usenix ;login:

APRIL 2012 VOL. 37, NO. 2

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix ASSOCIATION

UPCOMING EVENTS

4th USENIX Workshop on Hot Topics in Parallelism (HotPar '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGMETRICS, ACM SIGSOFT, ACM SIGOPS, ACM SIGARCH, AND ACM SIGPLAN

June 7–8, 2012, Berkeley, CA, USA
http://www.usenix.org/hotpar12

2012 USENIX Federated Conferences Week

June 12–15, 2012, Boston, MA, USA

2012 USENIX Annual Technical Conference (USENIX ATC '12)
June 13–15, 2012
http://www.usenix.org/atc12

3rd USENIX Conference on Web Application Development (WebApps '12)
June 13–14, 2012
http://www.usenix.org/webapps12

4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '12)
June 12–13, 2012
http://www.usenix.org/hotcloud12

4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '12)
June 13–14, 2012
http://www.usenix.org/hotstorage12

4th USENIX Workshop on the Theory and Practice of Provenance (TaPP '12)
June 14–15, 2012
http://www.usenix.org/tapp12

6th Workshop on Networked Systems for Developing Regions (NSDR '12)
June 15, 2012
http://www.usenix.org/nsdr12

5th Workshop on Cyber Security Experimentation and Test (CSET '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6, 2012, Bellevue, WA, USA
http://www.usenix.org/cset12

2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6, 2012, Bellevue, WA, USA
http://www.usenix.org/foci12

2012 Electronic Voting Technology Workshop/ Workshop on Trustworthy Elections (EVT/WOTE '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6–7, 2012, Bellevue, WA, USA
http://www.usenix.org/evtwote12
Submissions due: May 11, 2012

3rd USENIX Workshop on Health Security and Privacy (HealthSec '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6–7, 2012, Bellevue, WA, USA
http://www.usenix.org/healthsec12

6th USENIX Workshop on Offensive Technologies (WOOT '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6–7, 2012, Bellevue, WA, USA
http://www.usenix.org/woot12

7th USENIX Workshop on Hot Topics in Security (HotSec '12)

CO-LOCATED WITH USENIX SECURITY '12

August 7, 2012, Bellevue, WA, USA
http://www.usenix.org/hotsec12
Submissions due: May 7, 2012

21st USENIX Security Symposium (USENIX Security '12)

August 8–10, 2012, Bellevue, WA, USA
http://www.usenix.org/sec12

10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)

October 8–10, 2012, Hollywood, CA, USA
http://www.usenix.org/osdi12
Submissions due: May 3, 2012

26th Large Installation System Administration Conference (LISA '12)

December 9–14, 2012, San Diego, CA, USA
http://www.usenix.org/lisa12
Submissions due: May 17, 2012

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

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Arnold Gatilao
Casey Henderson
Jane-Ellen Long

TYPESETTER
Star Type
startype@comcast.net

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

http://www.usenix.org
http://www.sage.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 nonmembers are $125 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.

usenix ;login:

APRIL 2012, VOL. 37, NO. 2

OPINION

Musings

RIK FARROW

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

While I was at LISA '11, I ran into an old friend, Paul Ebersman. Paul was one of the first employees at UUNET, and every time I ran into Paul, usually at a USENIX conference, he would tell me how the UUNET office had grown onto another floor of a building. Eventually, UUNETs growth was taking over entire buildings. Paul would also tell me how fast the Internet was growing, with traffic doubling every few months. UUNET was the fastest-growing ISP in the 1990s [1] and was founded with help from USENIX.

These days, Paul is working for a company (Infoblox) that makes server appliances, and his own focus is on IPv6. Paul's article about issues with making the transition to IPv6 appears in this issue.

During our conversations at LISA, I asked Paul about security and IPv6, and Paul said, "It's like 1994 all over again." What Paul meant was that back in the mid-'90s, organizations were beginning to use the Internet without having more than the vaguest notion of security. Also, most IPv4 software stacks were largely untested, leading to root compromises and various denial-of-service (DoS) attacks, such as the Ping of Death [2]. In many ways, as the momentum to enable IPv6 on Internet-facing Web sites rolls onward, we will be facing another moment where most people will not be familiar with the new security issues that come with using a new network protocol.

Just the Same

Some things won't be any different: there are still 65535 TCP and 65535 UDP ports in IPv6, just as in IPv4. If your client or server application has a bug that can be exploited using IPv4, it most likely can also be exploited via IPv6 (once you have connectivity). But there may also be bugs, such as the PoD, lurking in relatively untested IPv6 stacks.

IPv6 brings with it an enormous address space, with billions of network addresses and four billion times four billion host addresses. Some people had hopes that this enormous address space would make automatic scanning for hosts unfeasible. Steve Bellovin, Bill Cheswick, and Angelos Keromytis explained in their 2006 *;login:* article [3] that there are a number of strategies worms could use to limit their attacks to existing networks. These same strategies can also be used by attackers when exploring networks they wish to attack.

Paul said that it is an excellent idea to monitor your existing networks to check for IPv6 traffic that might already be there. I tried this, both on my home network,and

looking at traffic outside my firewall, and found a fair amount of IPv6 traffic. I used tcpdump with the ip6 filter (Wireshark uses the same input filter) and discovered that Macs and newer Windows systems are both pretty chatty. Both types of systems would occasionally send out Router Solicitation ICMPv6 packets, and both also looked for plug-and-play neighbors. Macs use MDNS (Multicast DNS), Windows boxes are using SSDP (Simple Service Discovery Protocol), and both are sending out IPv6 multicast packets looking for or advertising services. If you've been on a public wireless network and noticed other people's shared disks showing up in a Finder or Explorer window, you have seen these protocols at play. And while these protocols are very helpful in some cases, I consider them harmful when used on non-private networks (hotels, coffee shops, airplanes, conferences, etc.). Nothing like an invitation to be hacked, and no scanning looking for victims required. The victims advertise their presence and potentially vulnerable services.

Most modern operating systems support IPv6 and will automatically bring up a link-local IPv6 interface on all network interfaces. Link-local addresses begin with the prefix fe80::/10 and, as the name suggests, are valid only on the local subnet. Link-local packets cannot be routed and are important to IPv6; they are used in network discovery protocol and for address assignment and DHCPv6 (see Paul's article). The host portion of these IPv6 addresses (EUI-64) is constructed using each interface's MAC (Media Access Control) address, with the 2 bytes FFFE inserted into the middle of these 6 bytes, and the 7th bit set to one [5]. Unless you are using DHCPv6, your systems will use the MAC address when automatically generating IPv6 addresses—unless you have a Windows 7 or newer OS.

Windows 7 systems use a randomly generated host suffix that changes every 15 minutes for privacy, since including your MAC address in a globally accessible IPv6 address makes your system easy to track on the Internet, as well as possibly identifying your system, since the first 3 bytes of the MAC address are assigned to vendors [5]. David Barrera wrote about privacy extensions for IPv6 address for *;login:* in 2011 [6], and perhaps the rest of the mobile system world—Linux, Mac, and BSD laptops, smartphones, and tablets—may catch up to Microsoft's ability to maintain more privacy when using IPv6.

IPv6 was designed so that we can use globally unique addresses, and this rightly implies the end of NAT. There are available address ranges (site-local addresses [7]), such as link-local addresses, that will not be routed over the Internet. But having IPv6 allows you to do away with the problems that NAT has caused, such as having to renumber networks when organizations merge or when once separate networks are joined. This also implies that your firewall had better be IPv6-enabled [8] and that you block packets destined for internal services at edges of your network. NAT has provided some protection against clients being visible more by accident than by design, but with IPv6, this limited protection goes away.

Negative NAT

Because you already have systems (Windows and Macs) on your network bursting with eagerness to use IPv6 (just look at those router solicitations!), you are already using IPv6—just inadvertently instead of intentionally. An attacker who manages to gain the ability to run a router advertisement daemon such as radvd [9] can announce whatever IPv6 prefix and routes the attacker desires. This allows the attacker to perform man-in-the-middle (MITM) attacks by routing packets through a system the attacker controls, while forwarding the packets using NAT-

PT [10] to convert them to IPv4 over your existing networks. This attack works just as well in public networks and will work whenever a Windows or Mac system decides to use IPv6 instead IPv4 (for example, as soon as an IPv4 DNS request times out on a Mac).

IPv6 brings with it a large share of annoyances, primarily the very long and awkward 128-bit addresses. It is one thing to memorize an IPv4 numeric address, expressible as four numbers, and another to memorize the IPv6 address for the same network interface. I found a nice little tool called ipv6calc [11] you can install to help you to interpret IPv6 addresses:

```
$ ipv6calc --showinfo fe80::21b:fcff:fefb:742c/64
No input type specified, try autodetection...found type: ipv6addr
No output type specified, try autodetection...found type: ipv6addr
Address type: unicast, link-local
Registry for address: reserved
Interface identifier: 021b:fcff:fefb:742c
EUI-48/MAC address: 00:1b:fc:fb:74:2c
MAC is a global unique one
MAC is an unicast one
$ []
```

In this example, I fed it the autoconfigured IPv6 address for a Linux server, and ipv6calc did a nice job of explaining it.

Firewalls will be another annoyance. In the BSD world, you can have a single set of firewall rules for both IPv4 and IPv6. But in the Linux world you have two sets of rules, one for each protocol, and you must remember that any changes you make to one set of rules you must also make in the other.

If you look at some default rules in a Linux ip6tables firewall file, you will see rules allowing all ICMPv6 packets. IPv6 relies on ICMPv6 for determining MTU, and fragmentation will not work without it. Network discovery, like the router advertisements, also rely on this, but this is something you may want to block at your firewall [12].

Older Linux kernels (pre-2.6.20) do not support stateful filtering of IPv6 packets (CONFIG_NF_CONNTRACK_IPV6) and leave a gaping hole (ports 32,768–61,000) in the ip6tables rules so that client packets can return through the firewall:

```
ACCEPT udp anywhere anywhere udp dpts:filenet-tms:61000

ACCEPT tcp anywhere anywhere tcp dpts:filenet-tms:61000

      flags:!SYN,RST,ACK/SYN
```

This charming behavior reminds me of old Cisco router packet filtering (ACL) rules, back in the days before stateful packet filtering.

Another wonderful feature of IPv6, since deprecated, is the routing header (RFC 5095). Routing header 0 (RH0) was included in IPv6 to support mobile devices so that they can roam between networks and continue to receive response packets. It also allows the construction of what we used to call source routing, a trick that used to work wonders with ancient protocols that trusted the source address for authentication. Today, RH0 headers can be used in amplification DoS attacks.

The Lineup

We start off this issue with an article about dark silicon by Nikos Hardavellas. I've read posts by people who consider dark silicon to be evil, but Nikos makes an excellent case for why we will be seeing dark silicon in future CPU designs.

Paul Ebersman is up next, with the article I've already mentioned about IPv6. Did you know that June 6, 2012, is World IPv6 day [13]? And that someday your IPv4-only servers will not be reachable by IPv6-only devices?

Simson Garfinkel approached me in December, wondering if we could publish an article about Unicode. To be honest, I wasn't excited about character encoding at first, but as I read his article, I discovered lots of things (and many annoyances) that I could finally recognize as Unicode artifices. I learned that my desktop display uses UTF-8, and the FFFE byte string I had been deleting from the beginning of text files was actually an indicator of the type of Unicode character encoding found in the file I was editing (UTF-16, little endian, mysteriously included in IPv6 EUI-64). If you write programs, you need to know about Unicode.

While Simson covers C, Java, and Python, Tobi Oetiker discusses Unicode for Perl hackers. Tobi tells us that there has been Perl support for Unicode for many years. More importantly, he explains where you must actively write Perl code that properly encodes and decodes Unicode, with examples, as well as some surprises you may encounter when dealing with Perl's support for Unicode.

Doug Hughes approached me during LISA with an article idea. Doug, like many of us today, works in an organization that manages huge amounts of data that, naturally, must not be corrupted. But Doug had encountered silent data corruption—that is, errors not reported by disk drives. Doug explains how he discovered this data and provides suggestions for how you can do this as well.

David Blank-Edelman takes us on a journey beyond Perl's core functions for handling error messages. If you have ever used Perl, you will be familiar with warn() and die(), but David shows us much more helpful error functions—that is, ones that provide more useful debugging information or allow you to catch errors the way other programming languages often do.

Dave Beazley also steps outside the bounds of Python, exploring add-on Python libraries. Python comes well-equipped with libraries, but Dave suggests two (currently) non-standard ones that extend regular expression (regex) and Web interaction (requests), the first with new features, the second with both more features and easier programming. I do want to note that I was unable to install regex on my older CentOS (5.7) system.

Dave Josephsen is still waxing enthusiastic about the features of Graphite, part of a suite of monitoring software. In this article Dave shows us some neat tricks for creating graphs using the URL interface to Graphite.

Robert Ferrell, having heard about Nikos's article, decided to create his own spin on dark silicon and other things considered "dark" today.

Many books are reviewed in this issue, including two by a new book reviewer, Mark Lamourine, one by Trey Darley, one by Brandon Ching (back after a long break), and five by Elizabeth Zwicky. Elizabeth covers intro to the command line and Python, while Mark covers two programming-related topics. Trey volunteered to read and review the massive second edition of Richard Stevens's *TCP/IP Illustrated*, and Brandon Ching takes a look at an intro to HTML 5.

This issue includes reports from LISA '11. We did not manage to cover all 34 sessions of LISA, but we covered most sessions. We also have a report from the Advanced Topics Workshop at LISA and one from CHIMIT, an ACM workshop on computer human interfaces as they apply to system administrators.

Now that you know that your network is already (hopefully) supporting IPv6 to a limited extent, perhaps it is time to learn, and do, more about IPv6. One of the best papers at LISA covered how Google approached adding IPv6 support, first internally, then externally [14]. Learning about and using IPv6 is the best way not to be surprised by attacks against your networks that utilize this protocol.

At the very least, you should be monitoring IPv6 traffic on your networks. As far as switches are concerned, IPv6 is just another Layer 3 protocol, and routers and firewalls are the only real barriers that prevent you from having a dark network, one that you are unaware of. Don't forget: it's 1994 all over again.

References

[1] UUNET: https://en.wikipedia.org/wiki/Uunet.

[2] The Ping of Death (PoD): https://en.wikipedia.org/wiki/Ping_of_death.

[3] S. Bellovin et al., "Worm Propagation Strategies in an IPv6 Internet," *;login:*, vol. 31, no. 1, USENIX: https://www.usenix.org/publications/login/february-2006-volume-31-number-1/worm-propagation-strategies-ipv6-internet.

[4] Link-local addresses: https://en.wikipedia.org/wiki/Link-local_address#IPv6.

[5] IPv6 Extended Unique Identifier, 64 bit: http://wiki.nil.com/IPv6_EUI-64_interface_addressing.

[6] D. Barrera et al., "Back to the Future: Revisiting IPv6 Privacy Extensions," *;login:*, vol. 36, no. 1, USENIX:https://www.usenix.org/publications/login/february-2011-volume-36-number-1/back-future-revisiting-ipv6-privacy-extensions.

[7] Unique local address: https://en.wikipedia.org/wiki/Site-local_address.

[8] See the April 2008 issue of ;login: for two articles that discuss IPv6 firewalls: https://www.usenix.org/publications/login/april-2008-volume-33-number-2.

[9] Router advertisement daemon (radvd): http://www.litech.org/radvd/.

[10] IPv6 MITM attack using NAT-PT: http://resources.infosecinstitute.com/slaac-attack/.

[11] ipv6calc homepage: http://www.deepspace6.net/projects/ipv6calc.html.

[12] IETF, "Recommendations for Filtering ICMPv6 Messages in Firewalls," May 2007: http://www.ietf.org/rfc/rfc4890.txt.

[13] Internet Society, World IPv6 Day: http://www.worldipv6day.org/.

[14] Haythum Babiker et al., "Deploying IPv6 in the Google Enterprise Network: Lessons Learned" (Practice & Experience Report): https://www.usenix.org/conference/lisa11/deploying-ipv6-google-enterprise-network-lessons-learned-practice-experience.

The Rise and Fall of Dark Silicon

NIKOS HARDAVELLAS

Nikos Hardavellas is the June and Donald Brewer Assistant Professor of Electrical Engineering and Computer Science at Northwestern University. His research interests include parallel computer architecture, dark silicon, memory systems, optical interconnects, and data-oriented software architectures. Prior to joining Northwestern University, he contributed to the design of several generations of Alpha processors and high-end multiprocessor servers at Digital Equipment Corp. (DEC), Compaq Computer Corp., and Hewlett-Packard. Nikos holds a PhD in Computer Science from Carnegie Mellon University. nikos@northwestern.edu

Industry experts predict that transistor counts will continue to grow exponentially for at least another decade. Historically we were able to harness all of these transistors to deliver exponential increases in computational power by capitalizing on both technological improvements and micro-architectural innovation. However, after decades of reaping Moore's bounty, processors eventually hit a power wall. Technological limitations will soon prevent us from powering all transistors simultaneously, leaving a large fraction of the chip powered off ("dark"). Short of a technological miracle, we head toward an era of "dark silicon," able to build dense devices we cannot afford to power. Without the ability to use more transistors or run them faster, performance improvements are likely to stagnate unless we change course.

In this article we (see Acknowledgments) discuss the technological trends that give rise to dark silicon and outline our current efforts to curb them. One of the most important realizations is that instead of wasting the dark silicon, we can harness it to implement specialized cores, where each core is able to perform a narrow set of tasks at significantly lower energy. We envision an architecture that provides a sea of specialized cores, with the executing workload powering up only the most application-specific hardware, while the rest of the chip is switched off to conserve energy. The new architecture shows promise in solving the problem of dark silicon for the foreseeable future, but requires us to overcome several challenges across the entire computing stack to realize it.

The Energy Cost of Computing

Computer systems have already become indispensable and ubiquitous in every aspect of our life. Our continuous reliance on computers generates data at exponential rates. For example, during March 2011, over 1.6 PB of data was generated and transferred for processing among the Tier-1 sites of CERN's Large Hadron Collider computing grid [1]. Typical business data sets have grown by 29% annually over the last decade, surpassing Moore's Law [2], and personal data has been shown to grow at similar rates. Processing all this data requires unprecedented amounts of computation, with commensurate energy demands. To put the energy demands into perspective, it suffices to note that a 1,000m^2 datacenter is a 1.5 MW facility. Gartner estimates that personal computers and servers consumed 408 TWh of energy globally in 2010 [3]. Computer energy consumption in the US alone is estimated at 150 TWh annually, accounting for 3.8% of domestic energy generation, for a total of $15B. This appetite for energy has created an IT industry

with approximately the same carbon footprint as the airline industry, accounting for 2% of the greenhouse gas emissions [3]. With a forecasted 10% annual growth on installed computing systems [3], these figures rise rapidly, negatively impacting both the environment and the economics of computing.

It is not surprising, then, that energy consumption is quickly becoming a limiter for big science. Building an exascale machine with today's technology is impractical due to the inordinate power draw it would require, hampering large-scale scientific efforts. The average energy-per-instruction needs to decrease 200-fold (from 2 nJ to 10 pJ) for exascale machines to be within a reasonable (20 MW) power target [4]. Even this target would require 3% of the output of an average nuclear plant to feed a single machine.

Unfortunately, a large fraction of this energy is wasted. Processor chips account for 38% of the power consumption of a typical computer system [5], but the general-purpose computing substrate they provide is highly power inefficient. For example, conventional multicore processors consume 157–707 times more energy [6] than customized hardware designs (ASICs—application-specific integrated circuits). To enter an environmentally sustainable path and lower the operational costs of large computing installations, we must find ways to eliminate these energy overheads.

While the energy demands of computing at the macro scale have received widespread attention, there is another quiet revolution taking place at the chip level that threatens to topple today's hardware landscape.

The Rise of Dark Silicon

In the past several decades, technological progress and micro-architectural innovation allowed processor performance to ride Moore's Law. However, a few years ago the semiconductor industry hit a power wall. When processors approached the limits of air-cooling technologies, the on-chip frequency growth halted, and micro-architectural techniques alone proved inadequate to continue the upward performance trend. With the traditional performance driver gone, processor manufacturers turned to multicores to satisfy users' need for performance.

Since the number of transistors on chip is growing exponentially, fueling the multicore revolution, operating all transistors simultaneously requires exponentially more power per chip. However, whereas the power requirements grow, chip power delivery and cooling limitations remain largely unchanged across technologies [7]. We are already at a point where we cannot deliver and cool efficiently more than 130 W per chip [7], and as a result we will soon be incapable of operating all transistors simultaneously, pushing multicore scaling to an end [8, 9].

The reason behind this profound change is that while transistor counts grow exponentially, the voltage required to power them does not decrease fast enough: a 10-fold increase in transistor counts over the last decade was followed by only a 30% drop in supply voltage [7]. To the first order, power (P) is related to supply voltage (V_{dd}) by the equation

$$P = P_{static} + P_{dynamic} = a \times N \times I_{leak} \times V_{dd} \times K + (1-a) \times N \times C \times f \times V_{dd}^2$$

where N is the transistor count, a is the average fraction of transistors in the off state (not switching), I_{leak} is the leakage current, K is a circuit-design-style constant, C is the transistor capacitance, and f is the operational frequency. The

supply voltage V_{dd} cannot be reduced much, as it is limited by the threshold voltage (V_{th}) required to switch transistors reliably. To reduce V_{dd} one needs to reduce V_{th}, which itself cannot be reduced much as it will increase I_{leak} exponentially, raising the power consumption. Thus, while V_{dd} is stuck at virtually the same voltage level, the transistor count N grows exponentially, raising the power consumption of the chip.

Unfortunately, the power consumption of the additional transistors can no longer be mitigated through circuit-level techniques. Voltage-frequency scaling may decrease power consumption by lowering the operating frequency (f) and voltage, but its operational voltage range has shrunk by 70% over the last decade, rendering it incapable of solving computing's energy woes. At the same time, the frequency f cannot be reduced sufficiently to keep the power consumption at bay and simultaneously deliver reasonable performance [9]. With the power envelope remaining constant, and voltage scaling much slower than the exponential growth in transistor density, we'll soon be unable to power all transistors simultaneously. Short of a technological miracle, we head toward an era of dark silicon, able to build dense devices that we cannot afford to power [2, 8, 9].

Modeling Methodology

To assess the extent of dark silicon, we developed first-order analytical models of the dominant components of a processor's power consumption, bandwidth utilization, area occupancy, and performance, relating the effects of technology-driven physical constraints to the performance of workloads running on future multicores. We construct detailed parameterized models that conform to the projections of the International Technology Roadmap for Semiconductors (ITRS) [7] for future manufacturing technologies. Detailed descriptions of the models appear elsewhere [2, 9]. Our models have been independently vetted and used in a recent study of heterogeneous computing [10]. Similar models were independently developed and validated against PARSEC benchmarks, demonstrating that multicore performance will not scale with technology [8].

The goal of the analytical models is to describe the processor's physical constraints and derive peak-performance designs by jointly optimizing the models' parameters (including core type and performance, core count, silicon area, memory hierarchy characteristics, manufacturing process, transistor type, cache miss models, application data-set growth, application parallelism, 3D-stacking, supply and threshold voltage, and clock frequency). The models facilitate the design space exploration of multicores by relating a design's performance to its power consumption, bandwidth requirements, and area occupancy. The modeling methodology allows us to select the design that attains peak performance, out of all the potential designs that conform to technology projections and physical constraints. It is important to note that the goal of this modeling effort is not to offer absolute numbers of cache size or number of cores that produce the best design. Rather, the purpose of the models is to capture the first-order effects of technology scaling and unveil the trends that ultimately lead to dark silicon.

Example

To illustrate the use of the models, we explain the progression of the design-space exploration algorithm based on the results plotted in Figure 1. To develop Figure 1, we first constrain the models to a single technology node (20 nm), transistor

technology (high-performance double-gate FinFETs), core design (conventional general-purpose cores modeled after Sun UltraSPARC), memory technology (off-chip DRAM), die area (310 mm^2), and application characteristics (cache miss rates modeled after TPC-H on DB2, with 99% parallel code). The plot shows the aggregate chip performance as a function of the L2 cache size on the chip. Thus, for each point in the figure, a fraction of the die area is dedicated to an L2 cache of size denoted in the X-axis, 25% of the die area is used for supporting structures (e.g., memory controllers, interconnect, component spacing), and the remaining area can be populated with cores.

The *Area* curve shows the performance of designs that are constrained only in the on-chip die area (they have unlimited power and off-chip bandwidth). These designs can achieve high performance by populating the entire die with cores. As the L2 cache size grows to the right, even though fewer cores fit in the remaining area, each core's performance is higher due to the higher hit rate of the bigger cache, leading to an increase in aggregate chip performance. Eventually, the performance benefit of a large cache is outweighed by the cost of reducing the core count, leading to an aggregate performance drop at larger cache sizes.

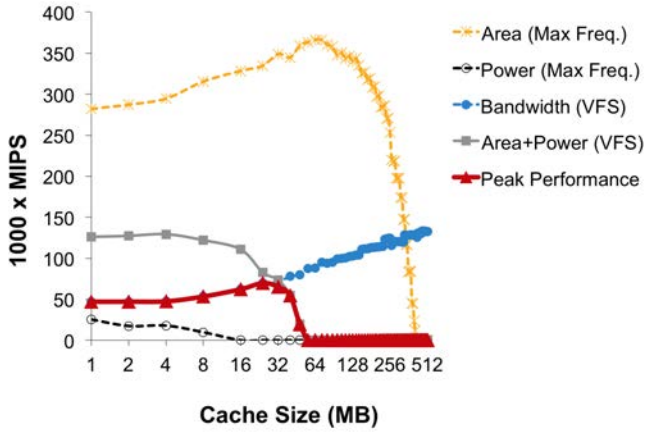

Figure 1: Performance of a multicore with general-purpose cores running TPC-H queries against a DB2 database at 20 nm

The *Power* curve shows designs populated with cores running at the maximum frequency allowed for the corresponding technology node, with power constrained by conventional forced-air cooling, but having unlimited area and off-chip bandwidth. Running the cores at maximum frequency requires so much power that it restricts these designs to a handful of cores, severely limiting the aggregate chip performance.

The *Bandwidth* curve shows designs that are limited only in off-chip bandwidth, permitting unlimited area and power use. The core count and core frequency are jointly optimized to find the peak-performing configuration, subject to bandwidth constraints. Larger caches reduce the off-chip bandwidth pressure, leading to improved performance. Conversely, the *Area+Power* curve shows designs limited in area and power but permitted to consume unlimited off-chip bandwidth. The Area+Power curve jointly optimizes the core voltage and frequency, selecting the peak-performing design for each L2 cache size.

Finally, the *Peak Performance* curve shows only the feasible multicore designs that conform to all physical constraints. At small cache sizes, the off-chip bandwidth is the performance-limiting factor. Beyond 24 MB, however, power becomes the main limiter, and the peak-performance design lies at the intersection of the power and bandwidth constraints. The gap between the Peak Performance and Area curves at 24 MB cache indicates that the majority of the silicon area of the best possible design for this technology node cannot be used for more cores, because they cannot be powered up.

Forecasting Dark Silicon

Using an identical analysis, but varying all the other parameters as well (e.g., core type, transistor technology, memory technology, workload, etc.), we find the multicore design that attains the highest performance on average across all workloads for a given process technology. We determine the technology trends by repeating this process for each technology node. Our workload suite includes online transactional processing (TPC-C on Oracle and DB2), Web servers (SPECweb on Apache), and decision support systems (TPC-H on DB2).

Figure 2 presents the results of this analysis, where the X-axis plots the year that each corresponding technology becomes mainstream according to ITRS and the Y-axis plots the number of cores. The dashed lines indicate the maximum number of cores that fit on chip, and the solid lines indicate the number of cores in the peak-performance design estimated by our models. There are two pairs of lines, one for multicore processors with conventional general-purpose (GPP) cores, and one for embedded (EMB) cores modeled after ARM11.

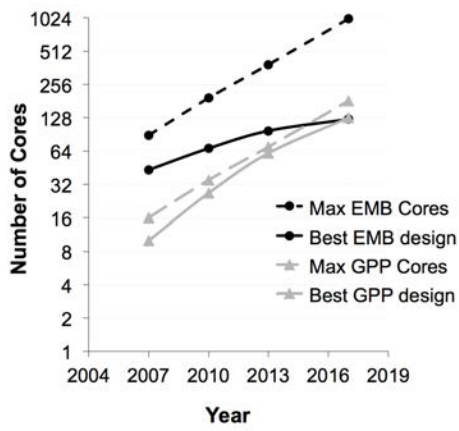

Figure 2: Number of cores for peak-performance designs across technologies

The gap between each pair of lines is an indicator of the impeding dark silicon. Even though up to 1000 cores can fit in a single die at 20 nm, populating the chip with an order of magnitude fewer cores is close to ideal. Placing more cores would require more power and bandwidth, which would force the supply voltage down and the entire system to run slower and lose performance.

The advent of dark silicon is even more pronounced once the bandwidth wall is alleviated—for example, through the use of 3D-stacked memory. In this technology, DRAM chips are stacked within the same package on top of the logic chip

that implements the processing cores. Communication between the stacked dies is very fast: the signals propagate vertically for only a few microns, as the dies are exceedingly thin. Retrieving data from a stacked DRAM is 60x more energy-efficient than using the conventional pin interface to off-chip DRAM. Also, the tight integration and small area footprint allow vertical buses to deliver three orders of magnitude higher bandwidth.

As a result, processor packages with 3D-stacked memory realize superior performance without the need for a large L2 on-chip cache, freeing both area and power to be used by the cores. Even these systems, however, because of power limitations can support only a fraction of the cores than can fit in a die. As a result, a significant portion of the die real estate remains unutilized, or dark. Our models predict that, under 3D-die stacking, as much as 54% of the die area may be left unutilized at the 20 nm technology node (Figure 3).

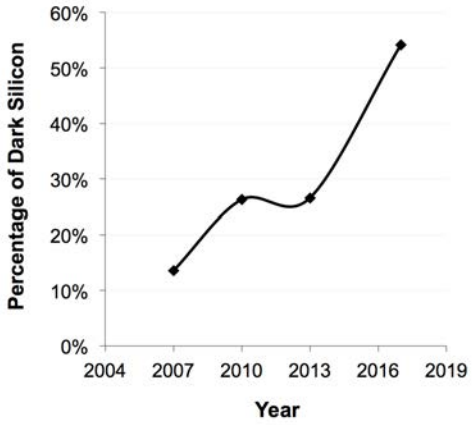

Figure 3: Dark silicon for peak-performance multicores with 3D-stacked DRAM

It becomes apparent that the power constraint is the main cause of dark silicon. While techniques like 3D-die stacking and photonic interconnects push the bandwidth wall far enough to practically free processor chips from the bandwidth limitation, there is little we can do to free future chips from the perils of the power wall. Even if exotic cooling technologies were employed, such as liquid cooling coupled with microfluidics, power delivery to the chip would likely impose a new constraint. Based on ITRS, power delivery poses significant challenges due to poor signal integrity when large currents are delivered at low voltages, for which no mainstream solutions have been proposed to date.

Without much hope of fixing the problem by raising the power budget, the only solution to dark silicon seems to be frugality: we need to identify the sources of energy overheads and remove them to increase the energy efficiency of computing. Today's processors are wasting energy at the circuit layer to provide reliable execution, and at the architectural layer to provide general-purpose computation and transfer data to the cores. An ideal solution would attack all these overheads.

The Energy Overhead of the Circuit Layer

The power consumption of conventional processors is high in part because the supply voltage is determined by conservative guardbands based on worst-case scenarios. To ensure the correct operation of circuits, designers supply the transistors

with a voltage high enough to guarantee that signals will be produced and communicated successfully through the chip under all operating conditions, no matter how rare worst-case conditions may be. However, this design approach results in significant power and area overheads. Even a 40% reduction in guardbands results in 13–19% reduction in power consumption, and 13% reduction in area occupancy and wire lengths [11].

To mitigate the overheads of wide guardbands, recent research advocates the near-threshold-voltage operation of circuits. Keeping all else constant, decreasing the operating voltage (V_{dd}) reduces power consumption at a quadratic rate. However, as V_{dd} approaches V_{th}, some transistors become slower than others, and some signals are not propagated correctly through the circuit and violate timing constraints, causing errors. The surge of errors at the circuit layer is for the most part the result of process variations: material impurities and the uneven distribution of dopants result in a variation in the electrical properties and switching speed of transistors, making them unreliable. The smaller the transistors become, the more susceptible they are to variations.

Building reliable circuits from unreliable components requires a host of techniques to mitigate the influx of errors, including wide guardbands and specialized error detection and recovery circuits. Unfortunately, these techniques induce significant power overheads [11]. But, what if we let go of these guardbands and allow the components of the processor to fail sometimes?

Elastic Fidelity

While traditional designs correct all errors and provide accurate computation, not all computations and all data in a workload need to maintain 100% accuracy. Rather, different portions of the execution and data exhibit different sensitivity to errors, and perfect computation is not always required to present acceptable results. For example, computations that involve human perception (e.g., image, audio, motion) provide a lot of leeway in occasional errors, as visual and auditory after-effects compensate for them. Some applications (e.g., networking) already have strong built-in error correction facilities, as they assume unreliable components [12]. Computations performed on noisy data (e.g., from sensors) are typically equipped with techniques to correct inaccuracies within reasonable error bounds. Computations that iteratively converge to a certain value will continue to converge successfully after the introduction of small errors, albeit a few iterations later. However, the freedom to introduce errors allows the processor to operate components at significantly lower voltage, quadratically reducing the dynamic energy consumption.

Elastic Fidelity capitalizes on this observation by occasionally relaxing the reliability guarantees of the hardware based on the software requirements, and judiciously letting errors manifest in the error-resilient portion of an application's data set. Programming language subtyping designates error-tolerant sections of the data set, which are communicated through the compiler to the hardware subsystem. The hardware then operates components (e.g., functional units, cache banks) occasionally at low voltage to reduce the power and energy consumption. Portions of the application that are error-sensitive execute at full reliability, while the error-tolerant computations are scheduled to run on low-voltage units to produce an acceptable result. Similarly, error-tolerant sections of the data are stored in

low-power storage elements (e.g., low-voltage cache banks, low-refresh-rate DRAM banks) that allow for the occasional error.

By not treating all code and all data the same with respect to reliability, Elastic Fidelity exploits sections of the computation that are error-tolerant to lower power and energy consumption, without negatively impacting executions that require full reliability. Overall, 13–50% energy savings can be obtained without noticeably degrading the output [13, 14], while reaching the reliability targets required by each computational segment [14].

The Energy Overhead of Data Transfers

While relaxing the conservative guardbands offers some respite from dark silicon, there remain significant overheads in conventional multicore computing. One of the most important ones is the high cost of data movement: fetching operands requires orders of magnitude more energy than computing on them. While a typical integer arithmetic operation consumes 0.5 pJ at 28 nm, and a 64-bit double-precision floating-point operation consumes about 20 pJ, reading two operands from a nearby cache memory requires around 100 pJ, which is 5x–200x more than the floating-point and integer operations, respectively. Retrieving the data from a cache 10 mm away requires 356 pJ, a 712-fold energy increase over an integer add, while retrieving them from across a 400 mm^2 chip requires 1100 pJ, a 55x–2200x increase over the floating-point and integer operations, respectively [21]. Even worse, retrieving the data from off-chip DRAM requires 16,000 pJ, three to five orders of magnitude more energy than the energy required to compute on the data!

Elastic Caches

As a result of the high cost of data movement, there has been a significant research effort to minimize data transfers. Aggressive scheduling techniques aim to schedule data-sharing threads together [14], advanced caching aims to minimize trips to main memory [18], and elastic data placement schemes [16] and memory hierarchies [17] aim to bring data and computation within physical proximity. Similarly, recent efforts have shown the advantages of moving computation (i.e., code, which is typically small) close to the data (which is typically large), rather than the reverse [19].

Even simple optimizations to minimize data transfers with negligible implementation cost have been shown to reduce the energy demands by a large margin. Elastic Caches, for example, adaptively co-locate data with computation [16] and can reduce the energy demands of a processor by 17%; by placing the on-chip directory entries close to the computation, they provide an additional 13% energy savings [15].

The Energy Overhead of General-Purpose Computing

While a simple arithmetic operation requires around 0.5–20 pJ, modern cores spend about 2000 pJ to schedule it. This tremendous source of overhead is the price we pay for general-purpose computing. Fetching instructions described in a generic ISA, decoding, tracking instructions in flight, discovering dependencies and forwarding the right values, renaming registers, reordering instructions, and predicting branch targets and target addresses all contribute to this overhead. As a result, compared to ASICs, conventional multicore processors consume two to three orders of magnitude more energy [6].

This profound energy waste contributes not only to the excessive energy consumption of modern computing, but also to the onset of dark silicon. We propose to harness the second problem to fix the first: instead of wasting dark silicon, we should embrace it and use it to realize energy-efficient computation.

Repurposing Dark Silicon for Specialized Computing

Our models indicate that chips will not scale efficiently beyond a few tens to low hundreds of cores, while upwards of 1000 cores can fit in a single chip [2, 9]. Thus, an increasing fraction of the chip in future technologies will be dark silicon. We envision SeaFire, an architecture that implements a sea of specialized cores on dark silicon, where each core is able to perform a narrow set of tasks at significantly lower energy. The executing workload would fire up only the cores most useful to the computation at hand, while the rest of the chip remains switched off to conserve energy. Examples of specialized cores include cores to execute Java bytecode and JVM natively (like Azul's Vega and aJile's JEMcore), or cores that implement server functionality common in the target workloads (e.g., compression, encryption, video decoding). Similarly, some cores could be ideally suited to data-parallel computation (e.g., SIMD), while others could be optimized for long-latency operations (e.g., memory accesses) or ILP (e.g., out-of-order cores). Finally, some cores could simply provide commonly used principles (e.g., string manipulation, convolution, file scanning and filtering).

To estimate the potential of SeaFire, we evaluate an extreme application of this approach. Rather than representing a specific core design, we consider a heterogeneous multicore populated with specialized cores that exhibit ASIC-like properties on the code they execute. We model such cores after ASICs running a CABAC (Context-Adaptive Binary Arithmetic Coding) segment of the H.264 video encoder. CABAC is a form of entropy encoding used in H.264/MPEG-4 AVC video encoding. While it provides higher compression than most alternative algorithms, it requires a large amount of processing, it is difficult to parallelize and vectorize, and its highly control-intensive nature does not lend itself to an efficient hardware implementation. Thus, we choose to model the specialized cores after CABAC in order to obtain conservative bounds, as opposed to modeling them after data-parallel computations that may provide significantly lower energy in a specialized design.

While conventional multicores require extreme parallelism by the software to be fully utilized, a processor with specialized cores on dark silicon attains peak performance with only a handful of cores powered up at a time [2, 9]. This observation holds across technologies, and for applications with up to 99.9% parallelism. The low core count across technologies hints that peak-performance designs with specialized cores can be realized in an increasingly smaller silicon area, leaving an exponentially larger portion of the chip powered-off (dark). SeaFire repurposes the otherwise-dark silicon to implement a large collection of specialized cores to increase the likelihood of finding a core suitable for the computation. Overall, we estimate that SeaFire reduces the energy consumption 12-fold over homogeneous architectures implemented within the same physical constraints [2, 9]. Thus, SeaFire promises not only to reduce the energy consumption by a large margin, but, in doing so, to utilize the otherwise wasted dark silicon.

Concluding Remarks

It is apparent that, unless we change course, we'll soon find ourselves unable to utilize fully the chips we build, and the inordinate energy consumption of computing will make its expansion prohibitive. The culprits seem to be the overheads associated with traditional implementation choices: we waste energy to guarantee correct circuit execution under all circumstances, no matter how rare they may be or how much correctness really matters; we waste energy to move data from faraway locations because until now maybe we didn't have a strong enough incentive to solve the problem; we waste energy in computations, a price we have been willing to pay until recently to gain generality in computing. However, our current path is unsustainable and needs to change.

But change doesn't come easily. Significant challenges need to be addressed to realize the solutions identified above. Every aspect of computing will need to be revisited to make this vision a reality. How to modify programming languages and applications to identify and specify error tolerance? Which computations are ideal candidates for off-loading to specialized hardware, and common enough to be utilized by several workloads? What are the appropriate language and runtime techniques to drive the execution migration across specialized cores? How to restructure software and algorithms for heterogeneity? The list quickly grows long, and the fight seems tough. However, the stakes are high enough to make it a fight worth fighting.

Acknowledgments

The author acknowledges the contributions of his collaborators in various aspects of this research: M. Ferdman, S. Roy, A. Das, K. Liu, T. Clemons, S.M. Faisal, B. Falsafi, A. Ailamaki, S. Parthasarathy, S. Memik, M. Schuchhardt, G. Tziantzioulis, G. Memik, A. Choudhary, I. Pandis, R. Johnson, and the members of the PARAG@N Lab.

References

[1] CERN, Worldwide LHC computing grid, realtime VO-wise data transfer: http://lcg.Web.cern.ch/lcg/, 2011.

[2] N. Hardavellas, "Chip Multiprocessors for Server Workloads," PhD thesis, Carnegie Mellon University, 2009.

[3] *Financial Times*, "Computing: Powerful Argument for Cutting IT Energy Consumption": http://www.ft.com/cms/s/0/4e926678-bf90-11df-b9de-00144feab49a.html#axzz18EH%YGujk, September 2010.

[4] B. Dally, "Power and Programmability: The Challenges of Exascale Computing," DoE ASCR Exascale Research PI Meeting, 2011.

[5] X. Fan, W-D. Weber, and L.A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," Proceedings of the 34th Annual International Symposium on Computer Architecture, 2007.

[6] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B.C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding Sources of Inefficiency in General-Purpose Chips," Proceedings of the 37th Annual International Symposium on Computer Architecture, 2010.

[7] Semiconductor Industry Association, The International Technology Roadmap for Semiconductors (ITRS): http://www.itrs.net/, 2008.

[8] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," Proceedings of the 38th Annual International Symposium on Computer Architecture, 2011.

[9] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward Dark Silicon in Servers," *IEEE Micro*, vol. 31, no. 4, 2011.

[10] E.S. Chung, P.A. Milder, J.C. Hoe, and K. Mai, "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPUs?" Proceedings of the 43rd International Symposium on Microarchitecture, 2010.

[11] K. Jeong, A.B. Kahng, and K. Samadi, "Impact of Guardband Reduction on Design Outcomes: A Quantitative Approach," *IEEE Transactions on Semiconductor Manufacturing*, vol. 22, no. 4, 2009.

[12] A. Mallik and G. Memik, "A Case for Clumsy Packet Processors," Proceedings of the 37th International Symposium on Microarchitecture, 2004.

[13] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-Power Computation," Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, 2011.

[14] S. Roy, T. Clemons, S.M. Faisal, K. Liu, N. Hardavellas, and S. Parthasarathy, "Elastic Fidelity: Trading-Off Computational Accuracy for Energy Reduction," CoRR, abs/1111.4279, 2011.

[15] S. Chen, P.B. Gibbons, M. Kozuch, V. Liaskovitis, A. Ailamaki, G.E. Blelloch, B. Falsafi, L. Fix, N. Hardavellas, T.C. Mowry, and C. Wilkerson, "Scheduling Threads for Constructive Cache Sharing on CMPs," Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2007.

[16] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Near-Optimal Cache Block Placement with Reactive Nonuniform Cache Architectures," *IEEE Micro*, vol. 30, no. 1, 2010.

[17] A. Das, M. Schuchhardt, N. Hardavellas, G. Memik, and A. Choudhary, "Dynamic Directories: Reducing On-Chip Interconnect Power in Multicores," Proceedings of the Design, Automation, and Test in Europe, 2012.

[18] A. Jaleel, K. Theobald, S.C. Steely Jr., and J. Emer, "High Performance Cache Replacement Using Re-Reference Interval Prediction," Proceedings of the 37th Annual International Symposium on Computer Architecture, 2010.

[19] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki, "Data-Oriented Transaction Execution," Proceedings of the VLDB Endowment, vol. 3, no. 1, 2010.

[20] J.H. Patel, "CMOS Process Variations: A Critical Operation Point Hypothesis," Computer Systems Colloquium, Stanford University, 2008.

[21] Stephen W. Keckler, "Echelon: NVIDIA & Team's UHPC Project," DARPA Ubiquitous High Performance Computing Program Meeting, Houston, TX, May 2011.

