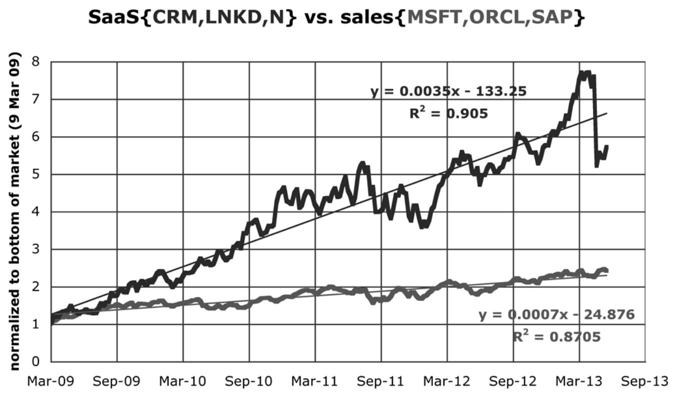
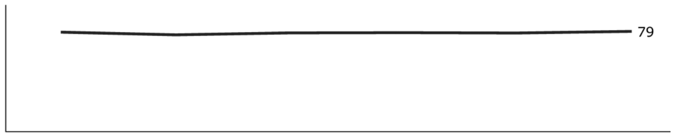


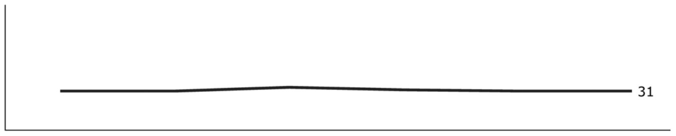
Figure 3: The market capitalization of the top three SaaS vendors is growing five times as fast as the top three sales vendors.

chance of catastrophes in cyberinsecurity is proportional to deployed interdependence, meaning that installed base is a strict lower bound indicator for what is TBTF in cyberinsecurity, i.e., the likelihood of a successful attack on one component of that interdependence may be a consequence of a successful attack on some different component.

Excepting TBTF, there always comes a point where risk transfer (like insurance) is a better investment than continued risk reduction, particularly when risk reduction is proven difficult even for firms that want to do it. Veracode's SoSS 5 report [10] shows two examples where among committed firms and repeated interventions, their cyberinsecurity score is all but constant. Over six quarters, analyzed software in the aggregate had a ±1.7% score



and Web applications subject to SQL injection had a ±4.7% score



Yes, those are flat lines.

Although it is true that the number of cyberinsecurity insurers is rising, so far as we know there is not yet a reinsurance market for cyberinsecurity insurance. Because reinsurance is a necessary condition for a robust market among primary insurers, and because the optimal number of reinsurers is the square root of the number of primary insurers [11], if the number of primary cyberinsecurity insurance issuers is to grow, so will grow the need for market makers in insurance futures. Despite being inconsistent with a free people, when a jurisdiction requires that its citizens buy insurance, capital must be sequestered to cover probable losses. A market that capitalizes the reserves needed for cyberinsecurity insurance is thus essential by observation: the risk is already collectivized even if merely ignored through the rolling over of cyberinsecurity debt society-wide.

One can argue about what is the "interest" on the cyberinsecurity debt, but it is unclear which of several models is relevant to the fundamental decisions—unless the interest rate is near zero, which it can only be by fiat rather than being market derived. The supply side makes exactly that assertion: the high-order bit on every page of every EULA is "It is not our fault," and courts have tended to agree that if the end user accepted such license terms, then they do govern. We do not think that cheaply rolling over cyberinsecurity debt can indefinitely continue, and therefore there needs to be a way to do risk transfer—one where objective measures of cyberinsecurity debt help price the transfer of risk. It would be wise to have that pricing in place before the rate shock hits.

References

[1] Ward Cunningham, "The WyCash Portfolio Management System," March 26, 1992: c2.com/doc/oopsla92.html.

[2] Chris Wysopal, "Application Security Debt and Application Interest Rates": http://www.veracode.com/blog/2011/02/application-security-debt-and-application-interest-rates.

[3] Elizabeth Nichols, "State of Software Security": http://www.veracode.com/blog/2013/05/soss-one-figure-at-a-time/.

[4] "Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-Day Vulnerabilities": http://www.acsac.org/2010/openconf/modules/request.php?module=oc_program&action=view.php&a=&id=69&type=2.

[5] Data courtesy of osvdb.org.

[6] "Vulnerable Compliance": https://www.usenix.org/system/files/login/articles/geer.pdf.

[7] Dan Geer and Dan Conway, "The Price of Anything Is the Foregone Alternative,";login:, vol. 38, no. 3, June 2013: geer.tinho.net/login/geer.login.1306.pdf.

[8] "Adobe Kills Creative Suite—All Future Features Online Only": http://www.theregister.co.uk/2013/05/06/adobe_kills_creative_suite_for_cloud.

[9] Data courtesy of Yahoo Finance.

[10] "State of Software Security Volume 5," https://www.veracode.com/images/pdf/soss/state-of-software-security-report-volume5.pdf.

[11] Michael Powers and Martin Shubik, "A 'Square-Root Rule' for Reinsurance," *Revista de Contabilidade e Finanças* (Review of Accounting and Finance), vol. 17, no. 5, pp. 101-107.

/dev/random

ROBERT G. FERRELL

Robert G. Ferrell is a fourth-generation Texan, literary techno-geek, and finalist for the 2011 Robert Benchley Society Humor Writing Award. rgferrell@gmail.com

On a lark (it was big for a lark, but I still had to scrunch up a bit) I applied for an infosec manager position at Facebook. I got to the "upload your resume" segment and there was also a statement indicating the desirability of a code sample. The job is not a software engineering job, but apparently their world consists of nails and code is the hammer. I scratched my head, as I haven't written any substantial code in at least seven years, and finally included a Perl module I coded in 1999. I don't even know if it will run.

I hope it gives them a good laugh, if nothing else. I did put in the comments that I'm a security geek, not a coder. They wanted me to solve a programming puzzle, too, but I ignored that part. They tried to convince me. I ignored them even more vigorously: they don't know that I solve all puzzles by cheating, and I intend to keep it that way. So, no Facebook job for me. It's probably for the best. I would be tempted to give users an encryption option, so that only those friends with whom they choose to share the key could read their posts. To everyone else they would look like shared photos of cats, funny mash-ups involving celebrities and/or cartoon characters, and long misspelled diatribes on half-understood government policies. One might be excused for thinking that feature has already been implemented.

The Facebook adventure put me in mind of other avant-garde hiring practices I've encountered, and they've almost always sprung from these high tech Silicon Valley-esque outfits. Do left coast companies vie for "Weirdest Hiring Process"? Is there an industry award for that now? Is it the sun and sand, or just the ready availability of pharmaceutical enhancements? Does an abundance of sea gulls, or the aerial poop thereto appertaining, affect the rational thought processes? Can someone hand me a sparkling mineral water?

Give me the traditional interview questions any day. I have my answers prepared.

Q: "What do you consider your strengths?"

A: "I can eat an entire box of Nutri Grain bars in one sitting. Two, if I have milk."

Q: "What about your weaknesses?"

A: "I have never made it all the way through the third box without getting sick."

Q: "Where do you want to be in five years?"

A: "Sitting on the beach in St. Thomas. Actually, that's where I want to be next week, too. "

Q: "Why do you think you are the best candidate for this position?"

A: "Because I have the largest collection of emails and text messages between you and the girlfriend your wife doesn't know about."

Tallying ho (that sounds like a gardening store owner taking inventory, doesn't it?), lately I've been pondering the expansion of virtualization to other areas of daily life. What if you could, for example, instantiate a cheery, optimistic virtual mien over the rotten, grouchy virulence of your personal underlying operating system? Your popularity would soar even as your indictments declined. It would revolutionize Hollywood, too. A director could take generic thespians and run whatever virtual personalities on them the script calls for. Easy-bake actor-in-a-box: no need for messy, time-consuming auditions. Spouse mad at you? Pop in the "Irresistible Lover" LCM (Loadable Cranial Module) and you'll be making beautiful music together in no time.

Pointy-haired bosses are so much easier to handle when you load up the "Respectful Obedient Employee Secretly Plotting To Do You In" module (also known as the "Eddie Haskell"). Maybe choose the "Terminator" one when the visiting in-laws have outstayed their welcome: "If you want to live, you will leave now. And do not come back" in a thick Austrian accent is sure to get the message across. Harsh, yes, but effective. You gotta be firm sometimes.

I'm wondering where the portable computing device market is heading next. At some point in the very near future the miniaturization craze is going to slam with considerable force into the wall of human optical limitations (and the video will undoubtedly appear minutes later on Live Leak). It doesn't matter to me if my postage stamp-sized Ultra-absorbent iPad Mini (sorry, got my pads mixed up) has quad octo-core processors and a 1760 x 1140 HD screen if the pixels are only one photon in size. We're rapidly approaching the point where all I/O will be have to done via Bluetooth or something equivalent—presuming they can continue to produce smaller and smaller transceivers—because the CPU housing itself is too small to accept any physical cable adapter or memory stick visible to the naked eye. I call this entire class of devices "Barbie boxes."

Rolling without discernible segue into an entirely different topic, there's an interesting discussion underway on a mailing list to which I belong concerning "back doors" in software intended for forcing updates. Rather than leap directly onto that philosophical log jam, though, I will approach it in my customary oblique fashion. If software can have back doors, what other architectural features might it possess? Well, windows to begin with. I suppose the screen intensity control could then be thought of as blinds or drapes. Cleaning said windows would be removing unnecessary icons, of which I have more than my share at present. I've always believed that one of the primary reasons Microsoft decided to call its graphic interface "Windows" is that it was such a pane to run. "Scraping the bugs off your Windows" then takes on a whole new meaning as a euphemism for installing a patch.

In this architectural scenario the firewall would of course be the front gate, where packets drive up and push the little buzzer to be let in. Now, the roof keeps out rain, and too much rain is a flood, and the thing that prevents floods (syn, for example) would be the…firewall. So, apparently in this house your visitors enter through the roof. I guess they get to the ground floor by climbing down the TCP/IP stack.

It's became glaringly obvious to me that this analogy is not built on a solid foundation. Further, the timbers are just as apparently rotten; the entire construct has collapsed from its own weight. I hope everyone got out all right.

Sometimes, despite our best efforts, the product of a labored metaphor is stillborn. At least it didn't suffer. My apologies if the same cannot be said for you.

P.S. I would be remiss if I didn't occasionally toot my own horn here, so if you're a humorous fantasy fan, look for *Goblinopolis* on bookshelves both virtual and real. Buy a copy for yourself and everyone you've ever met. I might even sign them if I can remember how to write in cursive.

Book Reviews

ELIZABETH ZWICKY, MARK LAMOURINE, AND RIK FARROW

Beyond Software Architecture: Creating and Sustaining Winning Solutions
Luke Hohmann
Addison Wesley, 2003, 302 pp.
ISBN 978-0-201-77594-5
Reviewed by Elizabeth Zwicky

When you go to build real systems, you will discover that the architecture is driven by a giant pile of considerations that are not theoretically part of software architecture. ("Why don't you log it in this log file?" I asked the other day. "Ah. Well. That log file is written by this other part of the software, and this part of the software only writes this log file. They have different lead programmers, see. And there's no way to log across the boundary without rewriting both components.") This is a book about the space where software architecture meets the need to actually make money and satisfy the people who build it.

Most of the book is about the relationship between what you sell to customers (the marketing architecture, or "marketecture," as the author calls it) and what you build (the technical architecture, or "tarchitecture," which sounds sticky and unpleasant to me, but makes sense as an abbreviation). It suggests ways in which they should interact, while being clear on the differences between them.

If you are making the leap to designing whole systems, particularly commercial software (which is the focus here, although the author does consider internal systems in passing), this is a nice balanced look at the problem, which does not present either the customer or the marketing department as an enemy. It encourages sensible, honest, and human behavior without pandering to the whims of programmers. For instance, it suggests that you should not try to make water flow uphill or ignore the personal needs of your team to feel like they have important, relevant tasks, but it also suggests avoiding getting sucked into new technologies because they'll look good on resumes or marketing collateral.

The section on security is OK but a bit perfunctory and not fully integrated. It does not have the same feel of hard-learned lessons that most of the rest book does, but it does at least hit the high points (you can't add security at the end of a project; do not write your own encryption; be sure you know what you're trying to protect against and protect sufficiently for that rather than trying to protect everything everywhere all the time against everybody).

Bad Data Handbook
Q. Ethan McCallum
O'Reilly Media, 2013, 256 pp.
ISBN 978-1-449-32188-8
Reviewed by Elizabeth Zwicky

Continuing in this month's theme of ruthless realism, here's an entire book on analyzing the data you can actually get, which is never quite the data you wanted. It's a collection of essays, not a cohesive whole, but it's full of interesting and useful insights. They range from generalizations (why and how you should cross-check data) to very specific techniques (how to process text of unknown origin and dubious character set in Python).

Anybody who is new to data analysis can use this book. Most new analysts have a wholly unfounded faith in the data, which leads them to produce charming flights of fancy that fall apart when you look at them at all closely (hey, according to this the sum of the parts is greater than the whole!). When they lose this faith, they are not sure what to substitute for it, and poke gingerly at the data like a small child presented with a new food. This book suggests concrete steps to take to verify and sanitize data, as well as pointing out the importance of understanding exactly where your data comes from and how.

As always, some parts are weaker than others. As somebody who works with human-generated data a lot, I found the section on "liars" less useful. It's good to understand that the people who use systems often have complicated meta-goals and almost always behave in ways you don't expect (the example the author uses is a great one, and I don't want to give it away); the users understand how the system works, mostly, but they don't really get all the implications of the difference between automated and human systems. Their attempts to optimize are damaging to the system as a whole, but it's not exactly because they intend to lie, and the author fails to draw out the implications for human-based systems. There are always people who are trying to cheat the system, but there are also always people who are simply at odds with your design goals, intentionally or unintentionally. New analysts, in general, assume that you can divide all the data points into "good" and "bad," and anything you weren't expecting is somebody being Wrong on the Internet. In fact there are at least four relevant categories—"good," "bad," "bug in the processing system," and "not what we were hoping the user would do."

Data Insights: New Ways to Visualize and Make Sense of Data

Hunter Whitney
Morgan Kaufman, 2013, 296 pp.
ISBN 978-0-12-387793-2
Reviewed by Elizabeth Zwicky

This is a very pretty book, with nice examples of visualizations and with generalizations and advice that are both engaging and accurate. It is suitable for smart people who aren't clear on where to aim for when going from a pile of numbers to something meaningful, or a manager who wants or needs to understand what comes beyond Excel's defaults for pie charts. It is not, however, at all specific on how to get to where you are aiming at. If you want step by step advice, or actual techniques at all, you'll need to look elsewhere, so absolute beginners and experts alike are probably going to be frustrated.

I didn't disagree with the content; I think the author's approach is right, but I felt it kept getting close to being useful, coherent, and organized, and then getting distracted by another pretty visualization and wandering off again. Plus, QR codes as ways to provide URLs with examples sound like a great idea, but they're eye-catching without being illuminating to the reader. And, in my case, the first two I tried don't work on the device I had at hand; one is a Flash animation and the other loads but for some unclear reason doesn't do a lot. This is an object lesson in something about visualization, but it's not clear that it's what the author had in mind. (And I worry about the level of impermanency implied in using goo.gl-shortened links to a magazine Web site; six months from now, are these QR codes going to lead anywhere at all?)

Finally, there's an illustration labeled "The Esperanto of visualization." I think that's meant to be a compliment, but without any further explanation of the illustration, I can't be totally sure, since the illustration conveys no data at all to me. It could be meant as a scathing condemnation of both Esperanto and the visualization. This was an unusually clear example, but I often found myself unsure of exactly what message I was supposed to take away and what I was supposed to do about it. Ultimately, I ended up treating it like a fashion magazine; look at the pretty pictures, maybe some of the captions about what they are.

Generation Blend: Managing Across the Technology Age Gap

Rob Salkowitz
Wiley, 2008, 243 pp.
ISBN 978-0-470-19396-9
Reviewed by Elizabeth Zwicky

Sadly, I got through the entire book without being convinced that in fact there is a significant difference between grouping people by generation and grouping them by astrological sign. Supposedly, Millenials (the youngest generation at work) like to mix life and work by having lives at work, and Boomers (the oldest) and Generation X (mine, which is apparently why I'm so cynical) don't—Boomers take work home, and Generation X works to live. I fail to see how this goes with the popular statistics about lost work time for the Super Bowl and Thanksgiving shopping, and while anecdote is not data, I promise you I get mail (to my personal address) from the personal address of my Boomer colleague during work hours. I also read it. Apparently it's not just the kids these days.

If you believe in firm distinctions between the generations, and dislike one of them, this is a useful introduction to how all of them are useful in the workplace. Otherwise, the most useful part of the book is a chapter on how to teach computers to older people who don't have much experience with them.

Digital Capture After Dark

Amanda Quintenz-Fiedler and Philipp Scholz Ritterman
Rocky Nook, 2013, 190 pp.
ISBN 978-1-9333952-66-6
Reviewed by Elizabeth Zwicky

Digital cameras these days will do all kinds of fascinating things with almost no human help. So, of course, people want to do the things they're bad at. Photographing in the dark is one of those things—your camera will be great, aside from the fact that it won't focus or correctly calculate exposure for you, so pretty much, you're on your own. Once it gets dark your easy-going camera that shows you something like you see and can let you review it on the spot becomes a lying tyrannical magic box which must be kept still and shielded from stray light but then will show you things you can't see.

