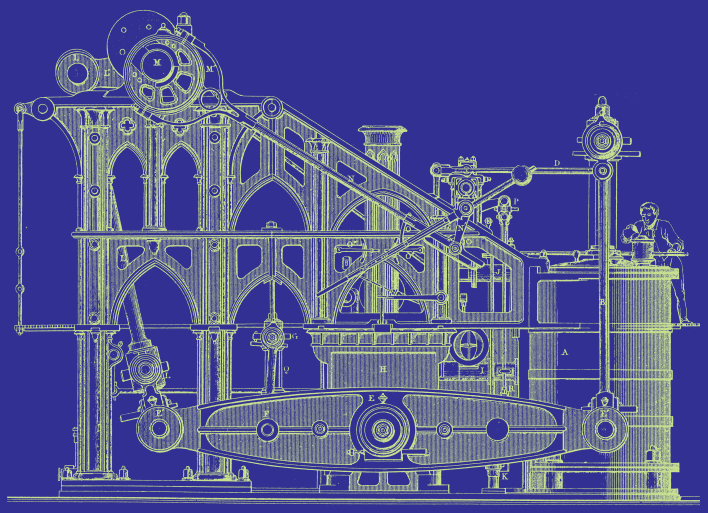
AUGUST 2008 VOLUME 33 NUMBER 4

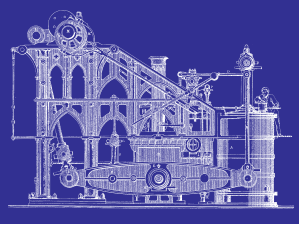
;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events



22nd Large Installation System Administration Conference (LISA '08)

Sponsored by USENIX and SAGE

NOVEMBER 9–14, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/lisa08

Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT '08)

Sponsored by ACM in association with USENIX

NOVEMBER 14–15, 2008, SAN DIEGO, CA, USA
http://www.chimit08.org

ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008)

DECEMBER 1–5, 2008, LEUVEN, BELGIUM
http://middleware2008.cs.kuleuven.be

Fourth Workshop on Hot Topics in System Dependability (HotDep '08)

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/hotdep08

First USENIX Workshop on the Analysis of System Logs (WASL '08)

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/wasl08
Paper submissions due: September 2, 2008

Workshop on Power Aware Computing and Systems (HotPower '08)

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/hotpower08
Paper submissions due: September 11, 2008

8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)

Sponsored by USENIX in cooperation with ACM SIGOPS

DECEMBER 8–10, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/osdi08

Third Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML08)

Co-located with OSDI '08

DECEMBER 11, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/sysml08
Submissions due: September 26, 2008

7th USENIX Conference on File and Storage Technologies (FAST '09)

Sponsored by USENIX in cooperation with ACM SIGOPS, IEEE Mass Storage Systems Technical Committee (MSSTC), and IEEE TCOS

FEBRUARY 24–27, 2009, SAN FRANCISCO, CA, USA
http://www.usenix.org/fast09
Paper submissions due: September 12, 2008

2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)

Sponsored by ACM SIGPLAN and SIGOPS in cooperation with USENIX

MARCH 11–13, 2009, WASHINGTON, D.C., USA
http://www.cs.purdue.edu/VEE09/
Abstracts due: August 29, 2008

First USENIX Workshop on Hot Topics in Parallelism (HotPar '09)

MARCH 30–31, 2009, BERKELEY, CA
http://www.usenix.org/hotpar09
Submissions due: October 17, 2008

6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

APRIL 22–24, 2009, BOSTON, MA, USA
http://www.usenix.org/nsdi09
Paper titles and abstracts due: October 3, 2008

12th Workshop on Hot Topics in Operating Systems (HotOS XII)

Sponsored by USENIX in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)S

MAY 18–20, 2009, MONTE VERITÀ, SWITZERLAND
http://www.usenix.org/hotos09

For a complete list of all USENIX & USENIX co-sponsored events, see http://www.usenix.org/events.

contents

;login:

VOL. 33, #4, AUGUST 2008

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
David Couzens
proofshop@usenix.org

PRODUCTION
Casey Henderson
Jane-Ellen Long
Michele Nelson

TYPESETTER
Star Type
startype@comcast.net

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

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.

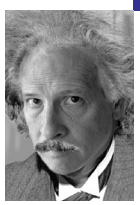
$90 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $120 per year.

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.

RIK FARROW

musings



rik@usenix.org

I FEEL LIKE I'VE FINALLY ENTERED THE future. I now have a microwave dish on my rooftop, watch television using a computer, listen to radio from online streams, and am even installing my own solar power plant. But when it comes to security, we are all still stuck in the Dark Ages.

In this issue you can read about a Web browser designed with security in mind. Don't think for a minute that current Web browsers include security in their design, because security is, at best, something added on as "desirable." Sure, Firefox, IE, Safari, and Opera are all carefully vetted for bugs and poor programming practices. And all also have security "features." But these browsers all share one thing, something that all browsers have done since Mosaic: They download and execute remote code in your own user context.

Just think about it for a moment. Would you routinely go off to strange Web sites, download code, and run it? Well, of course you would, because five nines (99.999%) of the people I know do exactly that. The 0.001% don't use Web browsers. And I am not kidding. I know one security geek who is still using nc for "browsing" the Web. That guy sure has to work hard to read Web pages (especially these days!), but he is secure, as the text he reads has been eviscerated of any danger because it is treated just as text.

Drive-by Downloads

I wrote about drive-by downloads exactly one year ago. Niels Provos and his team at Google [1] are still trolling caches of Web pages looking for pages that cause the downloading of first-stage exploits. And they are still finding them—thousands every day. The vast majority of these Web pages weren't made that way by their owners, but their Web server was exploited, and small changes were made not just to stored pages but also (in some cases) to dynamic content as well. All it takes is the insertion of an `iframe` or `script` tag to take control of almost any Web browser that visits the page.