IPv6: It's Not Your Dad's Internet Protocol

PAUL EBERSMAN

Paul Ebersman first worked on UNIX and TCP/IP for the Air Force in 1984, serving at the Pentagon. He was one of the original employees at UUNET and helped build AlterNet and the modem network used by MSN, AOL, and Earthlink. He has maintained his roots in the Internet and the open source community, working for various Internet infrastructure companies, including the Internet Systems Consortium and Nominum. Paul currently works in the Infoblox IPv6 Center of Excellence as a technical resource, both internally and to the Internet community. pebersman@infoblox.com

Functionally, IPv4 and IPv6 seem pretty similar. They connect a bunch of virtual wires or local networks together so that we can all run our favorite Internet apps. But once you start looking under the hood, there are some obvious differences. There are lots more subnets and addresses available in IPv6, and they are really big and ugly and hard to type (try "ssh 2001:db8:a32:8f3e:3c32:abcd:829:1a" without a typo).

But there are some more subtle twists and turns that may surprise you, and some old habits and "conventional wisdom" rules that we will all need to rethink and relearn. There are also some changes in how addresses are configured, which will change how we all design and build our networks.

Let's crack open the hood and take a peek.

Don't Get Tied Up in IPv4-Think

The Vast Reaches of Space

We are so used to constructing our networks from a patchwork of IPv4 ranges that we've begged, borrowed, or acquired that we've forgotten that a subnetting plan can actually make our lives *easier*. You can get enough IPv6 space to allow this.

Think about how you organize your company, define security zones, and delegate control. Then subnet in a way that follows those boundaries, your process, and your security model. It may be by department, function, or geography. Figure 1 provides a simple example of a location-based subnet plan.

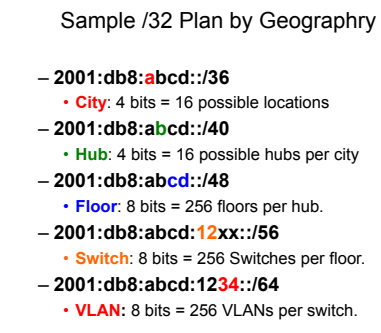

Figure 1: A subnetting scheme that takes advantage of the enormous addressing potential of IPv6

With the example in Figure 1, I can easily make ACLs at whatever level makes sense—by city, VLAN, etc.—because my subnet boundaries match my delegation scheme.

You can make your life easier by subnetting on nibble (4-bit) boundaries, just as we used to do subnet IPv4 address on 8-bit boundaries in classful subnets.

In IPv4, the 32 bits are written as four decimal characters, separated by dots: e.g., 192.168.1.1. Each one of those decimal characters is 8 bits. If you do subnets at 8-bit boundaries (where the prefix length is divisible by 8, such as /24, /16, /8), you don't have to "split" within any of those decimal characters.

With IPv6 and the written representation of the 128 bits as 32 hexadecimal characters (2001:0db8:0100:0000:1111:2222:3333:0001), if you use prefix lengths divisible by four (i.e., on nibble boundaries), you don't have to "split" within any of those 32 hexadecimal characters.

There's no technical reason you couldn't use non-nibble prefixes; it's just way easier for humans to not have to split any of those hexadecimal characters.

Hosts No Longer Require a Single Hard-Coded Default Route

With IPv6, host network stacks now deal gracefully with having multiple addresses on the same interface, with addresses coming and going without any user-visible disruption of most apps. Interfaces can have multiple default routes at the same time. And assignment of default routes is all done with router advertisement messages; there's no need to run an active routing protocol on each host. You may even be able to simplify your current HSRP/VRRP setup by having redundant default routes.

Changing My IP Address No Longer Breaks My Connections

In the old days of modems, you got users off a modem pool by letting existing calls stay up but not taking new calls. As active users hung up, you wound up with an empty modem pool for your maintenance window without having to hang up all of your user calls. We called this quiescence.

With IPv6, each host can have multiple addresses per network interface, choosing which to use based on RFC 3484 [1] and any local rule changes. Addresses come and go, with old addresses kept open for existing connections but not used for any new ones. As with modems, this quiescence allows new addresses to be used by the host without any user-visible disruption in existing connections.

RFC 1918: Can We Please Kill It?

The "conventional wisdom" is that using private addresses is "more secure," because you hide multiple users behind one single public IP address. The bad guys can't tell how many users you have and can't target one particular machine from the outside.

While this does add a layer of security, the reality is that we've convinced ourselves that RFC1918 [2] space is a security feature because we actually ran out of enough IPv4 addresses years ago, and using private address space and NAT let the Internet keep running. We've been trying to make the best of a bad situation.

With the abundance of IPv6 addresses, you can use public addresses in IPv6 for everything. Use ACLs to prevent accidental or malicious traffic from getting into

or out of your network. You know that everything has a unique address, so merging networks isn't nearly as painful. And you do have the stateful firewall option if you like things slow and complicated.

If you're bound and determined to use private addresses, IPv6 does have Unique Local Addresses (ULA). However, they have the same issues as RFC 1918 (not guaranteed to be unique, must be NAT'ed to get global connectivity, and break end-to-end apps).

DHCP: Not Just for Servers Anymore

In IPv4, you either statically configure all hosts, or you centrally manage your network and network policy via DHCP server. With IPv6, you can still statically configure hosts, but it's sure painful. The two new choices are SLAAC (StateLess Address AutoConfiguration) or DHCPv6.

With SLAAC, you push all the configuration to the edges. Each host is given certain information via router advertisement (RA) messages, such as default route and what network prefix(es) the host should create itself addresses from. Then the host configures its own network interfaces.

RAs can come from either a router, a switch port configured to send RA messages, or a server running RA daemon software (such as rtadvd).

With DHCPv6, much like with DHCPv4, the DHCP server provides the client host with most of the information it needs to configure itself. DHCPv6 can give you addresses, rDNS server, TFTP server, and lots of other configuration information, but it can't (currently) give you a default route.

RA messages give you default routes and prefixes but, until recently (RFC 6106 [3]), couldn't give the client a recursive DNS server. Even with RFC 6106, rDNS via RA support in DHCP clients is not yet widespread. And you still can't get NTP server, TFTP server, and other information via RA messages.

Another big change from DHCPv4 is that clients don't make DHCPv6 requests unless they are told to via RA message flags using the M (ManagedAddress) flag and/or O (OtherConfiguration) flag.

The end result is that you have to run both an RA and a DHCP server for most installations. That means implementing network policy requires both DHCP server configuration and router or switch configuration on every subnet with clients.

Mac Address vs. DUID

DHCPv4 uses the client MAC (Media Access Control) address as the unique identifier to the DHCPv4 server. The problems with MAC addresses are well known:

- Not guaranteed to be unique
- Can be changed or forged
- Doesn't identify a host with multiple network interfaces clearly

In spite of all these problems, there wasn't a better answer in IPv4, and we've all learned to deal with the problems. Most end hosts on the Internet use DHCP and MAC address-based identification.

For IPv6, the IETF decided to try to tackle the flaws with MAC-based identification by using a DUID (DHCP Unique IDentifier), one per DHCPv6 client and one per DHCPv6 server.

For every network interface on a host, there is an Identity Association (IA) that is the collection of all the addresses for that interface. The combination of DUID and IA for any given host/network interface uniquely identifies that interface. That allows you to use DHCPv6 for every interface, something that a MAC address-based system can't do.

DHCPv6 is a Layer 3 protocol (the client has a valid IPv6 link local address and uses a multicast IPv6 address to reach a DHCP relay/server). The client no longer has to send a broadcast using its MAC address; it sends the DUID/IA pair instead.

Because DHCPv6 clients don't need to broadcast with their MAC address in the DHCPDISCOVER packet, and DHCPv6 doesn't have the htype/claddr option that DHCPv4 uses to pass a MAC address through DHCP relays, there is no longer a way for the DHCP server to get the client's MAC address. This makes it hard to determine that a particular IPv4 and IPv6 address represent the same host.

Another problem is that you don't generate the DUID until the first time you start up DHCP (as server or client). It is not based directly on the MAC address, nor can you know in advance what the DUID will be until you contact DHCPv6 for the first time. This makes reserved addresses or host statements hard to pre-provision.

There is a proposal at the IETF to address this [5], but it will take time before these become standards and are available in production client and server code.

Failover

DHCPv4 failover was an attempt to deal with two issues: IPv4 client address changes tend to be user-visible, and the shortage of IPv4 addresses made pool sizes small/scarce. The failover protocol used by ISC was one attempt to solve these problems.

In IPv6, both issues are no longer relevant. Any IPv6-compliant network stack will deal gracefully with address changes with no user-visible impact. And a /64 subnet has 4 billion times 4 billion addresses. If you are running out of addresses in a network with that many to use, DHCP and failover aren't the problems you need to solve.

With IPv6, just assign half of the /64 to one DHCP server and half to the other. Yes, if server A is down and a client of server A needs an address, it will wind up getting a new IPv6 address from server B. But we've already established that new addresses aren't an issue for IPv6 clients. And 2 billion times 4 billion addresses is probably still sufficient for the subnet. If you want more redundancy or a remote server as backup, split the pool into three or four chunks and relay as needed.

This solves the problem of always having a working DHCP server on a subnet and avoids much of the fragility and complexity of trying to synchronize a lease file across a failover pair. The interim IETF draft that ISC based its failover on was never finalized in the IETF, due to these complexities. The odds are good that an IPv6 failover draft will not be finalized any time soon.

ICMPv6

Another place where the IETF has really taken lessons learned from IPv4 and improved things is ICMPv6. Function is much more precisely defined, making filtering by type much more accurate.

But ICMPv6 is more than just improved with IPv6; it's now a vital part of making things work. In IPv4, if you filter all ICMP regardless of type, you make your NOC's life miserable but things still mostly work. With IPv6, if you filter ICMPv6, you break:

- Duplicate Address Detection (DAD), the way you make sure two hosts aren't using the same address on the same LAN segment
- SLAAC RA messages
- DHCPv6
- Path MTU Discovery (PMTUD), the replacement for fragmentation of large packets
- Connectivity testing (echo request/response)
- Other network errors

There is a very useful RFC (RFC 4890 [4]) which has best current-practice recommendations on what to filter with ICMPv6.

Security

Meet the New Security Holes, Same as the Old Security Holes

While there are some complexion changes between IPv4 and IPv6 (arp cache corruption vs. neighbor cache corruption, rogue DHCP server vs. rogue router advertisements), most problems with IPv6 are the same as we've been living with in IPv4. Most issues with IPv6 are going to be due to lack of familiarity leading to misconfigurations.

There are some administrative issues—double work in ACLs in a dual stack environment, twice as many ports to check, etc.—nothing that says we should avoid IPv6 completely.

The big problem seems to be the impression that we don't already have IPv6 on our networks. If you're running a current OS X, Linux variant, FreeBSD, or Windows7/Vista/W2008, you probably have hosts using IPv6. If you specifically disable IPv6 completely in Win7/Vista/W2008, Microsoft considers this an unsupported configuration.

The best thing you can do for security is to turn on enough IPv6 that you can at least monitor and see it. Validate your firewalls with your vendor to ensure that they can detect and monitor IPv6, particularly various tunneling technologies. Get a lab up with IPv6 so that your staff starts to build experience and test procedures and toolsets.

IPsec Is Built into IPv6

Yup. Sure is. Per RFC, if you want to have a network stack that claims to support IPv6, you must support IPsec. Sadly, OS vendors have chosen to interpret this as meaning "We ship it with IPsec. If you want to actually enable it, knock yourself out."

The missing piece of IPsec is key management. There is no standard PKI infrastructure. This leaves you with the same situation as in IPv4. You have to configure the end nodes and the VPN/tunnel termination point with a shared secret. No worse than IPv4, no different from IPv4, just no better than IPv4 (at the moment).

The ray of hope here is Microsoft. Since Microsoft Windows authentication already has a built-in PKI, they can use the standard Windows login and credentials to configure a client with no extra effort on the user end. DirectAccess uses this to create an IPsec-secured tunnel from the end user to a W2008 server. The user logs in and is able to securely see any shared resources on the W2008 server as if on the local net, through firewalls, hotel NATs, etc.

If Microsoft can do this, surely the open source community can do it too.

Getting IPv6 on Your Network

How do you eat an elephant? One bite at a time.

By now, you're probably itching to get started implementing IPv6 in your network. The good news is that you don't have to do it all at once. You can do the phases as distinct projects and take time between them as you need to.

Start with an inventory of your IP address usage, subnetting, hardware, software, apps, and make sure everything can support IPv6. Upgrade or replace what can't (or figure out what transition technology might keep it limping along). Then decide on a new subnetting plan and network architecture, if needed.

You will next want to get the IPv6 addresses you need, working with your ISP(s) or applying directly to an RIR (Regional Internet Registry). Also make sure your ISP provides full IPv6 connectivity and is willing to route your new prefix (if you got it directly from the RIR).

For most companies, the easiest place to start is their external Internet site:

- Ensure thatyour firewall/IDS is IPv6 ready.
- Get IPv6 connectivity to your site.
- Create a test site where you can run IPv6 experiments separate from your current Internet-exposed production servers.
- Test in IPv4 to validate it reproduces your production site.
- Convert the test site to IPv6 only, put NAT64/DNS64 in front of the test site, and see if it all still works.
- Remove the NAT64/DNS64, go to IPv6 only, and see what was using IPv4 that you thought was using IPv6.
- Do a beta version of your production site using a subdomain (ipv6.example.com).
- When everything works for your beta testers, add the AAAA record for www .example.com.

Once you know that your production servers can work with IPv6, and have implemented IPv6 on your external Internet presence, you can work on adding IPv6 on your internal (or more complicated) environment:

- Ensure that your firewall/IDS is IPv6 ready (if it's different from your production site).
- Get IPv6 connectivity to your site.
- Get IPv6 running on just your core networking gear and switches.
- Make sure your network monitoring and logging infrastructure can monitor IPv6.
- Simulate your internal site in a lab as you did with your external site. Do the IPV4/IPV6-NAT64-DNS64/IPv6-only dance.
- Work with your vendors on any bugs or lack of features.
- Rinse and repeat until it all works.

Conclusion

These are some of the issues you will encounter as you implement IPv6. There will probably be others, although I'd expect most of those to be hidden in internally written code rather than in OS stacks or large commercial products. But the techniques and experience you'll gain dealing with the topics discussed here, especially as you run pilot IPv6 operations in labs, will make you much better prepared to find them in your own code.

IPv6 requires learning about the differences between the more familiar IPv4 and IPv6, as you will undoubtedly be using both for years to come. And, in some ways, it is 1994 all over again: you will be venturing into an unfamiliar networking technology, with a new set of warts to figure out how to overcome.

References

[1] http://tools.ietf.org/html/rfc3484.

[2] http://tools.ietf.org/html/rfc1918.

[3] http://tools.ietf.org/html/rfc6106.

[4] http://tools.ietf.org/html/rfc4890.

[5] http://tools.ietf.org/html/draft-halwasia-dhc-dhcpv6-hardware-addr-opt-00.txt.

Programming Unicode

SIMSON L. GARFINKEL

Simson L. Garfinkel is an Associate Professor at the Naval Postgraduate School in Monterey, California. His research interests include computer forensics, the emerging field of usability and security, personal information management, privacy, information policy, and terrorism. He holds six US patents for his computer-related research and has published dozens of journal and conference papers in security and computer forensics.
simsong@acm.org

Many people I work with understand neither Unicode's complexities nor how they must adapt their programming style to take Unicode into account. They were taught to program without any attention to internationalization or localization issues. To them, the world of ASCII is enough—after all, most programming courses stress algorithms and data structures, not how to read Arabic from a file and display it on a screen. But Unicode matters—without it, there is no way to properly display on a computer the majority of names and addresses on the planet. My goal with this article, then, is to briefly introduce the vocabulary and notation of Unicode, to explain some of the issues that arise when encountering Unicode data, and to provide basic information on how to write software that is more or less Unicode-aware. Naturally, this article can't be comprehensive. It should, however, give the Unicode-naïve a starting point. At the end I have a list of references that will provide the Unicode-hungry with additional food for thought.

Unicode is the international standard used by all modern computer systems to define a mapping between information stored inside a computer and the letters, digits, and symbols that are displayed on screens or printed on paper. It's Unicode that says that a decimal 65 stored in a computer's memory should be displayed as a capital letter "A"; that a decimal 97 is a lowercase "a"; and that the "£" is hex 00A3. Every Unicode character has a unique name in English that's written with uppercase letters—for example, POUND SIGN. Every Unicode character also has a unique numeric code, which Unicode calls a "code point," that's written with a U+ followed by a four or five character hex code. One of the most common Unicode characters that will not fit in 8-bits is the EURO SIGN (€, or U+20AC); my favorite character is SNOWMAN (☃, or U+2603). Code points over 65,536 are particularly troublesome. Fortunately, they are mostly used for unusual Chinese characters and letters in ancient scripts, such as AEGEAN NUMBER SIX THOUSAND (𐄇, or U+10127) [1].

As the preceding paragraph demonstrates, Unicode is dramatically more complex than the 7-bit American Standard Code for Information Interchange (ASCII) and so-called "Latin1" (actually ISO 8859-1) systems that it replaces. But Unicode brings more complexity than an expanded character set. For example, Unicode accented characters can be represented with a single code point (for example, LATIN SMALL LETTER E WITH ACUTE—é, or U+00E9), or with a character followed by a so-called combining character (for example, Unicode character LATIN SMALL LETTER E (U+0065) followed by Unicode character COMBINING ACUTE ACCENT (U+0301).

The Unicode standard also has rules for displaying and correctly processing writing systems that go right-to-left such as Arabic, Hebrew, and Dhivehi. Before Unicode, it was a complex task to build computer systems that could properly display text that way, let alone intermix right-to-left with left-to-right. Unicode's support for bidirectional text [9] largely eliminates these problems, making it possible to freely use Arabic words like سلام (Salam) in English text. What's particularly clever is that the characters are stored inside the computer in their logical order, not in the order that they are displayed.

Because you can paste Unicode into Web browsers and email messages, Unicode makes it relatively easy to embellish your writing with a wide variety of interesting symbology, including boxed check marks (☑), circled numbers (both ① and ❶), and even chess pieces (♛). Some programmers discover that it's significantly easier to paste a well-chosen Unicode character on a button than to hire a graphic artist to make an icon. And these icons look really great when magnified or printed, because they are typically encoded in the font as vector graphics, not bitmaps.

So why do so many US programmers and technologists have a poor understanding of Unicode? I suspect it is our cultural heritage at work: by design, most written communications within the United States can be encoded in ASCII. Ever since the move to word processing in the 1980s, it's been exceedingly hard for Americans to type characters that are not part of ASCII. As a result, many of these characters, such as the CENT SIGN ¢ (U+00A2), are now rarely used in the US—even though they were widely available on typewriters in the 1970s.

Indeed, for nearly 30 years it has been possible to have a very lucrative career in the US using only ASCII's 96 printable graphemes, never once having to venture into a world where characters are 8, 16, or 32 bits wide. Alas, the ASCII-ensconced lives of most US programmers is increasingly an anachronism, thanks to the integration of the world's economies, the increasingly international aspects of the computing profession, and even simple demographic trends—more people in the US have accented characters in their names and want those names properly spelled.

Code Points and Characters

Like ASCII and ISO8859-1, at its most fundamental level the Unicode standard defines a mapping between code points and print characters. The big difference is quantity: Unicode's most recent version, Unicode 6.1, defines more than 109,000 printable objects.

Most of the code points map to characters, which the standard sometimes calls "graphemes." A grapheme is a unit of written language such as the letter "a" (U+0061), the number "1" (U+0031), or the Kanji for man "男" (U+7537). Most graphemes are displayed as a single glyph, which is the smallest printable unit of a written language.

Unicode terminology is precise and frequently misused: the confusion is frequently reflected by errors in Unicode implementations. For example, a grapheme (such as the lowercase "a") can be displayed by two distinct glyphs (in the case of an "a", one of the glyphs looks like a circle with an attached vertical line on the right, while the other looks like a slightly smaller circle with a hook on the top and a tail in the lower-right quadrant). Both glyphs are represented by the same code point. But some glyphs can represent two characters—for example, some typesetting programs will typeset the letter "f" (U+0066) followed by the letter "i" (U+0069) as

an “fi” ligature (U+FB01), which obviously has a distinct code point. Typesetting programs do not expect to see a U+FB01 in their input file, for the simple reason that people don’t type ligatures. It’s the typesetting program that replaces the U+0066 followed by the U+0069 with a U+FB01, adjusting spacing as appropriate for a specific font.

Arabic is even more complex. Arabic graphemes are written with different glyphs depending on whether the grapheme appears at the beginning, at the end, or in the middle of a word. In the case of Arabic, most of the letters actually have four code points assigned: one that represents the abstract character and three “presentation” code points that represent each printable form. The abstract character might be used inside a word processor document, while the presentation forms are used in files that are rendered for display—for example, in a PDF file. Microsoft’s implementation of Arabic is different for various versions of Word, Excel, and PowerPoint on the Mac and PC. As a result, a file containing Arabic that looks beautiful on one platform can look horrible on another. Google Docs has similar problems when it is used to edit Arabic documents.

Every modern programming language supports Unicode, but frequently with types and classes that are different from those taught in school. For example, C and C++ programmers should use the wchar_t type to hold a Unicode character (or any other printable character, for that matter), defined in the <wchar.h> header. Unicode strings are best represented in C++ with the STL std::wstring class defined in the <string> header. In Python3 the “string” type can hold 0, 1, or multiple Unicode characters; there is no separate type for an individual character. If you are still using Python2 (and you should stop), the “string” class holds ASCII and you need to specify “unicode” (or u"") to create a Unicode character or string. In Java the “char” primitive type and the “Character” class hold Unicode code points, but only those characters with values less than U+FFFF—what’s called Unicode’s Basic Multilingual Plane. Code points that require more than 16 bits are represented by a pair of code points, as we will see below.

Expanding the Basic Multilingual Plane

Before Unicode, many manufacturers developed systems for representing Chinese or Japanese using 16-bit characters, sometimes called “wide” characters. One of the original goals of Unicode was “Han Unification”—that is, using the same code points to denote ideographs that were the same in the two languages. The designers were able to get nearly all of the characters they needed to fit into the 65,536 code points allowed by a 16-bit word. This 2-byte encoding was called UCS-2, for Universal Character Set, 2-byte encoding.

Alas, 65,536 characters were not sufficient to represent all of the Chinese logograms, let alone characters for all of the world’s ancient languages. Today, Unicode supports a total of 17 “code planes,” each with 65,536 characters. Code Plane 0 is the Basic Multilingual Plane and covers Unicode characters U+0000 through U+FFFF (although U+FFFF is explicitly not a valid Unicode character). Code Plane 1 is mostly used for additional symbols, Plane 2 for additional ideographs, Plane 14 for special purpose, and Planes 15 and 16 for private use. Planes 3–13 are currently unassigned. With so much room to grow, it is highly unlikely that Unicode will ever need to be expanded beyond the 17 code planes—even if we encounter an alien race that has 100,000 or more characters in its written language (see Table 1).

Plane 0:	0000–FFFF	Basic Multilingual Plane (BMP)
Plane 1:	10000–1FFFF	Supplementary Multilingual Plane (SMP)
Plane 2:	20000–2FFFF	Supplementary Ideographic Plane (SIP)
Planes 3–13:	30000–DFFFF	Unassigned
Plane 14:	E0000–EFFFF	Supplementary Special Purpose Plane (SSP)
Planes 15–16:	F0000–10FFFF	Supplementary Private Use Area (S PUA A/B)

Table 1: Unicode code planes

The assignment of code planes is arbitrary, of course, but there is some sensibility in the allocation. Plane 0 was first. The extension mechanism that the standards body adopted allows an additional 4 bits to be optionally specified; Plane 1 has all of those additional bits set to 0, Plane 16 has them all set to 1, and Plane 0 is marked by the absence of those optional bits. Planes 1 and 2 are really the only additional planes that were mapped out by the Committee. Because Planes 3–13 are unassigned, they can be trapped as errors. Frankly, I doubt that characters within Planes 3 through 13 will ever be assigned, although these might conceivably be used for some other purpose at some point in our lifetimes.

Many systems (e.g., Java) were designed to be Unicode-aware back when Unicode only had 16-bit characters. Unicode 2.0's creators thought that they would not be able to get programmers to change their systems to use 32-bit characters—especially when most of the bits would always be 0. Instead, a coding scheme was developed that allowed pairs of 16-bit code points in the BMP to represent characters with code points greater than 65,535.

Consider the SQUARED FREE (🈓, or U+1F193). This code point can be represented with 4 bytes using a sequence of four hex characters, 00 01 F1 93. But the code point can also be represented with a pair of Unicode characters called "surrogates." The 18 bits of code points outside the BMP can be divided into two halves, with 9 bits encoded by a first surrogate in the range D800–DBFF and 9 bits encoded using a second surrogate in the range DC00–DFFF. In the case of U+1F193, the two surrogates are U+D83C and U+DD93. Java always uses surrogates to represent characters outside the BMP.

Python can be compiled to use either 16-bit or 32-bit Unicode code points for its internal representation. If sys.maxunicode is 65535, then your Python interpreter is using surrogates internally to represent characters outside the BMP; if sys.maxunicode is 1114111, the Python interpreter can represent all Unicode characters without surrogates internally [4].

No matter how it is compiled, Python allows code points outside the BMP to be specified with a pair of 16-bit surrogates or as a single 32-bit value. Here we ask Python to print a character string from the two surrogates using the Python "\u" escape, which lets us specify any Unicode character in the BMP with its 4-character hexadecimal code:

```
>>> print("\uD83C\uDD93")
```

And here we use the Unicode "\U" escape and enter an 8-character hexadecimal code to print the same character without the use of surrogates:

```
>>> print("\U0001F193")
```

```
>>>
```

The way C and C++ handle characters outside the BMP depends on the platform. Under GCC/G++ 4.2 on Mac and Linux systems, wchar_t is a 32-bit value, allowing Unicode characters outside the BMP to be stored directly. But compile a program with Microsoft's Visual Studio or the mingw cross-compiler, and wchar_t is a 16-bit quantity. Frankly, it's hard to notice the difference, provided you always allocate memory in sizeof(wchar_t) chunks and never depend on the size of a Unicode string being related in any way to the number of characters that it contains.

All of this is less complicated in practice if you use a high-quality Unicode implementation that hides these details. Indeed, I wrote and deployed several Unicode-aware C++ programs before I realized that sizeof(wchar_t) was 4 on my primary development system but 2 when cross-compiling for Windows.

Normalization and Collation

Since Unicode allows the same string to be represented with many different but logically equivalent sequences of code points, the standard provides a way of normalizing any Unicode sequence of code points so that different strings can be compared for equivalency.

Actually, Unicode has two kinds of equivalencies between characters: "canonical equivalence" and "compatibility equivalence." Canonical equivalence resolves the ambiguity introduced by combining characters. LATIN SMALL LETTER E WITH ACUTE (U+00E9) and LATIN SMALL LETTER E (U+0065) followed by a COMBINING ACUTE ACCENT (U+0301) are considered to be equivalent. Compatibility equivalence is used to denote sequences that have the same semantic meaning but may appear visually distinct—for example, SUPERSCRIPT ONE ("¹", or U+00B9) and DIGIT ONE ("1", or U+0031) have compatibility equivalence, as do the characters LATIN CAPITAL LETTER I ("I" U+0049) and ROMAN NUMERAL ONE ("Ⅰ", U+2160). One of the main uses of compatibility equivalence is to improve the recall of string search, but it can also be used to address some of the many security issues caused by having forms that are visually identical but have different encodings [5a]. These two equivalence algorithms mean that determining whether or not two strings are equal is a multi-step process. First you must decide what kind of equivalence you want. Then both strings must be normalized. Finally, they can be compared [5b].

Yet another issue is the sort order of Unicode characters, something known as "collation." The complexity here is that different languages (and sometimes different usages within the same language) require different sorting of the same characters. A common example is that Swedish sorts "z" (U+007A) before "ö" (U+00F6), but German sorts "ö" before "z". Unicode's combining characters add to the complexity. The Unicode Collation Algorithm provides a unified, locale-aware approach to sorting. Although it's described in Unicode Technical Standard #10 [6], most

programmers will be better off using a collation implementation that's widely used and well-debugged rather than implementing their own. For Python users, the function locale.strcoll performs a basic implementation of the ISO 14651 collation algorithm but not the full Unicode algorithm. For a more complete implementation, use IBM's International Components for Unicode library, which has bindings for C, C++, Java, and Python [3].

Encoding and Decoding

So far, this article has been discussing Unicode in the abstract and has avoided the messy issue of reading and writing Unicode data. The issue is messy because modern computer systems read and write data in 8-bit bytes, but Unicode needs a minimum of 16 bits to represent characters in the BMP and 21 bits [10] to represent all possible code points (or two 16-bit pairs, if surrogates are used).

Early Unicode implementations, such as the one in Microsoft Windows, took the rather straightforward approach of storing everything as 16-bit UCS-2 characters. When Microsoft needed to store Unicode on disk—for example, in a file name—it simply wrote the bytes in the same order that they were stored in memory. This process of transforming abstract code points to a specific set of 8-bit codes stored in a file or sent down a wire is called "encoding."

Clearly, there are two ways for a 16-bit code point such as U+0061 to be encoded: as a 61 followed by a 00 (called "little endian," because the little end comes first), or as a 00 followed by a 61 ("big endian"). Rather than mandating that Unicode be encoded one way or the other, Unicode supports both byte orders. UTF-16LE (UCS Translation Format—16-bit Little Endian) is what Windows uses.

For example, the FAT32 file system stores both legacy ISO8859-1 8.3 filenames and UTF-16LE filenames that can be up to 255 characters long. NTFS file systems store only UTF-16LE filenames. This means that the filename README.TXT is stored as the UTF-16LE sequence 52 00 45 00 41 00 44 00 4d 00 45 00 2e 00 54 00 58 00 54 00. Most of the Windows API functions that operate on files have two versions—one that takes ISO8859-1 names terminated with a 00, and one that takes UTF-16LE "wide" names terminated with a 00 00. For creating files, these are the CreateFile() and CreateFileW() functions.

Similar to the 2-byte encodings, Unicode also supports 4-byte encodings UTF-32LE and UTF-32BE. With the UTF-32LE encoding the string "READ" would encode as 52 00 00 00 45 00 00 00 41 00 00 00 44 00 00 00. Such encodings are rarely used outside of a computer's memory, because of the storage cost.

Unicode provides a special code called the Byte Order Mark (BOM, U+FEFF) that can be stored inside a file and used to unambiguously indicate whether the file is encoded as UTF-16LE, UTF-16BE, UTF-32LE, UTF-32BE, or using the variable-length UTF-8 code (see below). The BOM approach works because the byte-swapped character U+FFFE is defined to be an invalid code point. Thus, by looking at the first 2 or 4 bytes of a file, it is possible to determine the encoding (see Table 2).

Initial Bytes	Encoding
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian
FE FF	UTF-16, big-endian
FF FE	UTF-16, little-endian
EF BB BF	UTF-8 (see below)

Table 2: Unicode Byte Order Mark signatures, from http://unicode.org/faq/utf_bom.html

C programmers have the biggest problem with UTF-16 and UTF-32 encodings, for the simple reason that the C standard library uses the NULL character (00) to denote the end of a string. The obvious way to avoid this is by using the wide-character version of the string library—for example, use wcslen() instead of strlen(). Personally, I recommend abandoning C-style strings and using the C++ STL std::wstring type instead.

From the preceding examples it may seem that a program can trivially convert between Unicode and ASCII by simply removing or inserting alternating NULL characters. You should never do this!!! This simplistic approach will fail if the Unicode string contains anything other than code points in the range U+0001 through U+007F. Instead, programs should explicitly encode Unicode strings from an abstract internal representation when strings are transformed for operating system APIs, sent over a network connection, or persisted into a file. Likewise, when data is read from an external source it should be decoded from the wire format into the program's internal representation.

UTF-8

UTF-8 is a system for encoding Unicode code points using a variable-length sequence of 8-bit characters. UTF-8 has the property that 7-bit ASCII characters are directly coded as a single UTF-8 byte, making UTF-8 upwards compatible from ASCII. Characters in the range U+0080 through U+07FF are coded as 2 bytes; the remaining characters in the BMP are coded as three bytes; characters outside the BMP are coded as 4 (see Table 3).

The UTF-8 scheme makes it possible to identify the start of a UTF-8 character from within a randomly chosen block of UTF-8 encoded bytes. If the most significant bit is a 0, then the character is a 7-bit UTF-8 character. If the high bits are "10", then it is a continuation character: move forward or backwards until a byte is found that begins "0" or that begins "11". The number of leading "1"s in the first byte of a UTF-8 encoding indicates the number of bytes in the sequence.

As the table makes clear, UTF-8 is great for Americans, since documents coded in UTF-8 are the same size as documents coded in ASCII. For Europeans, the advantage of UTF-8 is that all of their accented characters can be displayed, with only the non-ASCII characters taking up 2 bytes. For the Chinese, UTF-8 is not so good, as for most text it results in a 50% increase in required storage space compared to UTF-16.

Bits	Code	Point	Range	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	-	U+007F	0xxxxxxx			
11	U+0080	-	U+07FF	110xxxxx	10xxxxxx		
16	U+0800	-	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	-	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Table 3: UTF-8

One of the primary advantages of UTF-8 is that the NULL character is never used. This means that the POSIX APIs can be used more or less transparently with UTF-8 encoded Unicode. The confusion here, however, is that UTF-8 is not Unicode—it is a Unicode encoding. Keeping your program's internal data encoded in UTF-8 is fine as long as the strings are viewed atomically and no string operations such as comparison, search, case change, or splitting ever need to be performed by the program. All of these operations require decoding the UTF-8 sequences into Unicode characters and then re-encoding the Unicode characters back to UTF-8.

Source Code

Encoding and decoding comes up in two primary places when coding: the encoding and decoding of files that are written and read by your program, and the program's source code itself. To see how this works in practice, consider this simple Java program with some embedded Unicode characters:

```
class test {
       public static void main(String[] args) {
            String LETTER_A = "A";
          System.out.println("size("+LETTER_A   +")="+LETTER_A.length());

            String IDEOGRAPH = "男";
          System.out.println("size("+IDEOGRAPH   +")="+IDEOGRAPH.length());

          String SQUARED_FREE = "🆓";
          System.out.println("size("+SQUARED_FREE+")="+SQUARED_FREE.length());
       }
}
```

Java programs are files, of course, so they need to be encoded with a particular Unicode encoding when they are stored in the file system. This file was written with Apple's TextEdit application and saved in UTF-8. In UTF-8 the LETTER_A above takes a single byte, the IDEOGRAPH takes 3 bytes and the SQUARED _FREE takes 4 bytes. So when we run this program we get this result:

```
size(A)=1
size(男)=3
size(🆓)=4
```

Whoops! The values in the Java String are not Unicode code points! Instead they are literally the encoded bytes. This can be confusing (it's confusing to me, at least).

Instead of saving the file in UTF-8, the source file can be saved in UTF-16. I've done that, renaming the class from "test" to "test16." Now when the program is compiled, the Java compiler needs to be told the encoding used by the file:

```
$ javac -encoding utf-16 test16.java
$
```

When I run the program I get these confusing results:

```
$ java test16
size(A)=1
size(?)=1
size(?)=2
$
```

The question marks display result from the fact that the program was run inside a UTF-8 terminal. The A takes a single UCS-2 character, as does the 男. Java's runtime seems willing to convert the A to UTF-8, but not the 男. The SQUARED _FREE is represented in the Java source file with two surrogate pairs (I saved it as UTF-16, remember?), so the length of the string is 2, not 1.

I also tried saving my test program in UTF-32LE and compiling it with the Java compiler, but I got this error:

```
$ javac -encoding utf-32 test32.java
test32.java:1: warning: unmappable character for encoding utf-32
????????????????????????????????????????????????????????????????????????...
...
```

Clearly, my Java compiler does not support UTF-32 for input source encoding. For more information about how Java does this, see the Java documentation for the Character and String classes and the Java tutorial "Working with Unicode" [1a].

Python source code encodings are defined with a "magic comment" [1b] like this:

```
#!/usr/local/bin/python
# coding: utf-8
import os, sys
...
```

Reading and Writing Files

Similar problems occur when attempting to process files. Asked to open a file and infer its coding, some programs will attempt to guess whether the file's contents are in ASCII, UTF-8, or one of the UTF-16 dialects. Unfortunately, it is not always possible to guess correctly, for the simple reason that there are many hex sequences that can be decoded using multiple coding variants. Consider the sequence of hex bytes 41 42 43 44. This could be the UTF-8 sequence "ABCD" (U+0041 U+0042 U+0043 U+0044), but it could also be the UTF-16LE sequence 䅡䅢 (U+4241 U+4443) or the UTF-16BE sequence 格鈴 (U+4142 U+4344).

Python and Java address the encoding issue by allowing the programmer to specify an encoding when a file is opened. For example:

```
f = open("file.txt",mode="r",encoding="utf-8")
```

(In Python2.7, you can get the same functionality with the codecs.open function.)

In Java, one would use:

```
FileInputStream fis = new FileInputStream("file.txt");
InputStreamReader isr = new InputStreamReader(fis,"UTF8");
BufferedReader in = new BufferedReader(isr);
```

Some file formats allow you to specify the encoding inside the file itself, which is something of a trick, because you need to know the encoding in order to decode the file.

For example, if you edit files with EMACS, you can put a local variables line at the top of your file to tell EMACS which coding to use:

```
# -*- coding: utf-8 -*-
```

In XML, encodings are specified on the first line of the file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Python's XML parsers expect to read this line to determine the coding of the file. As a result, to read an XML file with Python3 and process it with expat, it's necessary to open the file as a binary stream:

```
f = open("file.xml",mode="rb")
p = xml.parser.expat.ParserCreate()
...
p.ParseFile(f)
```

Encoding a Python Unicode string to a particular representation is quite simple. Here's how s, a Python3 Unicode string with my favorite Snowman character, looks in UTF-8, UTF-16LE, and UTF-16BE:

```
>>> s = "This is a Snowman: ☃"
>>> s.encode('utf-8')
b'This is a Snowman: \xe2\x98\x83'
>>> s.encode('utf-16le')
b'T\x00h\x00i\x00s\x00 \x00i\x00s\x00 \x00a\x00 \x00S\x00n\x00o\x00w\
x00m\x00a\x00n\x00:\x00 \x00\x03&'
>>> s.encode('utf-16be')
b'\x00T\x00h\x00i\x00s\x00 \x00i\x00s\x00 \x00a\x00 \x00S\x00n\x00o\
x00w\x00m\x00a\x00n\x00:\x00 &\x03'
>>>
```

For encoding and decoding in C++, I recommend using the open source UTF8-CPP package [2]. The package contains C++ classes and iterators for interconverting between UTF-8, UTF-16, and UTF-32. If you want a more complete (and significantly larger) implementation, IBM's ICU is a better choice.

Encoding Errors

Many programmers get their first unpleasant taste of Unicode when they attempt to read a file and instead of getting data, they get an exception. For example, consider a file "file.txt" that contains the hex characters FF FE FE FF. Try to read this file with Python, and Python will throw an exception:

```
>>> f=open("file.txt")
>>> f.read()
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
      File "/opt/local/Library/Frameworks/Python.framework/Versions/3.2/
lib/python3.2/codecs.py", line 300, in decode
        (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf8' codec can't decode byte 0xff in position 0: invalid
start byte
>>>
```

Python assumes the file is in UTF-8, and FF is an invalid UTF-8 character. Of course, Python couldn't read this file if opened in text mode with any encoding.

If you are unsure if a file contains valid Unicode encodings, one approach is to open it in binary mode and convert it a line at a time. This approach works best with files encoded as UTF-8, since it's relatively easy to spot the line breaks, but it can be applied to UTF-16 and UTF-32 encoded files as well. If a line contains invalid Unicode, you can try converting it character by character. There are a number of packages that will do this, such as BeautifulSoup's UnicodeDammit class.

For longtime UNIX programmers, this need to open files in "text" or "binary" mode might seem like a step backwards—or into the wacky world of Windows programming. Sadly, the multiplicity of encodings leaves us no choice.

Problems resulting from improper encoding and decoding pervade modern computing systems. If you have ever opened a Web page and seen it filled with white question marks in black diamonds, your browser was showing the Unicode REPLACEMENT CHARACTER (, U+FFFD), because the bytes on the Web page couldn't be decoded using the encoding the Web page specified. This frequently happens on Web pages that specify no encoding at all but contain smart quotes; the default HTTP encoding is ISO 8859-1, and 8859-1 doesn't have any smart quotes in it.

I encountered a more egregious case of bad encoding after a recent trip to Japan. All of the little paper receipts had nicely printed katakana, hiragana, and kanji characters. But the online statement for my American Express credit card for a $39.30 transaction at the Tokyo Muji store looked like this:

MUJIRUSHI RYOHIN *	S L GARFINKEL	39.30
TOKYOTO TOSHIMAKU		
TOKYOTO TOSHIMAKU		
Foreign Spend Amount:	3,064 JAPANESE YEN	
Doing Business As:	MUJIRUSHI RYOHIN	
Merchant Address:	ncgÔcn nÇuh	
	ÏfæÇbiÐæhã 4-27-10	
	Æ]ÊcÄæjæb1Ïæ× 3F	
	JAPAN	

From the selection of characters, it seems that the merchant's bank transmitted the transaction information to American Express as an encoded UTF-8 string, but the data was then decoded by a computer in the US that assumed ISO 8859-1.

You don't have to travel to Japan to see these kinds of problems. A growing number of GNU tools now output UTF-8 error messages. Consider this buggy function:

```
int fail() {
        return "A";
}
```

Here is the error message that g++ generates when it is compiled:

```
$ g++ -Wall fail.cpp
fail.cpp: In function ‘size_t fail()’:
fail.cpp:2: error: invalid conversion from ‘const char*’ to ‘int’
$
```

The compiler outputs the error messages with smart quotes because the environment variables LANG and LC_ALL are set to 'en_US.UTF-8'. The compiler notices this encoding and sends the hex sequence E2 80 98 for the open quote and E2 80 99 for the close quote. If I unset both environment variables, the smart quotes go away:

```
$ unset LC_ALL; unset LANG; g++ -Wall fail.cpp
fail.cpp: In function 'size_t fail()':
fail.cpp:2: error: invalid conversion from 'const char*' to 'int'
$
```

One day I was running my compiles inside EMACS on a remote system and started seeing error messages like this:

```
fail.cpp:3: error: invalid conversion from âconst char*â to âcharâ
```

EMACS was not expecting the compiler to be sending UTF-8, so it assumed the default of ISO 8859-1. It received the E2 and displayed the â character (U+00E2); the 80, 98, and 99 have no mapping in ISO 8859-1, so they were not displayed. (A more clever EMACS implementation might notice that the sequence E2 80 98 is invalid ISO 8859-1 but valid UTF-8 and change the encoding mode for the buffer, but that might cause other problems.)

Conclusion

The information here should help many C, C++, Java, and Python programmers in developing Unicode-aware programs. There's certainly a lot more to learn, though. For those programmers with the time and the interest to delve deeply, here are some suggestions for further reading. Even if you aren't a Python programmer, it's quite instructive to read the Python "Unicode HOWTO." While there's lots to criticize about the Python Unicode implementation, Python3 gets a lot of things right. Python also includes a built-in Unicode database in a module named, aptly enough, unicodedata (http://docs.python.org/library/unicodedata.html). With the database you can trivially convert from code points to Unicode names and access other aspects of the standard.

Resources

Apple's Character Viewer, built into Mac OS X, is a great way to find Unicode characters. You can access it from the Apple menu bar if you enable the appropriate option in the System Preferences.

Wikipedia's articles on Unicode are really excellent.

The Unicode standard is available online. Start with the FAQs [7] and the Technical Reports [5, 6, 8].

Stackoverflow.com is filled with good information about Unicode.

The Web site FileFormat Info contains detailed information on each Unicode character, including test pages to see if your browser will display it. Find the snowman at http://www.fileformat.info/info/unicode/char/2603/.

References

[1] It's quite possible that this character won't show up properly on your system, since many programs still do not properly display Unicode code points over U+FFFF. You can see what AEGEAN NUMBER SIX THOUSAND is supposed to look like and test your browser's functionality at http://www.fileformat.info/info/unicode/char/10127/index.htm.

[1a] The Java Unicode tutorial can be found at http://docs.oracle.com/javase/tutorial/i18n/text/unicode.html. Documentation for the String and Character classes is at http://docs.oracle.com/javase/7/docs/api/java/lang/Character.html and http://docs.oracle.com/javase/7/docs/api/java/lang/String.html.