The authors go over the gear you need (a tripod, a remote shutter release, extra batteries, a flashlight, extra batteries for the flashlight) and then get into the techniques. Most of these are photographic techniques, but some of them are more practical. On the photographic side, you can't trust the camera to meter and you can't trust the preview or the viewfinder to show you what you're really getting, so you have to learn to figure out what you're probably getting and how to raise your odds of getting what you want. On the practical side, it's cold and dark at night. Digital cameras run on electricity. Batteries hate the cold. How do you maximize your ability to take photographs? How do you minimize the risk to your camera from coming back inside to the damp warmth?

Most popular kinds of night photography are discussed, including special lighting techniques like light painting and use of light bursts to combine motion and still in one picture. There isn't any discussion of lightning (oddly, because there's a lightning photo), but lightning is basically just a big fast light you don't control.

The book succeeded in making me feel that photographing in the dark was exciting and within my grasp, and its advice is consistent with the little night photography experience I've had. I'm looking forward to more experimentation.

Hacker's Delight, 2nd Ed.
Henry S. Warren, Jr.
Addison-Wesley, 2013, 494 pp.
ISBN 978-0-321-84268-8
Reviewed by Mark Lamourine

http://www.hackersdelight.org/

Hacker's Delight is a rare find: a clear, well-written book about a fundamental element of programming. I would put it on the same level as Stevens' TCP/IP Illustrated. Most people will never need to code binary logic and arithmetic operations at the level of individual RISC instructions, but knowing what's going on down there can only help.

Warren opens Hacker's Delight by describing the sandbox in which he will play: a basically unrestricted number of general purpose 32-bit registers and a "basic" and an "extended" RISC instruction set. All of the operations are simple integer arithmetic bit shifts and binary comparisons on registers or constants. Warren also limits branches and memory load and store instructions because they are expensive. The goal is to see what can be done within these constraints. The remainder of the book demonstrates that the answer is "a lot."

The first seven chapters cover a wide range of simple but sometimes obscure bit and byte operations. Where a reader might ask, "But why would I want to do that?" Warren provides a brief answer or a reference to more detail.

The chapters are broken into subsections of about one to three pages which describe and then explain a particular operation. The algorithms are presented in computer algebra (presented in Chapter 1) or in ISO C99. At least one algorithm is offered in Python, showing that the applications are not limited to machine-level code. Where a formal derivation or proof of the algorithm is needed it is presented in more traditional logical or mathematical form.

As Warren progresses, he moves from simple operations to more complex tasks. He shows how to implement such things as integer multiplication and long division, CRCs, and even how to implement floating-point arithmetic.

Warren briefly mentions issues that someone might find when trying to implement some operations in other languages, such as Java or on variations in real-world processors which may differ slightly from his abstract instruction set. As a high-level programmer, I would have liked to see some discussion of the applicability of these algorithms in modern scripting languages. While the logic remains correct regardless, I'm curious if the implementation of the scripting languages will carry the compact logical operations down into the resulting machine code.

There is also an associated Web site: http://www.hackersdelight.org. This refreshingly unadorned set links to copies of the code samples, errata, and additional resources.

Hacker's Delight reminded me that there is a case to be made for clever logical minimalism in specific cases, and this can have its own beauty and clarity. That said, I expect that it will be of limited direct use to the vast majority of the computing community. But utility isn't everything. Hacker's Delight will be a pleasure to anyone who started working with computers out of curiosity about how stuff works and an appreciation of the aesthetics of fundamental logic.

Dart in Action
Chris Buckett
Manning Publications Co., 2013, 398 pp.
ISBN 978-1-617290-86-2
Reviewed by Mark Lamourine

There have been a number of attempts to improve upon or replace JavaScript as the client-side programming language for the Web. Google's Dart programming language is one of these.

Dart is a young language. Chris Buckett introduces Dart and the entire development ecosystem: an IDE, testing frameworks, client- and server-side coding and deployment. The first section is a fairly traditional whirlwind setup and "Hello World" treatment.

In the second section Buckett does an excellent job of detailing the syntax of Dart. Especially important are the sections on two language features which appear to be unique to Dart. The first is the optional type system. In an attempt to address the concerns of both the fans of strongly typed languages (for compile time type checking and diagnostics) and those who prefer the flexibility of weakly typed languages, Dart allows the coder to choose whether or not to provide data type information.

The second feature, isolates, is a mechanism to provide concurrency in a single-threaded environment. Since Dart must be translated to JavaScript, and JavaScript is single-threaded, then Dart must be too. Isolates allow concurrency by creating functions with distinct execution contexts, and limiting interaction between them. Buckett provides a series of examples for using isolates to manage concurrent queries on a Web service written in Dart.

In the final two main sections of the book, Buckett addresses the requirements and capabilities of Dart on the Web browser and the Web server. He gives special attention to client-server communications and data storage. At the time the book was written

Dart did not yet have a browser GUI library, but the Dart Web site indicates that the current release does include one.

Buckett concludes with a pair of appendices which together provide a language reference to match the tutorial of the second main section of the book.

I like the general style of the Manning "In Action" series. At the end of each tutorial section there is a highlighted paragraph listing the significant points from the preceding section. Buckett's inclusion of the reference appendices at the end of the book means that it will continue to have use after the tutorials are finished.

Dart projects can be deployed either by translating to JavaScript (much like CoffeeScript is deployed) or by using the DartVM. Currently, the only Web browser to offer a DartVM is a specially built Chrome browser provided with the Dart SDK. Indications are 18 months after Dart's announcement no other Web browser producer has plans to include a DartVM, and even Google has no plans to include it in the main-line Chromium release.

If you're working on a project which already uses Dart or if you are interested in alternatives to writing Web applications in JavaScript, "Dart in Action" is a good place to start.

Distributed Network Data

Alasdair Allan and Kipp Bradford
O'Reilly Media, Inc., 2013, 155 pp.
ISBN 978-1-449-36026-9
Reviewed by Mark Lamourine

The subtitle of Distributed Network Data is "From Hardware to Data to Visualization." Allan and Bradford really do take the reader of this slim book from a hardware parts list to graphing the collected data.

To fully appreciate this book the reader really should commit to acquiring the parts and following along. This is a book for the adventurous beginning Maker. It is a little helpful (but not necessary) to have some familiarity with a soldering iron, a text editor, C, and a search engine. It's really important not to be intimidated by three- and four-letter acronyms. The authors explain the ones that matter and refer the reader to other sources when necessary.

Allan and Bradford have structured the book so that each chapter presents a nicely self-contained task. Each chapter builds on the one before. They begin with an explanation of the goal and any needed theory and then dive into a guided tutorial, ending with a demonstration of some new capability.

In the first few chapters the reader will get comfortable with Arduino, breadboarding, and some fairly easy circuits. The second section brings in the "network" from the title with XBee network devices. The final sections introduce three different pieces of software for visualizing both the plan for the project and the resulting data.

I haven't understood until now how a Mac OS user feels when presented with examples and screenshots from Windows or Linux. The examples and code in Distributed Network Data are all created on Mac OS. Two of the three visualization platforms, Processing and Fritzing, are open source, free to download, and available for all three platforms. LabView is a commercial product, but there is a free version tailored for use with Arduino.

Allan and Bradford close the book with a chapter giving references to the original sites for each of the software packages and for additional reading.

At the end of the process the reader will have a working wireless sensor system collecting data and the ability to plot the data. More importantly, the reader will have the confidence to try different configurations and pointers to additional resources, including a community of Arduino Makers, to tap for more ideas and explorations.

Absolute OpenBSD, 2nd Edition

Mike Lucas
No Starch Press, 2013, 491 pp.
ISBN 978-1-59327-476-4
Reviewed by Rik Farrow

Mike Lucas produces books that are clear and easy to read, and this book is no exception. He provides information that is specific to installing and maintaining the most recent version of OpenBSD at the time (5.3), and with a level of detail that is refreshing. Today, most of us just "ask the Web" when we want a quick answer. Each chapter in this book is more like a tutorial that goes beyond just telling you what to do, but why you should be doing it, or not doing it.

You won't find information about configuring Apache 2.2 in this book—in fact, Mike recommends that you use nginx instead, but has nothing to say about configuring nginx. This is a book about the OpenBSD system, not applications. Mike does have a thorough chapter about how to find applications, use the package system, or use ports if you cannot use the package system, but that's it. And as it should be.

You will find detailed information about configuring OpenBSD, from the options in login classes, to the purposes of the various files found in the /etc directory. There are two chapters about TCP/IP, one on theory, the other on the practical aspects, including IPv6. Mike also wrote two excellent chapters on PF, the OpenBSD firewall, which in itself is a good reason to use OpenBSD. If you want to try an operating system focused on security, and also want detailed instructions, this is the book for you.

Conference Reports

NSDI '13: 10th USENIX Symposium on Networked Systems Design and Implementation

Lombard, IL
April 2–5, 2013

Summarized by: Anish Babu Bharata, Weverton Cordeiro, Rik Farrow, Utkarsh Goel, Arpit Gupta, Imranul Hoque, Peng Huang, Murad Kaplan, Scott J. Krieder, Joshua B. Leners, Muntasir Raihan Rahman

NSDI '13 Opening Remarks and Awards

Summarized by Rik Farrow (rik@usenix.org)

Jeff Mogul, NSDI co-chair, opened the conference by telling attendees that there were 170 paper submissions, and each paper received three first-round reviews. About half the papers made it into the second round, and also received three or four more reviews. By the time the PC meeting occurred, there were 64 papers left. In total, each PC member performed 25.8 reviews on average. In the end, 38 papers were accepted, more than usual. Because of this, each presentation could only be 20 minutes, including questions at the end. And by the end of the first session, things were going well.

Jeff also pointed out that, with 258 attendees, this is almost the largest NSDI ever. Jeff then singled out three PC committee members for work above and beyond the "call of duty": Jeff Chase, James Mickens, and Renata Teixeira.

Glancing at the hot topics graphic, their most mentioned terms were: *network, data, system, applications, paper, and power*. Jeff magnified a small section of the chart, where the words "Looking for work" were visible as a significant topic. Finally, Jeff mentioned that Ratul Mahajan and Ion Stoica would be the co-chairs for NSDI 2014.

Nick Feamster, the other co-chair, took over for the presentation of awards.

The Best Paper awards went to: "Embassies: Radically Refactoring the Web," by Jon Howell, Bryan Parno, and John R. Douceur, Microsoft Research; and "F10: A Fault-Tolerant Engineered Network," by Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson, University of Washington.

The Community awards went to: "Composing Software Defined Networks," by Christopher Monsanto and Joshua Reich, Princeton University; Nate Foster, Cornell University; Jennifer Rexford and David Walker, Princeton University; and "Expanding Rural Cellular Networks with Virtual Coverage," by Kurtis Heimerl and Kashif Ali, University of California, Berkeley; Joshua Blumenstock, University of Washington; Brian Gawalt and Eric Brewer, University of California, Berkeley.

Howell et al. received their Best Paper award for being ambitious, thought-provoking and even controversial in their paper. Liu et al. got their award for applying simple effective insights to the co-design of network topology and protocols and evaluating them well.

Community awards were given to paper authors for contributions to the community above and beyond the typical paper. Monsanto et al. received their award for releasing the code for their software, which the award committee believed would be useful and built upon by the Software Defined Networking (SDN) community. Heimerl et al. received their community award on the basis that this will help with rural cellular deployments.

Software Defined Networking

Summarized by Weverton Cordeiro (weverton.cordeiro@inf.ufrgs.br)

Composing Software Defined Networks

Awarded Community Award!

Christopher Monsanto and Joshua Reich, Princeton University; Nate Foster, Cornell University; Jennifer Rexford and David Walker, Princeton University

Joshua Reich began by arguing that one can build a robust, large, complex network system using OpenFlow, but it is going to be a cumbersome, time-consuming, and even an error-prone task. In a brief example, he described how complex it is currently to implement even simple sequential composition of logic such as simple load balancing and routing functions. The problem is that existing platforms for network programming only expose part of the network, lacking proper support for enabling the programming of various aspects in an integrated fashion. Building network applications using existing frameworks is pretty much like building complex software using assembly language, he said.

After drawing such a picture of the current state-of-the-art, Joshua presented Pyretic, a modular and intuitive language (and system) for enabling network programmers to build sophisticated network traffic management applications. Pyretic approaches the problem of designing complex network traffic management applications by providing three abstractions to programmers: policy abstraction, network abstraction, and packet abstraction. Policy abstraction enables programmers to use composition operators (sequential and parallel), which opens the door for doing all sorts of functional composition. The network abstraction lies on top of that, enabling programmers to decompose network software logic based on topologies. Finally, the packet abstraction provides extensive headers, which form the foundation for the previous abstractions.

During the talk, Joshua navigated through a series of examples that highlighted the potentialities provided by each kind of abstraction, which now enables network programmers to focus on the meaning of their applications instead of concentrating on low-level details. "One single line, absolutely precise," he said. He also emphasized that Pyretic captures the full power of OpenFlow by providing one high-level policy for each natively supported policy, and that Pyretic can be used to implement both static and dynamic policies; a MAC-learning example described the power of these policies. Joshua concluded his technical discussion by sketching the implementation of the topology abstraction function in the context of a one-big-switch transformation example. Finally, he encouraged the audience to experience the full power of Pyretic by referring to the project Web site: http://www.frenetic-lang.org/pyretic.

Srimat Chakradhar (NEC Labs Princeton) asked about the things one can do with OpenFlow but cannot with Pyretic. In terms of capabilities, Joshua did not think there was anything one could implement with OpenFlow but not with Pyretic, though he noted that what one does give up is the ability to finely manage table resource usage, much in the same way that a Java programmer can no longer manage physical registers and memory. Chakradhar also asked about the impact on end-to-end latency. Joshua replied that the current prototype is mostly an interpreter, but with the micro-flow compiler that will be released shortly one would have the same performance. Rajesh Nishtala (Facebook) asked about the impact on queuing behavior. Joshua responded that they hadn't yet done this testing, but expected performance comparable to other systems.

VeriFlow: Verifying Network-Wide Invariants in Real Time

Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey, University of Illinois at Urbana-Champaign

Ahmed Khurshid started his talk by discussing the existing challenges for networking debugging: tons of devices running different protocols, from various types and vendors, and intricate relationships between devices, with several operators configuring them, among others. These challenges make it difficult for one to test every possible scenario. As a consequence, bugs may go hidden and affect production networks in a variety of ways (e.g., degrading their performance and/or making them more prone to attacks).

Ahmed enumerated some existing network debugging techniques, e.g., traffic flow monitoring and configuration verification. With regard to configuration verification, he emphasized a crucial problem: the only input taken is configuration scripts. Everything else (control-plane state, data-plane state, and network behavior) is predicted. To bridge this gap, Ahmed introduced a novel approach, data-plane verification, which considers data-plane state as input for the verification process, thus making it less dependent on predictions and closer to actual network behavior. He also emphasized running this verification task in real time in contrast to existing approaches such as FlowChecker, Anteater, and Header Space Analysis, as the network evolves.

This approach brings us to VeriFlow, a tool for checking network-wide invariants in real time. VeriFlow introduces a layer between the controller and devices in a Software Defined Network (SDN), thus enabling the verification of rules before they are deployed in the network. In summary, VeriFlow operates by intercepting new rules and building the equivalence classes associated to them, a set of packets that are affected by the rule. For each equivalence class computed, VeriFlow generates individual forwarding graphs, which model the forwarding behavior of the packets in the equivalence class through the network. Finally, VeriFlow runs custom queries to find problems. VeriFlow has a set of built-in queries to verify certain invariants; however, it also exposes an API for writing custom invariant checkers. During his talk, Ahmed extensively discussed the computation of equivalence classes (which use multi-dimensional prefix tries), always highlighting the aspects that make it an efficient and quick process. From the set of experiments carried out, one of the main takeaways was that VeriFlow checked 97.8% of the updates from a real BGP trace within one millisecond. Some updates took longer, however, because of the larger number of equivalence classes affected by new rules.

Sanjay Rao (Purdue University) asked about the coverage of error detection, the types of errors VeriFlow cannot detect. Ahmed argued that if the problem is visible at the data plane, it can be detected. Rao continued by asking about errors that span to multiple devices. Ahmed replied that yes, these type of errors can be captured as well, since VeriFlow has a global view of the network. Yan Chen (Northwestern University) asked whether VeriFlow is applicable to inter-domain routing. Ahmed replied that yes, VeriFlow can be used in this context as long as the controller is able to receive reports about changes. He also mentioned that the accuracy of the detection depends on receiving updates of network change in real time. Masoud Moshref (University of Southern California) asked whether one can reorder rules in order to improve VeriFlow's performance. Ahmed answered that they have not looked at this option yet, but they want to investigate it. He also said that as VeriFlow targets real-time processing of the stream of rules coming from the controller, it may not have the liberty to reorder those. Takeru Inoue (Japan Science and Technology Agency) asked about VeriFlow's memory requirements for verification. Ahmed replied it is expensive; for a BGP experiment shown in the evaluation section, VeriFlow took 6 GB of memory.

Software Defined Traffic Measurement with OpenSketch

Minlan Yu, University of Southern California; Lavanya Jose, Princeton University; Rui Miao, University of Southern California

Lavanya Jose started her talk enumerating some questions that existing measurement technologies are either unable to answer, or would require a prohibitive amount of resources to do so: who is sending a lot of packets to a specific subnet? how are flow sizes distributed? is there anyone doing a port scan? etc. NetFlow cannot answer many of the questions posed. For example, NetFlow typically does not sample light flows, such as port-scanning flows. Increasing sampling rate is an option, but then it becomes resource-consuming. Streaming algorithms could be used as well, though each algorithm answers one question only.