Exploiting personal computers is big business and is completely untaxed and unregulated (a libertarian's dream, in that sense). Exploited systems are used in relaying spam, exploiting other systems, and to collect identity information. Oh, I forgot to mention DDoS, but when you control thousands of distributed systems all with Internet connections, that should be obvious. All of these activities are il-

legal (apparently with the exception of the U.S. record industry [2]), yet occur routinely.

The OP browser's design (page 14) does what all browsers should—isolate one site's content from another site's. With the OP browser, you don't get multiple sites' content all running within the same security context. Each site runs within its own page-rendering process. And actual display of contents is done by yet another process, also running within a secure environment. The current browser design runs everything within your standard security context, and that means that anything you can do, your browser can do too. Sure, browser designers do make honest attempts to limit what your browser can do to your system, but these limitations have failed over and over again. And they must continue to fail, because the basic browser design is so terribly flawed.

We run our desktop systems as if they were mainframes with multiple users. The operating systems were designed for multi-user systems (unless you are still running Windows 95 or Mac OS 9), yet you (and your exploiter du jour) are the only ones using your computer. Our processors are also designed like old time-sharing systems. Neither has kept up with the way computers are actually used today. The biggest reason for this is inertia, in that OS designers build familiar systems, and CPU designers take advantage of their decades of building the same architecture.

I don't want to suggest that starting over will be easy. We need a way of transitioning from our insecure desktops, laptops, and smart phones to systems designed with security, performance, and efficiency from the ground up. The OP browser provides a model for the transition. But without all the sexy features and performance we have grown so accustomed to, the OP browser will not replace Firefox or IE.

Our current situation reminds me of the bad guys in the movie *Who Killed the Electric Car*? [3]. In that movie, General Motors (GM) recovers the hundreds of electric vehicles that people had leased and has them crushed—even when the owners offer to pay $24,400 for each several-year-old car rather than allow them to be taken away and destroyed.

The villains in that movie were all part of the status quo: manufacturers who owned factories that build internal combustion engine vehicles, oil companies with vast refineries and distribution systems, and even vehicle maintenance suppliers (since the GM EV1 had three items that needed to be replaced: tires, brake pads, and windshield washer fluid). So instead of another ten years' experience with electric vehicles, we have $4/gallon gas (perhaps it will be $5 when you read this) and a glut of SUVs that have lost their appeal (and most of their resale value). Note that consumers were also found guilty in the film (as they were not willing to change).

Have we reached the $4/gallon point in terms of desktop insecurity? I really don't know, but I do make Linux Live CDs for my friends who need to manage some of their finances online. Today, I wouldn't type anything on a Windows system that I wanted to keep private, and I almost feel the same about Macs and even Linux systems. They all share similar design flaws, in that one user runs remote code via the Web browser and mailtool as well. Linux, BSD, Mac OS X, and other UNIX-like systems at least separate the user from the administrator, making kernel-level exploits more difficult. But the installation of code within the browser will capture all of your keystrokes, and tools have existed for some time that selectively filter out those keystrokes that look like credit card numbers or what you have typed during a visit to a large number of domains, all related to finance. These captured keystrokes then get posted to a Web server under the control of the dark industry.

The Lineup

I really hate being so negative, but I do prefer to be brutally honest, even at the risk of sounding like a broken record. The Grier article about the OP browser gave me the perfect opportunity to sound off about this issue again. But all is not dark and dreary.

In this issue, we lead off with an article from Brandon Enright and some researchers from UCSD. While building an internal tool to track Storm botnet infections within the campus network, the team developed Stormdrain, an efficient means for discovering Storm bots wherever they are. You can read their LEET '08 paper [4], but I asked them to write about something a bit different: their experiences when they discovered that there were whole families of Storm-bot pretenders to uncover as well. Some of these look-alikes were created by other Storm researchers, and others represent attacks against Storm by other botnet owners. During the LEET workshop presentations related to Storm, it was common to hear comments like "That was you?" or "I owe you a beer" from researchers who had collided with them in the Storm network.

Colin Dixon and his co-authors wrote an article about their proposed DDoS defense for sites that generate dynamic content. DDoS is still a real issue eight years after these attacks first made headlines, and ways of providing access to servers with dynamic content while under attack—and that don't involve massive changes to the Internet's infrastructure—are rare. I liked what I heard during their paper presentation during NSDI '08 and asked them to write this article describing their very interesting approach.

Diana Smetters and her co-researchers had presented a paper at the UPSEC workshop that dealt with issues I felt we all face routinely. In the course of our work, we often need to share information electronically, yet doing so without running foul of security policies is just about impossible. Smetters and her co-authors interviewed people who needed to share data as part of their jobs, and they report just how these people actually handled sharing. I think there are lessons in this article for all of us, whether we write the security policies or violate them with file-sharing practices.

Rohan Murty and his co-authors present a different approach to providing better Wi-Fi bandwidth. In this article, related to their NSDI paper, Murty explains how they went about using existing infrastructure (plus some Wi-Fi cards) to control how wireless clients associate with access points, without modifying the clients.

Alva Couch thoughtfully provides us with his own viewpoint about why we need real standardization in sysadmin. Couch cogently explains not just why he thinks we need standards but what these might look like, and he provides real-world examples.

David Blank-Edelman presents us with cool Perl debugging techniques. Peter Galvin offers his favorite list of procedures for analyzing system problems, as well as opening this topic up for discussion in a wiki. Dave Josephsen demonstrates relatively unknown tricks that you can do with RRDtool, as long as you can grok RPN (Reverse Polish Notation). Finally, Robert Ferrell revels in cajoling us into considering using memes as a method of sharing encryption keys.

We have five summaries, including two from the recent BSDCan '08 conference in Ottawa. We also have the NSDI and LEET summaries. Finally, we have the short version of the summaries of WOWCS, the Workshop on Organizing Workshops, Conferences, and Symposia for Computer Systems, a

meta-workshop. Given the importance to researchers of getting their work published, the WOWCS workshop summaries should be required reading. WOWCS discussion covered many of the issues involving Program Committees, so if you plan to participate in a PC or submit a paper to one, I suggest you read this summary (or the longer version found at [5]).