[1b] For more information on Python's magic comments to specify file encoding, see http://www.python.org/dev/peps/pep-0263/.

[2] See UTF-8 with C++ in a Portable Way: http://utfcpp.sourceforge.net/.

[3] IBM's International Components for Unicode: http://site.icu-project.org/.

[4] Python 3.3 will be able to represent 1, 2, or 4-byte code points internally using the most efficient representation. For more information, see PEP393 Flexible String Representation: http://www.python.org/dev/peps/pep-0393/.

[5a] For a complete discussion of Unicode security issues, please see Unicode Technical Report #36, Unicode Security Considerations: http://unicode.org/reports/tr36/. There is also a very short but somewhat informative Security Issues FAQ at http://unicode.org/faq/security.html.

[5b] See Unicode Technical Annex #15: Unicode Normalization, for an in-depth discussion of the different forms of equivalence and the Unicode Normalization Algorithm: http://unicode.org/reports/tr15/.

[6] TR10—Unicode Collation Algorithm, Unicode Technical Standard #10: http://unicode.org/reports/tr10/.

[7] FAQ: UTF-8, UTF-16, UTF-32, and BOM: http://unicode.org/faq/utf_bom.html.

Python Unicode How To—http://docs.python.org/release/3.1.3/howto/unicode.html.

[8] TR17—Unicode Character Encoding Model, Unicode Technical Report #17: http://unicode.org/reports/tr17/.

[9] See "Unicode Bidirectional Algorithm" (Unicode Standard Annex #9), which formally defines the correct way for conforming Unicode implementations to display bidirectional text: http://unicode.org/reports/tr9.

[10] Twenty-one bits are required to represent an arbitrary code point: 16 bits for the character within the Plane; 1 bit indicating whether the code point is within the BMP or uses one of the higher planes; and 4 bits to represent the higher plane that is in use.

How Perl Added Unicode Support 10 Years Ago Without You Noticing It

TOBI OETIKER

Tobi Oetiker is the author of several wel-known open source applications: MRTG, RRDtool, and SmokePing. He co-owns and works at Oetiker+Partner AG, a consultancy and development company in Olten, Switzerland. Tobi's current pet open source projects are extopus.org, a tool for integrating results from multiple monitoring systems into a cool JavaScript-based Web frontend, and remOcular.org, a slick, interactive command line-to-Web converter. Read more from tobi on @oetiker, G+ or tobi.oetiker.ch
tobi@oetiker.ch

Imagine that your new German customer sends you a ton of text files that you have to add to your document database. You write a Perl script which neatly imports all the data into your shiny new PostgreSQL database. As you tell this to your DBA, she wonders what had happened to all the German ä, ö, ü umlauts and ß characters in the process. You had not suspected that there might be a problem, but, as you look, all is well—this, despite the database being UTF-8 encoded while the German text files were seemingly normal text files. Another shining example of Perl doing exactly what you want even when you don't know what you are doing.

All seems well, that is, until someone from accounting notices that all the Euro symbols (€) have been turned into ¤ symbols. That's when you start digging into how this really works with Perl and character encodings and Unicode.

A Short Introduction to Unicode

Back in the '60s, the American Standard Code for Information Interchange (aka ASCII) had become the lingua franca for encoding English text for electronic processing outside the IBM mainframe world.

As the use of computers spread to other languages, the whole encoding business became a jumbled mess. The vendors, as well as some international standardization bodies, fell over each other to come up with sensible ways of encoding all the extra characters found in non-English languages. Each language or group of similar languages got one or several encodings. In Western Europe, the Latin1, or ISO-8859-1, encoding became popular in the '80s and '90s. It sported all the characters required to write in the Western European languages.

Working with a single language, this was fine, but as soon as multiple languages were in play, it all became quite confusing; data had to be converted from one encoding to another, often losing information as some symbols from encoding A could not be represented in encoding B.

In the late '80s, work had begun to create a single universal encoding, capable of encoding text from all the world's languages in a unified manner. In 1991 the Unicode consortium was incorporated, and it published its first standard later that year. The current version of the standard is Unicode 6.0, published in October 2010. It covers 109,000 symbols from 93 different scripts. Each symbol is listed with a visual reference, as well as a name made up from ASCII letters, and is tagged with properties giving additional information as to the character's purpose.

With its huge number of characters, Unicode requires multiple bytes to store each symbol. A pretty wasteful undertaking, when you recall that most languages written in Latin script will require only about 70 different symbols to get by. Therefore, a number of different Unicode "encoding" schemes were proposed over time. These days the UTF-8 and UTF-16 schemes are the most popular. Both are variable length encodings, where symbols will use a varying number of bytes to be stored.

UTF-16 is the "native" encoding used by Microsoft operating systems since Windows 2000. It requires at least 2 bytes per character. UTF-8 is the primary encoding used in most Internet-based applications and also in the UNIX/Linux world. Its main feature is that it encodes all of the original 7-bit ASCII characters as themselves. This means that every US-ASCII encoded document is equivalent to its UTF-8 counterpart. All other Unicode characters are encoded by several bytes. The encoding is arranged such that most of the extra symbols required by Western European languages end up as 2-byte sequences.

These days UTF-8 is widely used. XML documents, for example, are encoded in UTF-8 by default. A lot of the Web content is encoded in UTF-8, and most Linux distributions use UTF-8 as their default encoding.

Perl Unicode Basics

In July 2002, with the release of Perl 5.8, Unicode support was integrated into the language. Since everything was done in a nicely backward-compatible manner, the six lines under the "Better Unicode Support" heading in the release announcement went largely unnoticed. The good news is that, in the meantime, most people are actually using Perl 5.8 or a later version, so all the information in this article should be readily applicable in your real-life Perl setup.

The fundamental idea behind Perl's Unicode support is that every string is stored in Unicode internally. If the string consists only of characters with code points (numeric IDs) lower than `0x100`, the string is stored with one byte per character. Since the Unicode code points `0x0` to `0xff` are equivalent to the Latin1 character set, nothing much changed to the casual observer. If any characters with code points `0x100` or higher are present, the string is UTF-8 encoded and flagged appropriately.

On IBM mainframes, Perl uses the EBCDIC encoding, and thus a matching UTF-EBCDIC was chosen to go with it. For the purposes of this article, if you are using Perl on a mainframe just think "EBCDIC" when you read "Latin1."

Perl Unicode Internals

So the main new thing is that now strings can be stored either as sequences of characters with one byte per character or as UTF-8 encoded character sequences with a flag. The perlunifaq(1) suggests that you not even think about all these things, pointing out that they will just work. This is largely true, but I found that until I understood what was happening internally, I kept running into interesting corner cases, driving me nuts. So here is your chance to get your mental picture cleaned up as well.

But, as the perlunifaq suggests, the functions shown in the following examples are not normally required in everyday tasks.

uft8::is_utf8($string) tells whether the UTF-8 flag is set on a string or not:

```
#!/usr/bin/perl
my %s = (
  latin1    => chr(228), # latin1 ä;
  utf8      => chr(195).chr(164), # utf8 encoded ä char
  smiley    => "\x{263A}", # unicode smiley
);
for (keys %s){
       print "$_: >".utf8::is_utf8($s{$_})."< $s{$_}\n";
}
```

Terminals set to work in Latin1 encoding, will show:

```
latin1: >< ä
Wide character in print at p1.pl line 8.
smiley: >1< â☺º
utf8: >< Ã¤
```

Terminals running in UTF-8 mode will display:

```
latin1: ><
Wide character in print at p1.pl line 8.
smiley: >1< ☺
utf8: >< ä
```

The presentation of the characters is entirely up to the terminal, hence the different rendering. Perl assumes that your output device is in Latin1 single-byte mode and warns that it will have trouble displaying the smiley character, which has no equivalent representation in Latin1.

The example also shows that Perl will keep strings in single-byte mode unless it is forced into UTF-8 encoding by the content of the string. Also, the UTF-8 encoded string is not automatically recognized as such.

A few more functions help to get things sorted. The `utf8` namespace holds a bunch of utility functions that allow you to move strings between encodings:

utf8::upgrade($string) in-place converts from single byte to UTF-8 encoding while setting the UTF-8 flag. If the UTF-8 flag is already set, this is a no-op.

utf8::downgrade($string[, FAIL_OK]) in-place converts from UTF-8 to single byte while removing the UTF-8 flag. If the logical character sequence cannot be represented in single byte, this function will die unless FAIL_OK is set.

utf8::encode($string) in-place converts from internal encoding to a byte-sequence UTF-8 encoding, and removes the UTF-8 flag in the process. Since Unicode strings are internally represented as UTF-8 already, all this really does, is remove the UTF-8 flag from a string.

utf8::decode($string) checks if a single-byte (non-encoded) string contains a valid UTF-8 encoded character sequence and sets the UTF-8 flag if this is the case.

```
#!/usr/bin/perl
my %s = (
  latin1  => chr(228), # latin1 ä;
  utf8    => chr(195).chr(164), # utf8 encoded ä char
```

```
  smiley  => "\x{263A}", # unicode smiley
);
utf8::upgrade($s{latin1}); # latin1 ⟶ internal utf8
utf8::decode($s{utf8}); # set the utf8 flag
for (keys %s){
       print "$_: >".utf8::is_utf8($s{$_})."< $s{$_}\n";
}
```

A UTF-8 terminal now shows:

```
latin1: >1<
Wide character in print at p1.pl line 12.
smiley: >1< ☺
utf8: >1<
```

All strings are in UTF-8 mode internally, so all should be well, but only the smiley character gets printed; the ä character is lost. The Latin1 terminal, on the other hand, shows:

```
latin1: >1< ä
Wide character in print at p1.pl line 12.
smiley: >1< â˜º
utf8: >1< ä
```

The reason for this effect is Perl assuming that our output device (STDOUT) is working in single-byte mode. Perl is "doing what you want" by encoding all the output strings into Latin1, and only if there is no Latin1 representation will it resort to UTF-8 native encoding. This leads to the question of how to tell Perl about the encoding in use on `STDOUT`. The `binmode` function helps:

```
#!/usr/bin/perl
binmode(STDOUT,':utf8');
my $smile = "\x{263A}";
print "$smile\n";
```

When running in an UTF-8 enabled terminal you now get properly encoded data *and Perl will also not complain about wide characters anymore*:

```
☺
```

While this works fine if you are running on an UTF-8 terminal, it would not work well for sites still running in Latin1 mode. Normally the `LANG` environment variable gives an indication as to the encoding in use on the system. If you want Perl to take this into account, you can use the `open` pragma and the `:locale` encoding:

```
#!/usr/bin/perl
use open ':locale';
my $umlaut = chr(228);
utf8::upgrade($umlaut);
print "$umlaut\n";
```

which will always output an "ä", taking the default encoding into account when reading and writing data on the system.

When interpolating a string containing material with the UTF-8 flag set (the smiley gets an automatic UTF-8 promotion due to its content, which cannot be

represented in a single-byte encoding), then the resulting string will be upgraded to UTF-8 mode as well:

```
#!/usr/bin/perl
use open ':locale';
my $umlaut = chr(228);
my $smile = "\x{263A}";
print "$umlaut $smile\n";
```

Running this on a Latin1 system gives:

```
"\x{263a}" does not map to iso-8859-1 at p3.pl line 5.
ä \x{263a}
```

If you want to write your Perl scripts in UTF-8 encoding, you can use the UTF-8 pragma to tell Perl about this.

```
use utf8; # assume utf8 program text
no utf8; # assume native program text
```

Note, though, that this only affects how Perl treats the text of the program, so it will understand an UTF-8 encoded "ä" in the program text, but it will still store it in native encoding internally. The UTF-8 flag will only be set on strings that do contain Unicode characters with code points above 0xff.

When a string has the UTF-8 flag set, all string handling functions will continue to work in an intuitive manner, meaning they will act on characters and not on bytes. This might cause some interesting side effects, as the length command will, for example, not tell you anymore how many bytes are in a string, but how many characters. Using the bytes pragma, you can force Perl to still look at the bytes and not at the characters:

```
#!/usr/bin/perl
my $umlaut = "ä";
print 'plain: ',length($umlaut),"\n";
utf8::upgrade($umlaut);
print 'utf8: ',length($umlaut),"\n";
use bytes;
print 'byte length:', length($umlaut),"\n";
```

As expected the UTF-8 version of the string uses 2 bytes of memory.

```
plain: 1
utf8: 1
byte length:2
```

PerlIO and the Encoding Module

The real fun begins when interacting with data from outside the program. The PerlIO layer goes a long way toward making this process as simple as possible. It allows you to do elaborate data processing steps as you are working with file handles. Using the three-argument open syntax and an appropriate PerlIO layer definition is all it takes:

```
#!/usr/bin/perl
open my $fs, '<:encoding(latin15)','euro-test.txt';
my $data = <$fs>;
print 'utf8 flag: ',utf8::is_utf8($data),"\n";
```

With this setup, PerlIO takes care of decoding the Latin15 encoded input file and stores the result in UTF-8 mode internally:

```
utf8 flag: 1
```

Latin15 is another name for the ISO-8859-15 encoding. It is the single-byte encoding commonly used in Western Europe these days. Latin15 is very similar to the classic Latin1 encoding, but a few characters have been replaced. Most importantly, Latin15 includes the € (Euro) symbol.

Using the `binmode` command, the encoding of an open file handle can be changed:

```
#!/usr/bin/perl
open my $fs, '<','euro-test.txt';
binmode($fs,':encoding(latin15)');
my $data = <$fs>;
```

The open pragma allows you to define the default encoding for commands creating file handles:

```
#!/usr/bin/perl
use open IN => ':encoding(latin15)', OUT=>':utf8';
open my $fs, '<','euro-test.txt';
my $data = <$fs>;
```

The encoding and decoding process can also be controlled directly by using the Encode module:

```
#!/usr/bin/perl
use Encode;
$x = decode('iso-8859-1', chr(228));
print 'flag: ',utf8::is_utf8($x),
       ' - ',encode('utf8',$x),"\n";
```

The decode step turns the "ä" character in Latin1 encoding into a Perl UTF-8 character sequence with the UTF-8 flag set. The encode step turns it into a UTF-8 multi-byte sequence without the UTF-8 flag set. A Latin1 terminal will show:

```
flag: 1 - Ã¤
```

The Unicode Bug

The Unicode standard not only defines code points (numeric IDs) for its characters, but it also provides properties such as *Letter, Number, Uppercase_Letter, Space_Separator* . Perl has access to this information and can use it in regular expressions and other commands. The example below demonstrates the behavior of the "\w" (word characters) regular expression match:

```
#!/usr/bin/perl
my $a = 'aäa';
$a =~ /(\w+)/ and print "standard match: $1\n";
utf8::upgrade($a);
$a =~ /(\w+)/ and print "utf8 match: $1\n";
```

The result is a bit odd:

```
standard match: a
utf8 match: aäa
```

While Perl uses Unicode for its internal strings, it seems to use the Unicode meta-information only when the string is in UTF-8 encoding. The "\w" does not match "ä". This can be fixed by using the good old "locale" module, but that is from a time when Perl did not assume all strings to be in Unicode.

This behavior has become known as the "Unicode Bug"; it has been present for quite some time and therefore could not just be fixed without breaking existing code. Perl 5.12 therefore introduced the `unicode_strings` feature, which "fixes" the bug:

```
#!/usr/bin/perl-5.12.0
use feature 'unicode_strings';
my $a = 'aäa';
$a =~ /(\w+)/ and print "standard match: $1\n";
utf8::upgrade($a);
$a =~ /(\w+)/ and print "utf8 match: $1\n";
```

Now Perl uses the knowledge from the Unicode standard in all cases and the result looks fine:

```
standard match: aäa
utf8 match: aäa
```

But also note that the bug does not affect you if you are using UTF-8 encoded and flagged character strings, the format you would end up with when using the PerlIO layer or the Encoding module functions.

CPAN and Unicode

When using CPAN modules, make sure to check their documentation for Unicode support.

The DBD module for PostgreSQL (`DBD::Pg`), for example, will return all string data with the UTF-8 flag set and properly decoded, unless the database encoding is set to `SQL_ASCII`. You can use the `pg_utf8_strings` flag to override the automated decision. The MySQL DBD module can deal with UTF-8 encoded databases, but you have to tell it explicitly by setting the `mysql_enable_utf8` flag on the database handle.

`XML::LibXML` will also work with UTF-8 characters without further ado.

Mojolicious, Dancer, and other new kids on the CPAN block will, in general, work graciously with Unicode. The only problem I ever ran into was that I was incorrectly encoding data for output which had already been encoded by the framework, ending up with doubly encoded data.

About That Missing Character...

And now, on to resolving the mystery from the beginning of this article. The German text files were in Latin15 encoding, which is pretty similar to the Latin1 encoding Perl uses by default except for the Euro sign (and some other bits). Your script read the text in as if it was Latin1 encoded. As the data went via `DBI` into PostgreSQL, the `DBD::Pg` module took care of properly UTF-8 encoding the data, which worked fine except for the Euro sign, which is not in the Latin1 character set. The fix for the problem is simple, though: the text files have to be opened with the `:encoding(Latin15)` PerlIO layer and it all works.

Recap

The road to Perl Unicode bliss:

- Be aware that Perl internally treats everything as Unicode (and make sure to keep all text information encoded in UTF-8 with the UTF-8 flag set to avoid the Unicode Bug).
- Whenever data enters the program from the outside, *decode* it from its outside encoding.
- When data leaves the program, *encode* it according to the requirements of the next step of processing.
- Consider that the modules you are using to access data might already be taking care of all (or part) of the encoding and decoding business.

For further entertainment have a look at the Perl Unicode documentation on http://perldoc.perl.org/.

Data Integrity

Finding Truth in a World of Guesses and Lies

DOUG HUGHES

Doug Hughes is the manager for the infrastructure team at D. E. Shaw Research, LLC, in Manhattan. He is a past LOPSA board member and was the LISA '11 conference co-chair. Doug fell into system administration accidentally, after acquiring a BE in Computer Engineering, and decided that it suited him.
doug@will.to

At LISA '11 in Boston, as I was sitting in a talk [1] on GPFS by Veera Deenadhayalan of IBM, I saw something that I instinctively knew was incorrect. It wasn't anything fundamental to his talk. To the contrary, the work that IBM is doing on GPFS is quite impressive and one of the reasons we had them come to give this talk. There are two main, salutary features of upcoming GPFS versions coming out of the IBM Almaden Research Center. The first is the de-clustering of RAID stripes from full disks, which, to be brief, allows very fast rebuilding of stripes of data across 100,000+ disk systems, where the expectation is that a RAID rebuild will be happening every eight hours or so. The second, and the focus of my article here, is the integration of superior integrity checks built into file systems.

This is nothing new, right? ZFS has been doing this for years. This is definitely worthwhile work, and I'm extremely glad to see this being integrated into other file systems, natively. The thing that struck me was about 33 minutes into the talk on slide 26. It was a reference to the paper "Evaluating the Impact of Undetected Disk Errors in RAID Systems" [2] published in 2009 in the IEEE International Conference on Dependable Systems. This publication clearly nailed the problem, but models indicated that a 1000-disk system would experience an undetected corruption every five years. This is where my mind jumped the rails a little bit, but not for the reasons you might think. I have practical experience that shows that this is extremely optimistic, and my own recorded failure rate is *much* higher than this! I mentioned this to Veera and he requested that I publish these results; I agreed.

The Problem

But first, it may be necessary to take a step back. What is an undetected error, how does one catch it, and why is this a problem? You are probably more familiar with the lengths to which most vendors will go to tell you how safe the data is that they are writing to disk. They will quote MTBF or MTTDL [3] numbers, call your attention to scrubbers for bad data, etc. But, from a data integrity perspective, the question is this: how do you know that the data that you are reading back is what you wrote? That's the crux of the matter. Verifying correct writes at the time of the write is obviously important, but equally so, or perhaps even more important in some cases, is ensuring that you are reading back the correct data.

Disks Lie

Do you trust that your disks are returning to you what you wrote? When you read back a file, there are a panoply of things intermediating between you and your data. First, you have to get the data off the disk. Disks are incredibly complicated bits of

near black-magic. It's amazing that they even work at all. They have little electromagnetic heads floating on air cushion mere microns above a rapidly spinning surface of mirror-polished rust—a 7200 RPM 3.5″ disk is moving at about 100 km/h at the outer edge. They have to be at exactly the right place at the right time to read off little chunks of data that are probabilistically encoded and decoded via ultra-fine magnetic fields at multiple depths using quantum effects [4]. The controller collects all of this analog data, analyzes it on the fly, accounts for surface expansion and contraction because of thermal effects, aligns the heads precisely, and uses complex error correction codes to reconstruct the data and turn it into streams of 0's and 1's. What could possibly go wrong!? When your disks lie to you, and they probably have already, do you know?

There are many things that could happen. You could write the data but, because of head alignment or a firmware issue, the data that you wrote might turn into a big chunk of 0's. This is one of the reasons that GPFS writes the data and the checksum to totally different disks. If the data and checksum were written to the same place, and these blocks erroneously became 0's, when you read it back the checksum would match! The checksum or parity would be correct as far as what is on the disk, but it would be wrong with respect to what you intended to be recorded (unless, of course, you are in the habit of storing 0's).

Another possibility is that the magnetic field degrades to such a point that the algorithms used to reconstruct the data guess incorrectly. After all, the retrieved data is an evaluation based upon mathematical best guesses based upon congruent magnetic fields. Add to this the possibility of firmware bugs and you've got a lot of potential for something to happen.

But disks are not the only ant in the colony. Connecting the disk controller to the rest of the system is a cable, which connects to a board with integrated circuits, which passes it over a bus, which passes the data through memory (usually) on its way through several levels of cache through the CPU to the operating system, which has a driver for the disk and a file system to aggregate multiple disks and which uses its own memory and usually passes through the CPU/memory systems several times on its way to a user program, which usually resides on multiple systems (whew). Sometimes there is also a RAID controller card with an ASIC, FPGA, or CPU involved. It just so happens, out of all of these, the disk is complicated and techno-magical enough that it is the most frequent source of errors.

What about client machines over the network? Some file systems, such as GPFS, include network clients natively as an option. Others use the more common NFS or AFS for remote access. These are beyond the scope of this article, for various reasons.

How ZFS helps

In the remainder of this article, I'm going to illustrate the problem using ZFS. ZFS has the advantage of telling you about prevarication up to and through the file system. ZFS verifies the integrity of everything that it touches, including the CPU, memory, cables, down to the disks. Since ZFS reports these problems in the form of easily accessible checksum errors, I can easily share them with you. ZFS is freely available. ZFS uses very strong checksums and verifies every single checksum on read, so you know when there is a phantom flip in any subsystem. It may not know exactly where the flip occurred. That would be hard. But it does associate the error with the disk holding that block, even though it may not be the disk that is at fault. We'll get to that in a little bit.

But, you ask, if ZFS detects it, how can it be undetected? It is because the disk (which I'll use as shorthand for all of the various components connecting the disk to the file system) did not detect that it was bad. It thinks that the data that it is returning is perfectly fine. This is where "undetected" comes from. It is the hardware's inability to realize that the equivalent of mischievous gremlins have been hopping through the data fields kicking over the bit grains without triggering any errors. This last part is important. There may be no other error! The head is fine, the disk platters may be totally fine. There are no timeouts, no bad blocks, etc. These physical media errors have been around for a very long time and systems already know how to deal with these fairly well. (Yet, somehow the exact failure states of disks, firmware issues, bus timeouts, and other ephemera still manage to torpedo us even after all of these years.)

I have been able to collect checksum failure data on a population of about 1000 disks over the course of a couple of years, and hopefully you will gain some appreciation that the scope of the problem is worse than the IBM people thought when they designed GPFS. Serendipitously, my population of ~1000 disks matches the prediction pool from the research paper mentioned earlier. A good detection and correction strategy is a shield from the bit gremlins.

Interpreting the Data

A normal zpool status output for the generic pool zpool1 made of a two-way redundant stripe (raidz2 in ZFS parlance, equivalent to RAID6) looks like this:

```
# zpool status zpool1
pool: zpool1
state: ONLINE
scrub: none requested
config:

NAME        STATE     READ      WRITE     CKSUM
zpool1      ONLINE    0         0         0
raidz2-0    ONLINE    0         0         0
c1t0d0s0    ONLINE    0         0         0
c2t0d0s0    ONLINE    0         0         0
c3t0d0s0    ONLINE    0         0         0
c5t7d0s0    ONLINE    0         0         0
c4t0d0s0    ONLINE    0         0         0
```

There are no read errors, no write errors, and no checksum errors. The checksum error is conveniently stored in the last column for each disk in the pool and all of the errors are tracked separately. Read and write errors are what you might expect: head alignment, media error, controller failure, bus timeout, etc. Most of the read and write errors are also visible to the system via normal error reporting: iostat –E, /var/adm/messages, and the Solaris (if using that) event manager. Checksum errors are as described earlier; the data read back from the disk does not match the checksum that was stored when the data was written. This could be because the checksum block is bad or because the data block is bad, but either way, ZFS flags this as an error in the CKSUM column and automatically corrects the data, if possible, by reconstructing from parity. This is the part that makes ZFS so integral to integrity. For the sake of space, I'll be stripping the headers and other extraneous rows for the following illustrations.

APR 7, 2011

```
c5t3d0 ONLINE 0 0 1
```

We had one checksum error on c5t3d0. Was it the disk? Was it an undetected ECC flip? Was it a bit flip on a bus somewhere? We don't really know. We wait for further flips and keep the disk under observation. But, had we not had ZFS here, bad data would have been returned to the program—that is certain.

OCT 8, 2010

```
c0t6d0  ONLINE     0    0    2  5K resilvered
```

Two checksum errors! We can reasonably conclude that this is the disk, because in a many-disk system other subsystem errors would be randomly spread around to other disks. Not only that, but ZFS informs us that it has resilvered the bad data to other disks for us. It's very polite and helpful. Resilvering is the name for the process by which all of the proper data is reconstructed onto the new disk to repair the raidz2 stripe, an homage to repairing old glass mirrors.

SEP 2, 2010

```
raidz2                  DEGRADED     0     0      0
   c5t5d0s0             ONLINE       0     0      0
   c0t6d0s0             ONLINE       0     0      0
   spare                DEGRADED     0     0  1.21M
     replacing          DEGRADED     0     0      0
       c1t6d0s0/old     FAULTED      0     0      0    corrupted data
       c1t6d0           ONLINE       0     0      0    2.95T resilvered
     c4t7d0             ONLINE       0     0      0    2.95T resilvered
   c2t6d0s0             ONLINE       0     0     20
   c3t6d0s0             ONLINE       0     0     20
```

We have a faulted disk, c1t6d0, that has been replaced, and a spare disk, c4t7d0, that has been swapped in. c1t6d0s0/old represents the original failed disk. c1t6d0 and c4t7d0 are mirrored during the sparing process. We also see 20 checksum errors each on c2t6d0s0 and c3t6d0s0. This is odd, particularly the exact equivalence, and most likely correlated with transient controller issues when the original disk failed, but ZFS was able to correct them on the fly. We chose to ignore these errors and clear them to see if there were any other issues after the resilvering was complete. There were none.

APR 5, 2010

```
c7t0d0s0          ONLINE     0     0      0
  spare           ONLINE   220     0    212
    c7t1d0s0      ONLINE    25     2    212
    c7t7d0s0      ONLINE     0     0    432    373G resilvered
```

That's a lot of checksum errors, above (April 5, 2010)! But also read (25) and write (2) errors. You can see that c7t1d0s0 was erroring all over the place and likely had fairly severe media defects, perhaps a head dip, a particle of dust, or a random manufacturing defect. c7t7d0s0 was hot-swapped into place and experienced 432 checksum errors while resilvering. /var/adm/messages was quite popular that day. Fortunately, the errors were corrected on the fly. That would have been a data

retrieval nightmare if they had been passed to the user program. It turns out that this resilver got stuck and that the replacement disk had an issue. We replaced the replacement and things worked much better.

DEC 15, 2011

```
        c1t4d0    ONLINE  0       0       1
```

Another single corrected error while I write this article. This disk is under observation for further errors.

Conclusion

Of the events recorded above, we have five separate events in an 18-month period. I think there are probably a few others that I was not able to find. Based upon rough memory, it seems that we have a corrected checksum error every six months or so. It would not be an over-stretch to call this 10 otherwise undetected bit flip events in five years, if we aggregate the co-temporal events. However, if we use the raw numbers, the actual number of bit/block corruptions is much higher than this (432 in one event alone)! The more conservative number is 10x the study projection, and I would consider our overall disk reliability to have been pretty good over the last four years.

Hopefully, I haven't frightened you too much with my tales of doom and gloom. There are other ways that you can protect your data without ZFS, such as keeping md5 checksums [5] on every file in the system. One thing that you should do is demand that your storage vendor, who is implementing RAID6 [6], check all reads and verify the parity on every block. This is relatively easy for them to do. It's slightly more overhead to read the blocks off the two parity disks and calculate the codes, but it helps to verify the data is correct: ASICs and CPUs are fast and inexpensive. Parity isn't as good as either the GPFS or ZFS checksum, which is verified on the main CPU, but it's better than nothing.

Many vendors will argue that they have strong guarantees that the data is written correctly. This is not enough! However, in some cases (images and videos come to mind), the bit flip phenomenon is inconsequential. Who cares if a pixel changes color in a movie in a frame? You can make your own judgment about your tolerance for data integrity and pick a solution that is appropriate for your enterprise.

References

[1] IBM talk on improvements to GPFS: http://www.youtube.com/watch?v=2g5rx4gP6yU (or the LISA '11 Web site).

[2] Rozier et al., "Evaluating the Impact of Undetected Disk Errors in RAID Systems": https://www.perform.csl.illinois.edu/Papers/USAN_papers/09ROZ01.pdf.

[3] Richard Elling, "A Story of Two MTTDL Models": http://blogs.oracle.com/relling/entry/a_story_of_two_mttdl.

[4] http://en.wikipedia.org/wiki/Giant_magnetoresistance.

[5] Andrew Hume, "How's Your OS These Days?" ;login:, vol. 30, no. 3, June 2005, USENIX: https://www.usenix.org/publications/login/june-2005-volume-30-number-3/hows-your-os-these-days.

[6] Robin Harris, "Why RAID5 Stops Working in 2009," July 2007: http://www.zdnet.com/blog/storage/why-raid-5-stops-working-in-2009/162.

Practical Perl Tools

Warning! Warning! Danger, Will Robinson!

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 late January, Dick Tufeld, the voice talent behind The Robot in *Lost in Space* (and many other fine shows, *Thundarr the Barbarian* notwithstanding) died at age 85. I thought I would use the immortal phrase he spoke from that TV show as inspiration for this issue's column and dedicate it to Tufeld's memory. For this column, we're going to look at a number of ways you can (and perhaps should) handle warnings and errors within your Perl programs.

When approaching a Perl question, my first inclination is to review what is available in the language directly and what ships with the distribution (i.e., is "in core") before looking elsewhere for solutions. With that in mind, let's start with the two built-in functions for signaling that an error has occurred:

```
warn()
die()
```

The former lets you print an error message while letting the program continue, and the latter does the same but under most circumstances (more on this later) terminates the program run. Both take as an argument a list used for printing the error message. In most cases, that list contains a simple scalar, as in:

```
open my $FILE, '<', 'ookla' or die "Couldn't open the file ookla:$!";
open my $FILE, '<', 'ookla' or die "Couldn't open the file ookla:$!\n";
```

In both of these examples, I've used the common convention of including the $! variable in the error message. That variable gets set by Perl (to quote the docs) with "the current value of the C 'errno' variable, or in other words, if a system or library call fails, it sets this variable." A slightly more subtle difference between the two examples is whether the error message passed to die() or warn() ends in a newline ("\n"). If it does, the message is printed as written. If it doesn't, Perl appends the script name and line number where the error was encountered in your program to the message:

```
"Couldn't open the file ookla: no such file at column.pl line 1920"
```

The Perl documentation for die() recommends adding ", stopped" to your message if you plan to leave off the newline. The error message then would be:

```
"Couldn't open the file ookla: no such file, stopped at column.pl line 1920"
```

For most of your small-scale programming tasks, warn() and die() will do what you need. But when you start to write longer, more compartmentalized programs, you may find at least one of their standard properties may be less useful to you. Let

me show you a small example of what I mean. Let's say I'm writing a program that consists of a module I wrote and a script that calls the module. Here's the super simple module:

```
# module.pm
sub feel_better {

        open my $DATA, '<', 'meditation_of_the_day' or
          die "Can't open my data file: $!";

        print while (<$DATA>);

        close($DATA) or die "Unable to close data file:$!";
}
1;
```

and an equally simple script to call it:

```
# caller.pl
use module;

feel_better();
```

Let's run caller.pl without the data file module.pm needs. When we do so, it complains thusly:

```
Can't open my data file: No such file or directory at module.pm line 3.
```

If we were trying to debug this situation, we'd be a little stuck. We'd know that something in the module included by caller.pl reported an error, but we have no indication just where in caller.pl that module had been invoked. What if caller.pl was much longer and it called feel_better() in 20 different places in the code? How would we know which call failed? This is where the Carp module comes into play. Carp is a module shipped with Perl since version 5.00 was first released. It provides the following alternative error message routines:

```
carp()
croak()
confess()
cluck()
```

The first two, carp() and croak(), work just like warn() and die(), respectively, but instead of adding location information to the message when called without a newline based on where the error was encountered, they add it based on where the erroring code was called. So, if I change module.pm to read:

```
use Carp;
sub feel_better {

        open my $DATA, '<', 'meditation_of_the_day' or
          croak "Can't open my data file: $!";
...
```

the error message it throws in the absence of a data file is:

```
Can't open my data file: No such file or directory at module.pm line 4
        main::feel_better() called at caller.pl line 5
```

Now we know not only where the error occurred, but also have an idea of just which code got us to that point. If we wanted even more information, let's say because

we are trying to track down a situation where our program called code that called something else, we could use confess() and cluck(). The confess() call is like die()/croak() except that it will produce a full stack trace as it shuffles off this mortal coil. cluck() does a similar thing but, like warn(), it lets the program continue to run.

Before we move on to the next section, I think it is worth mentioning a number of riffs on the Croak module that are available. Most of these take the idea in a direction that makes them even more useful. For example, there are a number of modules that redirect the output of carp() in some way. The first is one I've used to great effect in the past: CGI::Carp. CGI::Carp modifies the croak()/confess()/carp() trio so that the Web server's error logs will contain useful error messages from your script. It also has a special subroutine used like this:

```
use CGI::Carp qw(fatalsToBrowser);
die "Can't open my data file: $!";
```

This subroutine will send that error message to the browser of the person encountering the message instead of sending it to the error log. The fatalsToBrowser subroutine handles all the magic necessary to make sure HTTP headers are sent before the error message. There is a related subroutine available in CGI::Carp called warningsToBrowser which will let you send the output of any warn() calls to the browser as HTML comments (so they are not seen by the user, but are still available for perusal if desired). Both of these routines are really useful while debugging a Web application during the development process.

Another Carp redirection module that tries even harder to let you direct error messages in a useful fashion is Carp::Notify. Carp::Notify can actually email you the error messages. It will also let you designate certain global variables in your program as "storable," meaning you want it to tell you the value of all of those variables when it is reporting an error. It is very helpful to have the runtime context of an error at hand when examining an error report.

If you like that last idea, you are really going to like Carp::REPL. Carp::REPL will modify your die() (and, optionally, warn()) calls so that when you call die(), instead of having the program quit, you will find yourself in an interactive Perl session. In this session you can poke and prod at the current state of the program to determine just where and how things went awry.

Death Be Proud?

Some people prefer a programming style where Perl functions and their connected system calls automatically throw errors if they fail. Instead of having to write:

```
open my $DATA, '<', 'meditation_of_the_day' or
       die "Can't open my data file: $!";
 ...
close($DATA) or die "Unable to close data file:$!";
```

they prefer to write:

```
use autodie qw(open close);
open my $DATA, '<', 'meditation_of_the_day';
 ...
close($DATA);
```

safe in the notion that if open() or close() calls fail in some way, that will cause the program to stop. The plus of this approach is your code looks a little cleaner because it isn't littered with tests for success/failure every step of the way. The autodie distribution also contains a Fatal module, which is how people pulled this trick off before autodie came along. It is worthwhile reading the docs for both modules so you will be aware of their individual gotchas.

This isn't my preferred programming style, but there is one context where I do make use of a similar behavior: DBI programming. If you open a connection to a database as follows:

```
$dbh = DBI->connect("DBI:mysql:$database",
                    $username, $pw,{RaiseError => 1});
```

DBI will automatically call die() if the connect() or other subsequent DBI calls return an error. There's an equivalent PrintError parameter that performs a warn() instead of a die() if that is more to your liking.

Death Be Not Proud

The flip side of the previous topic is that sometimes when your program encounters a "fatal" error, à la die(), you may have a different idea for what the program should do at that moment. If you are able to write code that can recover from an error like this, you need a way to catch the error and proceed from that point with your recovery. The usual way to do this is to wrap the operation that could potentially fail in an eval()/eval{} block, as in this example from the DBI docs:

```
eval {

        ...
        $sth->execute();
        ...
};
if ($@) {
        # $sth->err and $DBI::err will be true if error was from DBI
        warn $@; # print the error
}
```

If the execute() fails, the eval() will trap its failure and set $@ accordingly. Eval() also gets used like this when you want to test for a potentially fatal condition, e.g., if a module you plan to use isn't available:

```
eval ('use Mondok;');
warn "Mondok module not available on this machine, skipping..." if ($@);
```

But eval() has a few issues (especially before Perl version 5.14, where a key one was addressed). These issues mostly surround $@, which can be cleared, clobbered, and generally messed with in ways that don't necessarily make it a reliable semaphore. The easiest way to get around these problems is to use a module called Try::Tiny (whose docs contain a lovely litany of eval() complaints). Try::Tiny lets you write code of this form:

```
try { something } catch { the results } finally { perform some clean up }
```

Real Perl code using Try::Tiny would look like this:

```
try { this_might_die() }
catch { warn "had a problem: $_";}
finally { $error_count++ if ($@); }
```

If you like the try-catch-finally construct—perhaps you miss it from other programming languages—you might want to check out Try::Tiny's more powerful sibling, TryCatch. TryCatch has a larger list of dependencies, but it lets you do things like put "return()" in your catch blocks, something Try::Tiny by itself cannot do. To get that specific functionality from Try::Tiny you would need to add another module, Error::Return.

But Maybe What You Really Want Is an Object

We're almost out of time for this column, but I wanted to at least mention one other approach to handling errors that you may wish to explore if you are writing larger and more extensive OOP-based programs. There's a good argument to be made for passing around exception objects instead of the simple scalar error messages we've seen throughout this column. With an exception object, you can define a persistent interface for reporting back more detail on errors. This means that instead of having to parse:

```
"Ran out of memory: 20k"
```

and then rewrite how you parse it when you decide the message should be:

```
"FATAL: memory exceeded max allocated by 20k"
```

you might want to use an object that can be queried for the error message and the amount of memory separately. Here's some example code:

```
use Exception::Class ( 'ColumnException' => { fields => ['memory_used'] } );

# we wrap this in an eval because throw() does a die() with the given object
eval {ColumnException->throw(
       error=> 'FATAL: memory exceeded max allocated',
       memory_used => '20000'
)};

my $e  = Exception::Class->caught('ColumnException');
print $e->error,": ";
print $e->memory_used,"\n";
```

There are a few modules that help make creating exception objects pretty painless. I'd recommend you look at Exception::Class (used in the previous example) or Exception::Base. If you decide to use the latter, you may want to check out Exception::Died, because it will automatically hook all of the die() calls in your program and cause them to return exception objects by default.

With that, we have to bring this column to an end. Thank you Mr. Tufeld for making my TV-watching days richer with your voice. Take care, and I'll see you next time.

“R” is for Replacement

DAVID BEAZLEY

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

As Python programmers know, there has always been a “batteries included” philosophy when it comes to Python’s standard library. For instance, if you simply download Python and install it, you instantly get access to hundreds of modules ranging from XML parsing to reading and writing WAV files.

The standard library is both a blessing and a curse. Because of it, many programmers find they can simply install Python and have it work well enough for their purposes. At the same time, reliance on the library and concerns about backwards compatibility tend to give it a certain amount of inertia. It is sometimes difficult to push for changes and improvements to existing modules. This is especially true if one tries to challenge the dominance of standard library modules for extremely common tasks such as regular expression parsing or network programming.

In this article, I’m going to take a brief tour through two third-party libraries, requests and regex, that have generated a bit of buzz in the Python world by aiming to replace long-standing and widely used standard library modules. Both have generated buzz in the Python world and, coincidentally, both start with the letter “R.”

Interacting with the Web

Python has long included a module, urllib, that gives you simple access to the Web. For example, if you want to download and print out the street address of every bike rack in the city of Chicago, you can write code like this:

```
import urllib
u = urllib.urlopen("http://data.cityofchicago.org/api/views/cbyb-69xx/rows.
csv")
for line in u:
        fields = line.split(",")
        print fields[1]
```

This works fine if all you want to do is pull down a simple document and read it. However, as you know, the Web is a complicated place. If you need to do almost anything else, such as supply custom HTTP headers, provide form data, upload files, perform authentication, or deal with cookies, you’re out of luck.

Some limitations of urllib are addressed by another standard library, creatively named urllib2. However, if you’ve ever used urllib2 you know that it feels “over engineered” and that seemingly simple tasks like authentication can be tricky. To give you some idea, here is a fragment of code that shows how you would initiate a basic authentication login to the Python Package Index (http://pypi.python.org).

```
import urllib2
auth = urllib2.HTTPBasicAuthHandler()
auth.add_password("pypi","http://pypi.python.org","username","password")
opener = urllib2.build_opener(auth)

r = urllib2.Request("http://pypi.python.org/pypi?:action=login")
u = opener.open(r)
resp = u.read()

# From here. You can access more pages
```

As you can see, the process has become quite a bit more complicated. It gets far more convoluted, if not practically impossible, if you want to do anything more advanced, such as invoke other HTTP methods (e.g., HEAD, PUT, DELETE, etc.), upload files, or read streaming data.

Although you could continue to hack away on urllib2 to try to make it do what you want, you might be better off looking at Kenneth Reitz's requests library instead (http://pypi.python.org/pypi/requests). Rather than trying to emulate existing functionality, requests provides an entirely different programming interface.

First, let's just download a simple Web page:

```
>>> import requests
>>> r = requests.get("http://www.python.org")
>>> r.status_code
200
>>> r.headers
{'last-modified': 'Fri, 27 Jan 2012 15:49:35 GMT',
  'content-length': '18882',
  'etag': '"105800d-49c2-4b7847185c1c0"',
  'date': 'Sat, 28 Jan 2012 19:13:11 GMT',
  'accept-ranges': 'bytes',
  'content-type': 'text/html',
  'server': 'Apache/2.2.16 (Debian)'
}
>>> page = r.text
>>> page
u'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"...'
>>>
```

That was certainly easy enough. Although it looks easy, there are some subtle things going on under the covers. First, access to the r.text attribute automatically performs the appropriate Unicode decoding, returning a Unicode string. Thus, you don't have to worry about it. Similarly, you can easily obtain status code and header information as shown.

Now, let's make a HEAD request to see if the document has changed recently:

```
>>> r = requests.head("http://www.python.org")
>>> r.headers
{'last-modified': 'Fri, 27 Jan 2012 15:49:35 GMT',
  'content-length': '18882',
  'etag': '"105800d-49c2-4b7847185c1c0"',
  'date': 'Sat, 28 Jan 2012 19:23:08 GMT',  'accept-ranges': 'bytes',
  'content-type': 'text/html',
  'server': 'Apache/2.2.16 (Debian)'
}
```

```
>>> r.text
u''
>>>
```

That was also easy. Believe it or not, this is practically impossible to do using urllib or urllib2, since they don't provide an interface for changing the HTTP method.