Given the above, the question now is: what measurement architecture can answer all the questions? The answer is OpenSketch. OpenSketch is a software-defined traffic measurement architecture that separates the measurement control and data-plane functions, and uses sketches (Count Min Sketch) as building blocks to provide a generic and efficient measurement API. Lavanya discussed the basics of sketches, highlighting the tradeoff between memory consumption and accuracy of the resulting measures. The error can be estimated, which thus can indicate the degree of confidence one can have in the accuracy of the obtained measurements.

There is an issue with sketches, however: each one can estimate only one function. To solve this, Lavanya indicated a solution based on a three-stage pipeline that can support many sketches. The first stage in this pipeline is to hash the packet (based on the header), then classify it (based on the packet header and hash values), and finally update a set of counters based on the results of the previous stages. This pipeline can be configured in the controller plane in order to obtain the required measures and implement the measurement tasks to solve the questions initially posed. Lavanya then discussed possible strategies for implementing sketches with the pipeline, how one can provision the pipeline so that one can implement multiple, different sketches, and the evaluation results. The main takeaway? OpenSketch truly adheres to the SDN philosophy: separate the measurement control and data-plane functions, and make measurement in switches efficient and easy.

After the talk, Dejan Kosti (Institute IMDEA Networks) asked about the possibility of achieving throughput of 10 Gbps. Lavanya said it is possible, but sequentially updating the SRAM might become a bottleneck for tasks that update many counters per-packet. Dejan then asked about the limitations of OpenSketch. Lavanya replied that the data plane is somewhat limited so that OpenSketch can be made simple enough to implement with commodity hardware and operate at line rate. For example, some sketches cannot be implemented as they use more complex data structures (such as binary trees or heaps) not provided by the data plane.

Pervasive Computing

Summarized by Scott J. Krieder (skrieder@iit.edu)

V-edge: Fast Self-Constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics

Fengyuan Xu, College of William and Mary; Yunxin Liu, Microsoft Research Asia; Qun Li, College of William and Mary; Yongguang Zhang, Microsoft Research Asia

Fengyuan Xu presented V-edge, a joint work aiming to improve the accuracy of battery prediction technologies. The idea is to consider power consumption and current system activities to accurately predict power runtime. By taking a snapshot of system activities the authors can determine CPU usage, backlight settings, and provide a calculation of power usage. A power model for smartphones needs to be an abstract concept, applying to the many different phone developers. Two of the most common ways to measure power consumption is through external metering and self-metering. This work uses self-monitoring, but applies unique algorithms for a fast and accurate measuring. The authors capture instantaneous current changes that lead to instantaneous output voltages. V-edge offers the advantages of accuracy and stability, and it's been tested on eight different batteries from two different smartphones. The reading is fast, as fast as the battery can provide an update rate. The V-edge solution consists of a training stage and an estimation stage. V-edge consists of an event-driven design, which provides a low overhead. After their power profiler runs, you will know which application consumes how much power and where.

Man Dong (Samsung Electronics) asked about the resistance in the battery pack, which is sensitive to temperature. Fengyuan Xu replied that in this case you don't need to know the absolute, as they work with the relative changes. Srimat Chakradhar (NEC Labs) asked that if they are sampling at one hertz, don't they miss quick events that only take a few milliseconds but happen with high frequency? Xu answered that V-edge only needs to provide power information during the training state. The event-driven design is quite accurate, comparable to fine-grained monitoring.

eDoctor: Automatically Diagnosing Abnormal Battery Drain Issues on Smartphones

Xiao Ma, University of Illinois at Urbana-Champaign and University of California, San Diego; Peng Huang and Xinxin Jin, University of California, San Diego; Pei Wang, Peking University; Soyeon Park, Dongcai Shen, Yuanyuan Zhou, Lawrence K. Saul, and Geoffrey M. Voelker, University of California, San Diego

Xiao Ma explained that the motivation for this work is abnormal battery drain (ABD). ABD is a condition on smartphone devices where the battery begins to drain at a significant rate without the authorization of the smartphone user. The authors developed an application called eDoctor which can identify the issue and suggest a solution to the user. ABD is often caused by an application update or software change, and eDoctor uses snapshots and installation logs to identify recent changes on the system. The application then suggests which apps to revert or remove to shift the battery drain back to normal consumption.

During a user study with 31 volunteers and 50 cases, the application was able to diagnose 47 of the cases accurately. This user study consisted of 6,000 device hours where a hidden bug was added to a user device. The bug was then activated and the users needed to use eDoctor to diagnose the problem.

Weverton Cordeiro asked whether they could identify bugs caused by updates to the Android OS system version. Ma said that they avoided telling users that a system update was the problem, as that is hard to undo. Cordeiro than asked about battery failure. Ma answered that most problems are software, and they can't detect hardware failures. Someone from Northeastern asked whether collecting data was required and was user privacy a concern. Ma said that they don't observe what the user does in an app, just the power utilized by an app. Also, everything is local on the phone.

ArrayTrack: A Fine-Grained Indoor Location System

Jie Xiong and Kyle Jamieson, University College London

ArrayTrack is an indoor tracking system that can calculate a position down to a 23 cm level of accuracy when using access points (APs) with eight antennas. The motivation for this work is that GPS does not work indoors, and even in cases when it does, provides accuracy at the level of meters. Some future solutions include augmented reality on smartphones, or wearable glasses. The use cases for such a technology include trying to find a book in the library, an item in a supermarket, or a piece of art in a museum. Many works have tried to use WiFi, Infrared, and other technologies over the years, but none are able to calculate with the level of accuracy that ArrayTrack provides. The theory of operation requires there to be multiple antennas and radios at each AP. By locating a device and its rate of movement from the AP, the authors can calculate the angle of arrival from the device. The authors then calculate the distance based on the rate of arrival. Another benefit of ArrayTrack is that it can accurately calculate position and distance based on a single packet. Future work includes conducting studies on how the height of APs and clients affects results.

Phil Levis (Stanford) asked how many antennas the client has and what effect does MIMO have? Jie Xiong answered that they assume the client has only one antenna, and if the client has more than one antenna there will appear to be multiple locations. For MIMO (multiple input, multiple output), they did the processing at the MIMO site, and there they needed multiple antennas. Keith Winstein (MIT) asked whether they needed to know the exact location and orientation of each AP. Xiong replied that it did make a difference with linear-oriented antennas. If they used a circular array of antennas, they could avoid concern with orientation. Winstein then wondered whether they could bootstrap to find the orientation of the linear arrays, and Xiong said that he thought that might work.

Walkie-Markie: Indoor Pathway Mapping Made Easy

Guobin Shen, Zhuo Chen, Peichao Zhang, Thomas Moscibroda, and Yongguang Zhang, Microsoft Research Asia

Guobin (Jacky) Shen explained that their research used WiFi client devices to generate maps of the given location. The motivation for their work is that mapping applications often assume that the map is predetermined or given. But in many cases this is not possible.

Consider the problem of generating electronic maps for a large number of buildings. Neither hiring someone to do this manually or requesting the floor plans for every building is realistic and both would be prohibitively expensive. Thus the authors set out to calculate maps dynamically based on where the WiFi client traffic was coming from throughout a building. The system then generates digital landmarks based on high traffic areas and connects these landmarks based on how client traffic traverses a building. Some challenges of this approach include noisy traffic that causes difficulty in the tracking data, the difficulty of tracking multiple users at a given time, and the need to handle user diversity as well as device diversity. The landmarks are easy to determine, but they simply need to be determined by the device not by the user.

The authors calculated maps by having users walk in a space for 20, 30, 50, and 100 minutes. This is where the work gets its name; the idea is that by walking you mark locations with WiFi landmarks. The maps improve with time, and future work is aimed to reduce the time it takes to collect an accurate map. One drawback to this approach is that the maps are limited to where the clients are walking, and some areas such as closets or some routes that clients do not walk on will not be included in the map. Future work also seeks to improve data captures from lifts and stairwells.

Phil Levis, the session chair, pointed out that signal strengths vary over space. Shen replied that they don't need WiFi marks everywhere as they can still run their algorithms. Steve Tarzen (Vaporstream) asked whether random walks change the problem. Shen said that they split the readings into one-minute segments, and even with random walks, this works as long as they got data from a lot of users. Someone asked whether building materials could cause problems. For example, hospitals have lots of metal objects. Shen replied that it was the stability of the environment that was the key to their system working. Moving objects would affect the measurements, though.

Network Integrity

Summarized by Arpit Gupta (agupta13@ncsu.edu)

Real Time Network Policy Checking Using Header Space Analysis

Peyman Kazemian, Michael Chang, and Hongyi Zeng, Stanford University; George Varghese, University of California, San Diego and Microsoft Research; Nick McKeown, Stanford University; Scott Whyte, Google Inc.

Peyman started by discussing the importance of policy checking. Network state changes all the time, rules keep on changing, and, as a result, there is the potential for policy violations. Thus it is important to have policy checkers that are real-time for the entire network. Kazemian proposed NetPlumber, which improves on Header Space Analysis (NSDI '12) by enabling real-time checking of updates. NetPlumber is suited for SDN as it can tap into the centralized control data to perform real-time policy checking. Also, NetPlumber can be used for legacy implementations.

The heart of NetPlumber is the plumbing graph, which captures all possible paths of flows through the network. Nodes in the graph correspond to the rules in the network, and directed edges represent the next-hop dependency of these rules. This graph is a perfect tool to check policy in a network, react to violations, evaluate complex policies, etc. Peyman also talked about Flow Exp, which is a regular-expression-like language, to verify policies in complex networks. The work was evaluated over three real-world networks: Google WAN, Stanford's backbone network, and the Internet2 nationwide network. Rule-update verifications are on the order of sub-milliseconds, whereas link updates are a few milliseconds.

Someone asked Kazemian to compare/contrast NetPlumber with VeriFlow. Kazemian replied that performance-wise they're equivalent, as NetPlumber can handle any sort of flow and is independent of where wild cards are used. NetPlumber has more of an agnostic approach and is more generic. Though NetPlumber can't verify performance, it can check for loops, blackholes, etc., but not throughput guarantee. Ethan Katz-Bassett, the session chair, asked about the possibility of using NetPlumber for non-SDN networks. Kazemian said that you can use NetPlumber, but it will be using a snapshot and might miss things.

Ensuring Connectivity via Data Plane Mechanisms

Junda Liu, Google Inc.; Aurojit Panda, University of California, Berkeley; Ankit Singla and Brighten Godfrey, University of Illinois at Urbana-Champaign; Michael Schapira, Hebrew University; Scott Shenker, University of California, Berkeley and International Computer Science Institute

Aurojit Panda started with the difference in time-scale of operations for control and data planes. Control-plane response to link failures is slow, and current solutions rely on precomputed backup paths. Such backup paths make sense for single link failures and are hard to generalize for multiple-link failures.

Panda suggested that the question to ask is whether we can push this to data plane. Such a solution will be impossible if we put constraints like no FIB (Forwarding Information Base) changes at packet rate and/or no additional data in packet headers. Their approach is to relax a few constraints—for example, allow changes to a few bits in the FIB at packet rates. Their solution is to take advantage of redundancy by extending routing tables with other paths to a destination, and to restore connectivity at data speeds using a strategy of reverse reconnect. Using reverse to reconnect means to start at the disconnected node to attempt to rebuild the DAG (Directed Acyclic Graph). Enabling reversals in the data plane means two challenges must be addressed, namely lost or delayed notification. A safe control plane is proposed,, which should not interfere with the data plane.

They evaluated their solution on WAN and datacenter topologies over NS3 to test for stretch, throughput, and latency. They also analyzed the effect of FIB update delays on latency and throughput, and end-to-end benefits of using DDC.

Omar Javed (University of Washington) asked how the concept of storing multiple links is different from the multiple path concept. Also, was it possible to enable link reversal for the wider Internet, where complex business agreements exist? Will it work for inter AS? Panda replied that link reversal is different from multipath as it is less restrictive. He also explained that current work focuses only on intra-domain routing and considers only a single AS.

Michael Freedman (Princeton) asked how to characterize the number of bits to change. Is the value three referred to in the paper a hard limit, and why was a higher value not chosen? Panda replied that the current algorithm is a simple one and he is not sure whether higher values will work as expected.

Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets

Rahul Potharaju, Purdue University; Navendu Jain, Microsoft Research; Cristina Nita-Rotaru, Purdue University

Rahul Potharaju explained that trouble tickets for network management are common and fixing them as soon as possible is currently the prime focus. What is missing is how to learn from such mistakes/problems/issues.

The main goal of this work is to analyze these tickets and extract three key features for summarization: problem, activities, and action. The information in a ticket has structured fields and free text. The goal is to use the free text to extract features. This choice is based on learning that structured field data is coarse grained and does not provide much information. A strawman approach of applying natural language processing (NLP) does not work because it is suited for well-written texts and ignores context. Their solution, NetSieve, combines NLP with semantics. In this context, knowledge is like a dictionary that needs to be built first before they choose to infer semantics for a ticket. It involves three steps: (1) repeated phrase extraction, i.e., extracting *n*-grams from words' pool-set, trading completeness for scalability; (2) knowledge discovery, which applies a pipeline of linguistic filters to determine domain-specific patterns; and (3) ontology modeling used to determine semantic interpretations. The evaluation was done for two standard metrics of accuracy percentage and F-score.

Yan Chen (Northwestern) asked about the status of the source code release because he envisioned using these techniques in other domains too, in analyzing security logs, for example. Rahul briefly replied that the IP for this project is owned by MSR, and there was a burst of laughter in the conference hall. Ahmed Khurshid (UIUC) asked what happens in cases where two errors are related to each other. Rahul replied that they found that such correlations showed improvements and are shown in use cases in the paper.

Abhishek Sharma (NEC) asked why the authors used area experts, and were other data mining (unsupervised) techniques used to compare results. Rahul affirmed exploration in that direction. He said they wanted to take a different approach with a semi-supervised approach; being domain-specific also helped in terms of performance.

Data Centers

Summarized by Imranul Hoque (ihoque2@illinois.edu)

Yank: Enabling Green Data Centers to Pull the Plug

Rahul Singh, David Irwin, and Prashant Shenoy, University of Massachusetts Amherst; K.K. Ramakrishnan, AT&T Labs—Research

Rahul Singh started his presentation on Yank by mentioning that applications hosted in modern datacenters assume always available stable servers. In order to guarantee server availability, datacenters employ a highly redundant power infrastructure, which is expensive. So applications often relax this strict stability assumption and compensate for it using low-cost high availability techniques in software. Singh then introduced a new abstraction of transient servers, which, unlike stable servers, have unpredictable availability; however, these servers receive advance warning either from the software (Amazon spot instances) or from the hardware (UPS units) prior to termination. Yank enables datacenters to use the mix of stable and transient servers transparently while maintaining application availability.

Singh presented two ways of supporting transient servers: by modifying individual applications and by providing system support. Yank adopts the latter approach because the former can be challenging for certain classes of applications. Upon receiving a warning, a transient server transfers its VMs to a stable server to ensure that the application is always available. The transfer has to be completed within the warning period to ensure that no state is lost. Singh mentioned two strawman approaches for transferring VM states: the live migration approach, which has low overhead but requires a large warning period, and the backup VM high availability approach, which supports low warning time but incurs high overhead. These two approaches fall on two ends of a spectrum. Singh pointed out that Yank covers the entire spectrum by adapting to the warning time. When the warning time is low, Yank is similar to the high availability approach. On the other hand, when the warning time is high, Yank behaves like the live migration approach. Singh then presented the high-level design of Yank, which consists of a snapshot manager that runs at each transient server and is responsible for sending VM states to backup servers; a backup engine that runs at each backup server and is responsible for storing multiple transient servers' snapshot; and a restoration service that runs at stable servers and is responsible for restoring VMs when the transient servers receive warnings.

Singh then presented their evaluation of Yank. First, he showed that when the warning time increases from five seconds to 20 seconds, the amount of data transferred from the transient to backup servers reduces by a factor of 70. This is because, with a higher warning time, the entire state of a program is small enough to be transferred after receiving the warning signal. Second, he showed that a five second warning time is sufficient to bring down the client-perceived response time by a factor of 20. He also showed that a 4 GB backup server can support 15 transient VMs and claimed that powerful backup servers will be able to support hundreds of VMs. Finally, Singh showed that transition from stable servers to transient servers and vice-versa do not have any visible difference on the response time—thus, he claimed that Yank masks applications from transiency due to changing power availability.

Rajesh Nishtala (Facebook) asked what percentage of an application's memory is needed to reconstruct a live instance and why Yank moves states as opposed to reconstructing states, which can be done in a shorter time period. Singh replied that their experiments showed that the state of an application is much smaller than the total allocated memory. So their design is guided by this observation of small working sets. Dave Andersen (Carnegie Mellon University) asked about the benefit of Yank in terms of energy saving or carbon footprint reduction, because Yank assumes a grid power supply in addition to the renewable power supply. Singh answered that a datacenter can leverage the on-peak and off-peak electricity pricing in order to decide when to switch to renewable energy and back. For example, the energy cost can be reduced if, during an on-peak hour, all the workloads can be moved to transient servers. Andersen was not satisfied with the answer and said that he would continue the discussion later. He then asked whether the same techniques could be used to ensure VM migration within a bounded time. Singh answered affirmatively by saying that Yank gives users a knob to change the warning period and thus reduce the overhead of maintaining VM backups.