As our personal computing devices get more powerful, and even as our cell phones may soon take over for our laptops, we need real security. What's the point of having computers if they can't be trusted? Our security model, our OS model, and even our CPU/system architecture were designed for a long-gone time. It is time for us to prepare for a secure future.

REFERENCES

[1] Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware: http://www.usenix.org/events/leet08/tech/full_papers/polychronakis/polychronakis_html/.

[2] MediaDefender shuts down legitimate BitTorrent tracker with DDoS attack: http://blog.wired.com/27bstroke6/2008/05/mediadefender-d.html?cid=117123750.

[3] http://en.wikipedia.org/wiki/Who_Killed_the_Electric_Car?.

[4] The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff: http://www.usenix.org/events/leet08/tech/full_papers/kanich/kanich_html/.

[5] WOWCS scribe notes: http://www.usenix.org/events/wowcs08/tech/WOWCSnotes.pdf.

BRANDON ENRIGHT, GEOFF VOELKER, STEFAN SAVAGE, CHRIS KANICH, AND KIRILL LEVCHENKO

Storm: when researchers collide

Brandon Enright is a network security analyst at the University of California, San Diego. He is primarily interested in malware and exploit research.

bmenrigh@ucsd.edu

Geoff Voelker is an associate professor of computer science at the University of California, San Diego. He works in computer systems and networking.

voelker@cs.ucsd.edu

Stefan Savage is an associate professor of computer science at the University of California, San Diego. He has a BS in history and reminds his colleagues of this fact any time the technical issues get too complicated.

savage@cs.ucsd.edu

Chris Kanich is a PhD student studying networking systems and security at the University of California, San Diego.

ckanich@cs.ucsd.edu

Kirill Levchenko is a PhD student at the University of California, San Diego. His research is focused on network routing and security.

klevchen@cs.ucsd.edu

WHEN IT COMES TO INTERNET THREATS, few topics get researchers and the media as excited as the propagation speed and vitality of modern malware. One such example is the SQL Slammer worm, which was the first so-called Warhol Worm, a term used to describe worms that get their "15 minutes of fame" by spreading at an exponential rate—infecting every vulnerable machine in under 15 minutes [1]. It is ironic, then, that the latest malware to capture the attention of researchers is not one of the shortest-lived but one of the longest, largest, and most successful bots ever: Storm.

Storm got its name from a particular self-propagation spam email subject line used in early 2007: "230 dead as storm batters Europe." Storm, also known as the Storm worm, is not actually a worm. It is hybrid malware: part worm, part bot (a program designed to perform automated tasks), part Trojan, and even part Peer-to-Peer (P2P) client. Unlike SQL Slammer, where all of the research and analysis is, by necessity, post mortem, the architecture and longevity of Storm have made it a veritable researchers' paradise, offering nearly endless opportunities to measure, interact, infiltrate, and even manipulate it. However, this interactive quality is also a researcher's greatest enemy; the ease with which anyone can poke at the Storm network means that to gather any meaningful research data requires identifying and filtering out the noise created by other researchers attempting to accomplish the same task.

History

The Storm botnet that exists today is not the same network or architecture the authors designed and built originally. In fact, the authors have modified the network architecture on more than one occasion in response to enterprising researchers, making life difficult for them. Our player in this cat-and-mouse game is Stormdrain. We originally wrote Stormdrain in early June 2007 as a way to locate existing and future Storm infections on our institution's network [2].

Originally the Storm bot was little more than a P2P client and downloader; the authors would publish URLs to download new components in the P2P network and new functionality (such as the ability to spam) would be downloaded as stand-alone

packages. Within a few months the authors designed and built a new tiered architecture to relay commands and adapted the existing command and control (C&C) P2P network to be an overlay/rendezvous system only. This new network design also allowed the authors to combine all the functionality that was once in separate components into a single unified package that could do everything. By mid-2007 Storm had garnered considerable research and media attention and in October the authors chose to separate the Storm P2P network from the original Overnet/eDonkey network by using a simple encryption scheme. Each big change has slowed researchers down and required additional efforts to continue to track and measure the network. The evolution of the Storm network now makes it necessary for researchers to mention what version and at what time their work on the Storm network took place. Unless otherwise noted, the network architecture discussed in this article is the one that was deployed in May 2008.

A View from Orbit

Much of the research interest in Storm is directly related to its novel architecture. The Storm network is actually two separate networks, each with its own protocol and function. The first is the UDP-based private P2P overlay network built on the Overnet protocol. It is this P2P network that gives Storm much of its resiliency [3]. No actual C&C between the Storm authors and the bots uses this P2P overlay network. The network's only function now is to facilitate advertising and locating proxy addresses in the separate TCP C&C network. This second protocol and network utilizes a custom TCP- and HTTP-based protocol for C&C. This C&C network makes heavy use of proxying to provide several layers of anonymity between the authors in control of the network and the actual bots doing their dirty work [4]. The Storm authors run a series of proxies on the TCP C&C network and UDP P2P network we call controllers. These controllers provide HTTP proxy services on the TCP network and locate Storm infections on the UDP P2P network to turn into proxies. Even though no actual bot C&C uses the P2P portion of Storm, all Storm infections participate in the P2P network so they can advertise/locate proxy services to connect to the TCP C&C network. Overnet does not provide any sort of authentication; in true egalitarian fashion, all peers have equal roles. Because of this, Storm's P2P network is an attractive place for researchers to measure and manipulate it. Additionally, any attack or disruption in the P2P network will directly affect both Storm's proper functioning and the work of other researchers. It is difficult to strike a balance between being a good citizen in the network and potentially damaging it through novel research techniques.