Let's look at an example of authentication. This example shows how to log in to the Python Package Index using basic authentication as shown earlier:

```
import requests
r = requests.get("http://pypi.python.org/pypi?:action=login",
                      auth=("user","password"))
resp = r.text
```

If you wanted to know whether any cookies were set on a page, simply access the cookies attribute:

```
>>> r = requests.get("http://pypi.python.org/pypi?:action=login",auth=("user
","password"))
>>> r.cookies
{'pypi': '0bf0722dee2203ee3accf4fef9650b2f'}
>>>
```

To pass cookies back on subsequent requests, simply supply the r.cookies dictionary as an argument:

```
>>> r2 = requests.get(newurl, cookies=r.cookies)
>>>
```

Let's write a request that reads real-time data from Twitter's streaming API for anything that mentions the word "python". You'll need to supply your own username and password for this:

```
import requests
import sys
import json
url = "https://stream.twitter.com/1/statuses/filter.json"
parms = {
    'track' : 'python',
    }

auth = ('username','password')
r = requests.post(url, data=parms, auth=auth)
for line in r.iter_lines():
    if line:
        print json.loads(line)
```

Here, requests will open a connection and simply feed you a stream of lines as they are produced. Again, it's relatively straightforward. However, don't even try it with urllib2. There is far more that you can do with requests, but this should have given you a small taste for it.

Regular Expression Pattern Matching

The standard library module for handling regular expression parsing is re. If I recall correctly, it is the second implementation of regular expressions, first appearing about 14 years ago in Python 2.0. However, just when you thought re might be the last word in Python regular expression handling, a new library called regex

has appeared. regex is the work of Matthew Barnett and has recently been officially blessed for inclusion in the standard library starting with Python 3.3 (not yet released). However, you can use it now if you simply download it from http://pypi.python.org/pypi/regex. (Editor's Note: You may not be able to install regex on top of versions of Python older than 2.6.4.)

regex is a drop-in replacement for the standard re library. Thus, any regex matching code that you might have written before should still work. An easy way to try regex without making too many changes is to simply change the import statement as follows:

```
import regex as re

# Use re library as before
...
```

The new regex library fixes a huge number of issues, annoyances, and bugs related to the old re library. These include various convenience features, such as showing you the pattern when pattern objects are inspected or printed, as here:

```
>>> pat = regex.compile("[a-zA-Z_][a-zA-Z0-9_]+")
>>> pat
regex.Regex('[a-zA-Z_][a-zA-Z0-9_]+', flags=regex.V0)
>>>
```

You also get a much simplified way to refer to capture groups.

```
>>> pat = regex.compile(r"(\d+)/(\d+)/(\d+)")
>>> m = pat.match("1/29/2012")
>>> m[0]
'1/29/2012'
>>> m[1]
'1'
>>> m[2]
'29'
>>> m[3]
'2012'
>>> m[1:]
('1', '29', '2012')
>>>
```

Behind the scenes, limitations related to the number of capture groups, concurrent operation with threads, and other matters are fixed, in addition to a number of tricky issues with Unicode (e.g., proper case folding).

However, besides subtle cosmetic and implementation improvements, regex offers an interesting range of new functionality. There are too many additions to cover in their entirety, but let's look at a few of the more interesting enhancements.

Suppose you wanted part of a regex to match a set of possible words or symbols. For example, suppose you wanted to match some of Python's math operators (+, -, *, /, and **). You might be inclined to write a regex like this:

```
import regex
pat = regex.compile(r'\*\*|\*|\+|-|/')
```

However, if you look at such a pattern, there are all sorts of tricky escapes (for * and +). Plus, you have to worry about matching in the right order (checking ** prior to *). Here is an alternate approach using named sets:

```
import regex
ops = { '+', '*', '-', '/', '**' }   # A set of everything you want to match
pat = regex.compile('\L<ops>', ops=ops)
```

In this version, you don't have to worry about escaping any of the possible matches or their order. You simply pass a set using the \L escape and specify an appropriate keyword argument (which contains a list or set of the literal symbols you want to match). regex will figure everything out, including the escaping and ordering, to make it work.

If you have written regular expressions, you probably know about the specification of character sets such as [a-zA-Z] or [^a-zA-Z]. regex takes this much further by allowing common set operations (union, intersection, difference), as well as an interface to the Unicode properties. Thus, you can start writing character set patterns like this:

```
# Match all letters except vowels
pat1 = regex.compile("[[a-z]--[aeiou]]+$",regex.V1)
pat1.match("xyzzy")     # Matches
pat1.match("plugh")     # Doesn't match

# Match any non-ascii character
pat2 = regex.compile(r"[^\p{ASCII}]+",regex.V1)
pat2.search(u"That's a spicy jalape\u00f1o")  # Matches
pat2.search(u"I want another torta")          # No matches
```

Finally, another interesting feature is support for fuzzy matching. This is a matching technique where text with errors in the form of insertions, deletions, or substitutions can be matched. Here is an example:

```
>>> pat = regex.compile("(?:spam){s<=1}")
>>>
```

This regex pattern specifies that the text "spam" is to be matched, but that, at most, one letter substitution is allowed. Here's what happens:

```
>>> pat.match("spam")       # Exact match
<_regex.Match object at 0x100547850>
>>> pat.match("slam")       # 1-letter substituted (match)
<_regex.Match object at 0x1005478b8>
>>> pat.match("slum")       # 2-letters substituted (no match)
>>>
```

There are additional options to specify insertions and deletions. For example, here is a pattern than allows, at most, one deletion, one substitution, and one insertion:

```
>>> pat = regex.compile("(?:spam){s<=1,d<=1,i<=1}$")
>>> pat.match("spm")      # Matches. 1 deletion
<_regex.Match object at 0x1005478b8>
>>> pat.match("sm")       # No match. 2 deletions
>>> pat.match("spaam")    # Match. 1 insertion
<_regex.Match object at 0x100547850>
>>> pat.match("slamm")    # Match. 1 insertion, 1 substitution
<_regex.Match object at 0x1005478b8>
>>> pat.match("slum")     # Match. 1 deletion, 1 insertion, 1 substition
```

Each insertion, substition, or deletion is counted separately and can be combined to match a wide range of words. If you wanted to narrow it down, you could just put a limit on the number of combined errors. For example:

```
>>> pat = regex.compile("(?:spam){s,i,d,e<=1}")
>>> pat2.match("spam")      # Match exact
<_regex.Match object at 0x100547850>
>>> pat2.match("spm")       # Match, 1 deletion
<_regex.Match object at 0x1005478b8>
>>> pat2.match("spaam")     # Match, 1 insertion
<_regex.Match object at 0x100547850>
>>> pat2.match("slum")      # No match
>>>
```

Needless to say, fuzzy matching opens up new areas of possible application to regular expression matching.

Putting It All Together

As a final example, it is now possible to present a short script that tries to identify people drunk-tweeting from the city of Chicago:

```
# drunktweet.py
'''
Print out possible drunk tweets from the city of Chicago.
'''

import regex
import requests
import json

# Terms for being "wasted"
terms = { 'drunk','wasted','buzzed','hammered','plastered' }

# A fuzzy regex for people who can't type
pat = regex.compile(r"(?:\L<terms>){i,d,s,e<=1}$", regex.I, terms=terms)

# Connect to the Twitter streaming API
url   = "https://stream.twitter.com/1/statuses/filter.json"
parms = {
   'locations' : "-87.94,41.644,-87.523,42.023"   # Chicago
   }
auth  = ('username','password')
r = requests.post(url, data=parms, auth=auth)

# Print possible candidates
for line in r.iter_lines():
   if line:
      tweet = json.loads(line)
      status = tweet.get('text',u'')
      words = status.split()
      if any(pat.match(word) for word in words):
        print tweet['user']['screen_name'], status
```

Final Words

Although this article has only focused on two modules, there are many other efforts to improve upon the standard library (too many to list). In a future issue, I hope to discuss alternatives to some of the system libraries—especially use of the subprocess module. Stay tuned.

iVoyeur

Changing the Game, Part 3

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

During the riots in London (the most recent ones), a friend pointed out that the top-selling items at Amazon.co.uk (as measured by their own "movers and shakers" page) were baseball bats. Before that, if you were to give me a data dump of all of Amazon's sales data for the past whenever, I confess I'd probably be at a loss for what to do with it. I would know that the data set held fascinating economic and sociological truths, but I wouldn't know the questions to ask to tease them out off the top of my head.

Given, however, the baseball bat tidbit and the accompanying revelation that the second highest selling item was baseballs, my head virtually spins with conjecture. I imagine my own city aflame outside my hastily boarded up windows. The shouts of looters and the screams of car-alarms penetrate my makeshift barricade while I click the Next-Day Air option on my order of a Louisville Slugger. My heart goes out to those people, truly, but what *were* they thinking? What UPS-man in his right mind drives a delivery truck through hordes of looters? Were they really so proper that they felt the need to buy some balls to keep up appearances while their city burnt down around them? Were the recipients of these orders even in London? Perhaps it was the surrounding burgs making preparations just in case?

At any rate, that little bit of knowledge makes me look at the larger data set with new eyes. It invites me to explore possibilities that hadn't occurred to me before, and, for me at least, that's the way it goes with data analysis. My imagination needs a bit of a kick in the shins to get it going. In the example above, we took our inspiration from the data itself, but it's also possible that a better understanding of, and easier access to, a few analysis techniques could inspire us to look at our data in new ways. Herein lies what I feel is the most fundamental difference between Graphite and RRDtool. The latter provides the means to perform whatever math we wish on our data before we graph it, while the former gives us a bunch of statically defined analysis functions that we may apply as we graph it.

Normally, I would argue for RRDtool's flexibility, but, in practice, Graphite's predefined functions do a much better job of kicking me in the shins and arousing my curiosity. In this, the last article in my series on Graphite, I'd like to share a few of these analysis functions with you, and hopefully show you how Graphite has me thinking differently about my data.

We'll start with the exception to the rule, the one area where Graphite is arguably more flexible than RRDtool: derivatives and integrals. As you probably know, RRDtool wants you to categorize your data into one of several types, including

"GAUGE" and "COUNTER." This is so that RRDtool can make some underlying assumptions about the data you're storing. For example, the reason RRDtool gives you a "COUNTER" data type is so it can automatically compute the derivative from counter data. So if you have a metric such as the number of packets passed through an interface, RRDtool will automatically take the derivative of this counter and provide you packets per second.

In Graphite, you'll recall that all data is stored the same way RRDtool would store a "GAUGE" data type, which is to say, Graphite just stores the raw data. This means that if you want to compute packets per second from a counter metric, you need to apply the "derive()" function to the data when you graph it. If this seems counter-intuitive, well, it is at first. At least I thought so before I became familiar with the rest of the functions, but first things first.

Graphite functions may be applied in the Graphite GUI via the "Graph Data" button in the graph composer, but now that I've been using Graphite for a while, I find it more expedient to work with the URLs directly. This is both because I type better than I click, and because the function names in the documentation don't exactly match those in the GUI, so one avoids the need to hunt around in menus by simply typing them into the URLs. To do this, I first get some data into the graph in the graph composer, then right-click the graph itself and select "copy link location," and then I simply paste the URL into a new browser tab.

Functions apply in a C-like manner, as you would expect, and most of them accept multiple metrics and even wildcards in lieu of lists. For example, I can apply the derive function to "some.counter.data" like so:

```
&target=derive(some.counter.data)
```

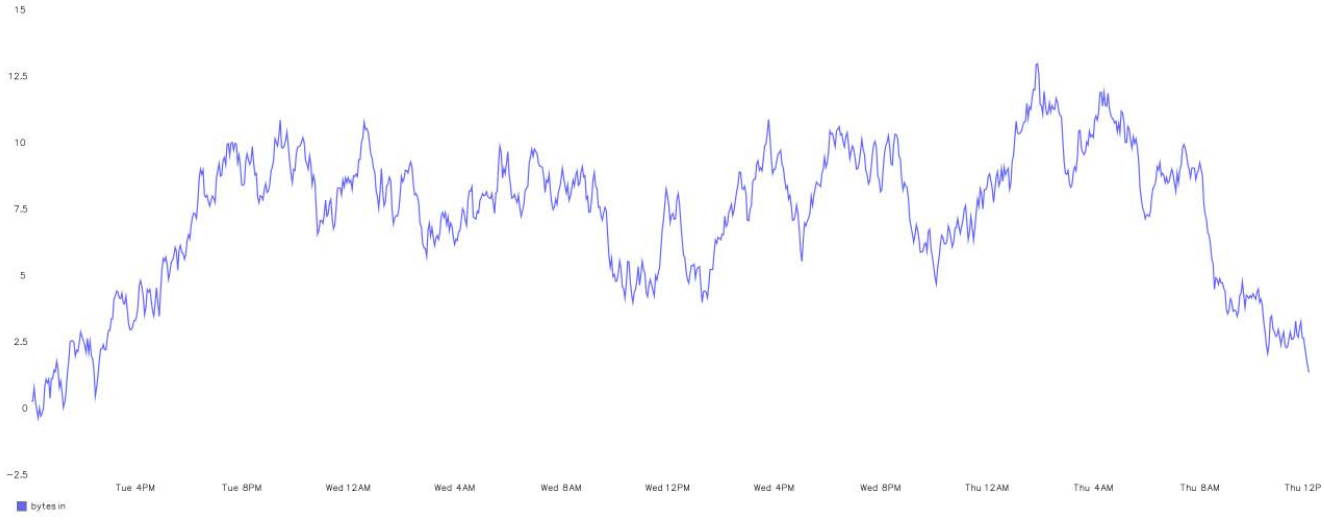

Figure 1: The derivative of a router's byte counter

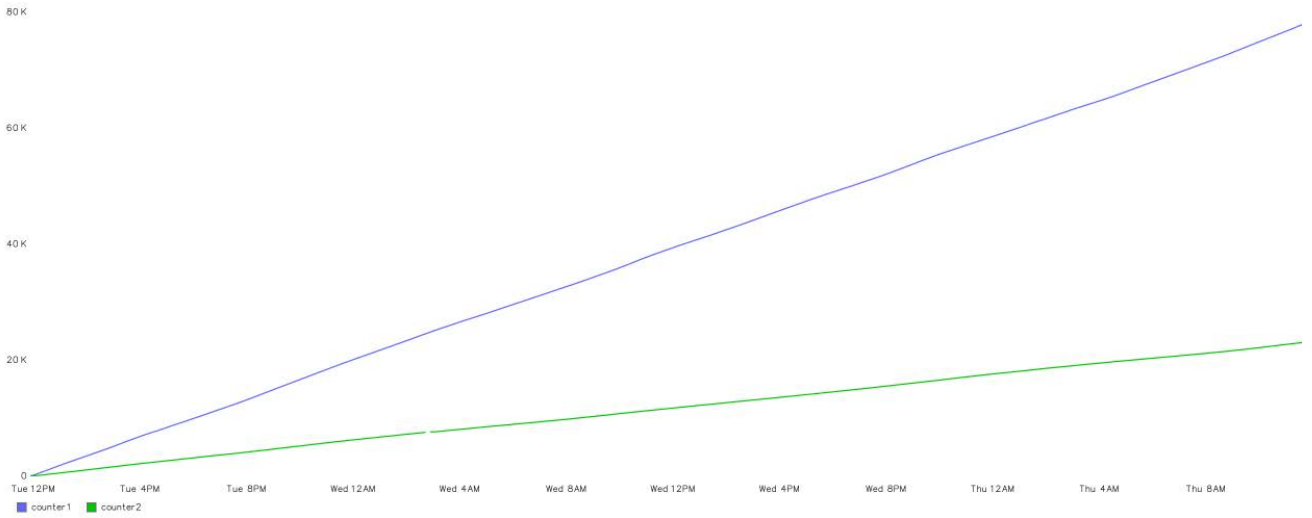

Figure 2: Raw byte count values from two routers

This function yields rate data as depicted in Figure 1, but that brings me to my first kick in the shins: namely, that sometimes looking at raw counter data is interesting. This wouldn't have occurred to me using RRDtool, but what if we compare the raw byte counters of two different routers, as seen in Figure 2? This could be useful capacity planning info, but it's not a fair comparison, because the routers have different total values, so one router will always appear to be growing at a smaller rate than the other. That's okay, Graphite provides us a "secondYAxis()" function, which easily allows us to draw one of these two data sets on its own Y-axis. So by graphing:

```
&target=router1.bytes&target=secondYAxis(router2.bytes)
```

we can get a clear picture of comparative rate of growth of the byte counters for these two routers, as seen in Figure 3. There's also an "integral()" function, which allows you to take GAUGE-based data sources and get counter-style data. If, for example you had a graph of widget sales per minute, you could apply the integral function to graph total sales for a given time interval.

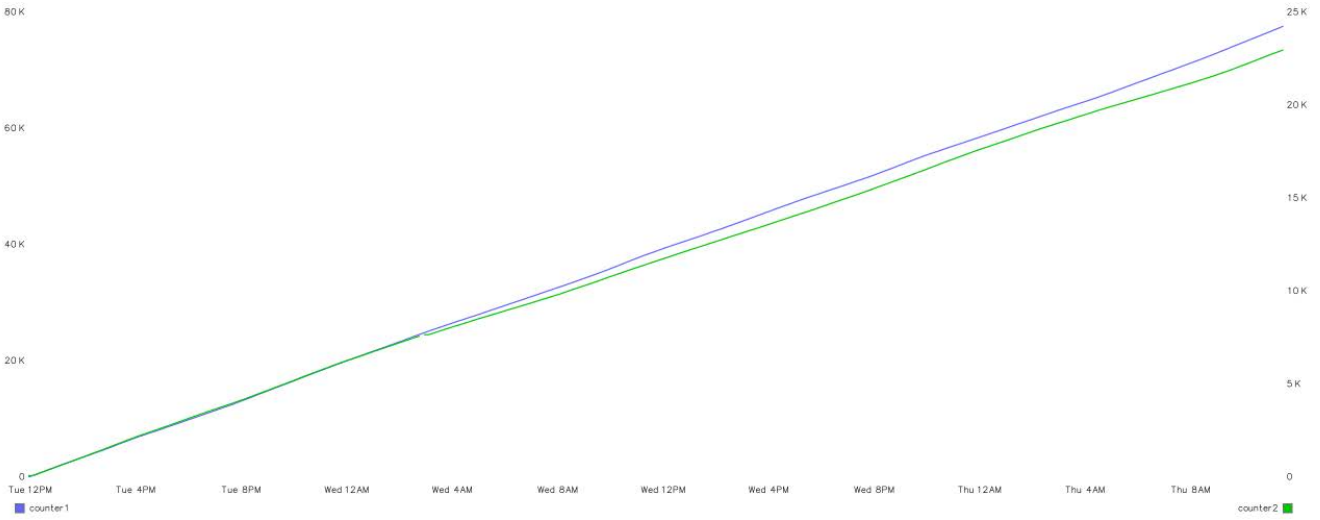

Figure 3: Raw byte counts from two routers compared with independent Y-axis

Now, if you're adept at RRDtool, take a moment and think about what it would have taken to "RRDtool graph" Figure 3, especially if you had been storing your counter data as type "COUNTER", as you should. It's probably possible, but I admit I don't know how to do it off the top of my head, and the idea of puzzling it out in Reverse Polish Notation somehow stops short of sounding appealing. Even if I did tease it out, I wouldn't be likely to apply the technique to other data sets for the benefit of my own curiosity, and the various RRDtool-based frontends out there wouldn't be much help to me in that endeavor. Graphite's functions invite me to visualize the data in new ways by virtue of their existence and accessibility. That's probably the biggest way Graphite has been a game-changer for me.

The functions themselves are fully documented at [1], and I can't cover all of them here, but let's take a look at some of my favorites, starting with "summarize()." This function allows you to re-compute the interval for a given set of time series data. So given a metric such as the number of users registering for an online service as depicted in Figure 4, we can imagine that the marketing team has a goal to maintain X registrations per hour and would like to display this data on a kiosk in the hallway, but they'll want it graphed as "registrations per hour" to reflect their goal. We can compute this graph, depicted in Figure 5, for them with:

```
&target=summarize(user.registrations,"1h")
```

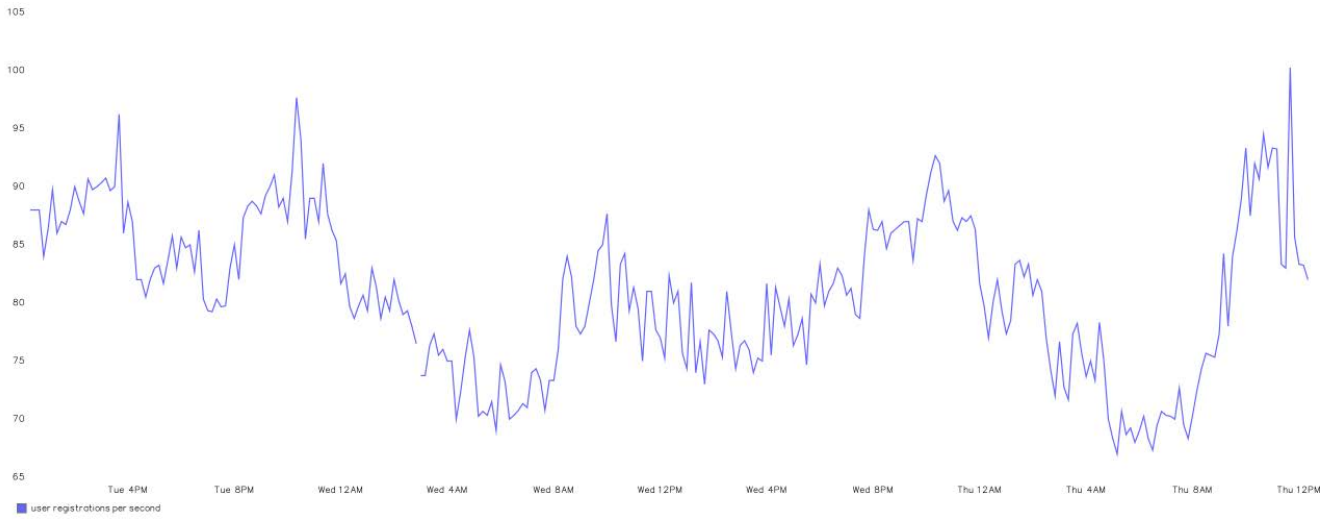

Figure 4: User registrations over time

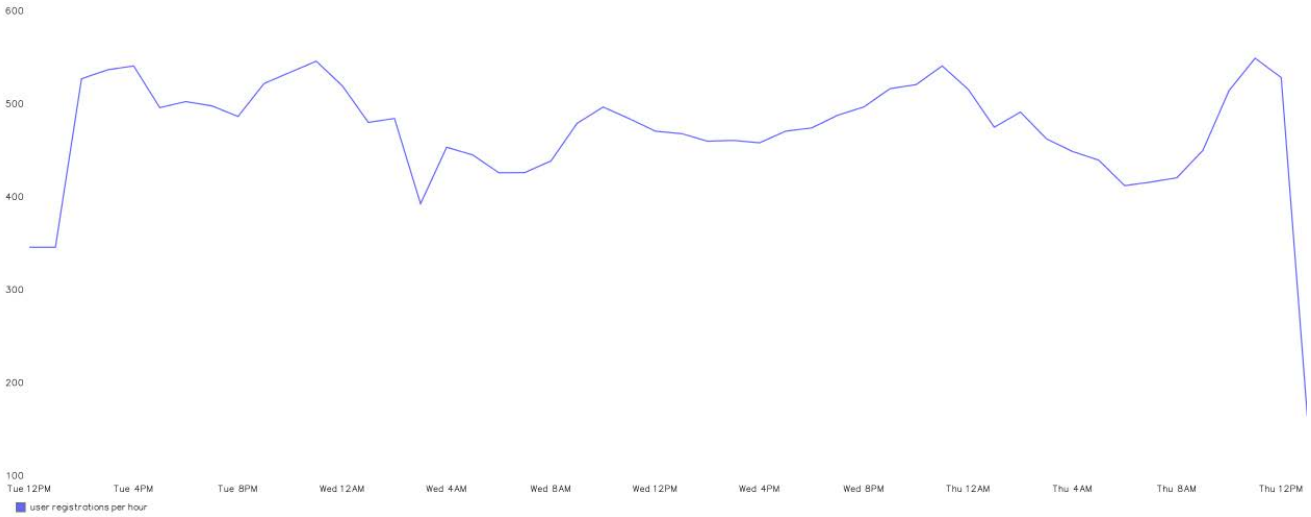

Figure 5: User registrations summarized hourly

To make their progress more obvious, we could add a horizontal line (constant) equal to their goal (Figure 6) with the "threshold()" function like so:

```
&target=summarize(user.registrations,"1h")&target=threshold(400,"Goal")
```

Figure 6: User registrations summarized hourly with a constant goal value

Functions are nestable, as in C, so we could add the data from last month to the graph by nesting the summarized target inside a "timeShift()" function. This would give the marketers some historical registration data from last month for context, while still maintaining a two-week period on the X-axis. This graph, drawn with the targets listed below, is depicted in Figure 7.

```
&target=summarize(user.registrations,"1h")&target=timeShift(summarize(user.
registrations,"1h"),"30d")&target=threshold(400,"Goal")
```

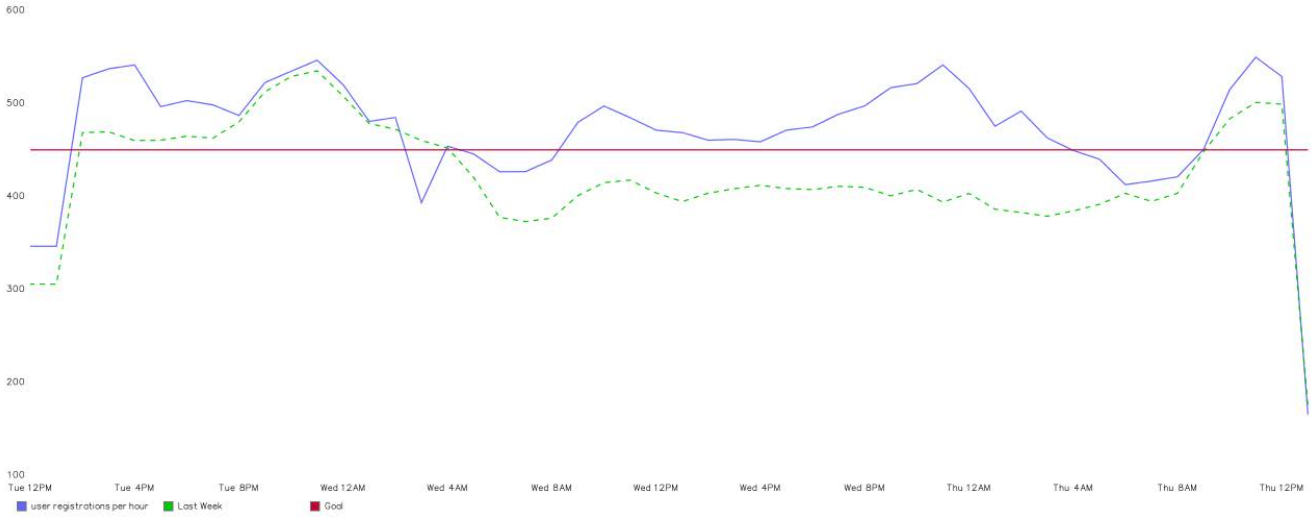

Figure 7: User registrations summarized hourly with a constant goal value and historical data

I really like the timeshift function. It's such an easy way to gain some context for almost any metric, and since I discovered it, every metric I graph seems to beg the question, "What was it doing last week at this time?" It's because of functions like this that Graphite feels more like an introspective tool and, by comparison, RRD-tool seems inflexible or perhaps even created for a different problem domain.

Various functions exist for combining multiple metrics into a single line: these are "sumSeries()," which creates a single line from multiple metrics by adding them together, "averageSeries()," which averages multiple metrics into a single metric, and "minSeries()" and "maxSeries()," which plot only the minimum or maximum value data points in the series. All of these functions support wildcards in the data-source field. For example:

```
&target=averageSeries(dc4.web.*.cpu)
```

plots a single line with the average CPU utilization of every Web server in dc4. Combinatorial functions are great for summarizing clusters or even datacenters. I find myself combining multiple averageSeries() of different metric types (CPU and disk, for example) using "secondAxis()." In this way I can get multiple metrics across entire datacenters on the same graph in a really usable way. Other functions exist for filtering individual metrics out of large lists. For example:

```
&target=highestCurrent(dc4.Web.*.cpu,5)
```

plots the CPU utilization of only the five currently most utilized Web servers in DC4. Combining these:

```
&target=averageSeries(highestCurrent(dc4.Web.*.cpu,5))
```

plots the average CPU utilization of the five currently most utilized Web servers in DC4. These are awesome for dashboards where you're just wanting to show things that are misbehaving, or aberrant behavior in general. I'm sure you get the idea by now. Although too numerous to offer a complete list here, filters include highest and lowest max, average, and current; filters which plot metrics that fall above or below static thresholds as measured by max, min, and average; and metrics that are most deviant from the rest of the series.

There are also a few advanced functions that bear mentioning. Included are functions for plotting the Holt-Winters Forecast, Confidence Bands (error bars), and Deviation. Holt-Winters is a statistical forecasting technique based on exponential smoothing. I wrote an article [2] about its inclusion in RRDtool, and I stand by what I said in that article: it's the coolest code that nobody ever uses.

I can't go into great detail here, but suffice it to say that the technique does a good job of predicting future data points based on existing data, even taking into account long-term and seasonal patterns (such as spikes or slow periods caused by human behavior on weekends and holidays). For many metrics, and especially system-based ones, problems can be detected by measuring their deviation from the "expected" value given to us by Holt-Winters, and Graphite makes this more accessible than it's ever been before.

Using Holt-Winters, I could create a dashboard that told us not only the five most utilized Web servers, but also their deviation, with:

```
&target=highestCurrent(dc4.Web.*.cpu,5)&target=holtWintersAberration(highest
Current(dc4.Web.*.cpu,5))
```

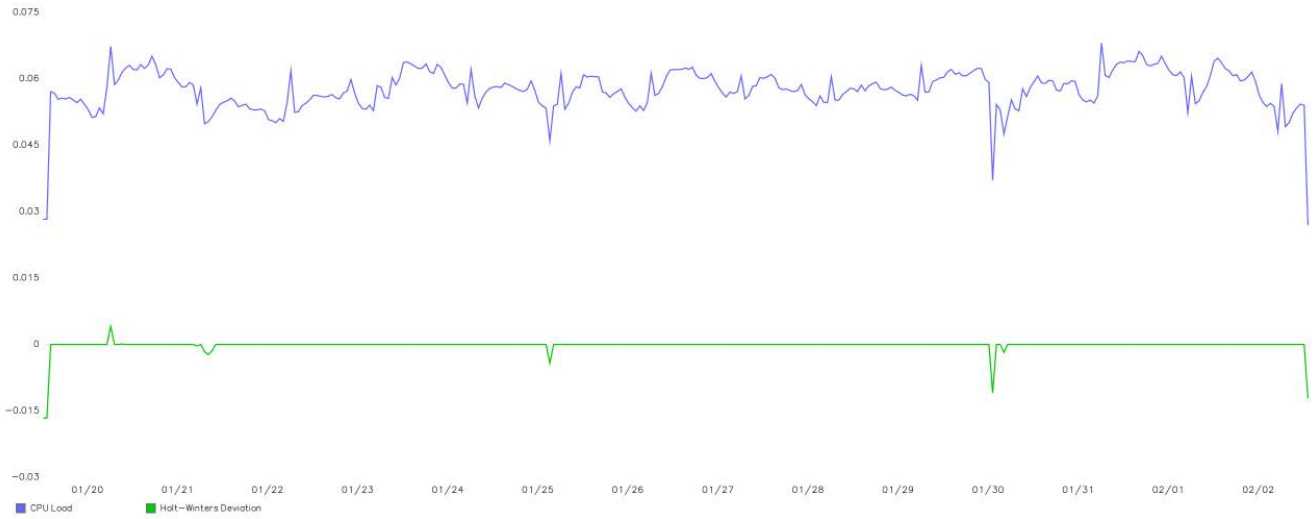

Figure 8: CPU utilization paired with Holt-Winters aberration

This graph might look something like Figure 8, where, while the lines on top would tell us what the CPU values were, the bottommost line would give us an indication of how "problematic" or at least how "unexpected" those values were.

There's a lot more to say here, but I'm afraid I'm at my word limit (to say nothing of having probably exhausted my Figures budget for all of 2012 (sorry, Jane-Ellen)). If my other articles on Graphite haven't convinced you to check out this truly excellent tool, I hope this last one has. I'm sure I'll revisit Graphite as it continues to mature, but I have so many excellent tools in the pipeline that I really must move on. Next time, expect the first in a new series of articles on a Nagios plugin that I'm really excited about called check_mk.

Take it easy.

References

[1] Graphite documentation—Functions: http://readthedocs.org/docs/graphite/en/latest/functions.html.

[2] Dave Josephsen, "iVoyeur: Hold the Pixels," *;login:*, vol. 33, no. 4, USENIX, 2008: https://www.usenix.org/publications/login/august-2008-volume-33-number-4/ivoyeur.

Should You Care About Solaris 11?

PETER BAER GALVIN

Peter Baer Galvin is the CTO for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is co-author of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter is also a Lecturer at Boston University. He is a Senior Contributor to *BYTE* and anewdomain.net. Peter blogs at http://www.galvin.info and twitters as "PeterGalvin."
pbg@cptech.com

With only minor fanfare, Oracle announced the first official customer-ready release of Solaris 11 on 11/10/2011. Oracle decided to announce on a Thursday rather than a Friday, which is a shame because releasing Solaris 11 on 11/11/11 would have been epic. Back in the day, say around the turn of the century, a major new Solaris release would have drawn quite a lot of attention from IT management worldwide. The luster has been somewhat lost due to the Solaris is open/closed/open/we-are-not-saying nature of its development history, the widespread growth of Linux as an enterprise OS, and the potential customer nervousness about the future of Solaris as now owned by Oracle.

However, that does not mean that Solaris 11 is uninteresting, unremarkable, or un-innovative. Quite the opposite. Below, I delve into the details of Solaris 11, what makes it pertinent, and why it should be given full consideration as a commercial-grade, full-featured, and powerful operating system. Also not to be ignored are the variants of OpenSolaris, each of which has an interesting take on the future of operating systems.

Features

Oracle is touting Solaris 11 as "The First Cloud OS" [1]. Certainly that statement comes with marketing hyperbole, but the features included in Solaris 11 (S11) do provide the basis for a scalable and manageable operating system. When combining S11 with the Oracle Enterprise Manager Ops Center [2] (free to use on systems with Solaris support contracts), Solaris gains some site-wide management features which help meet that claim.

Solaris 11 has several new features and enhances several older Solaris 10 features. To understand the potential utility of Solaris 11 it is important to understand the entire feature set. The list below is complete and indicates the nature of the feature (updated (U) or new (N):

- Package management system (N)—The new image packaging system (IPS) starts from scratch and solves many of the long-standing problems of the previous System V package management system. The new system is much more like Debian Linux in that all packages have versions, are network-update-able via the Internet package repositories, and are digitally signed for security. No before or after scripting is allowed, resulting in packages that are independent, removable, and reinstallable. The biggest overall change is the lack of a patching system. Rather than a patch, a new version of a package incorporates any changes. Packages

understand dependencies and thus an update or install of one package might result in a cascade of other updates or installations. Also, package installation can occur in parallel, so updating a system with many zones, for example, is much faster than before. The previous package and patching system still exists to allow installation and updating of pre-S11 packages.

- Boot Environments (N)—Live Upgrade is gone, replaced by a new boot environment facility and boot environment manager. Now that ZFS is the only root file system, its features are bearing fruit throughout the system, as exemplified in the boot environment manager. A new boot environment is created automatically by the package system if it is making major package updates, or manually at any point by the sysadmin. It is a ZFS clone of the existing root pool, with the new changes applied. The previous version is retained, allowing easy rebooting to a pre-changed environment. Further, many versions are kept by default, but are delete-able as needed. Booting to any of these reveals the system as it was at the time of that boot environment. Package upgrades take place on a live system (within the new clone), and just the downtime of a reboot is needed to switch between environments.
- Automated Installer (N) replaces Jumpstart and its brethren with one unified automatic installation tool. The new tool, of course, understands the new packaging and boot environment facilities. It also understands zones and can automatically create zones after its automatic installation or update of Solaris from the AI server.
- Network virtualization and quality of service (aka Project Crossbow) (N)—At long last Crossbow is available for the masses, rather than just in the open source preview releases. Crossbow is a breakthrough networking facility, layered on top of the previous network stack redesign that brought better performance and scalability. With Crossbow, the sysadmin can create an entire virtual network within a Solaris instance, including, for example, virtual NICs, switches, routers, firewalls, and load balancers. All those components can work together to route, filter, and balance traffic between zones within a system. The quality of service component gives fine-grained control over network flows, allowing bandwidth management on a per-protocol or per-NIC basis. If the sysadmin wants ftp to use at most 5 Mbps of network bandwidth, a couple of commands gets it done.
- DTrace (U) is now fully network-aware for exploration and debugging of network code, and has other minor changes.
- Zones (U) and virtualization are a bit confusing. Oracle renamed LDOMS to be Oracle VM for SPARC, while Oracle VM for x86 is a totally separate facility based on Xen. Of course, Zones are fully distinct from those two as well. The best course of action is to ignore the names and choose the right facility based on your use case. Zones have been updated and have gained some features, but also have lost some features. For example, Solaris 8 and Solaris 9-branded zones were supported with Solaris 10 as a way to capture a previous-OS system and run its apps within S10. S8 and S9-branded zones are no longer supported, but Solaris 10 zones within S11 are supported. That is, you can capture a Solaris 10 system and run it as a branded zone within S11. Unfortunately, you can only do so if the S10 system has no zones on it. If it does have zones, the best path is to virtual-to-virtual (v2v) the zones to the S11 system, making them S10-branded zones within an S11 system. Also gone from S11 are whole-root zones. Now that we have ZFS file system cloning, whole-root zones are no longer needed, since all S11 zones behave that way (they allow modification of even system directories). For sparse-zone-like operations, S11 zones have an "immutable" mode which

implements partial or complete read-only operation. Finally, zones now allow NFS server services. Unfortunately, zones themselves still cannot be stored on an NFS server.

- ZFS (U) is a revolutionary file system/volume manager that in S11 gets several new features, including block-level encryption.
- Oracle Enterprise Manager 12c (U) adds a host of features and adds support for some S11 features, including OS installation, package management, zone management, and system monitoring. Oracle's goal for the tool is to provide complete life-cycle management of Solaris.

There are many other changes within S11, including LDAP client and Active Directory client integration, improved role-based access control (RBAC), auditing and logging, and more cryptographic functions with hardware acceleration when run on SPARC hardware.

How to—and Should You?—Upgrade

It is important to note that there is no seamless upgrade path from S10 to S11. Rather, the zone management tools can capture S10 zones or the S10 global zone and turn them into an S10-branded zone that can run within S11. This method should be sufficient for most uses, but it is disappointing that no direct upgrade can take place. The lack of that upgrade path is an indicator of how major the changes between S10 and S11 are.

Which brings us to one aspect of the OS wars: ISV support. Without ISV support, a great operating system can become a footnote in history. ISVs need to understand the potential of Solaris and to determine whether their products will be supported on Solaris. To fully embrace Solaris 11 an ISV needs to adapt to the new package management system, but the old System V package system still works, as do older binaries. At a minimum, an ISV needs to test their existing application on Solaris, which is a low barrier to entry.

Impacts and Choices

Oracle, while unclear on some areas of its product plans, is being very clear in one area. Oracle is firmly committed to making Solaris and Oracle Linux the best places to run Oracle's other software products. The "run Oracle on Oracle" mantra is heard loud and clear at their conferences and within their documents and announcements.

Oracle is claiming, for example, that many internal changes were made to Solaris to support Oracle Database, including performance, reliability, and security improvements. Oracle also allows the use of specific features of Solaris to optimize Oracle DB license use. Consider that the Oracle rules for what is considered "hard partitioning" (and therefore is allowed to limit the CPU cores that need to be licensed) include several Solaris and SPARC options but fewer options for other technologies [3]. Certainly, running Oracle software products on Oracle hardware and operating system products makes sense for support reasons, although a given site should do a full analysis of price, performance, and features among the various options to determine which platform is the overall "best" solution for them.

While considering the platform options, a site should also consider the new kids on the block. (That is not an endorsement of the band by the same name.)

Teams of engineers have started from the last release of OpenSolaris and are creating their own distributions from that base. They are coordinating their efforts, in that they will contribute their changes into project illumos [4]. Think of that project as the new OpenSolaris. From there, several distributions are advancing rapidly, making use of the core and solving specific problems. OpenIndiana [5] is a general-purpose release. Nexenta [6] is a commercial release mostly designed to be a storage platform. And Joyent [7] recently released their SmartOS open source and free variant of illumos, with rich cloud-computing features. SmartOS keeps the core of OpenSolaris, replaces the new package management system with the one from BSD, and adds KVM-flavored containers to allow other operating systems such as Windows and Linux to run unmodified. While SmartOS can scale vertically, the design goals seem to be horizontal scaling with a management framework that allows monitoring, management, and automation of a farm of SmartOS systems.

The Future

No one outside of Oracle (and perhaps few inside) knows what the long-term future of Solaris is. Will it become an embedded OS that is used only for Oracle's Engineered Systems/Appliances? Is there enough demand, ISV support, and Oracle support for it to remain a leading general-purpose enterprise operating system? Certainly my discussions with IT management range from "We've moved on from Solaris," through "If only we were still running Solaris," and on to "We're moving back to Solaris." Fundamentally, it's my firm belief that no other common, commercial operating system has a better feature set than Solaris, especially as those features relate to production operation.

In the history of computing, there have been many failures of "better" engineering. Even as far back as Multics [8], the operating system that launched thousands of other operating systems, including UNIX, just having better functionality did not mean commercial or even cult-following success. Domain/OS [9] from Apollo was advanced for its time, as was TOPS-20 from Digital Equipment Corporation [10]. And let's not get started on the holy war of "operating system X was better than Microsoft Windows." In fact, all of the UNIX vendors of the time, in the 1990s, were worried about Windows NT winning the operating system wars and practically becoming the only major operating system. This continued until Scalability Day [11], when Microsoft tried and failed to prove that Windows NT could scale—at least according to the entertaining and informative presentation/rant by Bryan Cantrill at the LISA '11conference [12].

So where does that leave IT environments in terms of operating system choices? As usual, the choice of operating system will depend on feature need, in-house skill set, ISV support and recommendation, and a bit of arbitrariness. Solaris 11 does score well in several of those areas at many companies, and thus should be part of the consideration.

The Solaris offspring such as SmartOS, Nexenta, and OpenIndiana are worth considering if their features meet your goals. The case can be made for Solaris being a cost-effective platform, considering its free zone functionality, which provides very efficient virtualization, for example, or its capped zones, which can effectively reduce the number of licensed cores for some software products (including Oracle's). Add to that the production-ready functionality of DTrace and ZFS and it becomes difficult to not have Solaris on a short list of operating system platforms.

I would love to hear your thoughts about Solaris 11 and its future, and whether it makes your short list.

References

[1] The First [Oracle] Cloud OS: http://www.oracle.com/technetwork/server-storage/solaris11/overview/index.html.

[2] Oracle Enterprise Manager Ops Center http://www.oracle.com/technetwork/oem/grid-control/overview/index.html.

[3] Oracle partitioning options: https://www.oracle.com/us/corporate/pricing/partitioning-070609.pdf.

[4] illumos: https://www.illumos.org/.

[5] OpenIndiana: http://openindiana.org/.

[6] Nexenta, a commercial release mostly designed to be a storage platform: http://www.nexenta.com/corp/.

[7] Joyent, SmartOS with cloud computing features including KVM: http://www.joyent.com/.

[8] Multics: http://www.multicians.org/unix.html.

[9] Apollo Domain/OS: http://en.wikipedia.org/wiki/Apollo/Domain.

[10] DEC TOPS-20: http://en.wikipedia.org/wiki/Tops-20.

[11] MS Scalability Day: http://news.cnet.com/Scalability-Day-falls-short/2100-1001_3-279928.html.

[12] Bryan Cantrill at the LISA '11 conference: http://www.youtube.com/watch?v=-zRN7XLCRhc.

/dev/random
Dark Rhetoric

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

If I understand it (you probably don't want to put any money on that), "dark silicon" is what happens when you have more processing doohickeys on a chip than you can afford to keep powered up continuously, because of the excess heat that generates—something like the effect of a close-in shot of all the Dallas Cowboy Cheerleaders gyrating simultaneously. To minimize this (chip heat, not gyrating heat), manufacturers have taken to leaving parts of the chip that are unnecessary for the current processing tasks powered off, or "dark." That got me to thinking (keep your haz-mat suit handy): perhaps this approach would work equally well in other areas of technology.