Scalable Rule Management for Data Centers

Masoud Moshref and Minlan Yu, University of Southern California; Abhishek Sharma, University of Southern California and NEC Labs America; Ramesh Govindan, University of Southern California

Masoud Moshref started his presentation on vCRIB by pointing out that datacenters use rules to implement management policies. Rules are saved on predefined fixed machines (hypervisors, switches, etc.). On one hand, machines have limited resources (i.e., they can support a limited of rules). On the other hand, future datacenters will have many fine-grained rules ranging from millions to billions of rules. So, it is necessary to offload rules, which creates a tradeoff between resource and bandwidth usage. Moshref then pointed out some challenges of rule offloading. First, an offload scheme must preserve the semantics of overlapping rules. Second, the scheme must respect resource constraints. Third, the scheme must minimize traffic overhead. Fourth, and finally, the scheme must handle dynamics of traffic and rule changes and VM migration.

Next, Moshref presented the design of vCRIB, which is a virtual cloud rule information base. vCRIB provides a proactive rule placement abstraction layer to the operator. The input to vCRIB is a set of rules, and the output of vCRIB is a minimum-traffic feasible-placement. Moshref then pointed out how vCRIB addresses the challenges of rule-placement mentioned earlier in the talk. First, in order to ensure that the semantics of overlapping rules are unchanged, vCRIB uses a source partitioning with replication approach. Second, it uses a resource-aware placement algorithm known as First Fit Decreasing Similarity (FFDS) to find feasible placement of the rules. Third, vCRIB minimizes traffic overhead by refining the feasible placement using a traffic-aware refinement approach. Finally, vCRIB handles dynamism by re-running both the placement and refinement steps if the dynamics converted the feasible placement into an infeasible one. In the case when the placement is still feasible, vCRIB only re-runs the refinement step.

Finally, Moshref presented an evaluation of the vCRIB system. He compared vCRIB against source-placement, in which rules are saved at the traffic source. His simulation results revealed that vCRIB found low traffic feasible solutions. Additionally, adding more resources helps vCRIB further reduce traffic overhead. Moshref concluded his talk by mentioning several future works, which included supporting reactive placement of rules, splitting partitions when the number of rules becomes large, and testing for other rule sets.

Michael Piatek, the session chair, asked whether there is a way to limit the number of rule changes that happen at any point of time, because drastic rule changes may cause problems. Moshref replied that any such constraints will have to be set in the refinement algorithm by setting the appropriate budget. He also mentioned that it can be a good future step. Piatek followed up by asking whether setting a constraint will prevent the algorithm from finding a feasible placement. Moshref asked Piatek whether he was concerned about finding a feasible placement as opposed to refining for minimum traffic overhead. Piatek confirmed this and Moshref replied that in his algorithm he does not consider previous placements in order to find future placements; however, similar algorithms can be found for virtual machine placement and can be adopted for this scenario. Finally, Piatek asked what was causing the rule explosion in datacenters. Moshref replied this was happening because of the growing scale of datacenters and because of the way policies were written to regulate traffic within a datacenter (e.g., between pairs of virtual machines within a datacenter).

Chatty Tenants and the Cloud Network Sharing Problem

Hitesh Ballani, Keon Jang, Thomas Karagiannis, Microsoft Research, Cambridge; Changhoon Kim, Windows Azure; Dinan Gunawardena and Greg O'Shea, Microsoft Research, Cambridge

Keon Jang focused on how to share the network in multi-tenant datacenters. He pointed out that multi-tenant datacenters consist of both intra-tenant (VM-to-VM) and inter-tenant (VM-to-storage) traffic. Because the network is shared by multiple tenants, network performance of tenants is interdependent. Jang mentioned three requirements for network sharing: a minimum bandwidth guarantee, upper-bound proportionality, and high utilization. Jung pointed out that no prior work satisfies all three requirements—they focus on intra-tenant traffic only; however, inter-tenant traffic accounts for 10–35% of the overall traffic. Additionally, guaranteeing minimum bandwidth and ensuring proportionality for inter-tenant traffic is harder.

Jang then presented an overview of Hadrian, which satisfies the above three requirements. Hadrian uses a hierarchical hose model. In this model, tenants can separately specify inter-tenant bandwidth requirement and communication dependency. This guides the placement of VMs across the datacenter. Hadrian uses a hose-compliant bandwidth allocation scheme, which ensures upper-bound proportionality and provides minimum bandwidth guarantees.

Jang then presented an evaluation of Hadrian through real-world deployments as well as large scale simulation experiments. He showed that job completion time is 3.6x faster using Hadrian. This is attributed to the better and predictable network performance as well as efficient resource utilization offered by Hadrian. His results also verified that Hadrian satisfies upper-bound proportionality. Finally, Jang pointed out that a Hadrian cluster accepts 37% more jobs compared to a non-Hadrian cluster. Thus, Hadrian enables providers to offer improved service at a lower price.

Rajesh Nishtala (Facebook) asked Jang to comment on the multi-tenancy of low-bandwidth, low-latency applications and high-bandwidth, high-latency tolerant applications coexisting in a single environment in the Hadrian model. Jang replied that latency is out of scope of their work; however, bandwidth reservation prevents the network from getting over-utilized, so latency remains low. Sanjay Rao (Purdue University) asked how easy it is to predict the inter-tenant bandwidth and whether it changes a lot over time. Jang replied that it would depend on the application. For example, in MapReduce, where the input data size is known a priori and where users can reason about the job deadline, bandwidth prediction is easy. Rao wanted to know whether they make the assumption that the bandwidth is specified by tenants. Jung answered affirmatively. Rao then queried about the complexity of dependencies among tenants and whether Hadrian uses the dependency relationships among tenants to make placement decisions. Jung replied that tenants only specify their total bandwidth requirements and Hadrian's placement decisions are solely guided by these bandwidth requirements, not by the dependency relationships. Michael Piatek (Google) commented that in case of all-to-all communication the placement problem is tricky. This is specifically true for services like storage providers. He asked whether providing some hint to the placement algorithm to make better placement decisions in case of services is possible. Jung replied that in this work they did not consider this approach but certainly these hints can help the placement engine to make better decisions.

Effective Straggler Mitigation: Attack of the Clones

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, University of California, Berkeley

Ganesh Ananthanarayanan presented his work on effective straggler mitigation in the case of interactive data analytics. He pointed out that interactive jobs are common in today's clusters, and the number of interactive jobs is expected to grow further. These jobs are small in size, and low latency is crucial. For example, in Facebook's Hadoop cluster, 88% jobs operate on 20 GB of data and contain fewer than 50 tasks. These interactive jobs are sensitive to stragglers. Existing straggler mitigation techniques, such as blacklisting and speculation, are ineffective for these small jobs (6x–8x slower compared to the median task size).

Ananthanarayanan proposed proactively launching multiple clones of a job and picking the results from the earliest clone. This approach probabilistically mitigates stragglers. Because most of the small jobs use only a small fraction of resources in a cluster, to clone small jobs with only a few extra resources is feasible; however, cloning creates I/O contention. To avoid the contention, every clone should get its own copy of data. In MapReduce this data may either be the input data (replicated) or the intermediate data (not replicated). Ananthanarayanan considered the harder case of intermediate data and showed that job-level cloning is not effective in mitigating stragglers with a small number of clones; however, task-level cloning can solve the problem by using only a few (3) clones. Next, he presented two schemes for contention avoidance and straggler mitigation for intermediate data: contention-avoidance cloning (CAC) and contention-cloning (CC). He showed that CAC avoids contentions but increases vulnerability to stragglers. On the other hand, CC mitigates stragglers but creates contentions. He proposed solving the problem associated with CAC and CC by an approach called "delay assignment," where a small delay is assigned to get an exclusive copy before contending for the available copy. He also mentioned that jobs are cloned only if sufficient budget (resource) is available.

Finally, Ananthanarayanan presented evaluations of his system (called Dolly) by using workloads from Facebook and Bing traces. He compared Dolly against two prior approaches called LATE and Mantri. His experiments showed that jobs are 44% and 42% faster compared to LATE and Mantri, respectively. Additionally, the slowest task is only 1.06x slower compared to the median task (down from 8x). He also showed that the proposed delay assignment technique is critical for achieving better performance, and with the increasing number of phases in jobs, the benefit of this technique increases further.

Christopher Stewart (Ohio State University) asked how a small job was defined in terms of the number of tasks in that job. Ananthanarayanan replied that they avoided defining and specifying which jobs were small. This notion was captured by the cloning budget. A specific number of clones for a job are created as long as resources are available. Stewart followed up by asking whether there was any insight on how to set the cloning budget. Ananthanarayanan mentioned that it would require sensitivity analysis, which is mentioned in the paper. The cloning budget depends on the knee of the power-law curve of the job size in a specific workload. Sanjay Rao (Purdue University) asked whether cloning could backfire, i.e., whether the proposed techniques depended on the fact that stragglers were random and there was no correlation between them. Ananthanarayanan replied that cloning may not be effective in case of data skew (i.e., when one task has more data to process than the other). Rao then refined his question by asking whether straggler nodes were correlated or not. Ananthanarayanan replied that according to their observation, stragglers were uncorrelated. For this reason, techniques such as speculation were effective; however, what Rao suggested might be interesting in order to make placement decisions for speculative clones. If a specific rack is known to be faulty, then avoiding that rack might be better. They did not explore these cases in their work.

Substrate

Summarized by Joshua B. Leners (leners@cs.utexas.edu)

Wire Speed Name Lookup: A GPU-Based Approach

Yi Wang, Tsinghua University; Yuan Zu, University of Science and Technology of China; Ting Zhang, Tsinghua University; Kunyang Peng and Qunfeng Dong, University of Science and Technology of China; Bin Liu, Wei Meng, and Huicheng Dai, Tsinghua University; Xin Tian and Zhonghu Xu, University of Science and Technology of China; Hao Wu, Tsinghua University; Di Yang, University of Science and Technology of China

Bin Liu explained that name lookup is an important problem, both in the context of networking and in domains outside of networking. This paper focuses on a particular use case of name lookup: routing in a content-centric network. Content-centric networking uses hierarchical names (e.g., /com/google/maps) of arbitrary length rather than fixed-length addresses to route packets. To be useful, content-centric networking needs a lookup mechanism that supports longest prefix matching, has high throughput (saturates a 100 Gbps Ethernet link), and low latency (100 microseconds per lookup).

Satisfying these constraints requires a carefully designed data structure that can work efficiently with GPU hardware. Some straightforward approaches won't work: character tries and state transition tables require too much memory. Aligned transition arrays (ATAs) can greatly compact the information of a transition table, but they don't support incremental update and are still inefficient (each character must be looked up separately). To address both of these concerns, the authors implemented multi-ATAs, which can use multiple characters in each transition and support incremental update.

There are some challenges to implementing multi-ATAs on a hybrid CPU-GPU platform: the PCI-e bus and GPU processor are limiting factors in achieving both high-throughput and low latency. Two techniques are used to improve performance. First, pipelining lookup and data transfer improves PCI-e bus and GPU processor utilization. Second, interweaving the memory layout of the input names in GPU memory reduces the memory accesses of threads, improving performance.

The implementation of the multi-ATA data structure performs well on the CPU-GPU platform. Using the multi-ATA requires two orders of magnitude less space than a baseline state transition table. The techniques to use the GPU efficiently allow up to ~70 million searches per second, an order of magnitude more than a state transition table. Furthermore, these techniques can saturate a 100 Gbps link with latencies of less than 100 microseconds per search.

Dong Zhou (Carnegie Mellon University) pointed out that in their evaluation, they worked only on a local machine but didn't actually transfer data onto a NIC. Zhou wondered whether this was a fair comparison. Bin Liu replied that they only worked within the context of a single machine and that they were looking at using the NIC in future work. Because using the NIC would take additional CPU cycles, Zhou then wondered, would competing with the CPU affect their results? Bin Liu replied that this was something they needed to address but thought that other hardware acceleration could help. Srimat Chakradhar (NEC Labs) wondered if the state tables get larger, would they still fit in the GPU. Bin Liu said that they currently had a 10 million-entry table on the GPU, which uses about 1/3 of the GPU's external memory for a single GPU processor chip, and they had two processor chips on the GTX590 board. They estimated they could keep a 60 million-entry table on this kind of GPU. Newer GPUs had more space, and they thought they could keep a 120 million-entry table on the new hardware.

Michael Freedman (Princeton University) noticed that in the multi-ATA table it looked like collisions map into a second table. This seemed to imply that lookups would require multiple accesses. Bin Liu said that it didn't, and they have a proof in the paper. Freedman then asked whether they considered other hashing algorithms, such as cuckoo hashing, and Bin Liu said that they use a much simpler lookup algorithm. Gun Sirer (Cornell) suggested that they should be mining BitCoins with his GPUs, since, currently, their approach lacked security, as in self-verifying names. Lack of security as a first-class primitive plagues DNS today. Bin Liu said that security is future work.

SoNIC: Precise Realtime Software Access and Control of Wired Networks

Ki Suh Lee, Han Wang, and Hakim Weatherspoon, Cornell University

Ki Suh Lee said that measuring and controlling interpacket delays can enable better understanding of network characteristics and new applications. For example, measuring these interpacket delays can give better characterization of network traffic, and controlling the interpacket delays can create new covert channels. Unfortunately, current techniques are insufficient to measure these interpacket delays precisely; however, the precision of network measurements could be improved with access to the PHY layer: counting the number of PHY layer idle characters between packets can give network research applications sub-nanosecond precision.

Manipulating idle characters requires accessing the PHY, which is currently a black box that hides information (including idle character counts). One approach could use something like BiFocals, which uses physics equipment to measure the PHY layer. Unfortunately, this equipment is expensive ($500,000) and only works offline. Because there is limited access to hardware, the authors propose using software to count idle characters.

The authors implement their approach as a new platform called SoNIC, which ports some of the PHY layer functionality of a 10 Gbps Ethernet link into software. Specifically, SoNIC ports all of the functionality that manipulates bits into software (everything above the scrambler of the physical coding sublayer), but keeps the functions that translate bits into signals into hardware. This split requires high-performance software, which SoNIC implements using three techniques: (1) SoNIC software threads are pinned to an individual CPU core; (2) polling and an optimized DMA, rather than interrupts, are used to interface with the network device; and (3) software is tightly optimized (e.g., by replacing loops with bitwise operators).

These techniques give SoNIC precise measurement and control of interpacket delay and interpacket gaps, which allowed the authors to implement several functions that were previously impossible (including a new timing channel with data rates up to 250 Kbps that is undetectable by current techniques).

Dong Zhou(Carnegie Mellon University) asked whether there are any limitations to applications they can support, because SoNIC appears to require a lot of CPU. Lee replied that the applications must be able to work with data faster than 10 Gbps using the remaining resources of the system. Nikita Borisov (University of Illinois Urbana-Champaign) thought that it's cool that their timing channel attack works even if there are unmodified routers along the path, and wondered whether they had considered cross traffic while evaluating this attack. Lee said that they had, but their paper uses only unloaded routers to demonstrate feasibility. Hongyi Zeng (Stanford University) wondered about the requirements for an FPGA to implement SoNIC. Lee replied that the FPGA must have transceivers that can support more than 10.3 Gbps. Zeng then asked about the use of a CDF graph that showed variations in the interpacket gap, wondering why hardware would exhibit these variations. Lee said that SoNIC has errors because it's timestamping within the network stack, and within the kernel there's a lot of overhead: other tasks, interrupts, etc. Zeng asked about hardware timestamps, and Lee replied that hardware clocks have lower resolution than their techniques. What's cool about SoNIC is that they get really precise timing from counting the idle characters. Junda Liu (Google) noticed that every port had five dedicated kernel threads, and asked whether that required five cores. Lee answered yes, that they pinned each thread to its own core, but the other cores were shared. More CPU-intensive applications (e.g., full packet capture that requires disk access) are impossible right now because of application requirements (must handle > 10 Gbps of data).

Split/Merge: System Support for Elastic Execution in Virtual Middleboxes

Shriram Rajagopalan, IBM T. J. Watson Research Center and University of British Columbia; Dan Williams and Hani Jamjoom, IBM T. J. Watson Research Center; Andrew Warfield, University of British Columbia

Shriram Rajagopalan noted that elasticity, the ability to scale a Web service dynamically to meet demand, has been well-studied in the context of Web applications; however, these applications often depend on middleboxes, such as firewalls, intrusion detection systems, and load balancers, which are not well suited to dynamic scaling, because they maintain state. Because middleboxes are hard to provision dynamically, scalable Web services over-provision their middleboxes or stop using them entirely.

An important insight is that middleboxes are flow-oriented: most processing deals with a single flow in isolation, and that the state associated with these flows is partitionable. Using this insight, the authors classified middlebox state into three categories: partitionable (e.g., flow state), coherent (e.g., counters), and ephemeral, state that is local to a middlebox instance (caches, etc.).

To leverage this insight, the authors implemented FreeFlow, a VMM runtime for middleboxes that can dynamically provision middleboxes. To use FreeFlow, a middlebox developer must annotate the middlebox's state as partitionable, coherent, or ephemeral. At runtime, FreeFlow uses OpenFlow to migrate flows (and their state) correctly to dynamically allocated middleboxes, without disrupting existing traffic. To prevent coherent state from becoming a bottleneck, FreeFlow uses looser consistency semantics for keeping such state in sync. The lessened consistency is not problematic for most coherent state, such as counters for reporting statistics, monitoring thresholds, etc.