The inherently malicious nature of the Storm network engenders a carte blanche, "the gloves are off" attitude in some researchers. Initially Stormdrain was not prepared to handle the constant stream of misinformation thrown at it. For much of its early life, for example, Stormdrain crashed trying to handle various buggy and maliciously crafted addresses (multicast, 127.0.0.1, 0.0.0.0, etc). It wasn't until Stormdrain stopped making assumptions about the protocol, message fields, addresses, and content of the network that it was able to run for more than a few minutes without crashing.

DHT Modus Operandi

To understand how different research groups can affect Storm or each other, it is important to understand how Storm's P2P network works. The network is based on the Overnet Distributed Hash Table (DHT), which in turn is

based on the Kademlia DHT algorithm [5]. These names and acronyms are not particularly important. What is important is that DHT networks provide two basic functions: storage of some set of values and, later, retrieval of some of those values. In Storm's case, the values being stored and retrieved in the network are TCP C&C proxy IP addresses and ports. The mechanics of the storage and retrieval of data are at the heart of what so many researchers are poking at. To fully understand how Storm's DHT works, a simple analogy is in order.

Suppose you want to build a human-based social message storage system. One simple method would be to designate one person as the "note holding" person. Anyone who wants to leave a message for someone else can simply go to the note holder and give the holder a note. Anyone seeking that note can go to the note holder to get a copy. This system works pretty well in small groups but isn't resilient—if the note holder gets hit by a bus all is lost. A somewhat related issue to resiliency is that not all participants have the same role or responsibilities. A simple fix to this system would be to make everyone a potential note holder. How could that be accomplished? One solution is to introduce envelopes to put the notes in and create a simple addressing system for the people and envelopes. In this new system notes are not handed directly to a note holder. They are first placed in a numbered envelope and the envelope is given to the person whose own number is closest to the number on the envelope. To retrieve a note, you must find the person whose number is closest to the number on the envelope you are looking for. Without more structure, this process would get quite chaotic.

For the sake of simplicity in our note storage system, the numbers chosen by people or written on envelopes will be limited to 1 through 100. Rather than try to assign people numbers as they enter the system, they will be allowed to randomly choose a number for themselves. Also, to assist in finding numbers, any participant can be asked what their number is. Participants can also be asked to provide the number of any other participant that they know about. To aid in finding numbers quickly, participants will keep themselves in numerical order by standing in a line. With this additional structure in place, a dozen people could easily make a working note storage system. If one of the participants wants to store a note in the system, that person can write out his or her note, put it into an envelope, and number it, then seek others in line whose numbers are close to the number on the envelope the participant wants to store. To accomplish this the participant can approach someone else in the line and ask for his or her number. If the numbers aren't the same, the participant can ask for a list of people that the other knows about whose number is closer to the desired number. The person trying to store the envelope can keep doing this, slowly getting closer to a person with the same or a closer number. The would-be envelope storer keeps doing this until he or she is satisfied that anyone closer to the desired number can't be located, at which time the participant hands a copy of the envelope and note to that person. To reduce the negative effect of people holding envelopes who leave the line, it is best to give a copy of the envelope and note to several close participants. If new participants enter this system, they can choose a random number for themselves and go about using this same search procedure to locate the right place in the line to stand.

With this simple system several basic tasks are easily accomplished. For example, to store a note inside envelope "34" in this line, one could walk up to another and say, "What is your number and all the people you know about close to the number 34?" The response may be something like "I'm number 71 and I know persons with number 56 and 51 that are closer to 34."

The next step would be to approach numbers 56 and 51, asking the same question. The responses may provide numbers 37 and 30. When both 37 and 30 are asked the same question, no closer numbers are learned about. This means that 30 and 37 must be the closest numbers to 34. A copy of the envelope and note can be made and given to both people to hold. Putting new envelopes with notes into the line like this is called "publishing." If someone else comes along and wants to find a previously published note, the same system can be used. To search for envelope 34 the search would most likely end up slightly different. Perhaps the person doing the searching knows about some other set of people. The person can walk up to one of these and ask, "What is your number and all those you know about close to the number 34?" The response could be something like "I'm number 20 and I know about persons 29 and 30." When person 29 is asked, that person may respond with persons 30 and 37. The responses from 30 and 37 will be slightly unique, though. Because each has an envelope labeled with number 34 from a previous publish, they will both respond with others close to 34 and include themselves in the list. If, in the course of searching for an envelope, the person asked responds with him- or herself as an answer, it is an indication that the person has an envelope labeled with the desired number and can be asked for it.

Search failure can also be easily detected. If while searching for an envelope number no new close people are located and none offer themselves as a source then it can be assumed that the envelope does not exist and searching can be stopped. Without people standing in an ordered line the bound on the amount of searching someone would have to do before locating an envelope or giving up would be much higher. Finally, the person doing the searching does not need to start right where he or she is standing; the person can "skip" to the closest person he or she already knows about that happens to be close to the desired envelope. If someone searching for envelope 90 has already had contact with person 95, then that person can start the search with person 95.

Although this "line of people" analogy follows just a few simple rules, it should be readily apparent how dynamic the line can be. With people coming and going, reliability is not guaranteed. To try to keep content (notes inside of envelopes) fresh, the content is published not once but constantly, at regular intervals. Also, a person in the line may know about his or her neighbors at one point but, with people entering and leaving the line often, the line members must constantly search for people close to them and announce their presence to others. Acquainting yourself with another peer is called "connecting" in Overnet. Announcing your continued presence to someone you've already connected to is called "publicizing." Searching for another number (for whatever reason) is called "searching." As discussed before, storing content in the network is called "publishing."

The aforementioned setup is nearly exactly the way the Storm P2P network operates. The primary difference in the systems is in the scale. In the Storm Overnet network, rather than only use the numbers 1 through 100 for people and envelopes, Storm uses 0 through $(2^{128} - 1)$, an astronomical number. The number picked by a Storm peer (person) is typically called an Overnet ID or simply OID. The number on the outside of an envelope is typically called a hash or key. Additionally, rather than a mere dozen people in line, Storm has thousands of peers online at a time. The ability to skip around in line to the closest known person is roughly equivalent to the k-bucket system in Kademlia or Chord's finger tables.