Pray, let us take, by way of example, the nation's urban highway systems. The vast majority of these are lit by sodium vapor or halogen street lights. (Or, if they aren't, just pretend I'm right; my wife does it every day.) These things are bound to take a lot of power to keep shining, power that is wasted if no one is on the street. I'm talking about expressways and such with no routine pedestrian traffic or driveways along them, of course—not the road in front of your house where the corner streetlamp is burned out half the year, anyway.

Shoving aside (rudely) the considerable latency in firing up some of these lamps—because it more or less ruins the argument I'm about to make and therefore in true pundit style shall be tidily ignored—I propose that we install motion detectors calibrated for whatever the minimum legal vehicle size is for that roadway, so that the lights only come on when someone could actually need them. We could call this "dark highways." The motion detectors would be arranged in such a fashion that the highways in question were divided into zones. Only a currently occupied zone would be illuminated.

This would probably present a problem if you ran out of gas and tried to hoof it to the nearest gas station because: (1) a single person walking along the highway would not trip the motion detectors, so as soon as you left your zone you'd be in the dark; (2) since your car would no longer be in motion, even that zone would be dark by the time you got back. Maybe if you flapped your arms while you walked...

While this really isn't a particularly viable proposal for municipal energy-savings, people trudging up and down the bypass flapping their arms like goony birds trying to take off would provide a lot of entertainment value when viewed from the cameras that would almost certainly get posted every couple hundred feet or so along the roadways. Not to mention traffic helicopter views. Oh, and International Space Station videos of large cities blinking on and off in segments like mega-scale

billboards would be sweet, too. I can envision flash mobs being formed to spell out political or environmental messages as viewed from space. They might call themselves "flaptivists." (That's not what I would call them, mind you.)

We could further degrade this once useful principle by applying it badly to all sorts of inappropriate systems, but I want to take things in a slightly different direction now. The term "dark silicon" has put me in mind of the fate of other nouns that have had "dark" placed in front of them. Setting aside fictitious concoctions like "the dark side" and "dark lords," we shall begin with "dark matter."

This is a seriously messed-up concept. It's still matter, so it has mass, but it doesn't seem to reflect any light or glow when it gets irradiated or experience, in fact, any of the illumination-producing actions to which normal matter is subject. Zero albedo, as it were: interesting attribute. If we ever manage to corral some of it I can see a lot of covert operations applications, as well as just about any other activity that benefits from an absence of light (mushroom-farming and politics spring to mind). They might even have to add a new color to the crayon box rainbow: "Dark Black." Since it reflects no light at all, however, it won't be very popular for use in coloring books. I'm not sure whether it would look transparent, or simply create a gap in the optical field, as though that part of the page were missing and so was everything behind it. Thinking about that makes my head hurt.

This segues inexorably if not neatly into the nebulous "dark energy." I don't know what to say about this one except, "*Excuse* me?" This smacks of slapping a scientific-sounding term on something that may not really even be there. You can't just go around putting "dark" in front of anything you don't understand. If that ever catches on we'll have "dark algebra," "dark probate," and especially "dark romance." I'm not so sure that last one doesn't already exist.

...

"Sir, did you realize that you were going the wrong way on the street and speeding back there?"

"I'm sorry, officer. Traffic signs are dark for me."

...

"I'm afraid I'm going to have to disallow this $4,500 deduction for therapeutic ice cream sandwiches and the $875 one for marijuana-scented hair gel unless you provide some rationale beyond 'coz my old lady brings me down.'"

"I'm bummed about that, man. I guess this tax stuff is, like, dark."

...

At any rate, I think a much more descriptive term for it would be "missing energy" or its more dramatic cousin "fugitive energy." My own theory concerning dark matter, incidentally, is that the fundamental fabric of space itself has mass, and that explains all the mathematical anomalies. Whoever heard of massless fabric, after all? Problem solved. You can ship the Nobel Prize to my PO Box. Make sure to wrap it in an old towel or something so it doesn't get broken.

Last night I heard an odd noise in my backyard and went out to investigate. I realized that the dark itself was a mystery to me, which this terminology scheme would of course render as "dark dark." Unfortunately for the further exposition of this

thesis, the term "dark dark" reminded me of "Jar-Jar," which made me nauseous and I had to lie down.

It may well turn out that the culprit in my annoying middle-aged weight gain over the past few years has not been excessive chocolate and beer coupled with lack of exercise, after all, but "dark calories." Or perhaps I haven't really gained *any* weight; I'm just a victim of "dark gravity."

Here will I leave you all to your dark thoughts.

Book Reviews

ELIZABETH ZWICKY, WITH MARK LAMOURINE, TREY DARLEY, AND BRANDON CHING

The Linux Command Line: A Complete Introduction

William E. Shotts, Jr.

No Starch, 2012. 432 pp.
ISBN 978-1-59327-389-7

Some books I like because they fill my personal needs, some because they are good examples of something I have no interest in, and some because I can give them to other people. This book falls into that last category. This is the book that I can give to people who want to know how to do "that UNIX-y stuff you do." It assumes that you are a reasonably bright person with a grasp of how to use a computer, and you want to make the leap from using Linux with a GUI to using Linux from a command line. It introduces you to thinking like a UNIX person, without dragging in lots of history, and covers the most important commands you need to know, with a big helping of bash scripting.

Careful selection of topics keeps this down to a reasonable size. That means making lots of decisions I fully support, such as deciding to only cover Linux, and only modern distributions at that. Keeping the focus relatively narrow makes a book that's much more readable and usable. You're not forever skipping special cases. I am sad that this approach means that awk is only mentioned in passing, but if I'm going to support the drawing of lines, I'm going to have to live with some authorial choices that differ from mine.

I've been waiting for this book for quite a while, and will be enthusiastically pressing it on several people.

Beginning Python: Using Python 2.6 and Python 3.1

James Payne

Wrox, 2010. 558 pp.
ISBN 978-0-470-41463-7

Head First Python

Paul Barry

O'Reilly, 2011. 445 pp.
ISBN 978-1-449-38267-4

Learning Python, 4th Edition

Mark Lutz

O'Reilly, 2009. 1140 pp.
ISBN 978-0-596-15806-4

The Quick Python Book, 2d Edition

Vernon L Ceder, Daryl K. Harms, and Kenneth McDonald

Manning, 2010. 322 pp.
ISBN 978-1-935182-20-7

If somebody had given me column A, with the book titles complete with series names and subtitles, and column B, with an accurate description of what each one covers, and asked me to match them up, I would never have succeeded. I found this group of books both startlingly diverse and oddly titled.

Beginning Python is in the "Programmer to Programmer" series. It also starts with a description of how programming a computer differs from using a computer, and spends pages of its chapter on variables in a discussion of what a variable is. On the other hand, lambda functions appear not long after, and shortly after that you have left Python itself to gallop through topics that drag in extra protocols and topics, ranging from file typing and file system traversal through XML parsing, and on to creating your own fully functioning Web server with a database backend. (Input sanitization, however, is out of scope, so it comes with XSS and SQL injection vulnerabilities.) The Python it teaches is 2.6 with 3.1 enhancements; it uses 2.6 idioms, not 3.1 idioms. I wouldn't recommend it to anybody, and I'd particularly advise against it for anybody who is just learning to program. The example of quotes, which illustrates single, double, and triple quotes with something that's either a single quote, two single quotes, and three single quotes, or a single quote, a double quote, and some punctuation mark I've never seen before, is particularly problematic, especially since it is immediately followed by examples which use triple double quotes.

If you want a rapid introduction to Python for an experienced programmer, I'd suggest *The Quick Python Book* instead. It covers the basics of Python, plus some of the key libraries

(regular expressions, Tkinter, pickles, shelves) for Python 3 and Python 2. It does so with enough Monty Python references to suggest that the authors get the Python mindset, but not an unbearable number. (Yes, reviewing Python books will, perforce, involve evaluating them on the number of Monty Python references. It is as inescapable as the Spanish inquisition.) If you do not already understand some programming language—preferably an object-oriented one—you will not find it a rewarding experience.

I have not yet found a book I'd recommend as a Python introduction for your average person new to programming. *Learning Python* is only a reasonable introduction for somebody with a computing background and a burning desire for completeness. Its introductory chapter does not attempt to introduce you to programming as a concept, but it does list all the major varieties of Python implementations and explain them. You get to "What is Jython?" before you get to "Hello, world." It's a very complete introduction to Python, taking 3 as its point of reference but with information on 2, the differences, and how to code portably. It hews quite carefully to the language itself, avoiding more than the briefest of brushes with common libraries. It's a good, readable language reference, and if you like learning languages systematically, it's an unusually good example of a careful guide to the whole language.

Learning Python should put the other books' lengths in context. It takes a bit over a thousand pages to do a nice, thorough job of explaining the language, just the language, with explanations of the idioms, nice clear examples, and plenty of whitespace, but no major detours and the assumption that you already understand all the underlying concepts. *Quick Python* covers that territory, plus common libraries, in a third the space. *Beginning Python* does it in about a fifth the space, and tries to begin with fewer assumptions.

And then there's *Head First Python*, which I like better than *Beginning Python* even though it is even more of a breathless gallop. In fewer pages with more pictures, it not only walks you through creating your own fully functioning Web server with a database backend (and no input sanitization), it also has you create an Android app and move your Web server onto Google Apps. On the other hand, it does a believable job of explaining the things it does explain, and it makes no pretense to have taught you how these things work. It teaches a number of general programming concepts (not just why objects and exceptions are good ideas, but also some concepts in software design), and it explicitly walks the reader through a number of debugging situations, which is important for novices. Like *Learning Python*, its audience as a book to learn from is a relatively narrow one, but the right person will find it a fun and educational ride. But please, please, do not decide that it is a good idea to build your own fully functioning safety-free Web server, especially with a database backend.

Seven Languages in Seven Weeks

Bruce A. Tate

The Pragmatic Programmers LLC, 2010. 300 pp.
http://pragprog.com/book/btlang/seven-languages-in-seven-weeks
ISBN 978-1-93435-659-3

When I was a freshman in college, I learned seven programming languages. Computer concepts were taught in Pascal. Engineering was in FORTRAN and VAX and 68000 Assembly. Business used COBOL. Artificial Intelligence research was done in LISP and Prolog. I have always been glad that I had that grounding in the variety of ways it is possible to express a problem.

Seven Languages in Seven Weeks offers a similar survey of modern programming languages and language concepts. The creators of these languages each feel that there's something that needs to be expressed and that no other language they know does quite what they want. Tate sets out to show what makes each one special. He's chosen Ruby, Io, Prolog, Scala, Erlang, Clojure, and Haskell. Except for Ruby, most of these will be obscure or unknown to ordinary mainstream coders.

Tate's introduction is very clear about what this book is not: it's not a tutorial or an installation guide. It's not complete or comprehensive. He didn't pick the most popular or most academically acclaimed languages. He apologizes up front to those whose favorite working language isn't included, and explains that he was not interested in producing a "Best of" book. He chose a set of languages which covers the range of current practice. His goal is to explore the significant features of each language, how those help express different ideas clearly and concisely.

The book is divided into a section for each language. The introduction to each section provides the resources and information needed to install the language and to begin interacting or coding. Each section is further broken down into single-day sessions. Tate knows you have real work to do, so each section only contains three days.

The daily sessions start with the common language constructs: variables, types, logic, flow control, and so on. Tate glosses the basics and highlights how each language is special. By the third day, you're deep into the core concepts that make each language unique. Each day ends with a summary of the key concepts and a set of exercises to help you explore for yourself and to set them in your mind. The sections conclude with a wrap-up of the significant features and a little discussion of why they're important.

The book closes with a summary of the families of modern programming concepts and how each of these languages fits into those families. Tate highlights each language's strength, but he doesn't shy away from exposing the warts or showing how one problem or another might not be suited to a given language.

Tate's style is conversational and tutorial. He writes as if he's sitting down with you to show you something cool. He opens each day with some kind of informal anecdote or metaphor that leads to the day's topic. His preparation has included interviews with the language writers or researchers, and in some cases he includes portions of his interview if it highlights the character or taste of the language he's teaching. In at least one case he gets the author of a language to say what he'd most like to change if he could go back and start again.

When you've finished with this book, you should have a clear understanding of some of the more esoteric concepts of current programming languages, and some sense of the flavor of each of the individual languages. This book may be frustrating to someone who's not already familiar with at least a couple of programming languages. I'd steer away from it if your interest is solely in writing application code in any one of them.

I like exploring and understanding the capabilities of different programming languages, even ones I don't expect to use. There's no example in any of these sections that could not be implemented using one of the other languages. What I enjoy is seeing the elegance that each one brings to solving a problem. I suspect I'll pass it on to friends who also like that kind of thing.

—Mark Lamourine

The Art of Readable Code: Simple and Practical Techniques for Writing Better Code

Dustin Boswell and Trevor Foucher

O'Reilly Media Inc., 2012. 190 pp.
ISBN 978-0-596-80229-5

This is my first experience with an O'Reilly book from the "Theory in Practice" series. This series tries to "impart the knowledge and wisdom of leading-edge experts" (http://shop.oreilly.com/category/series/theory.do). *The Art of Readable Code* does feel like a series of lessons or conversations with a colleague or mentor. The authors claim that many of the examples come from their own real applications.

The Art of Readable Code opens by making a case that code should be written with the human reader in mind. Anyone who's ever read someone else's code (or even their own after a time) should be able to get behind that.

Sprinkled throughout the book are a set of "key ideas." Each one relates to the clarity of the style or structure of the code. They range from choosing good names to knowing your libraries. They also include a couple of examples of traditional structural refactoring. Some of this may sound quaint, but the authors illustrate their points in practical ways.

The first three sections cover cosmetic and aesthetics, then logic and branching structures, and, finally, application structure.

There is a fourth section with two unrelated chapters. The first makes a case for writing tests that can be read and that, when they fail, indicate clearly what failed. They also include a remarkably non-trivial application and work through three phases of development.

In most books I don't look at the table of contents much after I begin reading, but in this one the chapter headings make the best summary of those key concepts. It would have been nice to see a cheat sheet or a one- or two-page compact summary of the key ideas.

I've been coding for long enough that there's not a lot here that's new to me, but I did pick up a few tips, and the book presents the ideas in a concise and coherent way. For someone just starting out or who is interested in approaching coding for readability in a systematic way, there's something here for you that I haven't seen anywhere else.

My bookshelf is made up mostly of pure references. I have a few classics which don't get much use, but which I don't feel I can part with. I think this one may fit between those two groups. I won't be looking up function calls, but I can imagine scanning it again when I find myself facing something ugly.

—Mark Lamourine

TCP/IP Illustrated, Volume 1, 2d Edition: The Protocols

Kevin R. Fall and W. Richard Stevens

Addison-Wesley, 2011. 1017 pp.
ISBN 978-0-321-33631-6

There's no shortage of technical books. Most quickly fade in value due to the constant churn of innovation. A select few stand out, forming something like a canon of computer science. It is a testament to W. Richard Stevens's depth of knowledge and communication style that after nearly two decades people still refer to his books. Kevin R. Fall had big shoes to fill when he undertook the ambitious task of produc-

ing this updated edition. (Fall is certainly no slouch himself, having served on both the Internet Architecture Board and IETF.) The result is impressive, a true labor of love. It remains true to the spirit of the original while bringing it up to date.

Fall leads the reader gently up the OSI stack, from media layer framing all the way up to DNSSEC and TLS. He assumes a certain level of innate intelligence in his reader but tries hard not to assume much knowledge about TCP/IP. The text incorporates fascinating historical notes, from the ARPANET days to the present, which illuminate both the human politics and technical drivers for change.

One major difference between this and the former edition is how much material Fall elected to remove. The first edition was, in some respects, wider in scope, addressing such topics as NFS, SNMP, SMTP, and dynamic routing protocols. Fall has focused exclusively on core Internet protocols. One might well object that dynamic routing protocols *are* core but, as Fall explains in his preface, there's a world of difference between RIP and BGP/OSPF, and to properly treat the latter would have made this already sizable tome an unreadable doorstop.

While a good bit of material has been elided from this new edition, much has been added. The core protocols have substantially evolved over the past two decades. Fall has done a great service to his readers in assessing those changes. He's essentially read a great pile of RFCs, distilled the essence, and highlighted further reading on topics most relevant to you.

As in the first edition, this incorporates countless packet traces (both tcpdump and wireshark) to illustrate what's going down on the wire. On the inside front cover are three example network diagrams: a home network, a coffee house, and an enterprise. Fall uses these throughout the book, and it proves an effective trope. Back in Stevens's day, there were some pretty stark differences between different TCP/IP stacks. Things have improved, but Fall keeps to Stevens's penchant for mixing traces from different OSes (OS X, FreeBSD, Windows, and Linux), reflecting the heterogeneity of the real world.

Fall has tried to make each chapter self-contained (each chapter is followed immediately by its footnotes, for example). IPv4 and IPv6 are totally integrated within each chapter (except in a few cases where the topic is only applicable to one or the other, as with ARP vs. Neighbor Discovery). Security, too, is integral to the entire text.

It's worth commenting on the final chapter. Although Fall has made security an integral part of the entire book, his final chapter focuses exclusively on security issues. He starts off with an excellent refresher in crypto, then goes on to deal with EAP, IPsec, PKIs, DNSSEC, TLS, and DKIM. I would buy the book on the basis of this chapter alone.

Some who buy this book will just stick it on a shelf and only refer to it occasionally. But while this is most assuredly a reference book (and an excellent one at that), you definitely *can* read this book in its entirely, and I would argue that you are cheating yourself if you don't. Some material is a bit dense by its very nature, to be sure, but the writing is incisive and engaging. As I write this, we're up to RFC 6528. Nobody has time to read all of that. Who among us isn't constantly skirmishing with networks, be you coder, DBA, researcher, policy wonk, or sysadmin? The network is pervasive. Perhaps, like me, your knowledge of TCP/IP is an amalgamated hodgepodge gained through years of experience. If you take the time to read this book you will fill in your gaps and deepen your understanding of not only the "whats" of the Net but also the crucial whys and hows.

—Trey Darley

Head First HTML5 Programming: Building Web Apps with JavaScript

Eric Freeman and Elisabeth Robson

O'Reilly Media, 2011. 610 pp.
ISBN 978-1449390549

O'Reilly's Head First series is a definite departure from traditional technical publishing methods. Instead of pages and pages of text and code, the Head First series uses images, comedy, and a variety of methods to assist your brain in remembering what it is that you are learning. *Head First HTML 5 Programming: Building Web Apps with JavaScript* is one of the latest in this series and, like its predecessors, it does not fail to provide the reader with ample information in an understandable format.

Weighing in at 610 pages, you might think that *Head First HTML 5* is a bit of overkill for a relatively simple updated Web standard, and you'd be right. However, the extent of what is possible in HTML 5 is highly dependent on associated technologies such as JavaScript. Thus, the vast majority of this beefy text is actually focused on how JavaScript, in combination with HTML 5, can be used to usher in a new generation of Web applications and features.

In fact, of the ten chapters in the book, all but one are focused primarily on JavaScript. The book opens with a history and general overview of HTML 5 and how the HTML standard has come to be what it is today. The next three chapters offer a crash course in JavaScript. The overview provided

does assume some previous programming experience and focuses primarily on DOM parsing, events and handlers, and functions and objects. Chapters 5 through 9 provide a sort of "greatest hits" for Web applications, covering such topics as geolocation, canvas, AJAX and JSON, and video/media. The book wraps up with coverage of local Web storage and Web workers, for those beefy applications.

As with a lot of instructional texts, each new concept is supported by the construction of a dedicated mock project. All code samples in the book are concise, well explained, and relevant. One of the neat things about the Head First series is that visually, important gotchas and side notes are made easy to identify and remember and do much to help you understand what you are learning.

There was no single chapter that I thought was better than the rest. This is one of those rare books that I found to be well written and on target throughout. If I had to raise a complaint at all, it would be that all the examples are in standard JavaScript, as opposed to a JavaScript library such as jQuery. JavaScript libraries are so ubiquitous in the Web development community that it seems very little new development is done outside of them.

Head First HTML 5 is a book well suited to be on almost any Web developer's bookshelf. There is definitely something in here for everybody, from the junior developer to the expert. In the new era of HTML 5, JavaScript is no longer an option, it is a necessity, and *Head First HTML 5 Programming: Building Web Apps with JavaScript* offers a solid foundation.

—Brandon Ching

Conference Reports

In this issue:

USENIX LISA '11: 25th Large Installation System Administration Conference

Boston, MA
December 4–9, 2011

Opening Remarks, Awards, and Keynote Address

Summarized by Rik Farrow (rik@usenix.org)

The 25th Large Installation System Administration Conference began with the co-chairs, Tom Limoncelli (Google) and Doug Hughes (D. E. Shaw Research) tag-teaming their opening presentation. Carolyn Rowland will be the next LISA chair, with the conference occurring in San Diego in mid-December 2012. After many thanks to the organizers and USENIX, they announced the three Best Paper award winners: Herry Herry's "Automated Planning for Configuration Changes" won the Best Student Paper award, with his advisor, Paul Anderson, accepting the award for Herry. The Best Practice & Experience Report award went to a Google team for "Deploying IPv6 in the Google Enterprise Network: Lessons Learned" (Babiker et al.). Finally, Scott Campbell of Lawrence Berkeley National Lab won the Best Paper award with "Local System Security via SSHD Instrumentation."

Phil Kiser, LOPSA President, presented the Chuck Yerkes award to Matt Simon for his helpfulness and frequent activity on the mailing list. Matt said that he had never met Chuck, but that he must have been one heck of a guy.

David Blank-Edelman (Northeastern University and USENIX Board Liaison to LISA) presented the SAGE Outstanding Achievement Award to Ethan Galstad, the creator and principal maintainer of Nagios. Galstad said that he had created Nagios to prevent system administrators from being paged unnecessarily, and while he was pleased to receive the award, he is introverted and receiving it was not easy. He thanked his wife, family, friends, and the worldwide Nagios community, and the Linux community for providing him with a free compiler.

Keynote Address: The DevOps Transformation

Ben Rockwood, Joyent

Ben Rockwood, the Director of Systems Engineering at Joyent, gave a stirring explanation of the way he sees DevOps—not as a tool or a title, but as a cultural and professional activity. DevOps is not something that we do, but something that we are, said Ben. Ben declared that DevOps is more a banner for change (displaying a picture of a knight), not simply a technique. It is a journey, not a destination.

Ben went on to present both an interesting slide show and his detailed research into the history of systems studies. He emphasized that DevOps begins with the "Why," proceeds to the "How," and ends with the "What." The "Why" refers to the limbic system, the emotional seat of all animals, including humans, as does the "How." The "What" is the product, such as building awesome services. Another way to see this is that the "Why" represents motivation, such as quality through collaboration, and the "How" and "What" are the process and tools used. Ben said that DevOps does not mean starting with the "What," such as configuration management, and working backwards to the "Why."

Ben described Russell Ackoff's five contents of the mind: wisdom, understanding, knowledge, information, and data. He then tied these content types to levels of sysadmin, with System Architects related to wisdom and understanding, Senior Sysadmins with knowledge, and Junior Sysadmins/Support with information and data. Ben used these concepts as a way to shift into talking about systems thinking, with wisdom and understanding corresponding to synthesis and knowledge, information and data to analysis.

DevOps is about the entire system, not just Dev and Ops working separately, but with the two groups working together toward the goal of quality. While Dev and Ops are often silos, they truly are both part of the same system, but seeing this can be difficult. Ben quoted W. Edwards Deming: "A system cannot understand itself." This implied standing outside of both Dev and Ops to see how the two groups could work best together.

Ben was not content with spouting platitudes, but discussed methods for measuring quality in software and systems. You could tell that Ben had spent a lot of time studying potential metrics for measuring IT service, and after comparing many complex standards, he selected ITIL as the most complete and respected pattern for IT. Ben said that it is best to read the entire ITIL set of books, although he did suggest reading *The Visible Ops Handbook*, as a good place to get started.

At this point, Ben was just getting warmed up. He suggested Agile, said that the cloud has changed everything in the IT world (partially because it supports agility), and went into a history of operations management. In the next section of his talk, a section he said he feared might be boring, Ben explained where many of the ideas for scientific systems management came from. It turned out to be a fascinating look at the men who created this field and their contributions. He ended this section by suggesting that we stand on the shoulders of giants, and not ignore the lessons of the past. A simple summary of where we are today would include lean, Scrum, and agile operations concepts.

Ben concluded by bringing his focus back to DevOps, suggesting that boundaries between development and operations teams need to blur, and that both teams are fully accountable for both successes and problems.

Mark Burgess (CFEngine) opened the discussion by praising Ben for paying attention to great thinkers in history. He then mentioned Alvin Toffler's "Future Shock," and Ben agreed that people should read that book. Mark then asked if DevOps is an expression of wanting to make a difference, not just working in this monolithic tech environment. Ben responded that DevOps allows us to do what we always wanted to do, but were prevented from doing. Also, now more of the tools are present and we have moved beyond just building assembly lines.

Steven Levine (RedHat) commented on his world of tech writing, where he has to bridge the gap between marketing and engineering, and that DevOps concepts could address one of his lifetime issues. Ben answered by referring to Sun Microsystems tech writers, who had no access to engineers and just had to figure things out on their own—certainly not DevOps.

John Rouillard (Renesys) pointed out that if we don't know what went on before, we reinvent something else, perhaps worse. Ben responded by saying the problem is more fundamental than that—we don't know it exists! Ben told the story of walking his baby while at church, noticing a book on operations management (OM), then taking it home and devouring it. He hadn't even known OM had existed. Now he has an entire shelf in his office at Joyent devoted to OM books.

Perspicacious Packaging

Summarized by Carolyn Rowland (carolyn@twinight.org)

Staging Package Deployment via Repository Management

Chris St. Pierre and Matt Hermanson, Oak Ridge National Laboratory

Matt began the presentation by talking about their HPC environment and the concern for consistent security through automated package management. Using yum for package management did not allow for granular control of packages nor did it allow for a period of testing before rolling a package into production. The solution Chris and Matt presented included use of Bcfg2 for configuration management and a new tool called Pulp. Pulp is part of RedHat's CloudForms and a lean replacement for Spacewalk (minus the GUI and Oracle requirements). The authors wanted to control what was really in the package repository and how it moved into production, but they didn't want to do it manually. The repository has three areas: upstream—created daily from sources; unstable—automatically synced into from upstream filtering packages they don't want; and, finally, stable—copied from unstable after about a week. Development machines get their packages from unstable while everything else pulls from stable. For an HA server, one node pulls from unstable to test the package before the other node installs it. The only exceptions are security patches, which receive immediate attention. The authors said it was important to install the security packages right away.

Downgrading with yum was difficult, as was rollback. The authors examined and rejected a number of solutions, including functionality built into yum and Bcfg2 and other repository management systems. Pulp can do manual manipulation of blacklisted packages from upstream which can then be overridden with one command for machines that have an exception to the blacklisting rule. The authors' workflow uses the Pulp automated sync facility to promote non-blacklisted packages automatically. There are exceptions to the automated package management. Impactful packages (e.g., kernel packages) are not automatically promoted. The authors found that servers using Pulp were far more up-to-date with upstream package versions than the rest. Chris has also written Sponge, a Web front-end (written in Python/Django) to make configuration of Pulp more intuitive (Sponge is available on github: http://github.com/stpierre/sponge). Sponge is currently the largest instance of programming for the Pulp API outside of the CloudForms project. Matt and Chris aren't finished: they'd still like to add an age attribute to packages. Currently, packages move through the stages weekly, so a package may only be in unstable for a day before it is promoted to stable. They'd like to make sure that packages are at least seven days old before moving between unstable and stable. Currently, Pulp supports all RPM-based distros. The Pulp developers plan to support arbitrary content (e.g., Debian packages).

During questions, one person asked how the authors collected concrete evidence that packages were ready to go from unstable to stable. Chris responded that they depend a lot on HA to protect them, allowing the services to test themselves, but agreed that testing could use some improvement. Could the authors use RPM and yum to detect a rogue package installation? Bcfg2 could report these packages and they could be removed automatically; the authors are not currently doing this. How does the system know which packages are authorized? Bcfg2 has a list of base and specialized packages. Someone pointed out that a rogue package would only be detected if it were installed using RPM. The authors responded that one could use tripwire to detect such anomalies.

Does Pulp mirror the repositories or is there a separate tool to do that? In most cases, Pulp can do the mirroring; for RHN; the authors currently use mrepo. How can Pulp improve over the Cobbler+ custom scripts method? Chris admitted that Pulp was alpha code when they started (even as recently as a month ago it wasn't ready), but added, "We were also the first to contribute code back to it." How do the authors mirror packages from one site to another and still keep package consistency? Pulp has a content distribution system (CDS), but they hadn't used it in their efforts. How do the authors maintain an older version of a package even after that package is no longer available? They break the mirror and set that package aside. "We do have a couple of environments that demand that stability." Pulp uses hard-links to point to these packages.

CDE: Run Any Linux Application On-Demand Without Installation

Philip J. Guo, Stanford University

It's hard to package your software so that other people can reliably run it. You cannot predict someone else's environment. The Internet is full of forums trying to solve this problem. It's also tricky to create a package that runs across all Linux distros with no problems: missing dependencies and different environments can require a tiered dependency installation to resolve. Enter CDE, a tool that provides automated packaging of Code, Data, and Environment. According to Philip, there are three simple steps to using CDE to package a piece of software: (1) use CDE to package your code by prepending any set of commands with "cde"; (2) transfer the package to the new computer; (3) execute software from within the package on any modern Linux box using cde-exec

(root not required, installation not required). Step 1 creates a CDE package in the current directory that copies all necessary files into a self-contained bundle. This is similar to a chrooted environment, because every file or environment variable required to run the command is available within the local CDE directory. A second user of the program can also edit the script within the package and modify as needed, still running it with all included dependencies. CDE uses ptrace to hook onto the target process. It actively rewrites system calls. When a call is made to the kernel, CDE intercepts and copies the accessed files (e.g., "/home/bob/cde-package/cde-root/lib/libc.so.7") into the package, then returns control back to the program.

Addressing the impact of CDE on system resources, Philip said it depends on the number of syscalls the program needs to make. Having many files to access means more calls back to CDE and the package. A user can run CDE packages like any other command, so you can provide input or redirect output outside of the package. An example of this is cde-exec Python $CDE-package/var/log/httpd/access_log, which by default looks in $CDE-package/var/log/httpd/access_log first. If this doesn't exist, it then chops the path to look for the absolute path /var/log/httpd/access_log. You can stop CDE from looking for $CDE_package/path using rewrite rules to say ignore anything beginning with /var/log, which will cause CDE to look at the absolute path without prepending $CDE-package first. CDE also includes a streaming mode which allows the user to stream selected apps without installing anything: (1) mount distro (e.g., sshfs) and (2) cde-exec -s (streaming mode) eclipse. This will load from the cloud or server. CDE will cache a local copy into cde-root on the local machine. There have been ~4000 downloads of CDE up till LISA '11. (To find it, search for "cde linux".)

How does CDE deal with environment variables? CDE packages the environment variables first, from the original package build, and loads them as part of the cde-exec process. How does CDE keep the cache synced with the authoritative package source on a server or in the cloud? A checksum system would help this, but he hasn't yet implemented it in CDE. How does it rewrite the paths so that it knows where to go? Does it rewrite LD_LIBRARY_PATH? Philip responded that ptrace rewrites the strings to load the environment and said as an aside, "It's actually a big security hole," which made the audience laugh. How do you address signing? There is currently nothing built in and no verification of content in streaming mode. In capture mode, how do you know you've run the command long enough to capture all of the environment necessary to rerun the command? The solution is pretty low tech: you run some representative runs, do a find, and discover that most apps are well-behaved. You can rsync from run to run to manually copy missing files. How would things like plug-ins impact a package (e.g., eclipse)? Would it change the distributed version in the cloud? You would have to modify your own cached copy, or you could change the cloud copy. Does Philip plan to continue to develop CDE or is he done? Philip doesn't rule out further development—"I'm trying to graduate but maybe after that"—but he welcomed community support to make CDE more robust. CDE was really created to solve a specific problem but could have more broad applicability in the field.

Improving Virtual Appliance Management through Virtual Layered File Systems

Shaya Potter and Jason Nieh, Columbia University

Shaya started by asking some questions to make the audience think about how we manage virtual machines (VMs). How do we provision? How do we update? How do we secure? Virtual machines can increase complexity (sprawl) and this can introduce security issues, giving hackers a place to hide. Traditional package management works for a single system but falls down with many heterogeneous systems. Deploying virtual machines takes time; copying even a 1 GB VM takes a significant amount of time. Copy on Write (COW) disks/file systems are good solutions for provisioning homogeneous machines but are not so good in a complex environment. There are also no management benefits; once you snapshot/clone, you create independent instances, each of which needs to be managed.

Shaya introduced Strata, which models a Virtual Layered File System (VLFS). In reality, machines are administered at the filesystem level with large commonality even among disparate, diverse environments. Strata decomposes the file system into layers that can be shared out to VMs. There are three parts to Strata: layers, a set of file-like packages (e.g., mySQL, Apache, OpenOffice, Gnome); layer repositories, containing layers; and VLFS, which composes layers together into a single namespace.

Shaya's example of a VM used two VLFS layers consisting of a MySQL VLFS and an Apache VLFS. To use Strata in this example, perform the following three steps to build your VM: (1) create template machines;)2) provision machine instances;)3) maintain the VM. Updating the VM is easy. Shaya used the example of a security patch for tar. First you update the layer in the VLFS that contains tar, then the update occurs in all VLFSes that contain tar. All VMs get the update instead of having to update all individual instances of tar. Using union file systems, Strata composes the various layers together to form a single file system for a VM. The various layers have different rw/ro attributes depending on use. There is no copying of VM images, and the admin only

has to monitor a single, centrally maintained template for all VMs. The base layers are read-only (ro) while the user or local sysadmin has access to read-write (rw) layers for local system modification. These rw layers are sandwiched on top of the ro layers (hiding files in the ro layers that occur in both). This allows modification and deletion of files at the VM level. This layering also provides visibility into unauthorized modifications; they appear in rw layers and can easily be cleaned. This also reduces or eliminates the need for tripwire-like programs. Provisioning time is independent of data size. Even large numbers of layers provision quickly. An ssh server may have 12 layers while a desktop machine could have 404 layers. Updating traditional VM appliances takes time, while updates to Strata occur on system startup when VMs grab the updated layers.

A member of the audience asked where the shared layers were stored. Wherever you need, e.g., a SAN or NFS server. Through initrd, the client mounts the SAN/NFS file system, then determines the configuration and unions the layers. This becomes the root file system. Initrd does all of the work to connect the VM to the Strata VLFS. Another audience member asked if the unionfs was accessing the layers on the SAN to provide immediate updates. Shaya explained that updates means creating a new layer which then updates the configuration of the VM. The new layer becomes part of the VM and the old layer is marked as unavailable. Any program that is running currently will continue to run with the old layer, but new programs will not have access to the old layer. There was a concern from the audience that unionfs has had trouble getting into the Linux kernel. Shaya admitted that Stonybrook unionfs has been struggling to get support; there are others, but Shaya didn't know the status of these. Had the authors considered pasting a configuration management system into Strata? A CM system could be used with Strata but it wasn't part of the initial research. Is Strata limited to virtual machines? Could this be used with clustering? Shaya responded that he had presented a paper at USENIX ATC using Strata with a desktop machine. It certainly could be applied elsewhere.

Invited Talks I: Databases

NewSQL vs. NoSQL for New OLTP

Michael Stonebraker, MIT

Summarized by Timothy Nelson (tbnelson@gmail.com)

Michael began by reminding us what online transaction-processing (OLTP) looked like in the past. We bought our plane tickets on the telephone, and a human operator entered the transaction. Thirty transactions per second was normal, and 1000 per second was unbelievably large. Today the human intermediary is no longer around to be a rate-limiter; mobile phones and PDAs originate transactions and submit them directly via the Internet. OLTP systems today need to be able to consume a million transactions per second. More than just entering the transactions, the system must ingest them, validate them, and respond to them, and do so with low downtime. Broadly, there are three options available: "Old" SQL databases, which provide ACID (atomicity, consistency, isolation, and durability); NoSQL databases, which sacrifice ACID for performance; and the "new" SQL approach, which keeps ACID but uses new architecture to gain performance.

Traditional SQL products have problems, one of which is bloat. They are legacy systems, and once a feature has been added, they cannot easily remove it. They are also generic: when used properly, specialized software can outperform traditional SQL products by a factor of 50. Because of this fact, Michael encourages selecting specialized database solutions based on business needs. For data warehousing, column-store databases are 50 times faster than row-store, which is the standard approach. There are similar insights for OLTP. Most OLTP databases are under 1 terabyte in size, which is not an unreasonable amount of main memory.

Michael then discussed where traditional databases spend their time on OLTP transactions. Nearly a quarter of the execution time is used by row-level locking. Similarly, latching (to support multi-threading), crash recovery (maintaining a write-ahead log), and running the buffer pool (maintaining the on-disk store) each takes about 24% of the time. This overhead means that traditional databases are slow for architectural reasons, and we should rearchitect them when we can. Faster B-trees alone will not give us much.

NoSQL is not a cure-all. SQL gets compiled down to the same low-level operations that NoSQL requires, the compiler isn't a big source of overhead, and it's actually hard to beat the compiler these days. Giving up SQL will not give us much performance. Sacrificing ACID can improve performance (e.g., by abandoning row-level locking), but once an application gives up ACID, it's hard to recover if needed later. An application needs ACID if it has multi-record updates or updates for which order matters. For commutative, single-record transactions, NoSQL is appropriate. NoSQL is a great tool, and it is good at lots of things, but we should use the right tool for the right job.

Michael's company has a NewSQL product called VoltDB. VoltDB has no traditional record-level locking. It keeps the database (except for cold data) in main memory. Rather than keep a write-ahead log, it defaults to always failing over, which is what most OLTP applications do anyway. It handles multiple cores without latching, by partitioning

main memory between the cores. VoltDB can do a 200-record transaction in microseconds, and it beats an unnamed traditional SQL database vendor by a factor of 50. The product supports a subset of SQL; correlated subqueries are on the way. Finally, VoltDB is open source.

Is Amazon a good application for NoSQL and eventual consistency? Amazon implemented SimpleDB because, at their scale, they can afford to build their own solutions. The rest of us have to buy third-party engines. If there is no benefit to building faster B-trees, someone from Mozilla asked, should we work on improving latching instead? Better latching would absolutely be an improvement, and we should work on making our data-structures more concurrent, not just faster. Two audience members asked about benchmarking VoltDB in a data warehousing situation. Since warehousing is very different from OLTP, VoltDB would not do well in that space. For data warehousing, one should use a column store instead.

Eric Allman asked about the problem of moving data from OLTP systems to a data warehouse; wasn't it quite expensive? Michael agreed, but he said that you often need to duplicate the data anyway, since the warehouse schema may be different from the OLTP schema. Also, you often want to use different hardware for response-time reasons. So ETL (extract, transform, load) isn't going away, but there may be a market opportunity for seamless integration of separate OLTP and data-warehousing solutions.

What is the revenue model for open source software? RedHat has a good model: provide an enterprise edition with goodies that aren't in the free version, and provide support for the enterprise version. The free community version can be viewed as a vehicle for sales.

Invited Talks II: Newish Technologies

Summarized by Rik Farrow (rik@usenix.org)

Issues and Trends in Reliably Sanitizing Solid State Disks

Michael Wei, University of California, San Diego

Michael presented research from NVSL (Non-Volatile Systems Laboratory), first reported during FAST '11. The takeaway is simple: deleting data on solid state disks (SSDs) doesn't actually remove it. We all have confidential data on disk: private data, corporate and government secrets. We also known how to destroy data on hard disks, as we've been doing this for years with multiple overwrites. But SSDs are different because of the complex controller, the flash translation layer (FTL). The FTL maps page locations to physical pages, and there are always more pages than a device advertises as spares. So techniques used on hard disks fail to work for flash.

Recovery of data on SSDs is cheap: it costs a few hundred dollars. To do the same thing on hard drives, instead of unsoldering a few chips, requires hundreds of thousands of dollars. With SSDs, there is a lot of leftover data, which is easy to recover. It is also possible to recover data at the physical level, even overwritten data. You need to grind flash to destroy it. SSDs write to the first free block, and to overwrite it SSD writes to the next available block, leaving the original data untouched. This is unlike hard drives, where original data can be overwritten.

Their lab examined state-of-the-art mechanisms used in SSDs to remove remnant data. First, they wrote some patterns to SSDs. Then, they "'sanitized"' the SSD, removed flash chips, and used a custom hardware platform for reading chips, bypassing the FTL entirely. If a drive completes its sanitization successfully, there is no stale data left over.

Some drives worked correctly, but some drives didn't apply erase commands properly. They would report success, but data would still be present. Some drives could be mounted as if nothing had happened. Other drives could be reset and some or all data could be recovered. Some drives used a cryptographic scramble, which involves writing all data using some form of encryption and then discarding the key. But since they could not confirm that the key was actually discarded, they didn't trust this approach. Encryption is actually commonly used in SSDs, as it has the effect of randomizing data, which is desirable when writing to flash. For this to work, the key must be durably stored somewhere, and they could not prove that this key had been destroyed after a cryptographic scramble.

They also found that ATA security erase worked sometimes but not at other times *on the same drive*. They suggested combining cryptographic erase and erasing data. First, scramble the disk and delete the keys, then perform a block erase on the entire disk, which then can be verified by reading all blocks and checking for erasure. If done in parallel, a 256 GB disk could be erased in 20 seconds. Perhaps drive manufacturers would implement this.

If you try overwriting, you don't know if you have written to all blocks on the SSD. Their experiment showed that typically two passes were sufficient, but sometimes it took 20 tries. This had a lot to do with past errors on the drive and a lot of things that the FTL hides.

Sanitizing single files is an even bigger problem. You want to get rid of a confidential document, such as browser history or files with credit card data or passwords for customers.

When testing overwriting of a 1 GB file, they tried various standards, and all left at least 10 MB. The NIST standard left almost everything (overwrite once considered sufficient for this standard). Their suggestion for sanitizing single files involves adding a scrubbing command to the FTL. Scrubbing erases all stale data—that is, blocks that are unused but have yet to be erased. But this is not as easy as it seems. Flash is arranged in pages, and these are part of erase blocks. So erasing a block could also erase pages in the block that are still in use. In high reliability memory, they found they could overwrite pages instead of erasing the entire block. But in the more common MLC (a denser, less reliable flash), you have a limited overwrite budget before the write fails. MLC is the type of flash that is cheapest and most common.

They suggest overwriting using the scanning method, which overwrites just the page. If this fails, the entire block must be copied and the block erased (and likely linked to a bad block list as it is now faulty).

In conclusion, Michael pointed out that verification is necessary to prove sanitization effectiveness. Hard drive techniques do not work, and having drive-level support for sanitization is a requirement.

Someone mentioned a study that shows that you cannot recover data from flash disks after they have been microwaved. Someone else asked if the trim command works. Michael said they looked at that, as it appears very interesting. The standard describes "trim" as a hint, meaning the drive doesn't have to implement it, and, in fact, most drives do nothing when given a trim command. Is there any research on recovering data from erased SSDs showing that particular patterns work better? Flash is different from disk, and the way erasure works involves using a higher voltage level that makes distributions of bits impossible to recover.

Aren't some of these issues, such as relocating bad blocks, similar for hard drives? This is definitely an issue for hard drives. The difference is that hard drives don't do this very commonly—perhaps 30 blocks, after a lot of use. SSDs have as much as 10% of their capacity set aside for relocation of bad blocks. Someone else pointed out there has been a lot of work on extreme magnetic analysis on hard disks, where people could go back and see data patterns and infer some of the data. Michael repeated that flash is different.

Apache Traffic Server: More Than Just a Proxy

Leif Hedstrom, GoDaddy

Leif started with the history of traffic servers. Inktomi produced the first commercial version in 1998. Yahoo! acquired Inktomi but ignored the traffic server initially, as all they cared about was search technology. In 2005, Yahoo! began to build a traffic server, and once they had gotten the Inktomi solution under Linux, it was many times faster than Squid. They wanted to open-source their solution, but doing this with existing code required a huge amount of work because of patents. So they started over from source, and in 2010 they achieved this. Nginx is also an awesome proxy server, Varnish is good, and Squid is still around. When you look at traffic servers, what you want to do is comparison shop. Leif presented a table showing features of various traffic servers and suggested you pick the one that fits your tasks the best.