Using its Split mechanism, FreeFlow is able to keep latency low by dynamically allocating new middleboxes in the face of increased load. FreeFlow also keeps utilization high with its Merge mechanism: combining middlebox replicas when load decreases. FreeFlow has end-to-end benefits with existing middleboxes. For example, FreeFlow allows Bro, an intrusion detection system, to scale dynamically, performing as well as over-provisioning, with only a minor performance hit after a burst in load.

Masoud Mosharef (USC) asked whether they assumed that the flow entries were not dependent. Rajagopalan replied that they assumed that flows were independent. Mosharef then asked whether they had any suggestions for selecting middleboxes (e.g., to balance network traffic). Rajagopalan answered that implementing policy on top of their mechanism is future work. Anthony Nicholson (Google) pointed out that their implementation requires modifying code, and wondered whether FreeFlow could be made to work with unmodified middleboxes. Rajagopalan replied that, currently, modifying the code is necessary. The idea is that people are already moving away from custom hardware middleboxes to software-based middleboxes that can be deployed and scaled in the cloud. We have the opportunity to get things right the first time by designing these middleboxes properly. Srimat Chakradhar (NEC) asked whether they were assuming locking across middle boxes for coherent state. Rajagopalan answered yes, if the coherent state requires strong consistency. Yan Chen (Northwestern) pointed out that in Bro, there are multi-dependent flows and wondered whether they tried to evaluate the state explosion in keeping these flows together. Rajagopalan said that there is no state explosion as such. There are two ways of tackling this scenario: partition state at larger granularity or abstract the dependencies into a coherent state and synchronize as needed. There is definitely a tradeoff in the granularity of partitioning vs. the ability to finely balance load. They didn't find many interflow dependencies in their evaluation, so there wasn't much problematic state.

Steve Tarzia (Vaporstream) asked why not make the middleboxes stateless, since they were already re-architecting them. Rajagopalan replied that stateless Web applications can afford to look up session state from external systems like databases or key-value stores because they handle thousands of requests per second; however, middleboxes handle millions of packets per second. The latency requirements on middleboxes make accessing an external database impractical.

Wireless

Summarized by Arpit Gupta (agupta13@ncsu.edu)

PinPoint: Localizing Interfering Radios

Kiran Joshi, Steven Hong, and Sachin Katti, Stanford University

Currently, interference is the major cause of poor performance for WiFi networks, and localizing these interfering devices is desirable. Most of the previous work in this area required extensive pre-deployment calibration. Steven Hong presented their solution, which requires minimum calibration and is built on top of existing AP architecture.

Steven also emphasized that this solution cannot only be used for locating interfering devices but also for location-based advertising, indoor navigation, real-life analytics, etc. PinPoint can differentiate between multiple interfering signals, compute line-of-sight angle-of-arrival (LoS AoA) in a non-line-of-sight (NLoS) multipath environment, and aggregate and process noisy data from APs. Computation of LoS AoA in an NLoS multipath environment with multiple antenna requires usage of angle of arrival for interference signals, but the presence of multipaths obscures such an approach. Steven than explained how angle of arrival estimation techniques work in general. Also, LoS detection naively won't work due to a NLoS scenario. Their solution is to use feature vectors and arrival time differences. LoS signals arrive first, enabling identification of LoS signal and AoA eventually.

For evaluations, they used AoA+ CSSI (cyclic signal strength indicator) information and compared it against performance of RSSI-based techniques and MUSIC. Pinpoint leverages existing WiFi infrastructure, provides a better algorithm for LoS AoA estimates, and is capable of differentiating various interference sources. This makes PinPoint a better candidate for interference localization than existing solutions.

Sarthak Grover (Georgia Tech) asked about the CSSI approach, whether it is possible to differentiate between two WiFi signals. Steven replied that it works better if the interfering signals are of different protocols. In the case of two WiFi signals, CSSI information won't be important but AoA will surely be important. A researcher from UCL London said that MUSIC is not suited to indoor scenarios and asked whether theirs was a fair comparison. Steven admitted that the objective for MUSIC is different, and they do focus on the strongest multipath components.

SloMo: Downclocking WiFi Communication

Feng Lu, Geoffrey M. Voelker, and Alex C. Snoeren, University of California, San Diego

Feng Lu started with statistical figures to emphasize the dominance of WiFi radios in consuming power for energy limited smart devices. He then explained how WiFi sleep works and about the focus of researchers to develop better sleep policies in recent years. Most apps are real time and chatty in nature. Feng explained that the data rates for such apps are small, but these apps stay connected more than 62% of the time. In order to identify opportunities to save energy, knowing where energy is spent is important. WiFi radio goes to idle before sleep, which is almost the amount of energy of the transmit (Tx) and receive (Rx) states. As apps require a smaller data rate, but available rates are higher, time spent in Tx/Rx is small and most of the energy is spent in idle state.

Their solution is to downclock the WiFi radio, saving 30–40% energy. Clock rate is gated by the sampling rate, which is higher following the Nyquist principle for WiFi signals. Recent advances in compressive sensing allows them to cheat when the information rate is much less than the signaling rate (11 times). The simple idea is to sample groups of chips rather than a single one, enabling downclocking for WiFi radio. The solution enables downclocking for both transmission and reception.

SloMo is implemented over the Sora (SDR) platform and doesn't require any modification to WiFi APs, with full backward compatibility. Evaluation reveals that there is not much difference for cases where SNR is good and when SNR is poorer. As an example, SloMo energy consumption for the Angry Birds app goes up for Tx and Rx, but significant energy savings are observed for idle times. Similarly, apps such as Skype benefit significantly. They observed that the increase in airtime is less than 13%, ensuring that SloMo is useful and relevant for energy saving.

Philip Levis from Stanford pointed out that this solution works fine at the link level but wondered how it's going to play out at the MAC level. For example, if the receiver is downclocked and AP is sending RTS/CTS, it is possible that the receiver won't be able to decode these RTS/CTS signals. Feng replied that if RTS/CTS are sent at 1 or 2 Mbps, decoding them easily is possible, but for other data rates we may not be able to decode correctly. For the cases when RTS/CTS was sent at 1–2 Mbps, SloMo was able to achieve 80–90% detection rate for low SNR values, so it is not much of a concern. Men Dong (Samsung Labs) was curious how the authors determined the energy breakdown in their experiments. Also, the 700 mW power consumption mentioned in earlier slides does not match the specifications for Qualcomm or TI WiFi chips. Feng explained that the WiFi node sends a null packet to the AP for sleep activities, and they used that knowledge to determine when the AP went to sleep and vice versa. Finally, they combined the power models based on real smartphone measurements to map power consumed for different modes of operation. The energy consumption values mentioned are for data transmissions/receptions. Arpit Gupta (NC State University) asked about scenarios in which multiple apps (data-intensive ones along with chatty ones) concurrently use the network interfaces; what should be the mechanism to switch downclocking ON/OFF for such scenarios? Feng claimed that apps do not actually use network interfaces concurrently, and users interact with one app at a time. Masoud (USC) asked how downclocking affects user experience or the performance of applications like Skype. Feng said that experiments were carried out with Skype and no perceivable impact was observed.

Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks

Manjunath Doddavenkatappa, Mun Choon Chan, and Ben Leong, National University of Singapore

Manjunath Doddavenkatappa talked about the importance of data dissemination and the critical nature of its completion time, which for existing protocols is of the order of a few minutes attributable to contention resolution. The proposed solution, Splash, eliminates the need for contentions and therefore minimizes completion time.

Manju explained how Splash eliminates the need for contention resolution through the use of constructive interference and channel diversity, covering the network with fast, parallel paths (tree pipelining). He further explained how Splash utilizes transmit density diversity, opportunistic overhearing, channel cycling, and XOR coding techniques to strengthen reliability. He emphasized that any missing data is recovered locally and the fact that 90% of nodes have full objects makes local recovery practical.

Evaluation of this work was carried out over two testbeds: Indriya at NUS and Twist at TU Berlin. Various experiments strongly demonstrated that Splash reduced the data dissemination completion time by an order of magnitude. Contributions of individual techniques were also presented giving better insight to various factors responsible for this performance improvement.

Philip Levis complimented the work for the significant gain demonstrated in the paper and asked about the implications for the physical layer, whether there was a need for new physical chips. Manju replied that it depends on the modulation techniques. Masoud (USC) asked whether devices needed to be synchronized and what data rate was required to keep that accuracy. Manju agreed that there is a need for synchronization and that the time difference between different data transmissions should be less than .5 μs to result in constructive interference. Session chair Brad Karp asked about how constructive interference will scale out as you increase the density, because the heuristic used about leaf and non-leaf nodes might not work if the network is too dense and non-leaves are nearly the entire network. Manju said that it is totally random because of the capture effect; it depends on the placements of nodes, and it is difficult to find an optimal number of receivers.

Community Award! Expanding Rural Cellular Networks with Virtual Coverage

Kurtis Heimerl and Kashif Ali, University of California, Berkeley; Joshua Blumenstock, University of Washington; Brian Gawalt and Eric Brewer, University of California, Berkeley

Awarded Community Award!

Kurtis Heimerl mentioned that his talk was about cellular networks, which is not very common at networking conferences. Interestingly, he compared the invention of cellular networks with the light bulb to demonstrate the impact of this technology. This set the tone for the entire talk. Kurtis explained the reasons why rural areas lack cellular network coverage: the investment cost is high and user density is not enough to recover costs. He revealed that half the cost of running a cellular tower in remote areas is power related. Thus, to enable wider coverage for rural areas, power draw must be reduced. Kurtis explained various components of a rural base station tower and the costs for each of its components, making it clear that the lack of power infrastructure was responsible for higher costs.

The power amplifier draws 130 W constantly and is always turned on. Because the number of users is low in rural areas, most of the time nothing goes on and energy is wasted keeping the power amplifier constantly running. Their simple idea is to enable sleep for base stations when not in use. Implementation of this idea is relatively tricky for the user side. Kurtis spoke about two solutions for this problem: wake-up radios and wake-up phones. The fundamental change is to involve users for power provisioning, which is shown to be very common in rural areas. The proposed solution when compared to traditional schemes saves around 84% power.

Alex Snoeren (UCSD) asked about power consumption of receivers for scanning activities, as in the case of WiFi networks; scanning is a power hungry activity, so how does this work impact power consumption for handsets? Alex also wondered about the cost of maintaining armed guards for rural areas. Kurtis replied that scanning is a common activity for cellular networks and results in no overhead for this solution. Labor cost is trivial in these areas, so hiring guards is not much of a cost issue. Rajesh Nishtala (Facebook) asked whether this kind of solution could be used for data activities like checking email. Kurtis said that there is nothing about this work which limits it to calls; it can be used for data activities, too. Though, as these things are asynchronous in nature, some periodic synch activities need to be planned around such a service, but it is surely doable. Josh Reich (Princeton) asked about potential attacks on such a service. Kurtis replied that validation mechanisms make sure that such attacks are not a problem. Josh asked why the authors didn't considered power cycling. Kurtis replied that usage for such a service is mostly for emergency situations and thus timing is an important factor to consider. Daniel Turner (UCSD) asked an economics-based question: who will sponsor such a service, government or rural entrepreneurs? Kurtis said that GSM operation requires a license, and its usage by rural entrepreneurs would require policy changes.

Big Data

Summarized by Muntasir Raihan Rahman (mrahman2@illinois.edu)

Rhea: Automatic Filtering for Unstructured Cloud Storage

Christos Gkantsidis, Dimitrios Vytiniotis, Orion Hodson, Dushyanth Narayanan, Florin Dinu, and Antony Rowstron, Microsoft Research, Cambridge

Christos Gkantsidis presented research on optimizing data analytics in the cloud. Public cloud infrastructures are breaking the locality of data and computation by introducing separate storage and computation infrastructures. This approach has many advantages, but comes at the cost of network transfer. The paper has measurements that characterize the amount of data transfer required for this, and the author mentioned that it is quite significant.

The key insight is that most jobs only operate on a subset of the input data. So filtering the input data before transferring from storage to compute can yield significant performance gains. The authors propose generating network filters that get rid of unnecessary data before starting the network transfer. The filter has to be correct and transparent; however, to filter the data, we need some structure, whereas the input data is usually unstructured. The authors propose using static analysis of job byte-code and extracting row and column filters to discover the data that is actually used in computing. Filters are opportunistic, conservative, and transparent.

Christos also briefly outlined the system design, especially the construction of row and columns filters. A row filter discovers rows that generate some output data. Then column filters identify which substring of that row is of interest. One problem is that some MapReduce programs use state, whereas filters cannot rely on mutable state. The solution is to tag all mutable fields as output. On the other hand, column filters use abstract interpretation and tokenization methods to find interesting columns within a row.

The experimental setup was Hadoop on Windows Azure. The author presented results for eight jobs in the same datacenter; however, implementing filters on Amazon storage is not possible. So the authors took the data, filtered it using Rhea, and compared the job results with the pure input and the filtered input. The authors also observed that the overhead of Rhea is linear in the number of cores. They observed that the filter performance was bottlenecked by string I/O. They saw 30–80% improvement in runtime using Rhea. These numbers turn out to be lower than job selectivity in Hadoop.

Following the talk, Ryan McElroy (Facebook) asked whether the authors tested the filters directly, where storage and compute are already collocated, and whether there were any benefits. Christos replied that they did some experiments using local storage and machines for testing the filter, which means that storage and compute are collocated, and they saw some benefits. The session chair, George Porter, then asked whether declarative cluster computing frameworks such as Spark, Pig, or Hive could help with the static analysis required in Rhea. In a nutshell, George was asking about generalizing Rhea from pure MapReduce programs written in Java to more functional programs like Spark, which is written in Scala. Christos first clarified by saying that in declarative languages, select and project operators are explicit in the code, which Rhea had to discover for MapReduce programs. But even in that case, the user will still need to do static analysis on user-defined functions.

Robustness in the Salus Scalable Block Store

Yang Wang, Manos Kapritsos, Zuocheng Ren, Prince Mahajan, Jeevitha Kirubanandam, Lorenzo Alvisi, and Mike Dahlin, The University of Texas at Austin

Yang Wang presented Salus, a scalable and robust block store. The first priority of any storage system is not to lose data. The problem is exacerbated by remote storage used by most users. Salus adds robustness on top of scalable remote storage systems such as Amazon elastic block store (EBS). There are existing systems that provide strong protection against arbitrary failures; however, these techniques do not go well with online scalable storage systems. This is where Salus comes in; it matches well with scalable block stores. Salus inherits scalability from existing systems, while providing strong robustness with low overhead. The robustness guarantees are that a client never reads corrupted data, and the system remains durable and available despite a fixed number of failures.

Yang then described the architecture of existing scalable systems. The first component is a metadata server, which is rarely accessed to avoid being a bottleneck. Scalability is achieved via parallel access to data servers, whereas the availability and durability guarantees come from replication; however, these systems lack ordered write guarantees and have single points of failure. A block store needs ordering guarantees for implementing barrier semantics. A simple checksum does not suffice to resolve the corruption of data by a compute node.

Salus overcomes these problems by introducing end-to-end verification at the block driver level to prevent corrupt data reads by clients. Also, Salus uses pipelined commits to the servers to implement barrier semantics. The single point of failure is alleviated through active storage in the replication protocol. Yang then went into details about pipelined commits and active storage. A naive approach of waiting for all writes to finish does not work for barrier semantics, since it loses all parallelism. Two-phase commit also falls short because it cannot provide ordering among multiple transactions. Pipelined commit solves this issue by forcing a transaction leader to wait for an acknowledgement from the previous leader. So the pipelined commit still does the first phase of 2PC in parallel but executes the second phase sequentially for barrier semantics. The performance is still good because the message overhead in the second phase is much smaller than in the first phase. Salus handles active storage by decoupling safety and liveness. Safety is still achieved via f+1 replicas, whereas liveness is guaranteed by only storing soft state in compute nodes. Surprisingly, active storage helps with performance because the soft state compute nodes can now be collocated with storage nodes. Evaluation results show that Salus is always safe, and remains live when the number of computing node memory failures doesn't exceed two. The experiments also showed that the overhead of Salus doesn't grow as the system scales.

George Porter asked whether Salus can support multiple concurrent writers. Yang answered that it was still an open problem, but it could be achieved by sacrificing either linearizability for causal consistency or scalability. Salus can handle independent random failures. As a follow-up, George asked whether Salus can handle correlated failures. The author said that they did not experiment with correlated failures and it was left as future work.

MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing

Bin Fan and David G. Andersen, Carnegie Mellon University; Michael Kaminsky, Intel Labs

Bin Fan presented MemC3, which improves upon the basic Memcached via low overhead caching and better hashing. The goal of Memcached is to reduce space overhead and improve performance. The state-of-the-art systems improve performance by sharding, which eliminates inter-thread communication; however, this only works for uniform workloads. For skewed workloads, sharding can lead to hotspots and memory overhead. Instead, the authors try to use space-efficient concurrent data structures for their system via smart algorithmic tuning. Their system has 3x throughput improvement and requires 30% less space compared to the original Memcached system.

Before diving into the details of MemC3, Bin gave a brief overview of the original Memcached architecture and the typical workload for the system. MemC3 optimizes for this typical workload of small objects with high throughput requirements. The core data structures used in Memcached is a key-value index implemented via a chaining hash table, and a doubly linked list for LRU eviction. MemC3 replaces these data structures with efficient concurrent data structures without global locks for improved single-node scalability and reduced space overhead. For hashing, the authors use optimistic cuckoo hashing, which is space-efficient and highly concurrent. For cache eviction, they use CLOCK-based LRU eviction. The rest of the talk focused on optimistic cuckoo hashing.