Storm's Innovation

The two attributes that have not yet been well defined about this system are how to decide what numbers to assign envelopes and what useful information to actually write down for the notes. One of Storm's most interesting innovations is how it repurposes these pieces of Overnet. Storm has two different types of notes that it wants to publish. The first is used to advertise the ability to become a C&C proxy to the Storm authors. If a Storm node comes online and determines that it is publicly accessible to other Internet hosts, it first tries to announce itself as "proxy-capable" to the controllers.

The other type of note is used by peers who have already been transformed into a proxy by a Storm controller and want to advertise their proxy service to others by generating proxy-advertisement notes. Regular Storm peers that are not publicly accessible and simply want to perform nefarious tasks will seek out a proxy by searching for proxy-advertisement notes. Two different envelope numbering schemes are used but both are based on the current date and each can generate 32 different envelope numbers per day. The first scheme is only used for proxy-capable notes and a hash of the date is used directly. Only the Storm authors search for envelopes with these numbers because only they turn proxy-capable peers into actual proxies (by way of an RSA encrypted packet we term the "Breath of Life" or BoL for short). The second set of 32 envelope numbers is generated from a hash of the date where 1900 has been subtracted from the year. That is, the year 2008 uses 108 as the year. Storm nodes that have become proxies use these envelope numbers to publish their proxy-advertisement messages. Regular Storm peers search out these envelope numbers to locate the proxy service. By keeping the two groups of envelope numbers separate, Storm does not mix up peers looking to be proxies with peers that have already become proxies.

Of course, the trouble with rules is that they can easily be broken. Overnet specifies a simple set of rules and all (nonbuggy) versions of Storm implement those rules. Researchers, however, typically do not. Let's return to our analogy for a moment. If you want to determine every participant in the line (crawl the network) you can pick a random envelope number and search for it. As you learn about new people you can add them to a master list. If you do this fast enough and for long enough your master list will be nearly complete. Many researchers are crawling the network constantly, which can put a significant load on Storm bots and other researchers.

A View from the Stormdrain

One of the benefits of crawling the Storm network for many consecutive months is that anomalies are easy to spot. Stormdrain is set up to graph in real time many attributes gathered while crawling. Sudden bumps or dips are often the result of one or more misbehaving nodes. Initially there was no perfect way to tell which peers in the network were buggy nodes versus impostors under the control of third parties.

Only the most heinous abuses of the network were clearly the doing of third parties. Attempts to classify individual peers based on their behavior were often lost in a sea of data. For example, in late 2007 Stormdrain suffered a sudden drop in efficiency coupled with a large spike in CPU usage. The code was carefully reviewed for possible bugs but none were found that could cause the symptoms. Closer examination of network traffic revealed several IPs in China flooding Stormdrain with Overnet traffic to the point of Denial of Service (DoS). When the IPs were blocked a few hours later the attack resumed from a new set of Chinese IPs. A change had to be made to Stormd-

rain's select() loop to reduce the effect of the attacks. In the line analogy, Stormdrain is just a person walking around and contending with others doing the same. We put a lot of work into developing heuristics to differentiate Storm peers from third parties but none had the 100% accuracy we really wanted.

Through some reverse-engineering work on the Storm binary, it was discovered that Storm had a badly flawed pseudo random number generator that caused the OIDs it picked randomly to follow a specific pattern [2]. Using this knowledge as an oracle allowed Stormdrain to differentiate real Storm peers from impostors in the network. Suddenly it became very easy to measure the effect each of the other researchers had in the network and even to track the various attacks as researchers would evolve and adapt new techniques. It was an epiphany. Using this oracle to separate impostors from Storm has allowed us to create a virtual "police lineup" of others participating in the Storm network that are not Storm bots. Here are a few of the different types of "people" in that lineup:

Babbling Idiots

These people constantly walk up to Stormdrain and say something—anything. Sometimes it's a search; other times it's a publicize. It's never useful, but they just won't leave Stormdrain alone. Stormdrain currently does not bother to blacklist these peers because they aren't harmful enough.

The Deaf

These people periodically ask Stormdrain questions but don't even acknowledge the answers. When Stormdrain queries them, they don't respond. NAT can cause this sort of behavior but NAT has other qualities to it that these people don't exhibit. This also wastes Stormdrain's time but the design of Stormdrain keeps these peers from causing any real harm.

The Schizophrenic

These people answer all queries with imaginary friends. Stormdrain does not know this at first but later queries always fail to contact the peers they claimed to know. This is a very common attack in the network, and Stormdrain uses heuristics to identify and blacklist peers poisoning the network in this manner.

Pathological Liars

Some peers in the network respond with exceptionally poor results. When Stormdrain queries peers like this, the search result is intentionally crafted in a way to throw Stormdrain as far from the right peers as possible. In the envelope analogy, this would be like maliciously responding to a search for number 34 by claiming that person 90 is actually person 32. These peers can reduce the search efficiency of the Storm network but don't negatively impact Stormdrain, because crawling the network does not require actually locating content, just other peers.

Junk-mailers

A person can only hold a small number of envelopes without dropping some. To prevent legitimate notes being found, junk-mailing peers constantly publish random notes to everyone holding a particular envelope number. Those searching for that envelope number will then likely receive only the random notes. In the Storm network this attack is performed against any peer with an OID that happens to be close to the 32 daily rendezvous hash locations. This is known as the Eclipse attack and has proven very successful in tests [6].

Gangs of Friends

These groups of peers all choose OIDs very close to some content hash location. When Stormdrain searches these peers for content, they always respond claiming another in their group is closer. In this manner, these peers will pass any searching node around in circles until the peer gives up, thinking that no content has been published at that hash. This is referred to as the Sybil attack and is used by some researchers to attack the 32 daily Storm proxy-advertisement hashes [6].