Rather than just consider requests per second, where most traffic servers can do around 100,000 requests (and Apache traffic server does best), you need to look at latency. If you have 2 ms of latency per request and each page consists of 50 requests, that's a total of 100 ms of latency, so latency is important and a better benchmark (again, Apache traffic server does better).

Leif then explained proxy servers. Forward proxy can rewrite URLs which control which Web sites users visit, and cache content. A reverse proxy is transparent to the Web browser and uses rules to redirect, or forbid, access to remote servers. A reverse proxy can also cache content and do keep-alives. The last type of proxy is an intercepting proxy, which Leif called a "mixed bag," where the firewall forces users to go through a proxy. This can be done through an ISP that wants to save bandwidth through caching content. In general, when you think about proxies, you want to take advantage of caching. But the content has to be cacheable.

Caching improves performance for three reasons. The first is the TCP three-way handshake, which multiplies any latency by three before any request can even be made. The second is congestion control. If the server can only send three packets without getting an acknowledgment, and there is high latency, then only three packets can be in flight over that 100 ms. Having the proxy server close to the clients reduces latency to the client. The proxy server can also use keep-alives with remote servers, avoiding the costs of the three-way handshake. When Yahoo! tried a proxy with a keep-alive in India, they got huge performance improvement. The final issue is DNS lookups, which also involve latency. In particular, if you have, say, http://news.example.com and http://finance.example.com, each requires its own DNS lookup. On the other hand, if you use URLs such as http://www.example.com/news and http://www.example.com/finance, only one DNS lookup is required. In this case, you put a proxy near the servers and it can redirect traffic to a particular server (news or finance) as required.

The real problem is when you have millions of users all with powerful machines. Varnish has a great proxy, but you have

to have a single thread for each connection. Threaded code is also difficult to write and to debug. Squid uses event processing, which runs within one process, so there's no locking or potential races. But this doesn't solve the concurrency problem, because it uses only one core. Nginx gets around this by starting multiple processes, but then it has to deal with resource sharing between processes.

Traffic server takes the worse possible route: it does both. Traffic server has multiple threads, one or two per CPU, and runs several different types of threads: resource, event handling, and connection handling. Traffic server has lots less overhead than we see in Varnish. Great coders are required to work with multiple threads.

Configuration looks difficult, because there are many config files, but Leif pointed out that you will only be concerned with storage.conf, remap.config, and records.config. records.config is a normal key-value style, with plenty of comments, used to set flags, for example, and most of the defaults will just work out of the box. You will have to change storage.config since the server cannot figure this out itself. You also need to configure remap.config, especially if you are using the server in a reverse proxy environment.

Apache traffic server can use compression for stored objects, only uses 10 bytes to map to stored objects, and uses raw disk instead of the file system for performance. Traffic server has full IPv6 support on the client side but uses IPv4 on the server side. In the longer term, they want to add SSD to their stack of caches, instead of having just RAM and disk.

In summary, traffic server, and proxy servers in general, are great general-purpose tools. Proxy servers are outrageously fast (except Squid).

Someone asked about two features that chew up resources: content rewriting and header injection. Leif said that headers get rewritten using marshaling buffers. For rewriting content, use a plugin in your data stream. The plugin can inform the proxy if the content should be cached. Plugins are tricky to write, but there are examples.

Clusters and Configuration Control

Summarized by Rik Farrow (rik@usenix.org)

Sequencer: Smart Control of Hardware and Software Components in Clusters (and Beyond)

Pierre Vignéras, Bull, Architect of an Open World

Pierre Vignéras has built a system that shuts off power to systems and devices in an order determined by dependencies. The system was designed to work with one of the largest clusters in the world, Tera-100, composed of more than 4,000 nodes. Nodes can be shut down using commands, such as "ssh root@node halt", while many devices (e.g., switches) can only be shut down by turning off the power, using power distribution units (PDUs), for example. Dependencies include not shutting down a file server before the systems it relies on have shut down, or not turning off network switches before all network-delivered commands have been sent. The sequencer also needs to be fast and robust.

His system has three stages: a Dependency Graph Maker (DGM), an Instruction Sequence Maker (ISM), and an Instruction Sequence Executor (ISE). The input to the system consists of a table with one rule containing a dependency rule per each row. The DGM accepts this as input and creates graphs in the mathematical sense, then prunes the graphs. The output is an XML file with <par> tags indicating which sections can be completed in parallel. In tests on the Tera-100, they could shut down the system in less than nine minutes, compared to more than 30 minutes previously. Vignéras said this is an open source project.

Paul Krizak (AMD) asked what they use for feedback when running ISE. Vignéras said they expected the written code to provide feedback, the way a script provides return codes. Paul then asked how they know that an instruction, such as shutting down a system, has completed. Pierre said that the sequencer assumes each action is atomic, but it may also poll to ensure an action has been completed. Paul asked what type of system they run the sequencer on. Pierre said they use a Bull S6000, a big machine.

Automated Planning for Configuration Changes

Herry Herry, Paul Anderson, and Gerhard Wickler, University of Edinburgh

Awarded Best Student Paper!

Paul Anderson ran a slideshow presentation, as Herry was defeated by visa issues and wasn't able to enter the US. Herry (in a voice-over) began by pointing out that most common configuration management tools have a declarative approach, and that poses a critical shortcoming. Declarative tools imply an indeterminate order of execution which may violate a system's constraints. He illustrated this with an example of wanting to switch a client node C from server node A to server node B, then shutting down server A. If the configuration management tool executes commands out of order, shutting down server A before client C has a chance to mount a file system from Server B, there will be trouble.

They developed a prototype that uses IBM's ControlTier and Puppet, where ControlTier schedules changes and Puppet implements them. Their system works by collecting existing system state, translating it into a Planning Domain Defini-

tion Language (PDDL), having an administrator specify a declarative goal state, a planner generate a plan, and a mapper generate a ControlTier workflow, which sends Puppet manifest files that complete the work in the correct order. They tested their prototype by migrating an application from a private to a public cloud to address spikes in demand, with the constraints that there be no downtime, that the firewall be reconfigured, and that this be a true migration, not duplication.

Paul Krizak asked about the tools that make up the library, and Paul Anderson replied that their prototype uses a combination of various tools that are glued together: the standard planner, Puppet, and lots of glue. Someone else pointed out that they were pushing work on people to write the actions and prerequisites in order to produce the output. Paul responded that someone has to define actions for Puppet, and that the additional work to create prerequisites can be done over time. Norman Wilson pointed out that getting used to their system would not be that different from an earlier Sun system he had used, and Paul agreed. Paul also mentioned that sysadmins can forget things, and writing it down helps. Norman replied that he had never seen a system where he couldn't screw up something by leaving out a step.

Fine-grained Access-control for the Puppet Configuration Language

Bart Vanbrabant, Joris Peeraer, and Wouter Joosen, DistriNet, K.U. Leuven

Bart Vanbrabant pointed out that configuration management replaces needing to have root access on each machine. You manage configuration files on a central server, which performs root actions. This implies that if an attacker can insert malicious actions into the configuration, the attacker can affect all systems without being root. Their goal is to add access control to configuration management, but this is difficult to do, as there is no one-to-one mapping from configuration files to actions.

This work builds on ACHEL, a tool presented at LISA '09 (Bart Vanbrabant et al., "Federated Access Control and Workflow Enforcement in Systems Configuration") that they built to prove that this works with a real configuration management tool, Puppet. They use Puppet's compiler to produce an abstract syntax tree (AST), then normalize the tree before deriving the actions to be authorized. The normalization is necessary because there are multiple ways to do things such as user account creation. They use the XACML policy engine to perform the checking. Bart said that their tool does not support some Puppet constructs, such as switch cases that are handled at runtime. But their tool does work.

Paul Krizak asked if they had considered performing access control at runtime, letting changes go into the system, but refusing forbidden things at runtime? Bart said that it was possible, but with constraint languages it is almost impossible to determine which rules generated a forbidden action. Paul Anderson followed up by asking if they store information in the configuration about who made changes, and Bart agreed, but said that they had that ability in ACHEL and here they just wanted to test the tool. Anderson asked if they just look at changes to configuration, and Bart replied that they use the entire file, create the AST, and determine what changes have been made. Bart said you can do anything in XML if you are willing to write huge amounts of XML.

Invited Talks I: DevOps: Chef

GameDay: Creating Resiliency Through Destruction

Jesse Robbins, Opscode, LLC

Converting the Ad-Hoc Configuration of a Heterogeneous Environment to a CFM, or, How I Learned to Stop Worrying and Love the Chef

Dimitri Aivaliotis, EveryWare AG

No reports are available for these talks.

Invited Talks II: Panel

Summarized by Michael Wei (mwei@cs.ucsd.edu)

How University Programs Prepare the Next Generation of Sysadmins

Moderator: Carolyn Rowland, National Institute of Standards and Technology (NIST)

Panelists: Kyrre Begnum, Oslo and Akershus University College of Applied Sciences; Andrew Seely, University of Maryland University College; Steve VanDevender, University of Oregon; Guy Hembroff, Michigan Technological University

The main topic of discussion was how the university, which typically provides a rigorous formal education, could provide the kind of practical problem-solving skills that are necessary for effective system administrators. Each panelist brought with them the experience they had in training system administrators and the challenges they faced.

The first part of the panel covered the challenges of teaching and evaluating system administrators. In the words of Carolyn Rowland, "There's more than one way to skin a cat, and we all have the right answer." Kyrre Begnum agreed—sometimes a class can be about a specific way to solve a problem, but the student may have inherent knowledge on how to solve that problem in another way. However, the exam may test

only how well a student solves a problem in that particular way. As a result, a student who comes in with the inherent knowledge may actually be at a comparative disadvantage, and it may take time to get that student to accept that there are several ways to solve problems.

Andrew Seely felt that problem-solving in system administration is not a thing that can be taught. In the world of computer science, you can teach a student to build classes and data structures, and they can apply that knowledge to build a program. In the world of system administration, it is not clear what those basic building blocks are. Carolyn asked the rest of the panel whether they agreed. Kyrre Begnum and Guy Hembroff both disagreed. They gave the example of scripting and network infrastructure courses as similar building blocks that they teach at their universities, and felt that the problem was more that the literature for teaching these skills is limited and not necessarily practical.

The next section of the panel talked about the missing identity of system administration. Steve VanDevender pointed out that there was no standard that defined what a system administrator is, so attempting to figure out what skills are necessary to train good system administrators may be a futile question to ask until the identity of system administrators has been hammered out. Kyrre agreed, further pointing out that the lack of textbooks suitable for classroom use on system administration means that what system administration programs teach ends up being a hodgepodge of what the program administrators feel to be relevant at the time.

The panel ended with the difference between technical and theoretical education. There was general consensus that a good education in system administration would require both. Andrew Seely finished by explaining that other disciplines have fundamental theories, while system administration is composed mainly of best practices. If best practices could be changed so they did not expire with every new technology, then system administration could be furthered as a discipline with both a theoretical and a practical grounding on its own.

Security 1

Summarized by Timothy Nelson (tbnelson@gmail.com)

Tiqr: A Novel Take on Two-Factor Authentication

Roland M. van Rijswijk and Joost van Dijk, SURFnet BV

Roland van Rijswijk began by reminding us of a painful fact: we enter many username-password pairs throughout our lives. Entering a username and password remains the standard login paradigm, in spite of its risks: password re-use, for instance. Two-factor authentication is the act of logging in with two separate credentials, often something that you "have" plus something that you "know" or "are." For instance: a password and a one-time-use code, or a keycard and a fingerprint.

This work, tiqr, exploits the fact that nearly everyone has a mobile phone, and those who have them tend to carry them around. Even more, we tend to notice if they go missing, which is more than we can say for authentication tokens. However, most phone-based authentication solutions make their user retype complicated codes that appear on the phone screen. That requirement is an issue, especially for visually impaired users. tiqr makes progress in that direction.

Since the room had two separate projection screens, we were treated to a demo of tiqr. The screen on the left showed a Web browser, and the screen on the right showed an iPad display. Van Rijswijk showed tiqr scanning a QR code that appeared on the Web browser to generate an authentication response, which the iPad then transparently sent to the Web site via its own Internet connection (protected by 256-bit AES encryption). The speaker closed by showing us how tiqr can facilitate SSH login using an ASCII-art QR-code, and he briefly discussed an external security audit of tiqr. The external auditors said that tiqr's security was more than adequate.

What would happen if the user is already using the Internet on their mobile device? The QR codes contain a URL schema, and users merely have to click the code to switch apps. What about tiqr's API? They have a Web API, a demo version of which is available in PHP. They are working on making the API fully RESTful. What happens if the mobile device has no Internet access? In that case, tiqr will fall back to classic mode, giving the user a code to type into their browser manually.

Building Useful Security Infrastructure for Free (Practice & Experience Report)

Brad Lhotsky, National Institutes on Health, National Institute on Aging, Intramural Research Program

Brad Lhotsky, a recovering Perl programmer, is a Security Engineer at the Institute on Aging. Nobody likes getting "the call" from him, and his boss doesn't care about security. Lhotsky has found that security people need to justify their existence beyond saying that they make things more secure—to the organization, security is a minor concern compared to actually doing research. Their paper is about tools their team has built for useful security.

They do comprehensive, centralized logging of network events. They used to use syslog-ng, but some of its features

are not free. They now use rsyslog, which supports native encryption (among other things), but has a somewhat ugly configuration. They use postgreSQL for long-term storage of the events. Keeping logs in a RDBMS makes using the data easier. For instance, they can execute R (a language for statistical analysis) queries on the data.

This detailed logging lets them do correlation of data and provide it in a way that is helpful to their help desk operators. They answer questions such as "Where has a user logged in from?" "What devices has a user been assigned?" "Where is the user logged in now?" and "What is their network history?" They also feed the logs to Snort, an intrusion detection system, to discover potential data-loss risks.

An audience member noted that building infrastructure is one thing, but convincing auditors is another. They asked whether Lhotsky's team is externally audited. Lhotsky answered that they are indeed audited by health and human services.

Local System Security via SSHD Instrumentation

Scott Campbell, National Energy Research Scientific Computing Center, Lawrence Berkeley National Lab

Awarded Best Paper!

Scott Campbell's organization supports 4000 users worldwide, all of whom have shell access. NERSC doesn't want to interfere with their users' work, but does want to intercept intruders on their machines. They modified their SSH service to log sessions, authentication, and metadata, which involved identifying key points in the OpenSSH code.

To filter the raw data, they use the Bro intrusion-detection program, which is fed data from the captured SSH messages. Multiple data sources are mapped to one log file, which is then converted to a stream of Bro events. All the analysis is done in Bro, which looks at each event atomically to find what is considered hostile or insecure. For instance, they might want to detect someone remotely executing "sh -i". Their SSH modifications also allow them to collect soft data, listening in on intruders to learn their methods and skill levels. They actually captured a discussion between two intruders on their system.

Campbell's team plans to extend the work they have already done, to perform better analysis. For instance, they could analyze user behavior or pass more information (such as process accounting data) to Bro.

Someone asked about arbitrary whitespace, that is, what happens if an attacker uses "sh -i" (with extra spaces) instead of "sh -i"? Campbell answered that they do not currently catch that, but, with minor modifications, they could. Rik Farrow asked whether they capture passwords. Campbell replied that they can, but they try very hard not to. It is easier to capture passwords than not capture them, but they do their best to preserve their users' privacy.

Someone else asked what the false positive rate is. Campbell replied that his team is paged very rarely. Had they actually captured any intruders? While they have not captured anyone in the physical sense, they do intercept intruders 12 to 20 times per year. The full paper even contains some example conversations between intruders. Do they capture everything going on, because that would be a lot of traffic to log? They try to reduce unnecessary logging, stopping recording replies from the server after a period of user inactivity.

Invited Talks I: DevOps Case

The Operational Impact of Continuous Deployment

Avleen Vig, Etsy, Inc.

DevOps: The past and future are here. It's just not evenly distributed (yet).

Kris Buytaert, Inuits

No reports are available for this session.

Invited Talks II: Infrastructure Best Practices

3 Myths and 3 Challenges to Bring System Administration out of the Dark Ages

Mark Burgess, CFEngine

Summarized by Michael Wei (mwei@cs.ucsd.edu)

Mark Burgess began by describing the dark ages, a time when brute force was essentially the way problems were solved. In system administration, brute force is still widely used in many areas. For example, technicians are often attached to service tickets and thrown at problems wherever they show up. Mark proposed that there are three waves of innovation in system administration: the manual, brute force wave of the past; the automated wave that we are starting to move into; and the knowledge wave, where we build machines to serve us rather than just complete a singular task. Today we are moving from the second wave to the third wave, but we still face second wave myths that keep us tied to the first and second waves and third wave challenges that keep us from moving into the third wave.

Mark challenged three myths. The first is ordered sequential control. He believes that we are taught that sequentially is better, when it is not always the best solution. In fact, we sometimes artificially create sequentiality when no sequence necessarily exists. Mark gave an ordered XML document as

an example. The second myth Mark challenged was that of determinism and rollback. Mark said there is a belief that we can simply roll back changes to fix problems, but in reality this is a myth. If we accidently change our firewall configuration and a computer gets infected by a virus, we cannot simply roll back the firewall configuration; we need to repair the computer as well. This kind of rollback thinking is dangerous because it forces us to focus on the mistake instead of the outcome. The third myth Mark examined was "hierarchy or bust." The kind of hierarchical thinking that many system administrators thrive on creates many points of failure which are dangerous, while marginalizing problems.

After enumerating these three myths, Mark presented three challenges. The first is emergent complexity: we have to be able to accept that systems are becoming increasingly complex and accept diverse systems as they are. The second challenge is commerce alignment: we have to accept commerce's role in IT if we are to allow systems to grow. The final challenge Mark presented is knowledge management. We must be able to share knowledge efficiently, because individual skill is not replaceable.

Linux Systems Capacity Planning: Beyond RRD and top

Rodrigo Campos

No report is available for this talk.

Plenary Session

One Size Does Not Fit All in DB Systems

Andy Palmer, Global Head of Software and Data Engineering, Novartis Institute for Biomedical Research

Summarizedy by Ming Chow (mchow@cs.tufts.edu)

Andy Palmer talked about the changes and challenges of database fitting for biomedical research and scientific applications. The game has changed: the idea of one database engine (e.g., Oracle) fitting all research and scientific applications is no longer applicable. The reason for this is big data. Over the years, the amount of data and information has grown exponentially compared to storage of information, which is near flatline. The traditional DBMS architecture is roughly 25 years old and was designed mostly for write-based applications. The architecture has largely ignored CPU, disk, and memory trends. In short, while the RDBMS market became a billion-dollar market, traditional DBMS architecture has not caught up with the times. Vendors have addressed the issues by adding band-aids to their products such as bitmap indices.

The idea of OLTP (Online Transaction Processing) ran out of steam in the 2000s with the introduction of Big Table and Hadoop. Since then, there have been new database engines such as Mango, Couch, and Cassandra. Andy asked the question, "How do you pick the right engine for a specific workload?" He said we need frameworks and tools to characterize workloads to match engines based on empirical characteristics. This paradigm differs from "Oracle or MySQL is the answer for everything": now you need to stop and pick the right engine before starting a project. For scientific data, there are many options, including the traditional row store engine, column store, file-oriented, document-oriented, array-oriented, and federated.

Working with big data in a research and scientific environment is also much different from before: operations of new engines require integrated skill sets, including database, system administration, and clustering. Andy described the Novartis IT Data Engineering teams, which do hands-on application of new database technologies. The teams consist of an interesting mix of people who work at several different locations and frequently cross-train.

A significant issue with working with big data in a scientific and research environment is that the cost of fixing errors (e.g., quality) has grown exponentially. Andy stressed that data quality starts at the point of data creation. For an application, 70% of the work is creating or working with unstructured and structured data.

Andy noted a few myths, including "Oracle is the answer to everything," "one database for each app," and "one integrated database for everything." Scaling has been a huge challenge in working with petabytes of data. Andy noted that at Novartis, they have run out of space over 10 times this year alone. Unfortunately, given the importance of scaling, he has found it impossible to solve data challenges with Oracle. Working with big data involves numerous schema changes, and new data sets need loading and analyzing, which can be problematic in Oracle. At Novartis, a number of new database engines have been tested, including Vertica for gene expression, MapReduce, and CouchDB.

In-house, a few big questions have been asked as part of the framework to determine which is the best database engine to use in certain applications. The questions include the best performers, solubility, scalability, performance, and ease of adoption and integration. Andy and Novartis have taken a quantitative approach to answering these questions when working with new database engines.

Andy concluded the plenary by stressing the importance of considering up-front which database engine matches workload and that the operations of new engines require integrated skill sets.

From Small Migration to Big Iron

Summarized by Carolyn Rowland (carolyn@twinight.org)

Deploying IPv6 in the Google Enterprise Network: Lessons Learned (Practice & Experience Report)

Haythum Babiker, Irena Nikolova, and Kiran Kumar Chittimaneni, Google

Awarded Best Practice & Experience Report!

Irena's first slide summed up the move to IPv6: 96 more bits, no magic. She spoke of the IANA IPv4 exhaustion in February 2011 and how we are not reclaiming any of the current IPv4 address space. Sometime between 2012 and 2015 we will have assigned all IPv4 addresses. With the proliferation of smartphones, IPTV, virtualization and cloud, P2P, and network-aware devices, we are only increasing the demand for addresses. Google was motivated internally because they were running out of RFC 1918 addresses. They tried using NAT as a solution, but this just increased the complexity of the environment. Additionally, they thought that by implementing support for IPv6, they would help us break out of the chicken-or-egg problem: service providers in no hurry to deploy IPv6 due to lack of content, and content providers explaining the slow rate of adoption as being due to lack of IPv6 access to end users. To make a positive contribution to the Internet community, they knew they had to enable IPv6 access for Google engineers, to help them launch IPv6-ready products and services. Google has an internal process they call dogfooding—living the same experience as their users (eating your own dogfood). Since their internal teams wanted to work on IPv6 connectivity for Gmail and YouTube, etc., the networking team had a real need to provide them with a network that was IPv6-ready, so that they could help develop, test, and dogfood these products.

The Google approach was: IPv6 everywhere! They needed to build tools, test and certify code for various platforms, decide on routing protocols and policies, plan for IPv6 transit (WAN) connectivity, create a comprehensive addressing strategy, and request IPv6 address space. They began with dual-stack, using tunneling as a last resort. They assigned IPv6 address space as /48 to each campus, /56 at the building level, further dividing into /64 per VLAN. GRE tunnels were not a good option, because they created MTU fragmentation and other issues. This was pretty challenging, because lots of ISPs don't yet support IPv6. They tried DS-Lite to encapsulate IPv4 packets inside IPv6 networks. They also used SLAAC (stateless address autoconfiguration) instead of DHCPv6 for host address assignment. Sometimes they were the QA department for the vendors selling IPv6 products. The vendors were not eating their own dogfood. "Nothing could create a real-world scenario in the lab, so we discovered many IPv6-related bugs during deployment and testing." Also, IPv6 was still processed in software in many vendors' hardware platforms. Internally, training and education were the biggest challenges, although early information helped to fight FUD. They had some DevOps challenges as well; the engineers wanted to deploy new technology immediately, while the operations people were not so interested in early adoption. Words of wisdom: migration is not a Layer 3 problem, it is a Layer 7–9 problem. Migration is simple, but it takes time. A phased approach gradually builds skills and confidence. You want to make sure you design for the same quality standards as with IPv4.

Has Google worked with any VOIP technology? The Google Network Engineering Team is only responsible for migrating networks. Rik Farrow asked how organizations can allocate resources for an IPv6 migration. Irena admitted that most resource allocation is for IPv4 projects. If you have eight projects and two are IPv6 and six are IPv4, people are going to go for the v4 projects, because v6 is not a priority. An audience member said that now it's up to some other people to produce the other half of the equation. If you build it, they will come. So, are others starting to build stuff with IPv6? A lot of the Google external products are IPv6-enabled now. You can go to http://www.google.com/intl/en/ipv6/ to request access to Google products over IPv6.

Bringing Up Cielo: Experiences with a Cray XE6 System, or, Getting Started with Your New 140k Processor System (Practice & Experience Report)

Cory Lueninghoener, Daryl Grunau, Timothy Harrington, Kathleen Kelly, and Quellyn Snead, Los Alamos National Laboratory

Cory started his presentation with the obligatory HPC statistics slide. Cielo consists of 96 racks, 96 nodes per rack; two 8-core 2.4 GHz processors per node; 32 GB memory per node; a Torus network: 4.68 GB/s links; 142,304 cores; 284,608 GB total compute memory; and 1.11 PFlops measured speed. Cielo is currently used for simulations at Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratory, and Los Alamos National Laboratory (LANL). This project was a collaboration between Los Alamos and Sandia (but lives in Los Alamos). Other computers are used to support the management of Cielo. Cielito is used for testing before scaling up to the big system; Smog is used to try out new Cray software stack releases, configuration management, and other challenges of bringing up the big system; and Muzia is Sandia's one rack system (similar to Smog).

Cielo can be used as one big system to run one job or can be used to run smaller jobs. They use CfEngine for configuration management; this adds a layer of abstraction to make the nodes all seem the same. Cray provided monolithic install

scripts, hard-coded RPMs, and versions. Cory said his team asked lots of questions to try to understand the Cray configuration tools (e.g., one of the default install scripts checked whether it was being run on an interactive terminal, so you couldn't automate the install with | yes). Cray used a couple of different schemes, such as /etc/alternative (Debian), module files (more flexible and dynamic), and default links (links within the software tree to different versions of software). The question Cory's team asked was, "Do we hack the scripts so we can make it easier to automate?"

They began tackling these challenges with some CfEngine hackery. This got them past 80% of the challenges. They were able to tweak rules used on other production systems for the Cray. They resorted to outside scripts for the parts CfEngine didn't handle. Still, CfEngine could run these scripts keeping some form of automation. Sysadmins could still work with module files and changes would trickle down. Cory said they were basically telling Cray "your way is wrong," which isn't the message they wanted to send. A lot of people use Cray's tools just as intended, but Cory's team decided to do it a different way. Despite this deviation from Cray's standard strategy, Cray was extremely helpful. Cory's team submitted bug reports, Cray was helpful with the weird stuff, "and there were rainbows everywhere," said Cory. Maintaining positive vendor relations helped them. Cory admitted that the team needed Cray's support to do what they needed to do; without them, the team would still be fighting about the answers. Getting early access to test systems (Smog and Cielito) allowed the team to solve problems early before moving to the big system. Configuration management is a worthwhile investment. The team was able to rebuild the whole system in a day or two instead of weeks. They did this several times when they were reviewing security.

One audience member pointed out that Cory had provided a long list of things Cray did wrong. What did Cray give them? Cray gave them a big integrated system that was able to run jobs across the whole thing very quickly, and an underlying control system to control the booting, the management, reliability, and serviceability. They provided a lot of the underlying pieces. How long did it take from first power-on to production use of the system? Cory estimated 3–5 months. It wasn't something they tracked; there was a lot of shaking out of the system. Did Cory know the cooling number per rack? Cory looked to his teammates in the audience and someone responded, "5 or 6 MWs total for the whole machine."

Capacity Forecasting in a Backup Storage Environment (Practice & Experience Report)

Mark Chamness, EMC

Mark pointed out that IT behavior is reactive. We react to 100% disk capacity, failed backups, and late-night alerts. In those reactionary efforts, we take shortcuts such as deleting files to make space on a volume or decreasing the retention policy on backups so that we can hit our window. The solution to this is proactive prediction. First, start by choosing a time frame (e.g., the past 30 days). Next apply linear regression (e.g., over the next 90 days), then choose the time frame for notification (e.g., the next 90 days). Finally, run your analysis and generate a report. However, using a fixed time frame results in poor predictions and doesn't adapt for behavior. They tried two time frames (60 days and 7 days) and picked the best one. This also failed miserably, because both were wrong. The optimal prediction occurs when you use all possible models and choose the best one, selecting the maximum value of R(squared) (regression sum of squares). The maximum R(squared) occurs at the change in linear behavior. This is the best model to fit the data and to use for forecasting.

Mark showed some example graphs of different scenarios. This painted a clear picture so that you could predict the date of full capacity. The majority of systems were able to be modeled using linear regression. Mark used an example of a system that was at 60% for a long time and then started to grow. The graph then displayed a rollercoaster behavior, where system capacity went up and down. The schizophrenic graph, where capacity varied all over the place, did not work so well with linear regression as a predictor. In order for a model to work, one needs the following: goodness of fit r^^2 > 0.90; positive slope; forecast time frame < 10 years; sufficient statistics (15 days); and space utilization minimum of 10%. Mark raised the gotcha that the last data point trumps all. You can model a system fairly accurately using this method, until the sysadmin deletes a bunch of data to create free space. Then the system is no longer behaving in a predictable way. You can no longer predict behavior if the error is too great between the last data point and previous data.

Were there models that could have been used other than linear models (e.g., logistic regression)? If you attempt to use a more complicated model and show it to sysadmins or customers, it becomes too difficult to explain. More complicated models also tended to behave erratically; they would often go off exponentially and produce some strange predictions.

Why did Mark decide to model the percentage of capacity instead of rate of growth? Linear regression provided growth in GB/day: "Here's how much it's growing per day; here's when it is forecasted to reach capacity." He had the

ability to capture that data to model it. Did Mark consider daily spikes? No, the model does not monitor daily spikes, but his group uses Nagios to alert for 90% capacity, etc. On what percentage of systems does this model work? He could accurately model 60% of his systems. Did Mark see this as a tool running all the time that you push out to a consumer or is it a sizing exercise for presales (at EMC)? All of the above. When people buy a system they get support; they can go to the customer support portal and configure alerts. EMC will soon be releasing a new product called DD Management Station. It will allow a customer to manage 60 Data Domain devices. The last questioner said he didn't see any control for human intervention. Mark reiterated his "Last data point trumps all slides" statement. This is the example where the sysadmin went in and deleted a bunch of data. The model would not publish a prediction because of the error between previous data and the last data point. It's important for a model to know when it doesn't know.

Content-aware Load Balancing for Distributed Backup

Fred Douglis and Deepti Bhardwaj, EMC; Hangwei Qian, Case Western Reserve University; Philip Shilane, EMC

Fred started with a primer on deduplication: basically, you can avoid storing data at all if it already exists on backup. Using hashes, the system can check for changes; if there are none, then the system won't transfer any data. Deduplication is common in today's backup products. But what is the impact of deduplication? You desire affinity, sending the same client to the same appliance so it will deduplicate well. If you send a client to a new system, then all of the data will be transferred, because there is no sameness. You can also send similar systems to the same appliance, because they will deduplicate well, due to similarities in the data. Doing this can reduce capacity requirements and can improve performance of backups, because you copy less data.

For performance reasons, you don't want to send too much data to one place, so ensure you support simultaneous backup streams. One gotcha is not sending everything to the same appliance just because it deduplicates well. When sizing a system for load-balanced distributed backup, look for cases of affinity with high overlap among a small number of hosts. Virtual machines are a good example of good overlap. If you have a lot of similar hosts, then you can probably put them anywhere and they will deduplicate well. There are small penalties for migration, but the biggest penalties come when a client doesn't fit on backup at all.

Fred posed the question, "What are some algorithms for load balancing?" Fred's team started with brute force; this works okay if the system isn't too loaded. The first approach is "random." They started at a random appliance and used it as long as the appliance wasn't overloaded. If it had no storage capacity, they searched round-robin from there to find one with capacity. They took the best of 10 random configurations, using a cost metric that considered capacity, throughput, client movement, and other factors. After that, they tried bin packing: assigning an appliance based on size from large to small. Next, they tried simulated annealing, which starts with the bin-packing configuration but then iteratively adjusts the configuration to try to find a better one. This model is willing to temporarily move through a worse configuration to try to avoid local minima. To evaluate the algorithms, they used a synthetic workload with clients added incrementally. Appliances were occasionally added as well, and 1/3 of the existing assignments were dropped each time an appliance was added, to avoid always starting with the older appliances already overloaded. "Make sure to stress overlap affinity." Cost was a first attempt at weighing the relative impact of different factors, but it really has to be evaluated in practice. All of the cases have a high cost, but simulated annealing was the best of the worst. Fred referred back to the paper for more information on overlap computation, more examples, overhead analysis of simulated annealing, how to penalize things for not fitting, and impact of content awareness.

Matt Carpenter asked if Fred's team looked at any analytics other than simulating annealing. Fred responded that the team started with bin packing. They wanted to understand the right way to deal with the least movement while still getting the best result. They admitted that there may be other heuristics. Had they considered network latency issues in terms of the cost, assuming flat space/local with no cost to move data? Fred said that there was no assumption that there would be a greater cost to move from one appliance to another. They assumed the same cost per appliance. There is a moderately high cost if you miss the backup window, because moving a client to a different server means recopying data. This is still considered a moderate penalty compared to not being able to perform the backup at all. Someone commented that one of the issues is to understand exactly what "cost" means. Fred replied that it would have to be tuned for a particular environment. Rik Farrow asked if Fred was looking at the size of the content, not the actual content of the data. Fred said that they were looking at the size of blobs to see how full an appliance was and at fingerprints of the content to decide if a chunk of data on one client matched a chunk on another client. He also said that you may need to run something that scurries through the file system and creates the fingerprints. If you are using Networker and it writes one big file for a client, comparing the fingerprints that correspond to that file would mean never having to go to the raw file. Matt Carpenter asked about skipping over

full devices: he said there might be value to moving stuff off a full device, because the remainder might deduplicate well with the data you need to store. Fred replied that this is what simulated annealing might do, by moving one client away from an appliance, then putting better data on the one that just freed up space.

Invited Talks I: DevOps: Core

SRE@Google: Thousands of DevOps Since 2004

Thomas A. Limoncelli, Google NYC

Summarized by Ming Chow (mchow@cs.tufts.edu)

Tom Limoncelli started his DevOps talk by revisiting the '80s, when there was no monoculture and the software engineering methodology was based on the waterfall method. Back in those days, developers didn't care about operational matters after shipment, software developers were not involved with operations, no bug tracking system was necessary, and new software releases were far apart. Then came the '90s, with the Web and the modem. Software moved to the Web, but servers required producers and operators. The users used Netscape and Internet Explorer. Despite the new software development shift, the waterfall method still kind of worked. With the 2000s, everything changed: speed mattered, there was pressure to be first to market, there was feature one-upmanship, shipping constantly to compete, and reliability mattered (which was also a selling point). The major shift also caused tensions between developers and operators, where developers stress changes while operators stress stability.

Tom introduced how DevOps works at Google. The goal is to improve interaction between developers and operators. He introduced the Google Site Reliability Engineer (SRE) job role. The model is based on extreme reliability, high velocity, and high rate of change. In a nutshell, developers run their own service, and SREs create tools and services. This creates a workforce multiplier effect. Tom described the process of product creation at Google. First there is creation, followed by a live readiness review, then a launch to production, followed by a handoff readiness review, considered only after developers have run the product or tool for six months. Products and services that receive SRE support are those that have low-operation burden or high importance (e.g., Gmail), and those that address a regulatory requirement.

Tom also described the product handoff process at Google. The process involves reviewing frequency of pages and alerts, bug counts, maturity of monitoring infrastructure, and production hygiene. If a product does not do well in some of these areas, then it goes back to the drawing board. Tom also mentioned that products that are approved the fastest for handoff are those that worked with SREs early.

In order to make DevOps work at an organization like Google, Tom listed management support and balance of power between SREs and developers. Skills required include deep understanding of engineering issues (system administration, software design). An SRE role is half engineering and half operations. Tom also presented how development life is made easier at Google with tools, frameworks, monitoring, and plenty of information resources, including training, launch checklist, and mailing lists.

Tom went on to describe the release engineering policy at Google. How it works is based on the idea of the canary in the coal mine. A product or service is first run on a test cluster. Then it is run on some percentage of machines in a cluster. Product teams are given a reliability budget (tokens). New pushes are based on budget. Tom has found that more pushes during this phase equates to more rollbacks. The reliability budget for release engineering is numbers-based and gives incentives for developers to test harder.

In summary, Tom said that the model has worked well at Google. First, there is joint responsibility even after adoption of a product or service. For SREs, developers are committed to fixing issues so they will not be supporting junk, and it gives developers production experience. It has even removed the adversarial quality of a lot of relationships. Tom advised those in the audience that in order to have this model in their organization, developers must be empowered, practices must be adopted, and there must be management support.

Deployinator: Being Stupid to Be Smart

Erik Kastner and John Goulah, Etsy, Inc.

Erik Kastner started with an overview of Etsy. Etsy receives over one billion page views per month, has approximately 100 engineers, values speed and agility, and tries to limit barriers. The company believes in turning ideas into code within minutes, in open source software, that optimizing now leads to happiness, and that sad engineers are bad engineers. The development process at Etsy is an embedded reaction to stupidity; there is no fear, they don't aim for perfection, and correctness the first time is not important. In fact, the idea is to be wrong as fast as possible. Etsy accomplishes this by good communication, trust, openness, and constant improvement. In 2009, there was a single deploy master, developers rolled back in fear, and all deploys took all day. Currently, anyone at Etsy can deploy, there are no rollbacks, and developers deploy all day. John went on to describe the culture at Etsy. Doing the dumbest thing possible lets you learn as much as possible, such as committing to trunk, putting configuration into code, and having blameless post-mortems. Etsy feels that what

makes this works is to graph everything. John displayed a graph on memcached and change-related incidents: there were six in 2010.

Erik Kastner then described the Deployinator tool, which is released on http://etsy.githib.com. The tool is based on the Staples easy button, for building scalable Web sites and for deployment and monitoring. The monitoring is done via a dashboard, and practically everything is graphed. The Deployinator uses a combination of technologies, including SSH, rsync, and a number of Web servers that are synced to the deploy host. Erik then discussed how Deployinator can be used for iOS: it has been used successfully in that iOS code was deployed to a Mac Mini, then to TestFlight. Finally, Erik challenged the audience, "What's stopping you?" Just know what you're optimizing for.

Invited Talks II

Summarized by Deborah Wazir (dwazir@gmail.com)

Fork Yeah! The Rise and Development of illumos

Bryan M. Cantrill, Joyent

Bryan Cantrill gave a lively presentation on the history and current status of the illumos operating system. Beginning with the transition at Sun from SunOS 4.x to Solaris, Cantrill described the struggles to develop a version of Solaris that worked well, which were hindered when management was in charge, but was finally accomplished once the engineers took over. At this point, radical engineering innovation ensued, with features such as ZFS, DTrace, and Zones invented because engineers felt they should be part of the OS. Once Sun open-sourced the OS, the OpenSolaris community flourished, until Sun became too hands-on. When Oracle bought Sun, introducing a totally alien organizational philosophy and mission, several key engineers left. These were followed by many more, after OpenSolaris was effectively killed by Oracle's internal decision to stop releasing Solaris source code.

Meanwhile, illumos development had begun, with illumos meant to be an entirely open downstream repository of OpenSolaris. After hearing Cantrill describe the engineer-led culture of innovation at Sun, it became easy to understand the exodus of engineers after Oracle took over, and why illumos represents a continuation of that same culture. The project greatly benefitted from the participation of engineering talent that had left Sun. Cantrill emphasized that illumos provides innovations and bug fixes that will *never* become part of Oracle Solaris. At this point, he brought up a terminal window and demonstrated some of the new features and enhancements.

After the demo, Cantrill discussed numerous specific improvements added to ZFS, DTrace, and Zones, and new features such as the port of KVM from Linux to illumos. He also named and thanked the individual engineers who worked on these features. He went on to discuss different illumos distributions that are already available, which complement each other by addressing different deployment environments: desktop, server, and cloud.

Cantrill branched off from his prepared slides from time to time, sharing his personal experiences and interpretation of Sun's engineering culture, the Oracle acquisition, and Oracle leadership. These digressions received enthusiastic appreciation from the audience.

Afterwards, Cantrill was asked about the state of illumos's collaboration with other distros regarding DTrace. He replied that the most complete port of DTrace is on the Mac, with the FreeBSD implementation not quite as far along. illumos is collaborating with other distros as well. Another audience member asked if recent innovations to DTrace would flow to Apple, and Cantrill said that he expected Apple to take all of them.

GPFS Native RAID for 100,000-Disk Petascale Systems

Veera Deenadhayalan, IBM Almaden Research Center

Veera Deenadhayalan presented a feature recently added to the GPFS parallel file system, GPFS Native RAID. The presentation slide deck contains excellent diagrams that clearly illustrate all the concepts discussed. He began by explaining that disk drive performance has not kept pace with improvements to other components such as CPU and memory. Therefore, to produce comparable increases in system performance, many more disks have to be used.

Before describing GPFS Native RAID, he covered some background concepts and characteristics of the GPFS parallel file system. He also discussed challenges to traditional RAID, which leads to performance and integrity problems in 100,000-disk petascale systems. When you have that many disks, disk drive failures are expected to happen daily, resulting in performance degradation when the disks are rebuilt. There is also more incidence of integrity issues.

IBM's solution to these issues is to remove the external RAID controller and put native RAID in the file system. This results in higher performance through the use of declustered RAID to minimize performance degradation during disk rebuilds. Extreme data integrity comes from using end-to-end checksums and version numbers to detect, locate, and correct silent disk corruption.

The physical setup is an enclosure with 384 disks, grouped four disks to a carrier. RAID is set up such that each declus-

tered array could tolerate three disk failures. In a declustered array, disk rebuild activity is spread across many disks, resulting in faster rebuild and less disruption to user programs. When only one disk is down, it is possible to rebuild slowly, with minimal impact to client workload. When three disks are down, only 1% of stripes would have three failures, so only those stripes need to be rebuilt quickly, with performance degradation noticed only during this time. Regarding data integrity, Deenadhayalan not only discussed media errors, but gave a thorough coverage of "silent" undetected disk errors with their physical causes: distant off-track writes, near off-track writes, dropped writes, and undetected read errors. To cope with these issues, GPFS Native RAID implements version numbers in the metadata and end-to-end checksums.

GPFS Native RAID provides functions to handle data integrity, such as disk rebuild, free space rebalancing upon disk replacement, and scrubbing. Disk management functions analyze disk errors and take corrective action. By maintaining "health records" of performance and error rates, the system can proactively remediate potentially failing disks.

To the Cloud!

Summarized by Rik Farrow (rik@usenix.org)

Getting to Elastic: Adapting a Legacy Vertical Application Environment for Scalability

Eric Shamow, Puppet Labs

Eric explained that elastic means using a cloud as Infrastructure as a Service (IaaS that they get machines to use, perhaps with an OS installed. He then said that elasticity requires automation. You first need automated provisioning of servers, at a minimum. From power-on to application, startup needs to be much more automatic. More complex issues include when to expand or contract the number of servers.

As sysadmins, we tend to see the big picture, but devs understand the metrics when an app is impacted. If you lose a server but the business keeps running without interruption, then you can wait until tomorrow morning. Eric then asked how many people share logs with development teams. About 10% raised hands. Sysadmins need to help make developers aware of the production environment so that they can make decisions based on fact rather than assumptions.

Eric suggested monitoring everything, but not focusing on anything at first. Latency is variable in the cloud. You need to be prepared for change. Impose sanity limits on builds and teardowns. How many and how fast can you provision/destroy? Consider having a pool of offline servers. If you have spare db servers ready to go, you don't have to wait 20–30 minutes for them to come up. To automate, you must have DHCP, PXE, DNS, OS, and Patch provisioning, all with APIs or script-based management.

Rik Farrow asked about their application. Erik responded that it published a number of newspapers. Rik then asked about the uncertainty involved in moving to the cloud. Erik responded by talking about what happened when Michael Jackson died, and one of their publications was the top hit on Google News. Their applications could handle the first spike, but it was the continuous pounding that led to slow death. You definitely want to wake up a human at some point. They also turned off parts of some pages, such as banners. Did they have access to the TCP stack, and other aspects they could optimize? They didn't have control of that in most environments, and generally just decided not to count on that. Another person asked how to protect their data. Eric said that there were lots of protection techniques, such as using encrypted tunnels for LDAP. But if your data really must stay private, don't put it in the cloud.

Scaling on EC2 in a Fast-Paced Environment (Practice & Experience Report)

Nicolas Brousse, TubeMogul, Inc.

Tubemogul has grown from 20 to 500 servers, four Amazon regions, and one colo, and requires monitoring of 6000 active services and 1000 passive. They like to be able to spin up or shut down servers as needed, but it is difficult to troubleshoot failures in Amazon. Nicolas described a single point of failure where the image would get stuck in fsck on boot; they had to revert to an earlier image, and that image didn't match their current configuration, requiring editing the configuration on each instance.