The default Memcached architecture uses a chaining hash table, which has low cache locality and high pointer cost. Another alternative could be linear probing, which is cache friendly but has poor memory efficiency. Instead, the authors use cuckoo hashing, where each key has two candidate buckets, and lookups read both buckets in parallel. The value is actually stored in just one of the buckets, and this scheme has constant amortized insertion cost. Memory efficiency is further improved to 95% with increased set-associativity. Even though the system only supports single write, multiple reader concurrency, the problem is still hard due to false misses. This happens if a read happens while a recursive insertion is going on. The authors solved this using a two-step insert process. In the first step, the system finds a path to an empty bucket without editing buckets. In the next step, the empty slot is moved back through the reverse path. The only required guarantee is that each swap is atomic. The authors use optimistic locking for the atomic swap, which is optimized for read-heavy workloads. The system also avoids concurrent writes by serializing all inserts, which performs well with read-heavy workloads. Micro-benchmarks reveal that their hash table increases throughput from 1.74 to 21.54 M lookups/sec for all hit lookups, with further improvement for all miss lookups. They also showed that their system scales much better with increased number of threads.

Following the talk, Srimat Chakradhar (NEC Labs) asked whether the authors considered the impact of network performance on MemC3, since the network can be the main bottleneck. He commented that the experiment was masking the true network delay, and that any amount of local computation improvement cannot bypass the network delay. Bin wanted to answer this question offline. Arash Molavi (Northeastern University) asked whether the hash path could end up in a loop. Bin responded that they handled it with a fixed upper bound on retry attempts, and that if a lookup fails after 500 attempts, the table is most likely full. Arash then asked how the authors came up with the magic number of 500. Bin's response was that they used experimental parameter tuning to figure out the optimal number of retry attempts. Sid Sen (Princeton University) suggested that further improvement could be obtained by using overlapping instead of disjoint set-associativity for the hash table. The speaker responded that they tried that idea, but it did not work since they are only storing part of the key for performance reasons.

Scaling Memcache at Facebook

Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani, Facebook Inc.

Facebook is the largest user of Memcache in the industry space, and Rajesh Nishtala began by listing Facebook's key system infrastructure requirements: real-time communication and aggregation from multiple sources, access and update of popular content, and huge scalability. This means that any such system needs to support large read-heavy workloads, must be geo-replicated, and has to support evolution of products; however, Memcache does not need to worry about persistence, which helps it scale. The basic building block for Facebook's infrastructure is Memcached, which is basically a network attached in-memory hash table with LRU-based eviction policies. The rest of the talk gradually explored larger and larger scale usage of Memcached at Facebook.

The first phase was a single front-end cluster that dealt with a read-heavy workload, wide fan-out, and fault tolerance. Before delving into the design, Rajesh briefly mentioned that with the pre-Memcached architecture, a few shared databases sufficed to serve the workload. Obviously, this did not scale, as evident from an example of data fetching for a simple user request that Rajesh pointed out. So the first step was to add a few Memcached servers (around 10) to alleviate the load (about a million operations per second). The typical workload had about twice as much reads as writes, which required higher read capacity. The solution was to use Memcached as a demand-filled look-aside cache, with high probability of lookup success. Updates usually invalidate Memcache entries. Facebook prefers idempotent deletes rather than updates because they can be repeated without loss of data; however, the look-aside cache can lead to stale sets where the Memcache and the database are not consistent. This was resolved with a lease. This still leaves a problem called "thundering herds," where a huge number of Web servers flood the database that stores a popular item once the Memcache entry is invalidated through an update. This issue is resolved by using Memcache as an arbiter to decide who gets access to the database.

Next, Rajesh described the scale increasing to around 100 servers and about 10 million operations per second. Obviously, this required even more read capacity. At this scale, items are distributed using consistent hashing; however, this leads to all-to-all communication between Web servers and Memcache servers, which is bad for the network. One such problem is incast congestion, where multiple Memcache servers reply to the client at the same time, which overloads the client network connection. The solution is a simple sliding window protocol to control the number of outstanding requests.

In the second phase, Rajesh went on to talk about scaling with multiple front-end clusters, which introduced issues like data replication control and consistency. At this scale, there are thousands of servers and hundreds of millions of operations per second. Here the main problems are to keep each cluster of Memcache servers consistent and to manage over-replication of data. The solution is a simple push-based approach, where the database pushes invalidation updates to all Memcache clusters. The network overhead of this broadcast approach is handled using a middle layer of Memcache routers.

In the grand finale, Rajesh went on to the largest scale, that is, multiple regions, where data consistency issues really kick in. At this scale, there are billions of operations per second. The main problem is to handle writes to slave regions. This is resolved by only writing to the master, which works for read-heavy workload, but there can still be race conditions. This is handled via a remote marker, which is a special flag that indicates whether a race is likely to happen. This works because Memcache ensures that misses are rare. Again, the read-dominance of the workload is the reason why this works.

Rajesh concluded with three lessons learned from building massive scale systems. First, pushing as much complexity to the client as possible always helps. Second, operational efficiency is a first-class citizen. And third, separating cache and persistent storage allows systems to scale.

Anirudh Badam (MSR) asked about bottlenecks in a single server. Rajesh answered that their systems are always provisioned for the worst case, which means that single server bottlenecks rarely surface. Badam then asked about using flash-based storage for Memcached. Rajesh said that they use flash mainly for colder data, as opposed to using Memcached for hot data. Dave Andersen (CMU) asked whether the large scale usage of Memcache at Facebook was due to legacy issues. Rajesh answered that wasn't the case. He said scaling Memcache has worked well at Facebook and there is no reason to replace it. Dave then asked about using Memcached outside Facebook. As Bin pointed out in the last talk, the typical Facebook workload has a lot of reads of small data items, so Rajesh reiterated that as the dominant workload that would drive Memcache deployment outside Facebook. Amar Phanishayee (MSR) asked about the size of the cache and the average utilization. Rajesh answered that it depended on the popularity of the item, which is basically the principle of caching. Marcos Aguilera (MSR) asked for a clarification about pushing complexity to the client. Rajesh clarified that the complexity is actually pushed to the client-side library.

Posters

Summarized by Utkarsh Goel & Anish Babu Bharata
{utkarsh.goel, anishbabu.bharata}@cs.montana.edu

Demystifying Page Load Performance with WProf

Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall, University of Washington

Many techniques that focus on reducing the Web page load time (PLT) exist. Even though these techniques aim to provide a low PLT, they are still difficult to be identified due to the complexity of the page load process. The authors have abstracted a dependency graph of the activities that make up a page load and have developed a lightweight in-browser profiler, WProf, to produce this graph. The results obtained from WProf shows that synchronous JavaScript plays a significant role by blocking HTML parsing, and the computation is a significant factor that makes up as much as 35% of the critical path.

Towards A Secure Controller Platform for OpenFlow Applications

Xitao Wen, Yan Chen, Northwestern University, Chengchen Hu, Xi'an Jiaotong University, Chao Shi, Northwestern University, Yi Wang, Tsinghua University

The OpenFlow architecture suffers from trust issues as it embraces third-party development efforts. Most of the apps are highly flexible for defining network behavior, yet the openness and the absence of security enforcement make it difficult to keep such trust on the OF controller, especially on the third-party modules. To provide an intuitive impression, authors have envisioned four classes of attacks: (1) direct intrusion from control plane into data plane,(2) leakage of sensitive information, (3) manipulation of OF rules, (4) deactivation of other apps. As a part of the implementation, the authors have ported the app hub to evaluate the additional controller latency and throughput. Their results show that there were no additional latencies, and even the throughput was reduced by only 1.3%.

Syndicate: A Scalable, Read/Write File Store for Edge Applications

Jude Nelson, Larry Peterson, Princeton University

To preserve data consistency and its durability, coordination between cloud storage, network caches, and local storage is a must. The authors' work is mainly based on following design goals: decouple caching from consistency, decouple storage policy from implementation, and decouple read performance from durability. The authors' early work consisted of implementation of the user gateway as a FUSE file system, the replica gateway as a cloud-hosted process, and the metadata service as a Google AppEngine service.

Dynamic Layer Instantiation as a Service

Flavio Espositox, Yuefeng Wangx, Ibrahim Matta, John Day, Boston University

Current Internet architecture lacks scoping of control and management functions. This results in a challenge to deliver a communication service with required characteristics when the ranges of operations are wide. This deficiency, together with the desire to offer virtualized network services, has compounded existing network service management challenges. The main contribution of the authors is that RINA is capable of enabling private networks to be instantiated dynamically. This is done by customizing network management policies into a single layer, without the shortcomings of the TCP/IP architecture.

WASP: A Centrally Managed Communication Layer for Smart Phone Networks

Murad Kaplan, Chenyu Zheng , Eric Keller, University of Colorado

With the rapid increase in the use of smartphones-based interactive applications, overloading off the cellular network's capacity can be observed, which may quickly expend the limited bandwidth cap. In order to overcome these problems, the authors have proposed WASP, a communication layer within a smartphone that couples cellular connections with direct phone-to-phone links (over WiFi direct). The poster presented by the authors presented the design and implementation of WASP and discussed the design space that was enabled by this new SDN-based model. Their demonstration consisted of three parts: (1) they collected data from a small scale collection of phones, (2) they collected data from a larger scale simulation of the Android application within NS-3 (Network Simulator 3), and (3) they conducted a mixed-mode simulation involving real Android phones participating with the NS-3 based simulation.

Characterizing Broadband Services with Dasu

Zachary S. Bischof, Mario A. Sanchez, John S. Otto, John P. Rula, Fabián E. Bustamante, Northwestern University

The authors presented a crowd-sourced approach to broadband service benchmarking from end-systems. In the context of Dasu, the authors have described its prototype implementation. Dasu is a measurement experimentation platform for the Internet's edge based on the observation that, by leveraging minimal monitoring information from the local host and home routers, one can achieve the scale of "one-time" end host approaches with the accuracy of hardware-based solutions. The demonstration of Dasu consisted of broadband characterization functionality and its user interface. The results observed by the authors showed that by using an end-host approach, the large-scale view allowed them to capture the wide range of service performance as experienced by users over different times of day and across different geographic locations.

Auditable Anonymity

Sonia Jahid and Nikita Borisov, University of Illinois at Urbana-Champaign

The cloud is a common platform these days for storing and sharing data. Deploying applications in clouds brings some new challenges in security and privacy. The authors' work enables the data owner to get logs for data accesses, while hiding the information from the cloud provider at the same time. This is done at the data owner's end by decrypting the audit log and then getting access to all the logs provided by data access.

Self Tuning Data-Stores for Geo-Distributed Cloud Applications

Shankaranarayanan Puzhavakath Narayanan, Ashiwan Sivakumar, Sanjay Rao, and Mohit Tawarmalani, Purdue University

Most of the applications using the Internet share user data in an interactive manner. Moreover, these applications have stringent service-level agreements that are responsible for placing tight constraints. These constraints affect the performance of underlying geo-distributed datastores. The main challenge here is the deployment of such systems in the cloud as application architects have to maintain consistency between multiple replicas, minimize access latency, and ensure high availability. In this poster, the authors have adopted a systematic approach in which they were able to capture the performance of a datastore by developing analytical models. These datastores are based on application workload and are capable of building a system for optimal performance by automatically configuring the datastore.

A Declarative Framework for the Verification of Network Protocols

Jiefei Ma and Alessandra Russo, Imperial College London; Jorge Lobo, Universitat Pompeu Fabra; Franck Le, IBM T. J. Watson Research Center

Verifying network protocols is a challenging problem. In this poster, the authors have proposed a declarative framework that builds upon the recent realization that with simple extensions, database-style query languages can be used to specify and implement network protocols. This framework consists of three components: a protocol model, a communication model, and an analysis model. The authors have observed the following results: in a network running a link state protocol, the presence of persistent forwarding loops was revealed by using this framework, and the framework detected flaws in a MANET protocol that was designed for finding disjoint paths.

Natjam: Prioritizing Production Jobs in the Cloud

Brian Cho, Samsung Inc.; Muntasir Rahman and Indranil Gupta, University of Illinois at Urbana-Champaign; Cristina Abad, University of Illinois at Urbana-Champaign and Yahoo! Inc.; Tej Chajed, University of Illinois at Urbana-Champaign; Nathan Roberts, Yahoo! Inc.; Philbert Lin, University of Illinois at Urbana-Champaign

This project presents the Natjam system, which does both efficient resource utilization and better job run times during overload. This system introduces job and task eviction policies, thus achieving the co-existence of high priority production jobs and low priority research jobs on the Apache Hadoop cluster. This system is well tested, and they presented data that shows Natjam is better than existing systems and relatively close to an ideal outcome.

Consistent Packet Processing—Because Consistent Updates Are Not Enough

Peter Perešíni and Maciej Kuźniar, École Polytechnique Fédérale de Lausanne and Technische Universität Berlin/T-Labs; Nedeljko Vasić, École Polytechnique Fédérale de Lausanne; Marco Canini, Technische Universität Berlin/T-Labs; Dejan Kostić, Institute IMDEA Networks

This project explores the needs for consistent packet processing in software-defined networking controllers and describes the situations in which performance and scalability issues arise even in a straightforward OpenFlow deployment. The project proposes the use of transactional semantics within the controller and explains the benefits it poses over consistent packet processing.

IRate: Initial Video Bitrate Selection System for HTTP Streaming

Ricky K.P. Mok, Weichao Li, and Rocky K.C. Chang, The Hong Kong Polytechnic University

This project proposes IRate, which offers lightweight, fast, and yet accurate decision-making for selecting the best initial video bit rate for a given network condition. As one of its benefits, this system just needs modification on the server and not on any hosts. The results show greater accuracy of the IRate quality oracle against the duration of the probe kit measurement.

Software Defined Measurement for Data Centers

Masoud Moshref, Minlan Yu, and Ramesh Govindan, University of Southern California

Accurate detection of large flow aggregates and choice of better routes for these flows are needed for performing traffic engineering. For reducing latencies of a partition/aggregate, there is a need for identifying short traffic bursts. These measurement tasks vary according to what traffics are required to be measured, where to measure them, and when to measure. For automatically distributing the measurement tasks across all the switches, the authors have proposed leveraging a centralized controller in datacenters. The key challenge here is to identify the right division of labor across the controller and switches. Another challenge is that monitoring a source IP prefix is a collaborative task of multiple switches because a datacenter topology consists of several switches and no switch sees all the traffic.

IP2DC: Making Sense of Replica Selection Tools

Anish Bharata, Mike P. Wittie, and Qing Yang, Montana State University

Most of the cloud-based applications deliver the same level of responsiveness as stand-alone software, which leads to user dissatisfaction and slow adoption. In order to avoid poor user experience due to end-to-end delay or lag, back-end logic and application data are usually deployed across geographic locations. These distributed servers then direct all the user requests to the closest server. The challenge lies in the accurate selection of a server closest to a user, or a group of communicating users. Even though earlier studies have proposed a number of tools for identifying the closest replica server to a client IP, these tools suffer from incomplete coverage of the IP space and can make predictions based on stale network measurements. Hence, which of these tools makes the most accurate prediction in most cases remains unclear. The authors of this poster are interested in their coverage of the IP space and their accuracy in determining the closest public cloud datacenter to a given IP, relative to direct probing.

Reliability

Summarized by Peng Huang (ryanhuang@cs.ucsd.edu)

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson, University of Washington

Awarded Best Paper!

Vincent started by stating that multi-rooted tree topologies such as Fat Tree are preferred in today's datacenter networks for bisection bandwidth and cost concerns. Vincent proceeded to talk about failure detection and recovery problems inside these networks. The way they deal with failure is through heartbeat detection and centralized controller for recovery by exploiting path redundancy. But the detection and recovery are slow, which can lead to suboptimal flow assignment: DC networks use commodity switches that often fail and the particular topology doesn't allow local recovery. Vincent described the basic approach of F10, which achieves fast failure detection and local, fast, and optimal failure recovery by co-designing topology, routing protocol, and failure detection.

Fat Tree is not good at failure recovery primarily because there is no redundancy on the way down and the alternative routes are many hops away. They stem from the symmetries in the topologies: each node is connected to the same set of parents as the siblings and the same set of grandparents as their cousins. F10 breaks these symmetries. To this end, it investigates two types of subtrees, one with consecutive parents (type A) and the other with strided parents (type B). But type A and type B alone is essentially Fat Tree. The authors propose to mix them as an AB Fat Tree: half of the subtrees are type A and half are type B. With this topology, more nodes can have alternatives that are one hop away. Then F10 adds a cascaded failover protocol through local rerouting, notification, and a centralized scheduler.

Vincent went on to explain the failure detector in F10 in detail; it's faster than current failure detection because it looks at the link itself, monitoring incoming bit translation and rerouting the next packets. The key point is that rerouting is cheap in F10.

F10 was evaluated with both testbed (Emulab) and simulation on traces. Results show that failure recovery can be under a millisecond and with only 14% of the congestion loss compared to Fat Tree.

Brad Karp (UCL) asked how the solution compared to multipath TCP. Vincent said they didn't evaluate this, but as pointed out in the paper, they are targeting slightly different cases, where you aren't necessarily able to affect end-hosts, and you want them to use their own OS. Vincent speculated that even with multipath TCP, there would still be performance degradation in that you will loose one of the paths, which potentially could be worse than F10. Siddhartha Sen (Princeton) suggested adding more links, like VL2, so they have a fully connected topology. Vincent replied that he thought that was a Clos topology and that F10 will further optimize that. Even with a traditional datacenter network which has all-to-all connectivity, F10 probably would have a 50% - benefit over Fat Tree. Someone from USC asked whether the solution had impact on the nice load balancing properties of Fat Tree. Vincent replied that F10 doesn't change that. You can do the same thing in F10.