Tonya Harding Wannabees

Sometimes Stormdrain gets punished (via a DoS) for searching for particular content. When more than one peer is involved in this behavior the result is a DDoS. Stormdrain has had to contend with attacks like this on more than one occasion. It isn't clear whether this bat-to-the-kneecaps approach is the result of vigilante researchers, rival spam gangs, or some other players.

Researchers Collide

The trouble with having so many easily performed attacks is that invariably they are used often. Indeed, at one point or another all of these attacks have been performed in the Storm network—often several of them concurrently. To successfully crawl the network a researcher must put extensive engineering time into detecting and reducing the effectiveness of each of these attacks. This, in turn, encourages other researchers to perform more stealthy and difficult to detect attacks. There was a joke at a recent security conference [7] that eventually the Storm network would shrink to a handful of real bots and there would still be an army of rabid researchers fighting with each other to measure whatever was left!

Although this is certainly an exaggeration, it leads to the moral of the story: Engineering time and care must be taken not to inadvertently measure the activities of other researchers and network disruptors. Addressing this problem is perplexing because there is nothing fundamental that allows one to make this determination; a sufficiently sophisticated disruptor can design software that is indistinguishable from a bot at the network layer or simply infect a large number of honeypots with the bot itself and manipulate their behavior.

Thus, the question becomes, "At what point is the signal-to-noise ratio so low that there is no meaningful signal left to do research on?" We do not have an answer. One thing is certain, though: The unique architecture of Storm has given researchers around the world an unprecedented view of the future of botnets.

ACKNOWLEDGMENTS

Our thanks are owed to Vern Paxson and Christian Kreibich for detailed discussions and feedback on investigating the Storm botnet, Joe Stewart of SecureWorks for offering his insight into the workings of Storm, and Gabriel Lawrence and Jim Madden for supporting this activity on UCSD's systems and networks.

REFERENCES

[1] The Spread of the Sapphire/Slammer Worm: http://www.caida.org/publications/papers/2003/sapphire/sapphire.html.

[2] "The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff," *USENIX LEET '08*: http://www.usenix.org/events/leet08/tech/full_papers/kanich/kanich.pdf.

[3] Storm: http://www.usenix.org/publications/login/2007-12/pdfs/stover.pdf.

[4] "On the Spam Campaign Trail," *USENIX LEET '08*: http://www.usenix.org/events/leet08/tech/full_papers/kreibich/kreibich.pdf.

[5] Kademlia: A Peer-to-Peer Information System Based on the XOR Metric: http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

[6] "Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm," *USENIX LEET '08*: http://www.usenix.org/events/leet08/tech/full_papers/holz/holz.pdf.

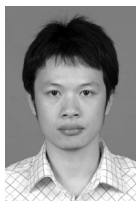
[7] First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '08): http://www.usenix.org/events/leet08/.

CHRIS GRIER, SHUO TANG, AND
SAMUEL T. KING

building a more secure Web browser

Chris Grier is a PhD student in the Electrical and Computer Engineering Department at the University of Illinois at Urbana-Champaign. He is interested in building secure software systems.

grier@uiuc.edu



Shuo Tang is a PhD student in the Computer Science Department at the University of Illinois at Urbana-Champaign. His research focuses on operating systems and system security.

stang6@uiuc.edu

Sam King is an Assistant Professor in the Computer Science Department at the University of Illinois at Urbana-Champaign. His primary research interests are in operating systems and security.

kingst@uiuc.edu

THE MODERN WEB BROWSER HAS evolved from a relatively simple client application designed to display static data into a complex networked operating system tasked with managing many facets of online experience. Support for dynamic content, multimedia data, and third-party plug-ins greatly enriches the browsing experience at the cost of increased complexity of the browser itself, resulting in a plague of security vulnerabilities that provide hackers with easy access to systems. To address the root of this problem, we designed and implemented the OP Web browser. We have partitioned the browser into smaller subsystems, isolated each subsystem, and made all communication between subsystems simple and explicit. Finally, we have used formal methods to prove the correctness of the communications between subsystems and the ability to limit the effects of compromised subsystems.

According to a recent report by Symantec [15], during the first half of 2007 Internet Explorer had 93 security vulnerabilities, Mozilla browsers had 74 vulnerabilities, Safari had 29 vulnerabilities, and Opera had 9 vulnerabilities. In addition to these browser bugs, there were also 301 reported vulnerabilities in browser plug-ins over the same period of time, including high-profile bugs in the Java virtual machine [3], the Adobe PDF reader [10], the Adobe Flash Player [2], and Apple's QuickTime [11]. Unfortunately, according to several recent reports [9, 12, 15, 16], attackers actively exploit these bugs.

The flawed design and architecture of current Web browsers make this trend of exploitation likely to continue. Modern Web browser design still has roots in the original model of browser usage in which users viewed static pages and the browser itself was the application. However, recent Web browsers have evolved into a platform for hosting Web-based applications, where each distinct page (or set of pages) represents a logically different application, such as an email client, a calendar program, an office application, a video client, or a news aggregate. The single-application model provides little isolation or security between these distinct applications hosted within the same browser

or between different applications aggregated on the same Web page. A compromise occurring within any part of the browser, including plug-ins, results in a total compromise of all Web-based applications running within the browser. This compromise may include all parts of the system that the user running the browser has access to, up to and including the operating system itself.

Efforts to provide security in this evolved model of Web browsing have had limited success. The same-origin policy, where the origin is defined as the domain, port, and protocol of a request, states that scripts and objects from one domain should only be able to access other scripts and objects from the same domain. This is the one security policy most browsers try to implement. However, modern browsers have different interpretations of the same-origin policy [6], and the implementation of this principle tends to be error-prone because of the complexity of modern browsers [4]. The same-origin policy is also too restrictive for use with browser plug-ins, and as a result browser plug-in writers have been forced to implement their own ad hoc security policies [1, 8, 14]. Plug-in security policies can contradict a browser's overall security policy and create a configuration nightmare for users, since they have to manage each plug-in's security settings independently.