They had started with a Tcl/Tk script called Cerveza and rebuilt it in Python after the meltdown. Instead of using customized AMI, they used public Ubuntu AMI images to reduce maintenance, and cloud-init for easy pre-configuration of new instances. They used Puppet for configuration management, and Ganglia and Nagios for monitoring. In conclusion, Nicolas suggested using configuration management tools early, keeping things simple, and not forgetting basic infrastructure management, such as backup and recovery processes. They are hiring: http://www.tubemogul.com/company/careers.

Bill LeFebvre asked how their instances self-identified. Nicolas said they start a server and keep this info in SimpleDB to find the right profile. Bill then wondered what key info they used per server. Nicolas answered that they provide a recognizable hostname during the startup process. Someone else

asked if they were using trending data for Nagios monitoring, and Nicolas said they collected data into an RRD file, and Nagios watches that.

Invited Talks I: DevOps: Puppet

Summarized by Scott Murphy (scott.murphy@arrow-eye.com)

Building IronMan, Not Programming

Luke Kanies, Founder, Puppet and Puppet Labs

Luke Kanies described how he went to his first LISA conference in 2001 and got hooked on configuration management. Puppet was conceived at a LISA conference, and LISA and other conferences influenced Puppet over the years and were instrumental in its development.

Luke then said that he was not there to talk about Puppet, but about DevOps/. He started with a description of what DevOps is not. DevOps is not development. It's poorly named. It is not about developers becoming operations. It is not about operations becoming developers. It is not about "not operations." Operations is not going away; however, it will likely be transformed over the next few years.

DevOps is about improving operations, primarily through cultural change—not through new tools or the company changing, but by the people at the company changing. A major way to effect that change is by improving the sysadmins. Sysadmins are operations. You can't talk about things changing operations without talking about sysadmins changing.

Another way to improve operations is to minimize process through better tools. Sometimes you can trade off process for tooling. The main way to improve is through collaboration. If you are a sysadmin and you do not work with anyone else, you are not doing what you can for the organization and you are not doing what you can to get your job done better.

Automation shows up a lot in DevOps. In our jobs, automation is a big part of what we need to be thinking about; it's what we need to be doing. It's either there or it's coming, and we can't avoid it—we need to embrace it, and scale is the reason. We are dealing with numbers we couldn't conceive of 10 years ago; 100+ machines in 1999 was a decent-sized infrastructure. Today it's barely a blip, as people run thousands, even tens of thousands, of machines. Speed of scale is also an issue. You can now add 1000 machines a week. Zynga added 1000 machines a week for months on end—imagine adding that many machines from any vendor 10 years ago. Now we want them in a week and to have them up and running, not just sitting on the loading dock. Services are now critical. Remember when we had maintenance windows? Remember when you had time to figure out why something wasn't working? That method is no longer valid. A weak corporate Web site 10 years ago is now 90–100% of your business today. These are all things that lead to automation. Just because this is the reality, it doesn't mean automation will replace us.

You get to pick the kind of sysadmin you can be. You can be a mechanic, know the rules, follow the rules, and not make mistakes. That skill can be automated and eventually you will be replaced. Artificial Intelligence (deciding and thinking) is difficult. AI is 10 years out, just like it has been for the past 50 years. If you just know obscure items from the operating system, you will be replaced. If you just perform the mechanical aspects of the job, you will be replaced. You are not a key differentiator for your business. You need to be good at understanding and deciding. You need to understand what is normal in your infrastructure, why the infrastructure exists, why the services are running and what is important to the end users, the customers, and the employers. You need to be good at deciding what to do based on the information you have available. Your real value lies in making decisions, good ones.

For example, the financial trading industry would love to fully automate, but it can't. We are talking big data, rapid response, huge amounts of money. The skilled traders get paid, make bonuses, etc., and if their function could be automated, it would happen. Why are they not automated? A lot of what they do is, but this is a skill-based system. Consider it to be the best video game in the world and they are skilled players and get paid to play games for eight hours a day. The software systems provide huge amounts of information coming in all day. They absorb this wealth of information and then make quick decisions. The software then rapidly performs the trades based on the decisions. The software gets data to them and once a decision has been made, acts on it. It does not make the decision; they do. This is their value.

In a similar way, you need to be the kind of sysadmin who is good at understanding and deciding. This makes you valuable to the organization. If you are good at following rules and not knowing who those rules impact, probably you will be replaced with software. It is likely that we all know somebody who could be replaced with a shell script. As an observation, this year is the first year that the majority of attendees are running configuration management systems. The first workshop was 10 years ago. The scary part is that the LISA community is an early adopter and it's taken 10 years to get here. We've been moving very slowly and the world has moved quickly. If we don't start moving and move quickly, this will end up with the developers taking over the operations role.

In 2001, a group of developers wrote the Agile Manifesto. They all worked in a similar way and they were breaking all the best practices and doing dramatically better. They didn't do it because they had a plan; it was because they recognized their own behaviors. This is not scrum. It's not extreme programming. It's not a tool or practice. It's a way of thinking. Read the Agile Manifesto. It doesn't describe practice, but, rather, the way you think about the world and applies that to how you interact with your team and the teams around you, which is why it is related to DevOps.

The Agile Manifesto has four main principles:

1. Care more about individuals and interactions than about processes and tools. Processes matter and they exist for a reason. Having good tools is great. However, if the processes get in the way, they are not doing their job. If the tools interfere with why you exist as an organization, you have the wrong tools. This is important for agile development, as there tended to be good tools and tightly defined processes in old-school development. If you are good at making decisions, then you don't need inflexible processes or to use the world's best tools. If you are bad at decisions, then you need processes to double-check you and better tools to keep you on track.
2. Value working software over comprehensive documentation. Documentation is great and typically very valuable. However, if the infrastructure is down, the best documentation in the world will not help you. The software must be running and must work. If the software doesn't match the documentation, it has limited value. You are far better off building software than documenting software.
3. Value customer collaboration over contract negotiation. This seems more applicable to software developers than system administrators, since software developers tend to do more consulting, bring in a team, write the software, and then move on, but sysadmins also have contracts and customers. We tell the employers we will keep the services and systems up, we will deploy the software, we will keep the systems stable and secure. We have customers we need to help.
4. Respond to change over following a plan. No plan survives contact with the enemy. Reality often changes while we are doing something important. New technologies show up. Facts on the ground change and you need to be able to adapt to that. Change happens all the time. You need to be thinking about the change and not the plan. Build a plan but realize that the plan will not last. It is a guideline, not an absolute.

Waterfall development was good at building an application under budget and on time but building the wrong software. They never talked to the customer about what they wanted and never worked with the customer to understand what to build. Operations has a similar problem. We are very good at building secure, stable systems that take 18 months to deploy, often longer.

Kanies then talked about a customer that had invested in this process to the point that they had forgotten how to deploy software. Another customer had the problem of it costing more to deploy their software than to write it. This was a Web company and had been around a while.

If you can't change, you can't meet business needs. The company exists for a reason and the company is paying you to enable that. Operations needs to see the world in this light. Operations needs to find a way to do the job better. If operations doesn't, then the business will step in and tell you how to do the job. This would not be a good thing.

Process is the bugaboo of the systems world. We all hate it and we all follow it. Even more rarely is there a process owner who will call us on not following process. This may have happened when something went wrong and the business decided that it didn't want a repeat. This resulted in a process to ensure that particular failure did not happen again. That's how process happens. Change management is an example of these processes. The sysadmin writes up a technical change procedure that needs to be signed off, typically by a non-technical person. How can a business person really understand that? They sign off and the change is approved. Does this make any sense?

This is the fault of operations. Operations made it extremely difficult for business to be confident in the work we do. We didn't help them be confident in our work so they felt they needed to be part of the process. They needed to have a non-technical sign-off because we didn't find a way to give them the confidence they needed without that sign-off. A big part of the cultural change we need is to find a way to give them confidence that we are doing what they want us to do, as well as providing security, compliance, and stability. One of the best ways to build confidence is to trade out process for tooling. Version control is a great example of this. It wasn't all that long ago that developers didn't use version control. This inspires confidence that the versions in development and deployment are the same.

In the systems world we can also use tools to manage the change control process without requiring sign-off.

In general, sysadmins are very conservative, as in, nothing should be done for the first time. The thought process is, "This is new, so we need to move slowly until we are confident about the technology." We need to find ways to get that confidence so that we can move quickly and be confident

without sacrificing the things we have to deliver. Needs are changing faster than we do, resulting in higher pressure on the operations people.

There is a huge amount of discussion of cloud computing. Luke's opinion is that the two most interesting versions are Software as a Service (SaaS) and the self-service cloud.

Two SaaS examples are Salesforce and Zendesk. SaaS is where you take things that are not your core competencies and have someone else do it. Very few companies take CRM as a core competency. You need to be able to talk to your customers, but it isn't critical that you be able to maintain the CRM system yourself. SaaS providers have operations people and consider it to be a core competency. They know they have to be fantastic at operations, so they hire the best, train the best, constantly adopt new technology, and require operations to adapt quickly.

The self-service cloud—I want to be able to do it myself, I don't want to have operations do it. I want the developers able to get a new machine or a new operating system without involving operations. I want to get 500 new machines right now without having to deal with operations. Self-service is more threatening for operations.

Developers like the self-service cloud, since they get what they want on request. The best case is that the operations group has provided a system that allows developers to get what they want while ensuring that business requirements are met. This is about collaboration, finding ways to deliver what your business needs and what you need to deliver to your organization.

Remember that operations is not why your organization exists. You exist to enable people and fulfill the purpose of your organization. You need to understand who you are trying to help. You are in the service industry. You do not directly produce; you assist other people to produce. If you don't think of your job as helping people, you need to be afraid of automation. Every day you need to decide what kind of sysadmin you want to be. If you decide not to decide, it will get decided for you.

Tim Kirby (Cray Inc.) said that "somewhere along the line it started to feel here as though your stance was that we should do everything they ask without saying, 'That may not be the right thing to do.' And I just want to make sure that I wasn't hearing that, because I think we have a role, not necessarily as gatekeepers, but as facilitators." Luke replied that it's absolutely not about just doing what they tell you to do or doing what they ask you to do. If you aren't great at your job, then your choices are to follow the rules or to do everything the customer asks. You want to be in a place where you can make intelligent decisions about how to interpret what they're asking, whether what they are asking is complete bonkers, and of course you can't do that. It's very important that you understand what they are saying and that you decide how to act on that. It's a lot like design. One of the things we are doing at Puppet Labs is that we are really pushing our whole organization behind design and design is about the user experience. It's like the famous quote by Henry Ford: if he had done what people wanted, he would have built a faster horse. He knew what they needed and built that. It's about understanding. You have to trust yourself and you can't just mechanically follow rules. That's how you get replaced by software. Humans are great at making intelligent decisions based on complex situations

Someone said that he is scared of self-service going so fast and giving developers free rein, because it's very easy to get yourself into technical debt. Yes, it's very easy to get something functional very fast, but backups, documentation, and all these non-functional pieces still need to be done. Operations typically takes care of that, but it takes time. What are some ways that we can work faster to be able to deploy faster but still take care of all these nonfunctional bits that are important and that we know are important but which they may gloss over? Luke replied that the first thing is to figure out where your time is going. Most places I've been, people's time is going to things that aren't critical to business—a lot of what amounts to menial work that you can automate. To me this is the best place to automate. Where is time going and is it valuable time? Is it adding real value or is it time that is not that valuable? If you can automate it and make it go away right now you have more time. And this is the time you are reclaiming without increasing budget, which you can then reuse and add on to things such as building self-service infrastructure, building better monitoring, and building more information for the user.

Kent Skaar (VMware Inc.) asked if Luke had seen new areas where he was surprised to see self-service. Luke replied that he isn't surprised by very much of it. Most money is spent on maintaining existing resources, 80–85% vs. adding new resources. It's about agility and moving quickly. It's about responding to the needs of the organization. People want to move things faster.

Betsy Schwartz (e-Dialog Inc) asked if Luke could talk a little bit about the intermediate stage. Right now we have a lot of human beings in QA who sign off on accepting the QA stage and managers who sign off on rollouts. Luke said that there is no such thing as a non-intermediate stage. All this stuff is asymptotic. You can always be better; you can always move faster. Greg LeMond has a great quote: "It never gets easier—you just go faster."

Invited Talks II

IPv6, DNSSEC, RPKI, etc.: What's the Holdup and How Can We Help?

Richard Jimmerson, IETF ISOC

No report is available for this talk.

Honey and Eggs: Keeping Out the Bad Guys with Food

DarkNOC: Dashboard for Honeypot Management

Bertrand Sobesto and Michel Cukier, University of Maryland; Matti Hiltunen, Dave Kormann, and Gregg Vesonder, AT&T Labs Research; Robin Berthier, University of Illinois

A Cuckoo's Egg in the Malware Nest: On-the-fly Signature-less Malware Analysis, Detection, and Containment for Large Networks

Damiano Bolzoni and Christiaan Schade, University of Twente; Sandro Etalle, University of Twente and Eindhoven Technical University

Auto-learning of SMTP TCP Transport-Layer Features for Spam and Abusive Message Detection

Georgios Kakavelakis, Robert Beverly, and Joel Young, Naval Postgraduate School

Using Active Intrusion Detection to Recover Network Trust

John F. Williamson and Sergey Bratus, Dartmouth College; Michael E. Locasto, University of Calgary; Sean W. Smith, Dartmouth College

No reports are available for this session.

Invited Talks I: DevOps Case: Scholastic

Summarized by David Klann (dklann@linux.com)

Fixing the Flying Plane: A Production DevOps Team

Calvin Domenico, Marie Hetrick, Elijah Aydnwylde, J. Brandon Arsenault, Patrick McAndrew, Alastair Firth, and Jesse Campbell, Scholastic, Inc.

Seven members of the DevOps team at Scholastic, Inc., described their heroic effort to "Webify" an entire application set while simultaneously switching datacenters. In six months. With a single weekend of down time.

The team of 10, led by Calvin Domenico and Marie Hetrick, took control of more than 20 custom-developed applications for 10,000 school districts. The apps were originally developed as locally run client-server applications that had been "jammed into" a hosted, managed service provider (MSP) environment. At the time they moved the application set from the MSP environment, there were about 400 school districts using the apps.

The Scholastic team described their previous MSP environment as a nightmare: a virtualized environment in a managed-hosting datacenter with limited visibility to the backend systems, no access to network or storage configurations, and, of course, no administrative access to anything.

The environment from which they migrated included untenable support issues. Operations team leader Elijah Aydnwylde described a scenario where they decided to build a workaround to mitigate network trouble that the MSP wouldn't help troubleshoot: Scholastic customers were reporting network trouble when accessing an application. Scholastic troubleshooting pointed to the load balancer in front of the app servers. The MSP disagreed: "Nope, can't be the load balancers." End of story...almost. The team built and deployed a layer that bypassed the load balancer, and successfully mitigated the trouble. Unfortunately, the time spent troubleshooting (in person-weeks), and the ongoing poor performance resulted in lost revenue and lost customers.

The Scholastic team, led by infrastructure team leader, Patrick McAndrew, chose not to build a datacenter from scratch. But they wanted all the elements and access they were missing at the time. They described choosing a datacenter partner who would perform all the initial heavy lifting, and subsequently turn over management and administration of the systems to Scholastic.

The panel described several keys to completing this challenge. One was the concept of merging development and the operations groups. They actually thought they invented the term "DevOps" (or "OpsDev" as they called it). By way of example, their operations team would lob an operations task over to the development folks, often a difficult or inefficient one. The pain of having to do the manual task would get them to automate it.

The team, led by Jesse Campbell (who was also the lead coder for the team), wrote its own configuration management tools. With the goal of creating an automated control engine for the existing underlying technologies, they created a system using NFS, git, Puppet, VSphere, Bash, and Perl.

They discussed two key points about working in the "agile" environment they created. First, every project needs an advocate to usher it along. Second, communicate the requirements by actually doing the job. Software engineer Alastair Firth said that the developers were directly involved in writing specifications so that "nothing gets lost in the translation from the stakeholders." Firth also asserted that "personality matters" with respect to building teams. Pay attention to teams' personalities.

At the end of their six-month project, the team chalked up all kinds of wins. Use of their new tools and improved processes

resulted in time reductions in all aspects of provisioning and deploying services and applications. In some cases, such as patch application, the time savings were several orders of magnitude (4,500 hours reduced to 3 hours), and many processes were automated. Project manager and database designer J. Brandon Arsenault presented a slew of impressive statistics, dramatically showing the improvements in the entire application suite.

What recommendations did they have for those who can't start over in a new datacenter? Virtualization is your friend. Start small, get bigger. Build a small-scale functional model, then prove it out at scale (Scholastic didn't have a datacenter for the first three months of the six-month time frame). Someone else noted that it's relatively unusual to integrate Dev and Ops; how did that happen? Hetrick said it was a conscious decision. This was the only way that it would work. The operations team was four people; the core software was developed by many houses of outsourced stuff. Their only chance to get this done right and on time was to bring developers into the operations team. They were excited about the LISA theme and how their model fit the notion being presented at the conference. Hetrick described their concept of culture: they had a really large development staff, but the wrong culture; they had a month to turn a set of apps into the SaaS model—just hire new developers. Domenico said that it really comes down to culture. The Scholastic staff needed to be able to get developers into operations to understand the scale of the thing.

Someone else pointed out that there are different types of developers (applications, systems, etc.). They focus on different aspects of the systems. Application developers spend much more time doing UI than anything else, while system developers spend *no* time on UI. Application skills aren't really applicable to systems development. Domenico said that's a great point. The core software engine was more than six years old, but some parts of it were completely opaque; how *do* we run hundreds of these right next to each other? Aydnwylde said that it was mind-blowing that we had no control—a terrible, terrible place to be. Getting control of infrastructure was the game changer.

Invited Talks I: Case Study: Big Launch

Summarized by David Klann (dklann@linux.com)

Releasing 9/11 Data to Satisfy FOIA: It's Just a Simple Web Site, Right?

David Pullman and Carolyn Rowland, NIST

All they wanted was a photo gallery Web site. It seemed a simple request of David Pullman, Carolyn Rowland, Stephen Barber, and Andrew Mundy from their managers at the US National Institute for Standards and Technology (NIST). It turns out it was a bit more complicated than it first seemed.

The initial design requirements included a site for photos, videos, and PDF documents primarily to satisfy US Freedom of Information Act (FOIA) requests. The constraints included time and money. The deadline: September 11, 2011 (the tenth anniversary of the terrorist attack on the US). The budget: about US $25,000.

The team consisted of four systems developers who worked with a research team at NIST. Their usual workload included support for robotics, sensor networks, and related manufacturing projects. This request came during a significant structural reorganization at NIST. The request to create the Web site seemed a simple additional task while integrating two disparate IT organizations.

The Building and Fire Research Lab had collected a "pile of stuff" (1.5 terabytes of data spread over 350 DVDs) during their investigation of the World Trade Center attacks. This request was an afterthought that was proposed as a way of satisfying all the different labor-intensive FOIA requests. One significant constraint was that NIST staff was not permitted to alter the source material in any way (including adding metadata to files). This limitation, along with the fact that there was almost no text accompanying the images and videos, led to the realization that conventional search tools (such as a Google search appliance) would be of very limited value. The team settled on a different approach to the Web site.

They considered using an existing open source content management system. After evaluating several, they settled on Gallery version 2 because of Gallery 2's rich MIME-type support. Using the Model-View-Controller architecture of Gallery 2, they configured a MySQL database to hold and organize the metadata. On poring through the data, the team found that the data and the small amount of metadata they had "were a mess." Rowland noted that they found they had all the metadata for the objects (images, videos, etc.), but that they couldn't release all the objects themselves due to copyright and other restrictions. Only 13,000 out of 90,000 files had any kind of metadata or tags.

Pullman described the iterative series of tests they wrote (in Perl) to match metadata to objects. At the end of the exercise they were left with only 200 of the original 13,000 unmatched objects.

Pullman remarked on their perspective as system administrators working on a development project. They put the task into a sysadmin's perspective: "How will we rebuild this?

What if the import process crashes?" and similar questions. They described designing scripts and processes to survive the bottom falling out during batch runs, and having to start the processes over (which Rowland noted happened "quite a few times").

A few months into the project, the laboratory hired a new director. The new director involved a slew of additional players, who brought with them all kinds of different and competing requirements. They learned Agile development via "trial by fire."

While showing off the live site at http://wtcdata.nist.gov/, Pullman commented that the collection of data includes over 100,000 objects and continues to grow, as objects are released by their owners.

Someone asked if a third party could take this and Rowland answered that archive.org already has it. Rowland noted again that the primary requirement was the integrity of the data. "Integrity is of the utmost importance, not availability or confidentiality."

Pullman described the development path. The path started with the golden copy of the data on a private server. From this pristine copy they imported data and metadata into the database. From there they copied things to the development server for testing, and finally to the production server (a cast-off Dell server running CentOS). This (largely automated) process enabled them to incorporate changes and additions quickly.

By March of 2011 they felt the project was well on its way to completion. Then the NYPD "helicopter video" went viral. Suddenly ("Oh my god!") scalability, availability, and performance became part of the requirements. The team also re-evaluated the potential demographics of the site's visitors. They reconsidered their deployment strategy and considered options like the Amazon's EC2 service. They eliminated this option after receiving Amazon's quote: between US $200,000 and US $600,000 per year. Performance on a limited budget became the new top priority.

They settled on a hardware and software configuration of two load-balanced servers running nginx (which raised the eyebrows of the resident NIST security officer).

The team also considered various ways of restricting download traffic in order to avoid being "slashdotted." They ended up moving the site to a combined NIST and NOAA site in Boulder, CO. This move was based on a search for bandwidth: at their Maryland location they had only 10 Mbps available; the site in Boulder offered a full gigabit link out of the 11 Gbps total bandwidth.

Next, the team performed load testing with an Ixia traffic generator. They tuned the setup so that responsiveness would degrade under extreme load, but would recover gracefully as the load decreased. Great! The site was ready to go. Andrew Mundy, one of the team, was in Boulder preparing for the launch. The rest of the team gave the NIST director one last demo. The NIST director asked, "What about availability? Won't a successful DoS attack make the agency look bad?" He was right, but nobody had asked that question until then.

Add another requirement, but this one came with some cash. The NIST director was able to dig around and said, "Here's some money. Put the app in the cloud." Unfortunately, even with the additional funding, the security-certified cloud solutions were still too expensive, due to content delivery network add-on charges. They decided to go directly to CDN provider Akamai.

The Akamai deployment was easy; figuring out caching issues, not so much. They discovered lots of errors, and rework was needed for the shopping cart application. The lesson they learned was not to cache dynamic content. Having satisfied the layer upon layer of additional requirements, they were finally ready.

They performed some last-minute legal due diligence (copyright holder notices resent, legal review, etc.). More changes: two primary content providers asked the team to remove their content due to lack of embedded copyright notices. The FOIA people delivered a new 75 GB data set full of high-resolution video from the FBI to add to the repository (in file sizes up to 15 GB).

They launched the site on August 12, a mere five months after the original target.

Lessons learned:

"We're Ops, not Dev."

"Getting data from people is really hard!"

Getting requirements is also difficult. The team was driving requirements by showing work as it progressed.

Getting data from others is messy, but can be automated to normalize the information.

Many stakeholders. This was difficult, but useful in order to get as much input as possible before the launch.

Real stress-testing beats any theory (stress-testing was contrived, but useful).

Last-minute stakeholders: get buy-in from as many interested parties as early as possible; identify stakeholders as well.

Having an exit strategy from Akamai was helpful in order to migrate away from their CDN after the initial push.

"We're operations." This was the team's first real "development project." They got to see firsthand how to do DevOps. Final word: "The DevOps paradigm really works."

The first question at the end of the presentation was about boiling down lessons learned to a sound bite, such as "Never demo anything until you launch." Rowland replied that there are two sides to it: launching without showing to enough stakeholders may result in more changes after the launch. It may be better to demo late in the project and delay the launch a bit. Someone else asked about avoiding all the stakeholders popping out of the woodwork. Rowland commented that the team didn't really know the customers, due to the reorganization. Normally, they would have worked harder at this, and now they would know who the stakeholders are. Rowland further noted that there were two kinds of stakeholders: researchers issuing reports, and directors awaiting the site. Look more proactively for stakeholders. In the future, this team will get them involved sooner.

The next person wondered if they really need to go to Akamai. Pullman said, "It probably would have been fine." On September 11, 2011, they experienced a peak of 29,000 daily total page views, with almost 350,000 cumulative views. Total site volume has now exceeded 9 TB of downloaded data. Someone else wondered about the continue Link, and Pullman said that it was necessary. Without this mechanism it would be easy for other sites to link directly to images and videos. The NIST legal department wanted this "wall" to prevent sites from making these links. Pullman noted that there may be a better solution to this constraint, but the current implementation works fine. The final question was, "Where is the biggest performance bottleneck?" Pullman answered, "We haven't hit any bottlenecks at this point." Rowland added that not everything is cached, but Akamai helps a lot. Traffic spikes didn't cause "running hot." Pullman added, "We'll see how it performs when Akamai goes away."

Invited Talks II: Storage

My First Petabyte: Now What?

Jacob Farmer, Cambridge Computer

No report is available for this session.

Invited Talks II: Security

Can Vulnerability Disclosure Processes Be Responsible, Rational, and Effective?

Larissa Shapiro, Internet Security Consortium

No report is available for this session.

Network Security

Summarized by Timothy Nelson (tn@cs.wpi.edu)

Community-based Analysis of Netflow for Early Detection of Security Incidents

Stefan Weigert, TU Dresden; Matti A. Hiltunen, AT&T Labs Research; Christof Fetzer, TU Dresden

Stefan Weigert noted that attacks are especially insidious when they are targeted rather than random and stealthy rather than immediately destructive. They assume that attackers know what they want, that they have incentive not to be noticed, and that they want to compromise machines in more than one company at a time. These attackers are hard to detect! Finding a single infected machine in a large organization is a needle-in-a-haystack problem. This work focuses on attackers who target a community instead of a single organization: for instance, compromising multiple banking or retail establishments at once.

If an IP address communicates often with many companies inside a community, it may be suspect. The problem is the sheer volume of data involved. Weigert's group begins by discarding non-community traffic and allowing certain addresses to be whitelisted. They also borrowed a concept called community-of-interest graphs from telephony, which allows them to focus on the most important addresses. In these graphs, a weighted, directed edge is drawn from outside addresses to community addresses. Weights decline over time, and are reinforced by net flows from source to sink. In the end, the graph will be sparse, with edges from only the most common external addresses. They construct a separate graph for each determining factor (e.g., ports used or number of transferred bytes).

For this work, Weigert's group looked at a heavily sampled one-day segment of traffic. They manually examined the list of suspicious IPs that their analysis produced. Weigert showed us two suspicious examples. The first address connected to many different addresses within the community, but very few addresses outside. After looking at whois data, they concluded that that address was indeed acting suspiciously. The second address turned out to be an anonymous FTP server. Wegert underscored that only the community members will be able to truly tell an attacker IP from a

benign IP, and that the goal of this work is to provide data to the community.

Someone from EMC asked whether a clever attacker could produce random traffic as a countermeasure. Weigert said that they could do that, but their work isn't meant to detect all possible attackers.

WCIS: A Prototype for Detecting Zero-Day Attacks in Web Server Requests

Melissa Danforth, California State University, Bakersfield

Melissa Danforth presented an attack-detection method based on an artificial immune system. Immune systems are all about pattern matching. Just as a real immune system looks for certain protein shapes, Danforth's system looks for shapes in the syntax of URIs. Artificial immune systems are mostly worried about distinguishing "self" from "non-self." They extended the idea to detect specific classes of attacks: information gathering, SQL injection, read-only directory-traversal, buffer overflows, cross-site scripting, and attempts to execute scripts on the Web server.

Danforth's system uses genetic algorithms to breed sensors for these attack types. Each URI is reduced to a fingerprint involving features such as length and number of times "%" occurs, and this fingerprint is consumed by the sensors. The sensor-generation process begins by randomly picking features and initializing them with random values. To reduce the chance of false-positives, newly generated sensors that trigger for a random set of normal traffic are discarded and regenerated. The system then feeds the sensors a representative sample of a single attack type and begins to breed sensors with an affinity for catching that attack type, adding a small amount of random mutation. After several generations the process stops, leaving the system with a set of sensors likely to find attacks.

They were unable to test their system on a live environment; permission to do so arrived too late. Instead, they analyzed existing Web-server logs. Effectively, this means that they tested the accuracy but not the scalability of the approach. They found that their sensors were best at detecting scripting and traversal attempts, but had difficulty detecting passive information-gathering attacks. In the near future, Danforth's group hopes to use the live data that have become available. They plan to have students try to craft attacks that evade the sensors. They also discovered that it was hard to distinguish read-only directory traversal and script-execution, and may lump the two classes together. Finally, they plan to look at more of each request than just the URI.

Tim Nelson asked why they opted to use genetic algorithms. Danforth replied that traditional artificial immune systems use genetic algorithms, and they wanted to see how they could extend them. Tim also asked whether they were worried about obfuscation, since their sensors are bred to detect syntax. Danforth answered that they had considered obfuscation, which is one reason why their sensors can see the number of "%" characters. There should still be indications of attack left in the URI, even after obfuscation. Someone else asked how they determine what traffic is "normal" when removing sensors that misclassify normal traffic. Danforth said that is a challenge, and they had to work hard to develop their normal-traffic data set.

Invited Talks I: Networking

Ethernet's Future Trajectory

John D'Ambrosia, Force10 Networks

No report is available for this session.

Invited Talks II: IPv6

IPv6: No Longer Optional

Owen DeLong, Hurricane Electric Internet Services

Summarized by Thang Nguyen (thang@ccs.neu.edu)

Owen DeLong came to LISA '11 to speak about the imminent depletion of IPv4 addresses, and how prepared we actually are. APNIC has already run out, with IANA following closely behind. Several technologies are also not ready for IPv6, including current WiMax handsets/providers, DSL systems, most IT staff/management, and various others. Technologies that are ready include current operating systems, DOCSIS 3, the WiMax specification, LTE, CPE, and early adopters/industry experts.

Following his intimidating statistics and graphics about how IPv4 is running out, Owen began talking about how people should be preparing. "We need to get everyone to a fully production dual-stack environment before IPv4 runs out." To reach new participants, services will need to be able to provide IPv6. While workarounds do exist to make IPv4 work for a little longer, they come with bad tradeoffs—mainly NAT on a carrier level being an absolute nightmare to work with. Continuing on the IPv6 bandwagon, there are several advantages compared to IPv4, including not needing to count hosts, a better address issue methodology, no need for NAT v4, and a stateless auto-config. Owen also reviewed the strengths and weaknesses of several different relevant technologies, including 6to4, Teredo, and RA Guard and noting that 6to4 was one of the last solutions people should use.

Implementing IPv6 on a Global Scale: Experiences at Akamai

Erik Nygren, Akamai

Erik Nygren spoke about his experiences with implementing IPv6 on a global scale, and the upcoming transition. Erik pulled several figures from www.potaroo.net/tools/ipv4 to display APNIC exhaustion in 2011. "Think of IPv4 and IPv6 as two different Internets that don't have direct compatibility with each other."

Erik then segued into IPv6 user adoption today, showing native IPv6 preference vs. 6to4 and Teredo (google.com/intl/en/ipv6/statistics). Adding to that, he mentioned that very few home routers today properly support IPv6,never mind actual service providers. With potentially broken IPv6 being a real problem, "Happy Eyeballs" was born. "Happy Eyeballs" is a proposed Internet draft to work around broken IPv6, with rules to fall back to IPv4 when necessary. Akamai addresses this by providing a dual stack to deliver content to users regardless of protocol.

The final part of Erik's segment was about how IPv6 has widespread implications: people and training are important, along with feature gaps and bugs. In September 2012, the US government will issue a directive to upgrade their services to support IPv6. Erik closed by saying it is critical to make progress with IPv6, but to also remember that IPv4 will be here for a long time. Prioritize your IPv6 on the most important areas now, and focus on the rest when you need to.

Refereed Papers: Networking 1

Summarized by Cory Lueninghoener (cluening@lanl.gov)

Automating Network and Service Configuration Using NETCONF and YANG

Stefan Wallin, Luleå University of Technology; Claes Wikström, Tail-f Systems AB

Claes Wikström started the session by describing his team's experience using NETCONF and YANG to improve their network configuration efficiency. He began with a description of their problem: managing and automating a large network of devices is a difficult task, and previously they had been relying on a lot of screen scraping to make automation possible on interactively configured devices. When they went looking for more efficient ways to manage their network devices, they quickly settled on a solution using NETCONF (an XML-RPC-like network device configuration protocol) and YANG (a hierarchical modeling language).

To make managing their NETCONF and YANG system easier, the authors built a system named the Network Configuration Server (NCS) to act as a central place to manage NETCONF-compatible products. This system was implemented in Erlang on top of a configuration database that acts as the authoritative source of configuration data. When initially run, NCS connects to a device, finds out what YANG modules it has, grabs the device's configs, and stores a copy. From there the device's configuration can be changed within the NCS system, human-checked with a diff, and committed to the devices being changed. Bad changes can easily be rolled back to a previous revision, giving the admin a useful security net.

Noting that running large network tests in the Amazon cloud is much cheaper than building an actual large network, Claes finished by describing their performance tests on a network of 10,000 virtual routers. They found that performing an entire NCS configuration sync took less than three hours on the 10,000 devices, and that after the initial sync was complete they were able to perform day-to-day operations (such as adding an IP address to every device) in a handful of minutes.

Are the protocols vendor-specific and are general-purpose OSes that act as routers supported? Many vendors are getting behind the same standards, and it would be great if more general-purpose OS systems would start supporting it.

How generic are the configurations? For example, how does NCS handle Juniper and Cisco devices that call the same thing by different names? NCS handles this abstraction at its service manager level.

An attendee doing a lot of load balancing in his environment wondered whether YANG supported ACLs. Claes responded that it absolutely does, and that there is continuing work in IETF on that support.

Adventures in (Small) Datacenter Migration (Practice & Experience Report)

Jon Kuroda, Jeff Anderson-Lee, Albert Goto, and Scott McNally, University of California, Berkeley

Jon Kuroda described his group's experience moving a 600 square-foot machine room one floor down in a very short period of time. A group at his university decided to remodel half of a floor of their building, which included removing an existing machine room. When they were notified of this, Jon's group followed the usual steps one follows when kicked out of an apartment: first they filed a complaint, and when that didn't help they started looking for a new place to house their machines and friends to help move them. Out of the places they found, they chose the "least bad" one and prepared the move without the luxury of datacenter help, network admin, or offers of help from those doing the remod-

eling. He described the move as being like DevOps, but with facilities: FacOps.

Due to university class schedules, there was a very small window of time in which the move could happen. Jon gave a great description of what had to happen during this time: electrical work, cleaning, carefully orchestrated machine shutdown, move, reconfiguration, and bring-up. He also described some of the nice things about the move: the ability to make a more sensible layout in the new room, reinforce hot-aisle containment, and generally clean up their area.

The move went much as one might expect: some things went very well (the electrical work, for example, was done quickly), but other parts put them behind schedule. Eventually, the team had to do work that they had expected others to do (such as network reconfiguration) to get it done in time, and they succeeded. However, Jon noted that he would try his hardest never to let this happen again. His strongest suggestions after the experience were to maintain good relationships with those around you, to be ready for external delays, and to work on good collaboration tools before starting something of this complexity.

Several people related similar experiences during the Q&A. One attendee was surprised that they had enough server room elsewhere to get rid of one machine room. Jon noted that he was surprised by that too and that they need to do better space planning in the future. Another attendee wondered if Jon's group had the opportunity to clean up the new room before moving in, and he said that, thankfully, they did.

Experiences with BOWL: Managing an Outdoor WiFi Network (or How to Keep Both Internet Users and Researchers Happy?) (Practice & Experience Report)

T. Fischer, T. Hühn, R. Kuck, R. Merz, J. Schulz-Zander, and C. Sengul, TU Berlin/Deutsche Telekom Laboratories

The final talk of the session was given by Thomas Hühn about his group's experience providing researchers with a wireless testbed. He began by describing how the Berlin Wireless Network Lab was created to meet this goal, giving researchers a flexible testbed with realistic traffic and full-campus roaming abilities. In doing this they had to balance a developer's freedom for change with the operator's desire for a robust and reliable network.

Thomas continued with a description of their final product: a network with three levels of production (desktop development, indoor testing, and outdoor deployment) and a flexible configuration with multiple OS images. Their initial attempt at configuration management involved pushing new configurations to the routers, but after this didn't work well they fell back to a pull model. Their final model has the router nodes contact a central server at boot time or when their configuration is stale to check for updates. By deploying this setup across around 60 outdoor antennas, they were able to meet the needs of both users and researchers.

With the network deployed, Thomas described how the authentication layer on the testbed works. They used FreeRADIUS as the basis for their authentication system, and they found that authenticating against many accounts from several different organizations got complex very quickly. Troubleshooting the system was especially difficult, and Thomas ran through the process they used to debug one issue to illustrate this difficulty. After weeks of work, they were able to find the subtle problem: one server certificate on the university side had expired. From this experience they learned the importance of version control and making stepwise changes to their authentications system.

Thomas finished with a few lessons they learned through their testbed rollout experience. One suggestion was, when designing a research testbed, use it in a realistic way. That helps make the transition to real life easier. He also noted that robust autoconfiguration takes a lot of work and that they found that pulling configurations was much more reliable than pushing them from a central server. Finally, he concluded that having a general router image with individual configs is much faster and more flexible than having an individual image for each router.

There were no questions for this talk.

Invited Talks I: Panel

What Will Be Hot Next Year?

Moderator: Narayan Desai, Argonne National Lab

Panelists: Kris Buytaert, Inuits; John D'Ambrosia, Force10 Networks; Jacob Farmer, Cambridge Computer

Summarized by Thang Nguyen (thang@ccs.neu.edu)

Narayan started the panel by asking about the biggest changes people should be looking into. Jacob Farmer said that SSDs will significantly replace spinning disks. John D'Ambrosia talked about the chips that are driving all of the new technology. Kris Buytaert is eagerly waiting to see DevOps move into the enterprise world.

Narayan then asked how the industry is going to change for small- to mid-sized enterprises. Jacob asserted that the balance between performance and cost was highly in favor of SSDs. This launched a discussion about where people hit a plateau of diminishing returns in terms of networking, CPU, or storage speed. John noted, "By 2015 we will produce more

data than we can store. How much data do we need? What data do we need to back up?" The discussion shifted to future bandwidth requirements, and utility of the upcoming growth of data to users.

The issue of power consumption arose as well. John said that hardware is simply one aspect, but intelligent software design is also an important factor to consider. The topic shifted to the future of data mining with the rising popularity and plausibility of SSDs. Better disk performance will allow faster indexing, enabling users to read and sort through information faster.

The panel continued on a broader topic: the different and exotic things we will be experiencing in the future. The consensus seemed to be that the future is already here, and not much is going to change. Narayan pointed out that pervasive computing, sensor networks, distributing computing nodes, etc. would be changing storage, power, and networking needs for future infrastructures. Kris made a valid point in that these sensor networks exist and significant data collection is happening today, but we still need to turn this into salient information for the user.

An audience member posed a final question about the future of supply chain manufacturing, highlighting recent natural disasters in Thailand and Japan which have interfered with the supply of hard drives and tape media. Jacob spoke of those events potentially being enough of a catalyst for SSDs to succeed hard drives, as the entry barrier for SSD production is lower than hard drives.

Invited Talks II: Beyond Technology

Summarized by Scott Murphy (scott.murphy@arrow-eye.com)

Customer Service for Sysadmins

H. Wade Minter, TeamSnap Inc.

Wade Minter got off to a quick start by asking, "How many people here think they are good at customer service?" and followed up with a few questions and comments that set the tone for the talk. Wade mentioned that his style is a little more interactive than the standard tech talk.

At TeamSnap they say, "You only have to talk to the customers you want to keep." Unlike most Web startups, they effectively started with a dedicated support person. They stumbled into it by accident. The wife of one of the founders was looking for some extra work, and she was hired to help out with customer support. That helped them quite a bit, as they were able to carve a niche in customer support. The competitors were large, well funded, and targeted to sell to big organizations. They took the opposite tack. They started with individual teams and grew the business with customer support and word of mouth. That set the stage for some success, and they built on this. They believe that they stumbled onto something that Web startups do not do well.

Wade went on to observe that the audience was composed of people who supported both internal and external customers. Regardless of the type, customers are not only the people who pay the company money (internal customers also pay, usually through some other mechanism), but they are the people whose lives you must make easier.

Wade continued with the question, "What is great customer service?" starting with examples of bad customer service from the audience. They included problem-report black holes (no response), reading from the script, automation hell, rigidity, and not taking responsibility for the problem report. This was followed by good customer-support examples, including setting expectation levels, making a connection with the customer, admitting to the customer that they did not have the answer right now, fast turnaround, and non-scripted follow-up. There are trends in both categories.

During this discussion, Wade had a slide up that paraphrased Clarke's Third Law. The slide said, "Any sufficiently advanced technology is indistinguishable from crappy customer service."

Wade went on to ask, "What can you do even if you are not directly answering external customer calls?" You can get engaged, treat the customer like a real person, follow up, and have a real dialog. Those are the positive things we remember from our own good customer-service experiences. You should strive to give this experience to your customers. Examples of customer comments from actual TeamSnap customers were shown that indicated that they follow this methodology.

He observed that we (the audience) are good at technology, even things we are unfamiliar with. He went on to describe a few situations where we may not be expert and how we'd feel about situations where we would need support. That is how our customers feel. Put yourself in their shoes. Empathy is a key skill in this area and we don't value it sufficiently. This will resonate with the customer and they will want to do business with you. Don't forget that business models are easy to copy, so you need to have a differentiator that attracts customers when a well-funded startup comes in to compete with you. People will go with the folks they want to deal with. The business or group that typically wins is the one the customers want to deal with.

Wade went on to ask why sysadmins are typically bad at customer service. In addition to lack of empathy, we are impatient with other people's technology issues ("It's just a

computer: how hard can it be?"), we have a propensity to say no, we tend to be overworked, we'll get to it later, we are reluctant to call people, and we resent being asked about things that can be answered with a short search. We value expertise and we need to remember that not everyone can be an expert.

A summary of customer types was shown: power users—you can respond with short technical answer); regular old everyday users—you need to respond, but they will accept "I'll get back to you easily); reluctant users, those using tech because they are forced to—empathy works well here; the totally clueless, whiteout on the screen, crayon on the big screen, etc.—they are difficult to deal with and you may not be able to give them an answer that works, so be patient, prompt response, etc.; asshats, the people who hate you personally, etc.—since you don't seem to be capable of giving them an answer they will accept, refer them to your manager. Wade then told a story about one customer who was less than impressed regarding language support and compared them to Gaddafi. With a little care and discussion on the issue, this customer was converted to a raving fan. The point being that if they can get upset enough to complain, they have an investment in success and want to use your product. This is an opportunity to create a fanatical supporter, sowork with them.

Wade then said that you should build tools for your users, make them nice to use; Twitter's bootstrap (http://twitter.github.com/bootstrap/) is useful, simple, and incapable of doing damage. This can reduce your nuisance calls. Another item that has good mileage is to make it super easy to interact via email. Make this happen even if you have a good phone system or a fantastic ticketing system. This is how a lot of people expect things to work.

Another TeamSnap practice is to put everyone in the company on customer support, where feasible. Have the CEO, marketing people, etc., do a customer-support shift once a month. You don't understand the customer's problems unless you are on the front line. Everyone should have a stake in the customer. Don't withhold support from free customers. If you can impress them with your support, then they are likely to think "these are the people I want to work with" when they are in a position to purchase support. You don't want them to get the idea that you will nickel and dime them to death. While they may want the world, you just need to manage expectations. When the answer is no, say no, but nicely. You don't need to be a jerk about it. When you say no, give a reason, be sympathetic.

Wade finished the talk with a short summary: be empathetic with your customers; listen to your customers; treat everyone the way you want to be treated; don't be the BOFH. Your career and your company will benefit from this.

Colin Higgs (University of Edinburgh) pointed out that from personal experience on automation vs. live interaction, it costs more to have live support.

Wade replied that having people interacting with people vs. having machines interacting people does cost more money. You are investing in people and there needs to be buy in on this from the top down; if there's no buy-in from the top down to support this type of methodology, you're just going to have to do what you can. If there's only one of you and the support load is going up like this, you're probably not going to be able to take 20 minutes for every person and make them feel like a unique snowflake. Apply some of the principles. If you have to blow them off, blow them off politely.