Someone from MSR asked two questions: when a node fails, will reacting too quickly to links cause more problems? Vincent answered that the actual number of messages broadcast is not that high. Also, F10 will do exponential back-offs. Does the global scheduler make an implicit assumption that traffic is stable? Vincent explained that previous work (Hedera and MicroTE) shows that there are LFN (Long Fat Network) flows that are predictable. Even if things are unpredictable, the scheduler can still provide benefits.

LOUP: The Principles and Practice of Intra-Domain Route Dissemination

Nikola Gvozdiev, Brad Karp, and Mark Handley, University College London

Nikola started with users' expectations of Internet reachability as the transport has become reliable and routing systems adaptive. Many real-time applications, such as VoIP and interactive gaming, become intolerant of brief interruptions. Routing is a major source of the unreachability. Nikola then explained the big picture of routing and the roles of iBGPs. Although previous work has looked into gateway protocols, the fundamental behavior of intra-AS route propagation is still unanswered. Typically, iBGP fails when updates of iBGP causes transient loops, which in turn cause collateral damage.

The authors propose new, clean-state intra-domain route dissemination protocols, SOUP and LOUP, that are loop free. These protocols use reverse forwarding tree (RFT) and forwarding activation (FA), require minimal configuration, and can be fully distributed. They can provably avoid loops when the underlying topology is stable. Nikola showed several cases of transient loops and how the protocol can avoid them.

The evaluation of SOUP an LOUP was done on synthetic topologies, and simulation was on a simulated publicly available network topology. Result shows LOUP causes no loops or blackholes.

Ethan Katzbatha (USC) asked what should be done for packets going to destinations that are being withdrawn. Nikola said there is a fundamental tradeoff between a loop and a blackhole. If you don't do anything, the packets will be dropped at the border. There can be smarter solutions if there is already a tunnel set up. The border router knows where the packet should be going to, and the router can forward the packet using this tunnel. Michael Freedman (Princeton) noted that even with the tell-me-when shortcut, there might still be strange issues in that the propagation of reverse activation where the announcements go to different parts of the network. Nikola said that there is one more mechanism that is not described in the presentation that fixes the issue.

Improving Availability in Distributed Systems with Failure Informers

Joshua B. Leners, Trinabh Gupta, The University of Texas at Austin; Marcos K. Aguilera, Microsoft Research Silicon Valley; Michael Walfish, The University of Texas at Austin

Joshua described the importance of failure handling in distributed systems. The typical method of timeouts doesn't tell what and why. For example, consider the ICMP "destination unreachable" error message; it's unclear why something's unreachable: is it a problem with the destination or network? is it permanent or transient? The same error message is delivered for all these types of failures. The reasons behind this is that the network was designed to hide find-grained information as described in a SIGCOMM 1988 paper on design philosophy of the DARPA Internet protocol. Joshua revisited this classic network design choice and argued that this choice is mismatched to today's requirements. Today's network environment is built with a large number of commodity machines and switches and, as a result, failures are common and diverse. Applications, on the other hand, lack failure information. To them, the diverse failures look similar and so they choose universal recovery mechanisms.

Joshua then provided an example where knowing the type of failure can help applications to act more efficiently for failure recovery. The thesis of the paper is to expose the failure types to application.

One potential problem of exposing failure types to applications is that it may be burdensome to applications. For example, an application now needs to understand the semantics of each failure type, and if the interface changes, the application code also needs to change. The approach they use to tackle this problem is to group failures: e.g., process crash, host crash, and host reboot can be grouped to represent the case where the target stops permanently.

The authors built a service, Pigeon (5400 LOC C++), which implements their new interfaces. Pigeon collects local failure information with sensors, transports the information to end-hosts, and provides an interpreter for applications to understand the information.

Evaluation of Pigeon on a 12-host Fat Tree-topology network connected by 16 physical routers with injected failures shows distinguishing the host and network failures reduces unavailability by 4x, from 6.9 seconds to 1.6 seconds. Also, enhancing Cassandra with Pigeon helps Cassandra optimize replica selection. Pigeon also helps RAMCloud avoid unnecessary recovery.

Dave Levin (University of Maryland) asked whether Pigeon can handle Byzantine failure. Joshua answered it's targeted for traditional crash failures. Someone from MSR asked how to ensure the information sent to application reflects the ground truth. Joshua said for stop conditions, Pigeon confirms that the process crashed by, for example, looking at the process table. In the case of network failure, the current prototype fetches the topology from the gateway. Michael Freedman (Princeton) asked whether using this in a large distributed system, where nodes might see the same failure as different types, would cause conflicting reports and problems for Pigeon. Joshua said no.

Applications

Summarized by Murad Kaplan (murad.kaplan@colorado.edu)

BOSS: Building Operating System Services

Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler, University of California, Berkeley

Stephen Dawson-Haggerty started by presenting facts on energy consumption in the US, how a big percentage of this consumption is related to building and how this is a computer science problem. As many buildings have digital controls that include sensors and actuators that run many applications, Stephen asked the big question: can we write portable building monitoring and control applications that help to reduce the energy consumption caused by the digital control? He pointed to a number of applications used in today's buildings and asked if researchers can be forward looking in the way these application use resources and energy. These applications break down into a couple of different categories. The challenge is how to integrate existing applications using the existing physical infrastructure, and how researchers can make these kinds of applications that pop up all the time widely deployable in a way that saves more energy. The other challenge is how researchers can incorporate these applications in wider scale control to better manage their energy consumption.

From a systems point of view, researchers face several challenges; the first is portability—how do we take these applications, which are pretty much demo work today, and make them run anywhere in the building? Second is the failure modes we have been introduced to in these applications and how we can deal with them. And third, accessing the control systems in the buildings, exposing the existing hardware to the world and building all our stuff on top of it. Researchers want to scale to a large number of sensors and to large buildings, and they want these applications to be portable and easy to use.

To accomplish these goals, Stephen and his team built BOSS, distributed building operating system services that together allow these applications and control processes to sit on top of and interact with physical sensors and actuators in the building, and that offer authorization, optimization, and personal comfort services.

Stephen described the BOSS architecture by presenting portability as a part of the abstraction layer. He also presented applications that were built on BOSS to use its services to perform analysis, provide personal control, and personalize the microenvironment and optimization. The team installed these applications in a number of buildings to make these measurements. Stephen concluded that while BOSS provides a lot of opportunities, there are also a number of issues since applying computer system design to buildings involves a lot of pieces. BOSS was able to provide a 30% saving in electricity and steam and a 60% saving in lighting in its test apps. These apps can be found at http://smap.cs.berkeley.edu.

Someone asked how BOSS would provide a good handle for coordination in the case of a multiple control system. Stephen answered that there is a tradeoff in the way that you handle these services; right now they are statically configured. Kurtis Heimerl (University of California, Berkeley) was interested in the user programming environment and asked how far along BOSS was in doing that. Stephen responded that it will be a cool idea to let people program but the key challenge is the really dynamic binding.

Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks

Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan, MIT Computer Science and Artificial Intelligence Laboratory

Keith Winstein started by highlighting some special characteristics of current cellular networks. He explained how cellular networks have probably became the means by which a majority of users access the Internet, but that cellular networks differ in many ways from the traditional Internet. One of these differences is that cellular networks are highly variable in link speed. Using Verizon LTE cell phones, they measured the downlink and uplink as they kept them busy and kept recording how fast packets crossed each link. Keith presented a graph that showed how extremely variable the network was. Keith also pointed out another difference from the "old fashioned" networks is that they are too reliable. We talk about best-effort network delivery, but in fact it is insane-effort network delivery. Another graph of one TCP download showed how there is a big delay in the round trip time.

Then Keith presented a graph showing Skype performance over the Verizon LTE network. This is to show when Skype is not using all the bandwidth available because Skype is being very conservative. Also when Skype is not conservative and sends higher than the available bandwidth, the result is a huge delay. To overcome this problem, Keith presented their protocol, Sprout. Sprout is a transmit protocol designed for real-time interactive applications over these variable, semi-reliable networks. Sprout has the goal of providing the application with the most throughput possible, but the higher priority is to bound the risk of accumulating a large delay in gateway queues.

In order to control the risk of delay, the authors have separate parts of their algorithm. At the Sprout receiver, the authors infer the current link speed and they do that with Bayesian inference on a stochastic model. In part two, the receiver tries to predict what is going to happen to the link speed in the future. It makes a "cautious forecast" and sends this to the sender as part of its acknowledgments. In part three, the sender uses this forecast to control an evolving window that compromises between the desire for more throughput and for controlling the risk that a packet will spend more than 100 ms in a queue.

Keith later compared Sprout with other applications, such as Skype, Google-Hangout, and Facetime. Sprout performed better in term of throughput and delay. Keith said that a Stanford networking class had reproduced these plots, and the two students who chose Sprout were awarded the best project in the class. MIT students were challenged to beat Sprout and came up with 3,000 different algorithms with different compromises between throughput and delay. Keith indicated several limitations in their measurements in the conclusion of his talk: Sprout was only evaluated in the case of video conferencing, and all measurements were made in Boston. The source code and directions to use it are at : http://alfalfa.mit.edu/.

Phil Levis (Stanford University) said that he sees the authors are using LEDBAT (Low Extra Delay Background Transport), but did they think they should replace LEDBAT with Sprout? Keith answered that LEDBAT tries to coexist with TCP on the same bottleneck queue to scavenge available bandwidth. Although there are similarities between LEDBAT and Sprout such as counter filter and one-way delay estimation, they did not test it in competition with TCP on the same queue, so they don't want to say that they do better than LEDBAT in this situation.

Kurtis Heimerl (University of California, Berkeley) asked if the authors had thought about using geographical knowledge to help in this. There are falls in throughput traffic when a user is switching towers, for example, or driving a car, so how would Sprout use that to make better prediction? Keith answered that there is previous work in NSDI on this kind of thing, and he is not sure if speed and location may affect his results or not. Kurtis pointed out that with cellular networks, there is a whole circuit switch piece of technology that essentially is designed to solve this problem for voice traffic; how is that related to the problem Sprout is trying to solve? Is it something Sprout could just solve by using that circuit switched network? Keith replied that circuit switching networks in cellular systems have admission controls that give users 64k bits per second and they don't have that for large bit rate flows that need to vary in speed. The problem is that the link quality varies, which you can't just solve with better policy. Someone asked, looking at the graph with the available bandwidth even if the mobile was stationary, whether they had tested Sprout with Sprout. Keith replied that they did that by testing six phones in the same cell.

Demystifying Page Load Performance with WProf

Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall, University of Washington

Xiao Sophia Wang demonstrated that the Web is everywhere and it is a critical part of the Internet. Unsurprisingly, page load is critical. Studies suggest that Amazon can increase its revenue by 1% by reducing page load time by 100 ms. Wang showed her measurement studies of page load time of about 200 Web pages. The results show that the median page load time is three seconds. And few Web pages take more than 10 seconds to load. Because Web page load time is slow, there are many techniques aimed to reduce this time. These optimization techniques include using CDN, small pages, etc. While these techniques can help some pages, they may have no effect on others. And, in fact, they may even harm performance. The reason is that the page load process is poorly understood. Because of that, the authors' goal in this paper was to understand the page load process. Wang explained how difficult it is to understand page load time because there are many factors that affect the page load. She showed an example on how a page's structure affects the page load time.

Wang presented the process of demystifying Web page performance. The authors modeled the page load process. And they presented WProf to implement the set of dependencies inside browsers to study in real Web pages. To identify the bottlenecks of page load, the authors applied critical path analysis on top of the dependences graph that WProf generated.

Their methodology includes three elements. First, they designed test pages to systematically uncover dependencies across browsers. They also examined documents as well as browser code whenever the source code was available. For designing the test pages, the authors started by thinking how Web objects are organized in Web pages. They found four categories of dependency policies: (1) flow dependencies, which is the natural order of the activities acquired; (2) output dependencies, which ensure the correct execution of multiple processes with access to the same resources; (3) lazy and eager binding, which are tradeoffs between data downloads and page load latencies; and (4) resource constraints resulting from limited computing power or network resources.

Wang reviewed some experimental results where they loaded real pages using WProf. They ran the experiments at the campus network using WProf in Chrome, looking at 150 Web pages. Their conclusion was that most object downloads are not critical, and JavaScript blocks parsing on 60% of top pages. Caching eliminates 80% of Web object loads, but it doesn't reduce page load time as much. Wang concluded her talk by pointing out the most important elements of WProf: it automatically extracts dependencies and analyzes critical paths. WProf can be used to understand performance of page load times and to explain the behaviors of current optimizations.

Someone asked whether WProf can predict what the benefit from parallelization will be. Wang replied that HTML parsing and CSS evaluation can be parallelization, but JavaScript evaluation is hard to parallelize. Someone from University of California, San Diego, asked if the critical path measurement can be used for a specific object, not only the entire page. Wang replied that their tool can do that. Brighton Godfrey (University of Illinois at Urbana Champaign) asked, assuming their results are based on the minimum over five trials for each page, would their conclusions change if they looked at the median and outliers? Wang replied that because there is large variation, they only looked at the median page.

Dasu: Pushing Experiments to the Internet's Edge

Mario A. Sánchez, John S. Otto, and Zachary S. Bischof, Northwestern University; David R. Choffnes, University of Washington; Fabián E. Bustamante, Northwestern University; Balachander Krishnamurthy and Walter Willinger, AT&T Labs—Research

Mario Sánchez talked about having measurement experimentation platforms located at the edge of network. The rapid growth of the Internet has been driven in part by advances in broadband technology and affordability which made the Internet not only larger but also spread out and diverse. Although many of the distributed systems are located at the edge of network, when we measure them from the code, they look different. The truth is that researchers like measuring platforms at the right scale, vantage point, and location in order to provide sufficient control for them to conduct needed experimentation. While there are platforms like PlanetLab that provide great fine-grained control, they don't scale very well. Also, their vantage points are located in research and academic networks. Other platforms, like Dimes, provide the potential for scale, and the actual view that researchers want, but they also provide little flexibility in term of control and experimentation. So there is a need for a platform that combines the best of both worlds. But for such a platform to be useful, the safety of the volunteered nodes need to be guaranteed, and researchers need both to be able to control the impact of having the experiment in their network and to have a way to share resources between multiple experiments and multiple experimenters.

To overcome these challenges, Mario presented Dasu, which is a prototype for such a platform. Dasu is a software measurement experimentation platform that is hosted by end users located at the edge of the network and that relies on the direct incentive of using broadband measurement for end users. Dasu was designed with two purposes: to allow end users to characterize the service from their broadband network and to support experimentation from the edges. By considering these design purposes, Dasu aligns the objectives of end users and experimenters in three different ways. First, both end users and experimenters can benefit from having large coverage in order to capture network and service diversity. Second, both users and experimenters benefit from having large availability in order to be able to catch changes in policies and scheduled events. Third, users and experimenters also benefit from being at the edge and extensible, in the case of end users in order to remain unaffected in the face of changes in ISP policies, and for experimenters in order to be able to expand the new platform with new measurements.

At the same time, Mario pointed out that researchers face a lot of challenges; one of them is the lack of dedicated resources, which means they cannot run arbitrary experiments on these machines. For instance, some experiments related to censorship might be out of the question because these may get end users in trouble. Mario presented the design implementation of Dasu and how it addresses these challenges.

To protect volunteer nodes, Dasu runs the experiments inside its own engine inside a sandbox. Also, Dasu's clients run a resource profiler to continuously monitor the resources consumed by the system, and there is also a watchdog timer that triggers client shut down if anything goes wrong. Mario presented the dynamic of the system by showing experiments run by clients using Dasu.

Dasu is currently an extension to BitTorrent and soon will be stand-alone in a DNS resolver. Mario also pointed out that Dasu has over 90,000 users in over 150 countries. Dasu's main components are the Dasu client and the infrastructure service that supports it.Indranil Gupta (University of Illinois at Urbana Champaign) asked how Dasu would stop a runaway experiment. Mario said they do have the resource profiler that looks at the CPU usage if anything goes wrong, and then it will automatically shut it down. Mike Weighty (Montana State University) asked if the authors have any plan on extending Dasu to mobile devices. Mario said there are other students in his lab who focus on wireless, and he is working on something related.

Security and Privacy

Summarized by Weverton Cordeiro (weverton.cordeiro@inf.ufrgs.br)

πBox: A Platform for Privacy-Preserving Apps

Sangmin Lee, Edmund L. Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov, The University of Texas at Austin

Sangmin Lee pointed out a recent study which has shown that only 17% of users pay attention to access permission requests when installing applications on their mobile devices. More alarmingly, just 3% of them actually fully understand these requests. But even though users read and fully understand these requests, there is no guarantee that apps will actually comply with them—in fact, there is evidence that some apps out there are misusing users' private data! The user may choose to believe that app publishers will not misuse their private data; but in which one of the 300,000+ publishers can they actually trust? According to Sangmin, such an obscure scenario for users' privacy strongly suggests that existing countermeasures to protect users' privacy are not working.

Sangmin proposed to shift the users' trust from the 300,000+ app publishers to a few major brands, such as Google, Microsoft, Apple, and Amazon. In addition to the fact that one already has to trust these companies to use their device anyway, they also have a reputation to maintain, and thus more incentive to work correctly. The key to realize the proposed shift? πBox: a framework that combines app confinement through sandboxes and control channels to provide explicit and useful privacy guarantees.