Given the importance of Web browsers and the lack of security in current approaches, our goal is to design and implement a secure Web browser. More precisely, we want to prevent as many attacks as we can with reasonable cost, limit the damage that the remaining attacks can do, recover swiftly from successful attacks, and learn how to prevent them in the future.

This article describes the design and implementation of the OP Web browser, which attempts to address the shortcomings of current Web browsers to enable secure Web browsing. OP comes from Opus Palladianum, which is a technique used in mosaic construction where pieces are cut into irregular fitting shapes. In our design we break the browser into several distinct and isolated components, and we make all interactions between these components explicit. At the heart of our design is a browser kernel that manages each of our components and interposes on communications between them. This model provides a clean separation between the implementation of the browser components and the security of the browser, and it allows us to provide strong isolation guarantees and to implement novel security features.

Building the OP Browser

In our current design [5] we break the browser into several distinct and isolated components, and we make all interactions between these components explicit. At the heart of our design is a browser kernel that manages each of our components and interposes on communications between them. This model provides a clean separation between the implementation of the browser components and the security of the browser, and it allows us to provide strong isolation guarantees and to implement novel security features. This architecture stands in stark contrast to current browser designs, which place all components in a single process and contain multiple paths for making security-critical decisions [4], making it difficult to reason about security.

DESIGN PRINCIPLES

Overall we embrace both operating system design principles and formal methods techniques in our design. By drawing on the expertise from both

communities we hope to converge on a better and more secure design. Four key principles guide the design for our Web browser:

- Have simple and explicit communication between components. Clean separation between functionality and security, with explicit interfaces between components, reduces the number of paths that can be taken to carry out an action. This makes reasoning about correctness, both manually and automatically, much easier.
- Have strong isolation between distinct browser-level components and defense in depth. Providing isolation between browser-level components reduces the likelihood of unanticipated and unaudited interactions and allows us to make stronger claims about general security and the specific policies we implement.
- Design components to do the proper thing, but monitor them to ensure they adhere to the design. Delegating some of the security logic to individual components makes the browser kernel simpler while still providing enough information to verify that the components faithfully execute their design.
- Maintain compatibility with current technologies. We do our best to avoid imposing additional burdens on users or Web application developers—our goal is to make the current browsing experience more secure.

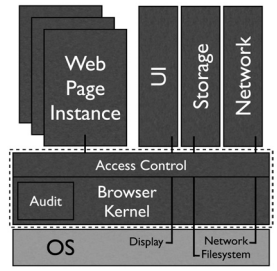
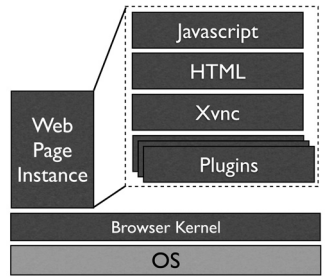


FIGURE 1: OVERALL ARCHITECTURE OF OUR OP WEB BROWSER

FIGURE 2: BREAKDOWN OF AN INDIVIDUAL WEB PAGE INSTANCE



OP BROWSER ARCHITECTURE

Figure 1 shows the overall architecture of OP. Our browser consists of five main subsystems: the Web page subsystem, a network component, a storage component, a user-interface (UI) component, and a browser kernel. Each of these subsystems runs within a separate OS-level process, and the Web page subsystem is broken into several different processes. The browser kernel manages the communication between the subsystems and between processes, and it also manages interactions with the underlying operating system.

We use a message-passing interface to support communications between all processes and subsystems. (See our recent paper [5] for a full listing of our message-passing interface.) These messages have a semantic meaning (e.g., fetch an HTML document) and are the sole means of communication

between different subsystems within our browser. They must pass through the browser kernel, and the browser kernel implements our access control mechanism, which can deny any messages that violate our access control policy.

We also use OS-level sandboxing techniques to limit the interactions of each subsystem with the underlying operating system. Each subsystem has a unique set of sandboxing rules specifically tailored to the individual component. For example, the Web page subsystem is denied access to the file system and the network, and the network subsystem is allowed to access the network, but not the file system. In our current design we use SELinux [7] to sandbox our subsystems, but other techniques would have been suitable for our purposes.

THE BROWSER KERNEL

The browser kernel is the base of our OP browser and it has three main responsibilities: manage subsystems, manage messages between subsystems, and maintain a detailed security audit log. To manage subsystems, the browser kernel is responsible for creating and deleting all processes and subsystems. The browser kernel creates most processes when the browser first launches, but it creates Web page instances on demand whenever a user visits a new Web page. Also, the browser kernel multiplexes existing Web page instances to allow the user to navigate to previous Web pages (e.g., the user presses the "back" button).

The browser kernel maintains a full audit log of all browser interactions. It records all messages between subsystems, which enables detailed forensic analysis of our browser if an attacker is able to compromise our system.

THE WEB PAGE SUBSYSTEM

When a user clicks on a link or is redirected to a new page, the browser kernel creates a new Web page instance. For each Web page instance we create a new set of processes to build the Web page. Each Web page instance consists of an HTML parsing and rendering engine, a JavaScript interpreter, plug-ins, and an X server for rendering all visual elements included within the page (Figure 2). The HTML engine represents the root HTML document for the Web page instance. The HTML engine delegates all JavaScript interpretation to the JavaScript component, which communicates back with the HTML engine to access any document object model (DOM) elements. We run each plug-in object in an OS-level process and plug-in objects also access DOM elements through the HTML engine. All visual elements are rendered in an Xvnc server, which streams the rendered content to the UI component where it is displayed.