Jay Faulkner, Rackspace, said, "I am a Racker, and top-to-bottom customer support has to be in everything we do. People tend to overlook that uptime is the primary method of serving your customers. If you build stable systems that do not crash and your customers never have to call, then you have reached the nirvana of customer support. Your customers are happy and you are happy, with no interaction required. Wade said that's an excellent point; you help yourself quite a bit by making systems that do not cause people pain. Eric Radman, D. E. Shaw Research, asked how you manage feature requests. Sometimes feature requests contradict what you just said. Wade said that they do, and if you are like TeamSnap you have three developers and a feature list of 200. You won't get to everything. Generally, say we are strapped for resources, we have serious uptime considerations, and we will consider the request. Promise that you will review the request and then really review it—don't just blow it off.

Marc Staveley asked, Isn't it true that this is one reason people like open source? You can track your feature request. You know what happens to it. Wade replied that open bug tracking may be a way to give your customers a view into what is really going on. If you are not going to get to it for years, or it's a single user request, mark it as such. Be honest. Other things may have higher priority, it may not be a good fit, etc. Marc then asked if Wade thought that visibility of the process is important. Other people chimed in and also said they would listen. Wade replied, "I think for us it is, but not for the Muggles. They will lob it over the wall and forget about it, unless it is critically important to them. It's a nice thing to have, especially if you are an internal person and you have technical customers, it would be a great thing to let them know things are being worked on; in a non-technical environment, not so much." Someone commented that it can backfire. He had an open bug open. There are a thousand people watching this bug that goes nowhere.

Somebody else asked for Wade's thoughts on transparency. He said he erred on the side of being open personally in terms of telling people, we don't have the money to do this, or, I can't do this because it has repercussions beyond what you know you're asking for. You're asking me to change the database. The more transparent you can be, the more users will think, "Okay, I'm not just getting a canned script blow-off. There is actually a reason behind why this is the way it is." If you give people a reason, the vast majority of the time they will be cool with that. People like knowing things. People like knowing that there's a reason behind something.

Jay Faulkner, Rackspace, said that the nutshell version is not just having empathy but also inducing empathy in others. In most cases, we want to fill that feature request but can't. Wade agreed, saying that by being open and honest, you allow them to understand that there are constraints. It helps your case and people no longer consider you to be an automaton.

Marybeth Griffin (Carnegie Mellon) pointed out that people in larger organizations have distributed responsibilities and everybody is a customer of everybody else. Due to the nature of the bureaucracy, things take time, and sometimes the customer service sucks. Tickets opened can sit in the queue past when the response time has elapsed. Shouldn't respect for each function be a two-way street? Wade responded that if you have groups that do not practice good customer service it will frustrate everyone and could severely limit your ability to do the same for other folks. That's a tough one to solve. His advice was to take care of your stuff and be an example, and other people may take notice.

Playing the Certification Game (No Straitjacket Required), a.k.a How to Become Certified Without Becoming Certifiable

Dru Lavigne, iXsystems, PC-BSD Project, FreeNAS Project, FreeBSD Foundation, BSD Certification Group

Dru Lavigne provided her background, which includes system administration, training, and writing and developing certifications (she's Chair of the BSD Certification Group). The tenet of the talk is that a person can derive value from certification. The session concentrated on system administration certifications, the message being that it's not all bad and ugly. There are good certifications out there.

Dru started with the bad. "Paper" certifications are a prime example of bad certification. What gives certifications a bad name is the idea that the certification itself is not worth the paper it's written on and that people who don't know what they are doing are getting certifications.

She used the "Dummies" book series as an example. The books themselves are not all bad. The basic idea is that you can take someone who knows nothing about a topic and they can be brought up to speed. The scary part is when we are dealing with system administration. You don't want someone who knows absolutely nothing about a topic very quickly coming up to speed, getting the piece of paper saying, "Oh yes, I know how to do this."

Brain dumps are also bad. If you search, you can find a list of the questions and answers for many certifications. Just by reading and memorizing you can be certified for something you have no ability to do.

Then she started getting into the ugly. What is involved in a particular program? The only way to pass an exam is to take the official training program, typically only available from the vendor, usually a week long, and costing several thousand dollars. This portion is not ugly, as who knows the product better than the vendor? The ugly part is if the only way to pass the exam is to take the program. Either the information is unavailable elsewhere or the official published information material has nothing to do with the exam, which you do not find out until you actually take the exam. The training is actually training on how to answer the exam questions. This type of exam is really an expensive brain dump.

More ugly are psychometrically invalid exams. Psychometrics is the science of assessment. The point of psychometrics is to see that if you have published a list of skills that are required to pass an exam, then the exam should test to see if you are able to perform the list of required skills, not to trick you into not passing the exam. The exam should test the skill level, not your ability to determine what question is being asked. That is not psychometrically valid.

Dru proceeded to give a number of examples of psychometrically invalid exams. Exams with pick lists, badly translated from another language or written by someone who is not a native speaker of the language the exam is written in, technically inaccurate, or possibly written by the sales team after a few beers.

More ugly, the current "hot" certification is required. This is seldom about content but, rather, the current trend. Experienced people have seen cycles of technologies go around. New terms for old concepts, new acronyms, and shiny new marketing spin generate new certifications that are unnecessary—clouds, virtualization, etc. No actual increase in your skill set results from the certification.

Dru then went on to talk about costs, not just for the holders of a certification, but for the maintainers of the certification programs. Psychometrics is expensive to achieve and not marketed as a value for certifications. This should change. Exams are expensive to maintain, and the bank of questions

will eventually get leaked. A lot of programs only have one version of the exam and it doesn't change. Over time, this cheapens the exam as more people are familiar with the content. Rectification in order to remain valid/current is expensive, especially when all that is changing is the feature list.

Having said all of that, Dru started covering the good side of certification. There is value in quantifying the tasks that make up a skill set. If you are aiming to become a good system administrator, you have a catalog of skills to measure against. If you are missing skills, then you have a learning map. If you are hiring people and they have a certification and there is a list of tasks that make up that skill set, then the candidate will have to prove that they can do these tasks. Chances are you learned your system administrator skills by doing the job. Chances are that there are knowledge gaps due to lack of exposure to some tasks. This skills catalog and a good certification program will help you fill in those gaps.

Dru described what to look for in a good certification program: Are the objectives available? Are they skill-based? Are there third-party reviews that indicate that the exam adheres to the objectives? What forms of training and study material are available?

If you are looking at certifications, you want to receive value. Sometimes the goals are different. Why are you doing it? Is this something your boss told you to get? Do you need it to get a job? Few training programs are geared toward skills, so build a lab setup. Find others who are skilled in this area. They can help you. Check IRC, forums, the local user groups, and system administrator groups.

You can gain from any certification, even the bad ones. Your employer wants you to have it, HR requires it, the vendor requires so many certified people, etc. You might even learn something new or develop the skills to wrestle that system into submission.

Dru advocates reinventing the certification game. The first system administrators didn't learn from a training program. Skills were developed by doing. How will the next generation of system administrators learn their skills? There are few practical programs, and while certification programs have improved, they are mostly vendor-specific. Look for and promote quality certifications. If you have the time, contribute to them. Most training is boot-camp based—three to five days training and you become certified. This is not sufficient time to be good or proficient at a task.

A good certification program is a tool that can be used to bridge the post-secondary knowledge-skill gap or bring new hires up to speed.

Aleksey Tsalolikhin asked if there is a list of the good programs. Dru replied that she thinks the BSD certification is a good program. Aleksey then asked her to tell us briefly about it. The BSD certification program was founded seven years ago. Prior to that, on BSD community mailing lists somebody would bring up a thread about every six months asking why there was no certification program for BSD, and that would immediately result in a flame war where people would list all the reasons why certification is a terrible thing. And some people would pipe in and say, "But you know there is some value in that." That went on for a couple of years. Finally, a couple of us got together and said, "We know there's some value; somebody just needs to sit down and do something," so we contacted all the people who said good things for years and we formed a nonprofit group.

That group was composed of system administrators and academics, and trainers and people who write programs. We had no idea what we were doing, but we just knew that this needed to be done, and we learned a lot along the way. We've run it like an open source project; all of our processes have been out there in the open, and we've written them down so other programs know what to do. Obviously, the questions themselves aren't open source and transparent, but everything else is.

We've always worked with the system administration community and psychometricians to find out how to make an exam very practical. What is it that people actually do in their day-to-day jobs, and what is it that employers and HR people are looking for in their employees? That's what we use to build the exams. Everybody's a volunteer, so we are on a shoestring budget. The only person we pay is a psychometrician, as I've never found a psychometrician who worked for free, but we pay for the exams through the cost of the exams themselves. And we offer a DVD where people have the tools to set up their own labs and practice skills they need to know.

Aleksey then asked if they found it necessary to keep updating the exam. Dru answered yes. The first step in creating exams is to define skills, and out of that there is a process for you to turn those skills into exam questions. After a set period of time, or after a certain number of people have taken the exam, you need to take all the psychometric data on how people respond to questions and see if there are any questions that need to be rewritten because they are too easy or too hard. You also have to look at those skills and say, "Since the last time we defined those skills, have new tasks been required of system administrators? Are there new features they need to know what to do with?"

Have they created training programs as well for this certificate? They decided they were only going to create the ques-

tions, the objectives. They have open sourced their objectives so that anybody can take them and create their own training programs and either contribute those freely or sell them commercially.

Christian Bauernfeind, Freudenberg IT, said that they are hiring, and he's been realizing that each interview is a kind of free-form certification exam that he's coming up with on the fly. How would psychometrics, doing proper exams, and basically testing for skill and not the ability to pass an exam apply to the interview process? Dru suggested two things: look at your own exam objectives and go through them, as they are basic system administration tasks. Their exam concentrates on BSD systems, but a lot of that would translate into any system administration. She suggested finding a subset that is important to you and have that as part of your process to make up your own mini exam. Second, if you find that a lot of the objectives would apply to the skills that you would want to see, the certification group could set it up to have a proctor come in and you could offer the exam to new hires. So do a dozen or so at a time and maybe make it a requirement of employment to be able to become certified.

Christian then wondered if they have a good source on how to turn a given skill set into a good set of interview or exam questions. Dru replied that they haven't looked at it that way. They have had some people in their group who actually work for very large companies that deal with a lot of new hires and have taken what they start with, something they call a JTA (a Job Task Analysis), and they have used the JTA list basically to see how people respond to those tasks. It's usually a list of 200 tasks, and if an applicant could answer 120 of these, that would indicate they have a very good skill set.

Eric Radman, D. E. Shaw Research, asked if they found they have to give some people hints that you can type "man ls" or "man intro" to find the answer. Dru said that's an interesting question; even when they put together their program they have two levels of exams. The first-level exam is very introductory, for junior-level sysadmins, and is a paper-based exam. It's multiple choice, but it was important that they put together a program that wasn't promoting memorization over understanding. You wouldn't get a question, for example, saying, "Which switch to ls do you use to do X?" That really is not for information. In the real world nobody memorizes those, and in the real world you've heard of "man" before. You do a "man ls" and you find your switch.

Their second level of exam is actually going to be lab-based and more involved with concept testing. The rules of the test will be that we won't tell you which operating system you use and we won't tell you what tools to use. They will ask test takers to meet an objective: for example, set up a mail server with certain attributes using whatever tools you want; at the end of the day, they want to see your number of users, does spam filtering work, etc. A lot of certification programs are either promoting memorization or they're promoting how you get to a certain screen. There's really no value if you end up with monkeys who don't know what they're doing. They try to test more concepts.

Aleksey Tsalolikhin wondered if anybody has experience with the O'Reilly certification for Linux/UNIX system administration and could talk about it. He has a new hire promoted from the help desk that he's apprenticing, is trying to make it go a little faster, and is looking at what resources are available. Dru asked Aleksey if they published any objectives. Aleksay said they have a course syllabus posted, but he didn't see any objectives. Someone suggested that Aleksey check out LPI (Linux Professional Institute); Dru said that their program is very similar to BSD's. LPI started before they did, it's Linux system administration, and they also started very open source. Their exam objectives are very well detailed, with the skills you need to know.

Migrations, Mental Maps, and Make Modernization

Summarized by Ming Chow (mchow@cs.tufts.edu)

Why Do Migrations Fail and What Can We Do About It?

Gong Zhang and Ling Liu, Georgia Institute of Technology

The goal of their paper was to understand the cause of incorrect migration. They hypothesized that the cause of most incorrect cloud migrations has to do with incorrect configuration. Gong said that his tool, CloudMig, analyzes configuration errors. Gong first discussed the cloud, which is utility driven, pay-as-you-go, and has elastic scalability. However, unlike in the past, physical nodes are connected to virtual nodes, and there are even virtual nodes to virtual nodes in datacenters. Thus, system migration is non-trivial, considering we are now dealing with a multi-tier, multi-server architecture. Alas, single host migration is not enough. Component checklists and knowing dependencies are more important than ever, as migration is a multi-step process and error rates are reportedly high and time-consuming. To make matters even worse, there are a plethora of dependencies and implicit things that occur. Currently, manual processes are used to fix migration errors. Unfortunately, this is very error-prone and the larger the data set, the longer it will take to fix them.

Their paper proposed a policy-based migration validation. The setup for the experiment included physical machines, one Hadoop server, and one Rubis machine. The aim was to

observe migration errors. Their paper put forth a categorization of errors: dependency preservation (which includes things like typos in dependency files), platform differences, network connectivity, reliability, shutdown and restart, and access control and security. They found that 36% of the migration errors are due to dependency preservation. Tools to manipulate and check configuration errors are critical, and this is the goal of CloudMig.

CloudMig is based on the idea of policy validation. It helps operators to weave important configuration constraints into continual query-based policies and periodically to run these policies to monitor the configuration changes, detecting and alerting for possible configuration constraint violations. CloudMig is semi-automated migration, and the architecture is two-tier: one server and one client. CloudMig has been tested on the same setup described above. Gong illustrated a configuration policy layer and an installation layer on the server. The experiment eliminated a good number of network and performance difference errors and mitigated platform, software, and hardware issues in Hadoop. The big lesson learned was that implicit and hidden errors are paramount in distributed apps.

Provenance for System Troubleshooting

Marc Chiarini, Harvard SEAS

Marc provided an overview of troubleshooting in a nutshell: troubleshooting is hard and frequent triage is detrimental to the construction of good mental models. The ideal way to develop such models is exposing hidden dependencies between components and build modes of component interactions that one can query. To accomplish this, Marc introduced the idea of provenance. "Provenance" means collecting and maintaining a history of interactions over time (i.e., where do things come from?). A provenance for system troubleshooting is a recorded history of digital process creation, including environment variables, execution time, parent process, arguments, and process destruction.

Marc described the use of an acyclic graph to organize all the information, where nodes are objects and edges are potential dependencies. It is important to note that the content passed between objects is not analyzed. Marc illustrated a simple example called wire_test that started with resolv.conf and led to net manager, which in turn led to netman socket endpoint on the left and other inputs on the right. Netman socket endpoint led to DBUS, which had two child nodes: dhclient socket endpoint as the left child and other inputs as the right child. The dhclient socket endpoint led to the dhclient, which led to the dhclient.conf file. Opening the dhclient.conf file revealed that someone commented out domain-search, host_name line. But what if a provenance graph has hundred of thousands of nodes? Marc proposed ranking edges based on the likelihood that the inputs affect expected behavior. A statistical approach is used: the number of times input F is read by program P divided by the number of times program P is invoked. As time progresses, rank is refined.

Marc also noted a few caveats with this approach, including that inputs of many programs can be changed by options or shell redirections, thus skewing the ratios. The solution is to treat invocations in which these differ as separate programs. In addition, for new executables such as different versions of applications, Marc proposed using a stack approach. In his work, many first-order dependencies for programs have been ranked (e.g., ssh). Edge-to-files residing in well-known directories may be increased (e.g., /etc/), while edge-to-files in log or temporary directories should have rank decreased. Popular objects are less likely to be the singular cause of problems. Using this statistical approach, users can explore "what if" scenarios by ranking paths and subgraphs. That is, take the arithmetical average of all of its edges.

Marc also introduced the PQL (pronounced "Pickle") language to query the graph. He showed an example to retrieve all the processes' output objects that use sendmail.

Marc concluded by reiterating building a model of interactions between system-like components. There are a number of works in progress, including performance, integrating the query warehouse with trouble ticket systems, exploring other methods for extracting patterns in provenance graphs, and making this work on a system with hypervisor.

Debugging Makefiles with remake

Rocky Bernstein

There was no presentation of this paper.

Invited Talks I: Security

Surveillance or Security? The Risks Posed by New Wiretapping Technologies

Susan Landau, Visiting Scholar, Department of Computer Science, Harvard University

Summarized by Erinn Looney-Triggs (erinn.looneytriggs@gmail.com)

Historically centralized technologies such as the telephone network lent themselves easily to wiretapping. As technology has progressed, certain facets, such as decentralized point-to-point networks, have removed that ease, while others, such as cloud architectures, have increased the ease of interception. The US government, attempting to keep pace with a perceived growing threat, has enacted laws broadening the scope of wiretapping and easing oversight on wiretapping.

The legal framework that underlies both the right to privacy and the laws that enable wiretapping starts with the Fourth Amendment, which grants the right to privacy. The first wiretapping laws were not enacted until 1968, with a revision in 1978. Since the 1994 advent of CALEA, there have been an escalating number of laws enhancing and broadening wiretapping capabilities.

As the use of wiretapping has increased, equipment from manufacturers such as Cisco has commoditized wiretapping abilities. This has in turn increased the attack surface for illegitimate use of said equipment for nefarious purposes. In short, with the increasing ease of wiretapping has come the increasing ease of illegal or unwanted wiretapping from third parties.

These increased risks have to be balanced against the need for wiretapping, just as increased encroachment onto privacy needs to be carefully balanced against the legitimate needs for encroaching upon said privacy.

How is law enforcement coping with the use of encryption? It appears that other tools, including pattern analysis, are able to extract enough information in some cases; in others, law enforcement will simply have to spend more time and money. The use of BlackBerry devices in India and how they were coping with RIM's end-to-end encryption was also discussed. The banning of BlackBerries and the changes that RIM is putting into place specifically for India should assuage the country's concern. Concern was also voiced about the costs of CALEA enforcement on ISPs; in a world using carrier-grade NAT, a larger ISP may generate up to a terabyte of tracking data a day. Susan responded that CALEA may have to be reworked to take this concern into account.

Invited Talks II: Sysadmin in/and the World

Copacetic.

David N. Blank-Edelman, Northeastern University College of Computer and Information Science

Summarized by Deborah Wazir (dwazir@gmail.com)

David Blank-Edelman presented several ways sysadmins could become happier at work, despite the stress, setbacks, and roadblocks we all experience in this line of work. The techniques were organized around the themes of mindset, motivation, and making change.

First, drawing on work published by Dr. Carol Dweck, Blank-Edelman covered the differences between a fixedand a growth mindset.

Dr. Dweck explains (http://mindsetonline.com) that people having a fixed mindset believe that basic qualities such as intelligence or talent are just fixed, so they focus on documenting these qualities rather than developing them, and they believe that success and perfect results will come from innate talent alone, without effort. Mistakes and great effort are then viewed as signs of failure.

People with a growth mindset view innate talent as the foundation, so abilities can be developed through hard work. Since working hard and making mistakes are considered to be necessary for improvement, they are viewed as signs of progress toward mastery, encouraging further effort.

Blank-Edelman emphasized that making mistakes is the way to innovation. He summarized the way that individuals can practice replacing fixed mindset thoughts with growth mindset actions.

Intrinsic vs. extrinsic motivation was discussed next, referring to Daniel Pink's book *Drive*. Although sysadmin work can contain many algorithmic (repetitious) tasks, quite a bit of it is more heuristic and can contain a lot of intrinsic reward. External rewards can motivate for a while, but the effect will wear off and result in poorer work quality overall. Elements of a work environment that can boost motivation are autonomy, mastery, flow, and having a sense of purpose.

Keeping these elements in mind, the discussion turned to making change. Blank-Edelman suggested changing just one thing as a way to start, such as increasing autonomy by negotiating to own a whole project rather than just helping with part of it. Mastery of new technology could be developed by using virtual machines to create an environment safe for experimentation that would not affect production. To effect change in the organization, the focus should be on persuading the large group of people who are neither advocates nor opponents of the new idea, as this would give a clear majority in favor.

Finally, Blank-Edelman explained ways to change your own perception—especially necessary as a sysadmin, where continually changing goals can become frustrating. He gave examples of ways to turn tasks into games and to increase motivation to do them. Several books were displayed for further reading and inspiration.

After the talk, one audience member recommended the "Quantified Self" Web site as an additional resource.

Project Cauã

Jon "maddog" Hall, Linux International and Project Cauã

No report is available for this session.

Closing Session

What Is Watson?

Michael P. Perrone, Manager, Multicore Computing, IBM T.J. Watson Research Center

Summarized by Rik Farrow (rik@usenix.org)

Michael gave the closing talk, thanking his audience for sticking around. He explained that although he had worked on Watson, he wasn't responsible for most of the algorithms that provided the magic sauce. He then went on to tell his audience how Watson is different from other forms of search.

Michael began by asking how many people in the audience had seen the PBS show, then if anyone was not familiar with *Jeopardy*? One man from Scotland piped up, another person said he lived under a rock. Michael dodged, then went on to say that in the past, grep was his search tool. Next, Google became the tool of choice, but using Google requires putting some thought into the proper search terms.

Jeopardy poses much more difficult problems to solve. Michael provided some example *Jeopardy* answers, using them to illustrate the importance of being able to parse natural language and tease out the important parts of each answer. Winning at *Jeopardy* also requires broad knowledge and quick answers.

Over 350 TBs of text were parsed to create syntactic frames, and the results processed again to create semantic frames. A semantic frame might be "Water is a fluid (.9)" or "Ships sink (.5)," where the number represents the degree of certainty. As more text is processed, it is cross-correlated with existing frames, increasing or decreasing certainty. Simply matching frames with answers doesn't work, because finding matches may involve temporal or geospatial reasoning, statistical paraphrasing, decomposition, and synthesis.

Michael took a break for questions, most of which he said he would answer later in the presentation. Then he presented some examples of early responses by Watson, provoking laughter and applause because certain key engines were lacking. For example, under the category Milestones, the answer "In 1994, 25 years after this event, [one] participant said, 'For one crowning moment, we were creatures of the cosmic ocean.'" While the correct question was, "What was the Apollo 11 moon landing?" Watson posited, "What is the Big Bang?" as Watson had no engine at that point that took time into account as a constraint. There are hundreds of engines in Watson.

Someone asked how they measured reliability, and Michael sighed and said they don't have a good answer for that. One way was how many times some particular evidence appeared and whether that evidence appeared in more documents. How did they monitor such a complex system? There was a team of about 20 people working on various aspects of certain engines, and if something was wrong with an answer, they could direct the result to the person or persons handling the related engine. An audience member pointed out that categories themselves can have puns built into them, and Michael agreed, saying that categories have to be analyzed by pun engines, just like words surrounded by quotes. Someone asked how much they studied humans, and the answer was "a lot." There are lots of strategies in *Jeopardy*, and they compared their strategies to human strategies. Humans frequently start at the top and work down. Watson doesn't. Watson goes to the bottom rows, going for the Daily Doubles, which allow players to double their winnings.

Michael then displayed a graph showing Watson's progress compared to human winners. Over the four years of the project, Watson went from very poor to the territory of the best human players. Michael said that he was worried that Watson would get too good, and that could result in a backlash against devices like Watson. On a single node, a single question would take about 2 hours on a 2.6 GHz to run. They parallelized the task over 2880 cores, reaching 2.6 seconds, which is what they needed to compete with humans. Adding more cores may not help much, as there is a certain amount of overhead.

The real goal is to make Watson useful. One of the first areas of interest would be in answering health questions. Other areas could be tech support, business intelligence, and improved information sharing in government. Someone quipped, "Skynet," at the mention of government, but the goal is improving citizens' ability to get answers quickly from government. Someone else asked how much time it would take to specialize Watson for something else. Michael answered that they have built tools for analyzing data and could reuse those tools. The same person pointed out that the *Jeopardy* version had betting that relied on the confidence that an answer is correct. Michael said that the betting engines wouldn't be needed, but a lot of other engines would prove useful.

Michael saved the hardware slide for last: 90 Power 750 servers, 2880 POWER7 cores at 3.6 GHz, with 16 TBs of memory and 20 TBs of disk, in 10 racks. These run SUSE Linux, UIMA (Unstructured Information Management Architecture) software, their own software, and Hadoop. Watson is good, but it takes 80 kW of power and 20 tons of cooling. The human brain fits in a shoebox, can run on a tunafish sandwich, and can be cooled with a hand-held paper fan. We have a long way to go, Michael said.

How cost-effective was this project? Michael said he didn't know, but the publicity was priceless. Does Watson teach us anything about human brains? Michael answered that he likes to think about this. The algorithms they use are statistically driven, and he wouldn't want to tie this to human brains too tightly. Michael said perhaps he could discuss this over a beer later.

Someone asked about the buzzer. Michael showed the setup for *Jeopardy*, and explained that buzzers are not active until the game host has finished reading the answer. Watson has control of a solenoid that presses its button. Michael also pointed out that humans can hit their button when they intuit that they know the answer, while Watson will not answer until it has calculated the answer. Were linguists involved? No, that while natural language experts were involved, what was most important was to create a system that could learn. How many sysadmins were used? Just one or two, as the whole project was run on a shoestring. Had they rated provenance? Yes, they ranked their import sources, where an encyclopedia was rated with more confidence than Twitter, as an example. How had they handled software updates? Michael didn't know for certain.

Workshop Report

Advanced Topics Workshop

Summarized by Josh Simon (jss@clock.org)

Tuesday's sessions began with the Advanced Topics Workshop; once again, Adam Moskowitz was our host, moderator, and referee. We started with our usual administrative announcements and the overview of the moderation software for the new folks (more than in any past year). Then we went around the room and did introductions. Businesses (including consultants) outnumbered universities by about 9 to 2 (up from 4 to 1); over the course of the day, the room included 6 LISA program chairs (past, present, and future, the same as last year).

For the third year in a row, our first topic was cloud computing. We still don't have a common definition, though the room seemed to agree that we're moving toward the "Whatever as a Service (WaaS)" model with software, platform, and infrastructure as the most common. One problem is the relatively low amount of data on the scalability of the services; when the cloud is abstracted away from our control, there can be problems if production has capacity or bandwidth or speed requirements. When you grow in 18 months as big as the current cloud, that won't work. Anything with growth may not be appropriate for the cloud, though that's not necessarily true for well-understood and well-behaved Web applications. For some, the consumer view of "A place somewhere out there to keep my data" is a good definition. This isn't new to most of us. Businesses with certain regulatory requirements (FERPA, HIPAA, and SOX) may have requirements preventing them from moving certain data (and thus the processing thereof) to the cloud. The ability to spin up a machine and provision it via some configuration management system without having to do actual work (racking, connecting cables, and so on) is a good thing. We'll probably come up with different terminology in the industry.

Next up we discussed increasing regulation of Internet activity. Governments don't seem to have a clue about the Internet; each country doesn't seem to understand that they don't control the whole Internet. There's a lot more censorship on national boundaries (Australia, China, and Egypt were mentioned, and before we went to press the USA had legislation pending as well), and we're concerned where this might be leading. SAGE-AU managed to get the Australian legislation put on hold. The room seemed to be split on whether lobbying would really have any effect, though stewardship (such as ARIN for IP addressing) might be a good thing. This was a fairly gloomy discussion. One person noted that we have to give politicians an alternative; they'll take the most expedient thing. We need more companies to help enable the environment we want.

Our annual lightning round of new-to-you tools seemed to fall into two categories: software (Augeas, cloud-based VM systems, Dropbox, Evernote, f.lux, git, Google+ Hangouts, Internet in the pocket (any smartphone), OneNote, OpenStack, Puppet, Trac, vimdiff, vpnc, WordPress) and non-technological (changing jobs, getting engaged, new mattress, paying others to do work for you, taking vacations, and team-building exercises at shooting ranges).

That segued into career paths. The general question was how to move out of a too-stable, unchanging environment where there's no opportunity for growth without going into management; several believe they're in that kind of workplace. Companies increasingly claim, "We're interested in people who've been around for a while," but the reality is that they're hiring younger, inexperienced people who are more willing to work ridiculous hours for less money. Becoming senior often leads to becoming siloed. We took a straw poll: one person has been at his job for 17 years; about half a dozen were at seven years or more. Having a technical growth path is important. The problem with even tall technical tracks is they get narrow pretty quickly. Having other senior people around (even in different silos) to learn from can be helpful.

On the subject of interviewing, understanding the deeper concepts is much better than trivia; make the interview questions open-ended so you can see how the candidate thinks.

When you interview for a new job, always target the job *after* that. Remember that you're interviewing them as much as they're interviewing you. It's also probable that you know more than you realize. Practice interviews are good. Honing your higher-level thinking and problem-solving skills is also useful. We all have contacts; use your networks and possibly bypass the formal recruiting process.

On the subject of hiring, several have had problems finding enough high-quality people in the pipeline. Finding those who're interested in looking at the big picture is problematic and frustrating. One person believes that intelligent companies don't care so much about you knowing everything already but just being "clue-ready" to pick up their oddities (although HR has been filtering a lot on the specifics). Hiring managers working with HR to build the filter may be helpful. However, another is seeing the opposite: word on the street is that managers want very specific things. This may be region-specific. Any company needs to understand there's a learning curve, and hire people with clue so they can learn the specific technology. It's not a bad thing to come in understanding the space even if you don't have specific expertise. Demonstrate proficiency on the stuff you're doing now and how you've been able to pivot in the past. However, it may depend on where you're going: Big outsourcing organizations nowadays seem to look for the specifics so they can hit the ground running at the client, whereas research organizations or universities may be more willing to hire clue-ready people without specific skills or experience in a specific technology. One person had a senior position open for six months; they'd find a candidate they liked, but would take too long to get back to them and the candidate would slip away. They wanted to hire a candidate to revitalize and reinvigorate the team. They got a new recruiter on the HR team and within a month they filled all three open positions with amazing people. Sometimes you really do need a good recruiter.

The next major discussion was about what the DevOps and sysadmin community needs but doesn't have. We already have contacts, national and regional conferences, some sorta-magazines, a mentoring program (LOPSA), and mailing lists. One immediate response was lobbyists, linking back to the previous discussion on regulation. Some disagreed, believing that improving public perception of what we do would be helpful even without political lobbying. One believes that "sysadmin" is too narrow a term; many of us do more than just systems. We'd be better served as a community if we had better labels (for example, service administration): it's systems, databases, services, networking and connectivity, and so on. DevOps is another facet of the whole, and it's being integrated, but names are important and the "sysadmin" name may be too restrictive. One possible problem is that a universal professional identity is missing from the field. The medical profession (doctor/nurse) was brought up as an example. However, humans only work in specific known ways; IT can work in many different ways, so it's more complicated.

One tangential discussion was on the term DevOps. Some see it becoming as much a buzzword as cloud. We're not integrating the big DevOps communities into the USENIX/SAGE/LOPSA community. Is it "deploy multiple times a day to Production"? "Continuous integration via Hudson or Jenkins"? It should also be remembered that what works (or not) for Web sites definitely won't for larger enterprises. Even configuration management hasn't penetrated as much as people seem to think it has. We don't have best practices for CM yet. We don't have best practices for code review yet. There are no white papers on this.

After our lunch break we took a quick poll: only 11 of the 26 present at the time still run their own email service (either at home or offsite), and nine more have stopped doing so in the past year. One hasn't outsourced because it keeps his skills sharp. Another has his public blog adminned by someone in Romania for $50 every time the blogging software needs to be updated.

Our next discussion was on large scalable clustered storage. One company represented generated a lot of data (1 TB/day) that they need to keep forever, and they see that growing to 10 TB/day. The question was, what are people looking at for data? Are they staying with spinning media or moving toward flash or other solid-state drives? Much depends on your use profile; one site uses EMC Celera for non-parallelized storage, but their profile is user home directories and scientific data in an NFS model. Most people with large storage needs seem to be using GPFS. Other mentioned products include Fusion I/O cards, Infiniband, NetApps, and Violin. The network wonks present wondered about the network behind this large storage; consensus seems to be to use dedicated networks, though some have upgraded their network switches to terabit backplanes. On the subject of failover, most seem to be failing over the servers but not necessarily the storage independently from the servers.

Next we took a quick lightning round asking what the next useful fad will be in the next two years. Answers included configuration management, death of tape, decline of social networking, increasing use of app store-like software distribution within companies, infrastructure as a service (IaaS) increasing, IPv6 deployment, JSON APIs, mobile security, more private cloud products, moving away from big iron databases toward NoSQL/MongoDB, moving away from running machines toward providing APIs, moving away from

the cloud back to local, SSD not spindles as primary storage, statistical analysis about systems, UI improvements (facial recognition, motion detection, and Siri- or Watson-like interfaces), and virtualization.

We next discussed workstation replacement. Only four people said they use virtualized desktops. Some environments reimage the workstation on logout (mainly in public labs), and most seemed to prefer physical workstations, due to performance issues. Environments that use spare CPU cycles for processing (such as Condor) prefer physical to virtual workstations for performance reasons. Virtual desktops assume high-bandwidth and low-latency networking between the user and the physical hardware, which is not universally true. Furthermore, most seem to think virtualized desktops don't save money; hardware costs are falling and local processors and capabilities are getting cheaper, so centralizing the services for anything other than administrative overhead may not have a benefit except in areas where power and cooling are your expensive limiting factors.

Next we discussed life balance and stress management. IT culture seems to still be 60- to 80-hour work weeks, which leads to a lot of burnout. Some places bought toys like ping-pong tables ("startup mentality"), but we should change the culture more toward mentoring the younger or juniors, learning how to say "No" despite pressure, and educating management to have them cause less stress. There's a difference between good stress ("I bet you can't do this over the weekend...") and bad stress ("... or you'll be fired on Monday"). At one represented employer it's good to hit or be within some percentage of the service-level agreement, but bad to be outside that percentage, even if it's responding too fast. In other words, meet but don't exceed your SLAs.

IT often manages to pull a magic solution when backed into a corner, so expectations are set (perhaps unreasonably) high. One method of pushback is to say, "Here's what I'm working on; what do you want me to drop to work on this new thing?" and let management make the call. If your work runs into your personal time, you can use some of the work day to recover (such as running errands, making doctor's appointments, etc.). One person noted that adding a fitness regime can help with stress as well, though not even half of those present have a regular fitness routine. Another person pointed out that there are strict rules for what overtime is allowed in Europe, and there was a brief tangent on cultural differences between US and European time expectations.

One person's employer allows everyone to take one day per month to not come to work (managing their time); the requirement is to remain in town and available if you're needed. They tend to use it for kids' functions or doctor's visits. The company president's general attitude is that if there's a crisis of technology, he asks whether anyone's going to die if it isn't fixed immediately. The trick is convincing your management of that. However, if management doesn't support having a work-life balance, you're working in the wrong place. The final comment was that you get more respect by respecting yourself and enforcing your own work-life balance.

We had a very brief discussion about patents. There have been a lot of lawsuits about technology. One person was subpoenaed in a patent suit between two companies he'd never heard of; having written a PAM module a decade ago was apparently evidence of prior art. Do software patents help or hinder innovation? The way the (US) law is written and the decision is done, except for clear prior art the Patent Office has to grant the patent because something isn't prohibited, which can hinder innovation. How sysadmins look at things is different from how the law is written. LISA is important because papers help show prior art. A couple of years back a commercial company tried to patent configuration management, but Anderson's 1994 paper was prior art such that the patent was denied. The best way to fight this is publish your work; once it's published, it's prior art. However, many are held back by fear of being sued for violating someone else's patent.

Next someone asked if there was management-level publicity about sysadmins going away. Some are seeing a lot of this, in part because developers see that cloud services let them jump right to release. Others noted that the thought of some new technology making sysadmins obsolete has been around for the past decade, such as with autonomic computing. One person suggested that we could change the sysadmin role away from "operations drone" toward "architect." With the automation and configuration management tools we have today, many of the "mindless" tasks can be automated away, and the sysadmin can take on more of a higher-level architect, designer, or decider role and improve the service and infrastructure. Another idea was to have a gatekeeper between Development and Production, selling that their knowledge of security, process, scalability, and so on is important and relevant. One person's environment bills every product and service back to the requesting department. It was noted that the real answer depends on the actual cause. What's tickling management's nerves? If it's cost, argue about the cost to the business in the event of outages, in terms of financial impact, publicity, and goodwill.

After the afternoon break, we discussed women in technology. One of our participants is involved in a number of research areas and focus groups. They asked if we're seeing women in technology, if they are showing up in applicant pools (under- or overqualified), if we have any outreach

ideas, and so on. One environment has a lot of women in both tech and leadership roles and is seeing qualified candidates of both genders, although women tended to be more on the development than the sysadmin side, and there were almost no women DBA candidates. Another environment has a lot of female developers and project and program management, but practically none in service engineering/SA.

Some say that IT in general has a lot of unfriendliness toward women. One person observed that when we say, "We're not creating a hostile work environment," it may be untrue. We need to treat candidates or colleagues solely on their technical merits, not on their gender. In the past, women needed to be aggressive enough to get past the Old Boys' Network to get in. Also there's the culture of "She's not that good" from guys, which may be subconscious from many men. One person noted that the unconscious gender-biased behavior is learned. One job he was at had 40% women in technology. They felt little to no bias against them because so many were there it was considered "normal."

One person has been interviewing students for a decade and in that time he's had all of three female applicants. He was able to hire one; the other two weren't the best for the job at the time. That one has since left, in part because she didn't have the skill set yet. It's too late if we wait until they're in industry; we need to get them involved earlier. We need to instill the interest in technology at a younger age (college is too late). He sees no non-US females and only a small number of women overall. Most in the room seem to agree that we need more women in the field; we need to get more women (girls) involved in science, technology, engineering, and math (STEM), especially where there are no tech classes. However, we've observed that SEM is easier than T.

Tangentially, someone was triggered to think about statistics by a previous discussion. We're likely to look at more complex metrics over time that are statistically defined (95% of requests under *n* milliseconds, 99% under *m* milliseconds, and so on). Running large-scale services, you can't use "10% above peak" as a metric. It's also an educational problem. We need to think statistically about latency and capacity and what's "good enough." Similarly, we need to move from "I ran this test 10 times and got results of x" and toward "I have a 95% confidence that...," which is a better metric. We're also seeing a drive for comparisons (such as this week versus last week) and trending analysis. Several people think this could make a good Short Topics book. The final comment was that you have to know what's *normal* before you can define *abnormal*.

We ended the workshop with our final lightning round, asking what is going to be new-for-you in the next year. Answers included architecture design and development, career planning (finding better jobs, increasing team visibility in a good way, managing the existing job, moving between generalized and specialized), data mining and statistical research, decommissioning old hardware, delegating everyday tasks, doing more DevOps type work, hiring more people, managing more data, publishing new books, recreating one's environment from the ground up, and training interns. However, for some, not much is changing that quickly.

5th ACM Symposium on Computer Human Interaction for Management of IT

Boston, MA
December 4–5, 2011

Summarized by Kirstie Hawkey (hawkey@cs.dal.ca), Nicole Forsgren Velasquez (nicole.velasquez@pepperdine.edu), and Tamara Babaian (tbabaian@bentley.edu)

The 5th ACM Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT) was held in Boston, Massachusetts, on December 4–5, 2011 (visit chimit.acm.org for detailed information about its organization and program). Information technology (IT) is central to modern life and occurs in our homes as well as in enterprises. It is important that we develop usable solutions for those using, configuring, and maintaining the software and hardware components (e.g., wireless access points, network routers, firewalls, virus scanners, databases, Web servers, storage systems, and backup systems) that support our work and personal lives. CHIMIT has been addressing human-computer interaction for IT management since 2007. As in past years, the CHIMIT symposium was held adjacent to the USENIX LISA (Large Installation System Administration) conference to encourage attendance by the system administrators whom CHIMIT researchers aim to support. However, it is important to note that CHIMIT research also considers the problems of home users, who too often take time and resources away from the real work at hand in order to manage the underlying IT infrastructure.

The morning session began with a keynote address by Marti A. Hearst, "Bringing HCI to the Federal Government." Marti is an HCI professor in the UC Berkeley School of Information who has been on leave of absence to serve as the Chief IT Strategist at the US Patent & Trademark Office. Her talk focused on how the infusion of usability techniques helped the current administration transform the way US federal agencies design and build information technology. In the afternoon, Kyrre Begnum (Oslo and Akershus University) presented the invited talk, "What a Webserver Can Learn

from a Zebra and What We Learned in the Process" on behalf of his co-author Johan Finstadsveen (University of Oslo). This engaging talk had the audience considering how the defensive tactics of wildlife species can inspire new defense mechanisms for our IT infrastructure. We also enjoyed two invited presentations that exposed CHIMIT attendees to relevant research published in other venues last year. These included "The Margrave Tool for Firewall Analysis" (T. Nelson et al.), which was prsented at LISA in 2010, and "Heuristics for Evaluating IT Security Management Tools" (P. Jaferian et al.), which won the Best Paper award at the Symposium for Usable Privacy and Security (SOUPS) in 2011.

The technical program included three presentations of papers related to the work of system administrators. The first, "Understanding and Improving the Diagnostic Workflow of MapReduce Users" (J.D. Campbell et al.) was a joint submission from nine researchers from Intel Labs Pittsburgh, Carnegie Mellon University, and DSO National Laboratories in Singapore. This was followed by research from IBM, "Description and Application of Core Cloud User Roles" (T. Bleizeffer et al.). Another presentation by IBM Research Brazil (C. de Souza et al.), "Information Needs of System Administrators in Information Technology Service Factories," provided a novel perspective on the work in large IT service factories. We had one presentation, "Third-Party Apps on Facebook: Privacy and the Illusion of Control" (N. Wang et al., Pennsylvania State University) that focused on end users and the privacy issues they face on Facebook due to the actions of third-party applications.

The poster session included four posters from academic and industrial research and generated a lively exchange between the conference attendees and poster presenters. Lance Bloom from Hewlett Packard presented design principles for IT storage capacity management. Ryan Dellolio from George Washington University described a service-oriented, user-centered approach to business process management for back-office IT operations. Kurt Keville from MIT described the technologies he and his colleagues used to create and test a High Performance Embedded Computing (HPEC) cluster. Kirstie Hawkey from Dalhousie University presented her research agenda in visual analytics for system administrators.

On Monday, December 5, a special joint workshop was held with LISA, bringing the CHIMIT research community and practicing system administrators together to discuss current topics and interests. The workshop hosted 12 participants hailing from several sectors, including academia, government, and both large and small organizations. Overlapping areas of interest identified in the workshop included, but are definitely not limited to:

- The importance of mental models in system administration. How can we help with the documentation and development of tools and visualizations?
- Related to mental models is complex system visualization. How can tools help system administrators "see" their system at various levels of detail and from different perspectives?
- Knowledge management and knowledge sharing, both within teams and across organizations. How can we more easily capture the information? How can we better structure that information for both browsing and searching?
- Communication and collaboration. System administrators work at the intersection of several groups of stakeholders: how can we better understand their work and responsibilities to help them communicate and collaborate more effectively and efficiently?
- User experience in system administration. Many of us focus on graphical user interfaces, which are very important, but what does a good user experience mean for a command-line interface? How can we improve the user experience in text-based environments? As an extension, how can we use usability studies to help us identify the pitfalls in scripting languages? What heuristics are useful?

Feedback from the workshop has been very positive, with participants indicating they were pleased with the topics, mix of participants, the opportunity to network and test ideas, and the overall concept of mixing the two groups. Outcomes of the workshop include joint research projects and a possible magazine article. The workshop organizers offer a special thanks to ACM SIGCHI, who provided a grant to allow this workshop to take place.

All in all, we consider CHIMIT to have been successful this year, as we achieved our goal of fostering collaboration between researchers in fields such as human-computer interaction, human factors, and management and service sciences, and practitioners in the management of large IT systems. However, we continue to seek ways to increase the number of submissions to CHIMIT and the number of attendees as we look to the future. We are currently in the process of restructuring CHIMIT to try to better meet the needs of a greater cross-section of its constituents. Kirstie Hawkey, General Co-Chair (hawkey@cs.dal.ca), and Paul Anderson, Technical Co-Chair (dcspaul@ed.ac.uk), are beginning their second terms on the Organizing Committee. They would welcome suggestions about how to improve the next CHIMIT.

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