Sangmin explained that πBox provides a per-user, per-app sandbox confinement. All users' data stays inside the sandbox along with an app instance, and the sandbox prevents the app from leaking the data. The sandboxes use a model which makes them span the users' device and the cloud—and thus use resources from both the device and the cloud. Private vaults (which are in both the device and the cloud) and content storages are provided for storing user and app-specific data, respectively. Both are secured to prevent privacy violation. The aggregate channels provide communication capabilities between the app and its publisher, and use differential privacy to introduce random noise and thus bound information leakage. Finally, The sharing channel allows one to have control of "when" and "with whom" to share, but provides weak guarantees on "what to share." However, it makes it difficult for an adversary to access users' private data directly. Sangmin mentioned, though, that shared channels are not entirely secure (for example, photo sharing remains subject to steganography).

Along with a three-level classification model (green, yellow, and red), πBox enables fine-grained control over which security features will be enabled for each app, and provides explicit privacy guarantees users can rely on. Sangmin closed the talk arguing that real, existing applications can benefit from ϖBox, and highlighting that overhead is low: only a few lines of code needed to be changed in existing, popular apps; and the measured information throughput was marginally degraded.

Fengyuan Xu (College of Willian and Mary) appeared concerned about the burden πBox will place on users (who will have to decide what, when, and with whom contents will be shared, for example). Sangmin argued that there will be not much difference from how it is currently done, and that apps will provide support for that decision process. In a follow up question, Fengyuan asked if πBox is vulnerable to attacks that use covert channels. Sangmin replied that it depends on whether the implementation of sandboxes and the design of πBox is orthogonal. He added that if there are any improvements on building better sandboxes, πBox can benefit from it by using it. Fengyuan then asked if πBox is something like a platform, in which you can plug in other schemes to make it more secure. Sangmin simply replied that yes, it is. Yan Chen (Northwestern University) asked what the aggregation channels can support. Sangmin replied that one big goal of aggregate channels is to provide information to app providers without violating users' privacy. He mentioned some examples of apps and how they could use aggregate channels to export information to app publishers. Yan then asked if ϖBox does protect social network information (i.e., the list of friends to whom one is sharing photos) from potential spammers. Sangmin replied that yes, it does.

P3: Toward Privacy-Preserving Photo Sharing

Moo-Ryong Ra, Ramesh Govindan, and Antonio Ortega, University of Southern California.

Moo-Ryong Ra started his talk by mentioning that cloud-based photo sharing service providers (PSPs) are very popular, in spite of the various privacy concerns on the way these providers provision their services. For example, shared photos may be over-exposed (either accidentally or because of poor PSP system design). The PSP could also use inference techniques to extract information from shared photos. In other words, he argued that one can trust the (mobile) device but not everything else (e.g., the network and the PSPs).

On one hand, these privacy concerns Moo-Ryong mentioned are real. He showed several news headlines that reported privacy issues with Photobucket, Facebook, and Instagram. On the other hand, PSPs provide valuable and useful services to users: a shared photo may be adapted to the various types of devices that access these photos (smartphones, computers, etc.) and perform image quality processing. The only question is, can we protect users' privacy while still performing cloud-side image transformation? The answer: yes, we can. In the remainder of his talk, Moo-Ryong described P3, a privacy-preserving photo encoding algorithm. The algorithm extracts and encrypts a small part of the photo that contains significant visual information (the "secret part"), while preserving the larger part (the "public part"), which contains no significant information, in a standards-compatible format. With P3, PSPs can still provide their services without being required to redesign their systems.

P3 concentrates on JPEG images, and exploits the fact that DCT (Discrete Cosine Transform) coefficients of natural images are sparse, with a few coefficients holding most of the useful information; put in an intuitive way, P3's "secret part" and "public part" correspond to the most significant bits and least significant bits of the DCT coefficients, respectively. There is a challenge, however: how to reconstruct the original image out of the secret and public parts? If the public part has not been modified, a straightforward series of linear operations can reconstruct the image. In case it has been modified, the PSP must also send the linear operator used for processing it. During the talk, Moo-Ryong discussed the (easy-to-deploy) architecture of P3, provided evidence regarding its privacy-preserving aspect (through a set of examples and evaluation experiments), and showed that it causes minimal storage overhead (in the worst case, the image file-size increased at most 20%). Regarding the privacy-preserving aspect, Moo-Ryong showed that P3 virtually broke face recognition algorithms: the number of successful face recognitions on the public part was decreased significantly.

Nikita Borisov (University of Illinois at Urbana-Champaign) asked if one can obtain the original picture solely with the secret part. Moo-Ryong answered negatively, saying that one needs both parts to obtain the original picture. In a follow up, Nikita posed the situation in which the P2P is fully malicious. Moo-Ryong said that in this particular case P3 is broken. However, it was emphasized that in this case it is a functionality problem rather than a privacy problem—the P2P will not be able to do anything malicious without the secret part anyway. The session chair (Krishna Gummadi, Max Planck Institute for Software Systems) asked for the high-level intuition as for why the smallest part of the image contained most of the visual information, whereas the larger part of the image contained almost no information. Moo-Ryong replied that this is related to why JPEG works, and revisited the discussion about the process of extracting the secret part from the image. Finally, Lakshminarayanan Subramanian (New York University) asked about the privacy guarantees of P3. Moo-Ryong replied that they are still trying to prove privacy guarantees of P3 using mathematical tools. However, they are unaware of any techniques that could break the algorithm.

Embassies: Radically Refactoring the Web

Jon Howell, Bryan Parno, and John R. Douceur, Microsoft Research.

Awarded Best Paper!

Jon Howell started his talk by stating that the Web promises us a really nice isolation model, in which users are protected from malice even though they click on "dangerous links." The reality is quite different, however, as browsers have vulnerabilities that might compromise the system, if explored by a malicious Web app. In this scenario, what kind of advice should one give to ordinary Web users? Do not click on "dangerous links"?

According to Jon, these security weaknesses are not caused by poor Web API implementation; even a correct browser implementation cannot protect users from external protocol flaws. "The API itself needs to be changed," he said. Why? He first explained that the Web API has a subset of methods that specify the behavior of the application when it arrives at the client; that subset he called Client Execution Interface (CEI). To provide isolation, CEI must be as simple as possible and have a well-defined semantics, so that it can be implemented correctly; in other words, it should pursue minimality. He also explained that the same API has served as a Developer Programming Interface (DPI), and developers have always wanted this interface to be richer and richer, so that fancier applications can be built with less and less code; in other words, it has also been pursuing richness. "This API has been pulled off in two opposite directions," he said. To solve this problem, he proposed to refactoring the Web API, separating the roles of CEI and DPI.

Jon argued that the CEI should provide Web applications an execution environment having the same philosophy (and isolation guarantees) of a multi-tenant datacenter but in the client machine—a "pico-datacenter," as he defined it. In his refactored API, binary code would be the core of the CEI, thus accepting binary programs from the vendor. In this case, Web developers would be able to develop their applications using the libraries that best suit their needs (e.g., libraries for HTML rendering, JavaScript processing, etc.); not only is browser incompatibility gone in this "Embassies" model, but users would also be able to run applications such as Gimp in their browsers. To run an application in the pico-datacenter, the client would just need to load the required libraries. From an experimental evaluation using a prototype, the Embassies model has been shown to introduce some overhead. However, Jon strongly emphasized that this overhead comes in exchange for a truthful Web promise of a nice isolation model. "Dangerous links"? No more!

Nikita Borisov (University of Illinois at Urbana-Champaign) mentioned he saw some similarities between the Embassies model and the one that does exist in the mobile platform, and then asked Jon if the two models would eventually become one. Jon replied that he definitively sees convergence there, since the mobile world is being pushed towards the right user model. However, he thinks that not all the opportunities of this model are being explored, since mobile apps do not depend on a clean and narrow execution interface. Someone asked about the effectiveness of Web-indexing in this new model and the burden it would cause to vendors such as Google and Yahoo!. Jon answered that this is a reasonable aspect to be worried about, but said that while Web app vendors can go anywhere they want (by using any shared libraries they please), they would still attach to popular transit protocols, and also expose interactions one would be able to sniff on. Lakshminarayanan Subramanian (New York University) proposed taking the work in a different direction, and asked if there is a problem with how Web pages should be designed. Jon suggested that a way one can look at this work is to ask why the Web API developers use to design their pages must be bound to the browser, instead of being just software libraries that they are free to choose. Jon said it is inevitable there will be complexity in the stack used to design Web pages. The advantage is that Web developers will be free to make choices over that complexity.

12th USENIX Conference on File and Storage Technologies (FAST '14)

Sponsored by USENIX, the Advanced Computing Systems Association **February 17–20, 2014, Santa Clara, CA**

Important Dates

Paper submissions due: *Thursday, September 26, 2013, 9:00 p.m. PDT (Hard deadline, no extensions)*

Notification to authors: *Sunday, December 8, 2013*

Final paper files due: *Thursday, January 23, 2014*

Conference Organizers

Program Co-Chairs

Bianca Schroeder, *University of Toronto*
Eno Thereska, *Microsoft Research*

Program Committee

Remzi Arpaci-Dusseau, *University of Wisconsin—Madison*
Andre Brinkmann, *Universität Mainz*
Landon Cox, *Duke University*
Angela Demke-Brown, *University of Toronto*
Jason Flinn, *University of Michigan*
Garth Gibson, *Carnegie Mellon University and Panasas*
Steven Hand, *University of Cambridge*
Randy Katz, *University of California, Berkeley*
Kimberly Keeton, *HP Labs*
Jay Lorch, *Microsoft Research*
C.S. Lui, *The Chinese University of Hong Kong*
Arif Merchant, *Google*
Ethan Miller, *University of California, Santa Cruz*
Brian Noble, *University of Michigan*
Sam H. Noh, *Hongik University*
James Plank, *University of Tennesee*
Florentina Popovici, *Google*
Raju Rangaswami, *Florida International University*
Erik Riedel, *EMC*
Jiri Schindler, *NetApp*
Anand Sivasubramaniam, *Pennsylvania State University*
Steve Swanson, *University of California, San Diego*
Tom Talpey, *Microsoft*
Andrew Warfield, *University of British Columbia*
Hakim Weatherspoon, *Cornell University*
Erez Zadok, *Stony Brook University*
Xiaodong Zhang, *Ohio State University*
Zheng Zhang, *Microsoft Research Beijing*

Steering Committee

Remzi Arpaci-Dusseau, *University of Wisconsin—Madison*
William J. Bolosky, *Microsoft Research*
Randal Burns, *Johns Hopkins University*
Anne Dickison, *USENIX Association*
Jason Flinn, *University of Michigan*
Greg Ganger, *Carnegie Mellon University*
Garth Gibson, *Carnegie Mellon University and Panasas*
Kimberly Keeton, *HP Labs*
Darrell Long, *University of California, Santa Cruz*
Jai Menon, *Dell*
Erik Riedel, *EMC*
Margo Seltzer, *Harvard School of Engineering and Applied Sciences and Oracle*
Keith A. Smith, *NetApp*
Ric Wheeler, *Red Hat*
John Wilkes, *Google*
Yuanyuan Zhou, *University of California, San Diego*

Tutorial Coordinator

John Strunk, *NetApp*

Overview

The 12th USENIX Conference on File and Storage Technologies (FAST '14) brings together storage-system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems. The program committee will interpret "storage systems" broadly; everything from low-level storage devices to information management is of interest. The conference will consist of technical presentations, including refereed papers, Work-in-Progress (WiP) reports, poster sessions, and tutorials.

FAST accepts both full-length and short papers. Both types of submissions are reviewed to the same standards and differ primarily in the scope of the ideas expressed. Short papers are limited to half the space of full-length papers. The program committee will not accept a full paper on the condition that it is cut down to fit in a short paper slot, nor will it invite short papers to be extended to full length. Submissions will be considered only in the category in which they are submitted.

Topics

Topics of interest include but are not limited to:

- Archival storage systems
- Auditing and provenance
- Caching, replication, and consistency
- Cloud storage
- Data deduplication
- Database storage
- Distributed I/O (wide-area, grid, peer-to-peer)
- Empirical evaluation of storage systems
- Experience with deployed systems
- File system design
- Key-value and NoSQL storage
- Memory-only storage systems
- Mobile, personal, and home storage
- Parallel I/O

Continued on next page

- Power-aware storage architectures
- RAID and erasure coding
- Reliability, availability, and disaster tolerance
- Search and data retrieval
- Solid state storage technologies and uses (e.g., flash, PCM)
- Storage management
- Storage networking
- Storage performance and QoS
- Storage security
- The challenges of "big data"

Submission Instructions

Please submit full and short paper submissions (no extended abstracts) **9:00 p.m. PDT on September 26, 2013,** in PDF format via the Web form on the Call for Papers Web site, www.usenix.org/fast14/cfp.

- The complete submission must be no longer than twelve (12) pages for full papers and six (6) for short papers, excluding references. The program committee will value conciseness, so if an idea can be expressed in fewer pages than the limit, please do so. Papers should be typeset in two-column format in 10 point Times Roman type on 12 point leading (single-spaced), with the text block being no more than 6.5" wide by 9" deep. As references do not count against the page limit, they should not be set in a smaller font. **Submissions that violate any of these restrictions will not be reviewed.** The limits will be interpreted strictly. No extensions will be given for reformatting.
- There are no formal restrictions on the use of color in graphs or charts, but please use them sparingly—not everybody has access to a color printer.
- Authors must not be identified in the submissions, either explicitly or by implication. When it is necessary to cite your own work, cite it as if it were written by a third party. Do not say "reference removed for blind review."
- 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.
- If you are uncertain whether your submission meets USENIX's guidelines, please contact the program co-chairs, fast14chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.
- Papers accompanied by nondisclosure agreement forms will not be considered.

The program committee and external reviewers will judge papers on technical merit, significance, relevance, and presentation. A good paper will demonstrate that the authors:

- are attacking a significant problem,
- have devised an interesting, compelling solution,
- have demonstrated the practicality and benefits of the solution,
- have drawn appropriate conclusions,
- have clearly described what they have done, and
- have clearly articulated the advances beyond previous work.

Blind reviewing of all papers will be done by the program committee, assisted by outside referees when necessary. Each accepted paper will be shepherded through an editorial review process by a member of the program committee.

Authors will be notified of paper acceptance or rejection no later than Sunday, December 8, 2013. If your paper is accepted and you need an invitation letter to apply for a visa to attend the conference, please contact conference@usenix.org as soon as possible. (Visa applications can take at least 30 working days to process.) Please identify yourself as a presenter and include your mailing address in your email.

All papers will be available online to registered attendees, no earlier than January 23, 2014. If your accepted paper should not be published prior to the event, please notify production@usenix.org. The papers will be available online to everyone beginning on the first day of the main conference, February 18, 2014. Accepted submissions will be treated as confidential prior to publication on the USENIX FAST '14 Web site; rejected submissions will be permanently treated as confidential.

By submitting a paper, you agree that at least one of the authors will attend the conference to present it. If the conference registration fee will pose a hardship for the presenter of the accepted paper, please contact conference@usenix.org.

If you need a bigger testbed for the work that you will submit to FAST '14, see PRObE at www.nmc-probe.org.

Best Paper Awards

Awards will be given for the best paper(s) at the conference. A small, selected set of papers will be forwarded for publication in ACM *Transactions on Storage* (TOS) via a fast-path editorial process. Both full and short papers will be considered.

Test of Time Award

We will award a FAST paper from a conference at least ten years earlier with the "Test of Time" award, in recognition of its lasting impact on the field.

Work-in-Progress Reports and Poster Sessions

The FAST technical sessions will include a slot for short Work-in-Progress (WiP) reports presenting preliminary results and opinion statements. We are particularly interested in presentations of student work and topics that will provoke informative debate. While WiP proposals will be evaluated for appropriateness, they are not peer reviewed in the same sense that papers are.

We will also hold poster sessions each evening. WiP submissions will automatically be considered for a poster slot, and authors of all accepted full papers will be asked to present a poster on their paper. Other poster submissions are very welcome.

Arrangements for submitting posters and WiPs will be announced later.

Birds-of-a-Feather Sessions

Birds-of-a-Feather sessions (BoFs) are informal gatherings organized by attendees interested in a particular topic; they are held in the evenings. BoFs may be scheduled in advance by emailing the Conference Department at bofs@usenix.org. BoFs may also be scheduled at the conference.

Tutorial Sessions

Tutorial sessions will be held on February 17, 2014. Please send tutorial proposals to fasttutorials@usenix.org.

Registration Materials

Complete program and registration information will be available in December 2013 on the conference Web site.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

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

LISA '13

27th Large Installation System Administration Conference

Sponsored by USENIX in cooperation with LOPSA

NOVEMBER 3–8, 2013 | WASHINGTON, D.C.

Keynote Address:
"Modern Infrastructure: The Convergence of Network, Compute, and Data"
by Jason Hoffman, *CTO, Joyent*

6 days of training on topics including:

- Configuration management
- Cloud computing
- Distributed systems
- DevOps
- Security
- Virtualization
- And more!

Plus a 3-day technical program:

- Invited talks
- Guru Is In sessions
- Paper presentations
- Vendor exhibition
- Practice and experience reports
- Workshops
- Posters and lightning talks

New for 2013! The LISA Labs "hack space" will be available for mini-presentations, experimentation, tutoring, and mentoring.

Register now! www.usenix.org/lisa13