THE USER INTERFACE, NETWORK, AND STORAGE SUBSYSTEMS

Our UI subsystem is designed to isolate content that comes from Web page instances. The UI is a Java application and implements most typical browser widgets, but it does not render any Web page content directly. Instead the Web page instance renders its own content and streams the rendered content to the UI component using the VNC protocol [13]. By using Java and having the Web page instance render its own content we enforce isolation and add an extra layer of indirection between the potentially malicious content from the network and the content being displayed on the screen. This isolation and indirection allow us to have stronger guarantees that poten-

tially malicious content will not affect the UI in unanticipated ways. The UI includes navigation buttons, an address bar, a status bar, menus, and normal window decorations.

The UI is the only component in our system that has unrestricted access to the underlying file system. Anytime the Web browser needs to store or retrieve a file, it is done through the UI to make sure the user has an opportunity to validate the action using traditional browser UI mechanisms. This decision is justified since users need the flexibility to access the file system to download or upload files, but our design reduces the likelihood of a UI subsystem compromise.

Since other components cannot access the file system or the network, we provide components to handle these actions. The storage component stores persistent data, such as cookies, in an sqlite database. Sqlite stores all data in a single file and handles many small objects efficiently, making it a good choice for our design since it is nimble and easy to sandbox. The network subsystem implements the HTTP protocol and downloads content on behalf of other components in the system.

Security in the OP Browser

We drew on the expertise of the operating systems community to make our browser architecture well suited for security. Subsystems within the browser are first-class principals, and communication between subsystems is explicit and exposed, thus providing mechanisms suitable for implementing browser-based security. Next, we explore two areas: security policies for browser extensibility and formal methods for proving invariants about our browser.

BROWSER EXTENSIBILITY

Modern Web browsers support extensibility through two main mechanisms: browser plug-ins and browser extensions. Plug-ins are a browser mechanism for hosting additional applications within a Web page, usually to render non-HTML content such as multimedia files. For example, browsers render "application/x-shockwave-flash" content using a flash-capable movie player such as Adobe Flash Player.

Extensions are a browser mechanism for extending browser functionality. Extensions interpose on and interact with browser-level events and data and provide developers with the ability to add user-interface widgets to the browser itself. Three popular extensions are the Yahoo! toolbar, which provides easy access to the Yahoo! search engine, the Greasemonkey extension, which allows users to script common tasks such as filling in form data automatically, and the NoScript extension, which provides fine-grained control over which pages can run JavaScript, thus preventing untrusted pages from running potentially malicious scripts.

These mechanisms for browser extensibility introduce unique challenges from a security perspective. Common uses of plug-ins often contradict a browser's overall security policy, so plug-ins operate outside of current browser security policies. Extensions need flexibility to integrate tightly within the browser itself and often run with full privileges. For a browser to be secure one must support these rich features securely without compromising the flexibility of commonly used plug-ins and extensions. Two current browsers that support extensions, Firefox and Internet Explorer, opt to pro-

vide flexibility at the expense of security and have no mechanisms or policies for running extensions securely.

As a first step toward our greater goal of securing browser extensibility, we have integrated plug-in security policies within the OP Web browser. In addition to supporting the ubiquitous same-origin policy, we developed two novel plug-in policies designed to provide security for the browser even if an attacker successfully exploits a plug-in vulnerability. You can learn about these policies by reading pages five and six of our paper [5].

FORMAL METHODS

Large software artifacts are typically built and maintained by groups of developers who contribute thousands of lines of code over a period of several years. This process is error-prone despite the best intentions of the developers. Moreover, for the software to be useful over an extended period of time, it must adapt to changes in user requirements. These extensions may be made by the original developers, but they are more typically made by a different group of people who may work for a different company. Ideally, formal verification that the code is correct with respect to the security requirements of the system would be an essential part of initial software development as well as the subsequent revision process. For many reasons, however, formal verification is usually not a central component in software development. We address this problem for the OP Web browser by making formal methods a fundamental part of our overall design process.

In our work to date on verifying security properties of the OP browser we use formal methods as a useful and practical tool in our overall design process. We develop an abstract model of the browser components and exhaustively search through the execution state space using a model-checking framework to look for states that violate our specified security invariants. We verified our implementation of the same-origin policy and we verified an "address-bar visual invariant" that states the URL displayed in the address bar should always be the same as the URL of the displayed page.

There is often a gap between the formal model used to verify properties and the system implementation. Although we recognize that this gap exists between our model and our system, we feel that for our uses of formal methods the difference is small enough that we are able to use the results of model checking to iterate on design and development. Since we implement each of the browser components separately and use a compact API for message passing, the model that we use to formally verify parts of our browser is very similar to the actual implementation. The model we create is focused on message-passing between components. We do not verify, for example, that the HTML parsing engine is bug-free; instead, we verify that even if the HTML parsing engine had a bug, the messages that a code execution attack could generate (potentially any message) would not force the browser as a whole into a bad state. To do this, we model each component, and aspects of every component's internal state are included. Messages are the means for the browser's internal state to change.

Our application of formal methods helped us find bugs in our initial implementation. By model-checking our address bar model we revealed a state that violated our specification of the address-bar visual invariant. The resulting state was actually due to a bug in our implementation, as we had not properly considered the impact of attackers dropping messages or a compromised component choosing not to send a particular message. Our model gives an attacker complete control over the compromised component, in-

cluding the ability to selectively send some types of messages and not others. We used the result to fix our access control implementation and we updated our model accordingly.

This preliminary work on formal verification of our browser represents a first step toward our larger goal of full formal verification of the OP Web browser.

Conclusions

In this new era of Web-based applications and software as a service, the Web browser has become the new operating system. Unfortunately, current Web browsers are unable to cope with the complexity that accompanies this new role and have fallen subject to attack. In this article we showed how, by treating Web browsers like operating systems and by building them using operating system principles, we can make a first step toward a more secure Web browser.

We plan to have a version of the OP browser ready for download by the end of the summer.

ACKNOWLEDGMENTS

We would like to thank Jose Meseguer and Ralf Sasse for their valuable feedback on our use of formal methods. We would also like to thank Joe Tucek and Anthony Cozzie for discussions about the design of our browser, and Frank Stratton, Paul Dabrowski, Adam Lee, and Marianne Winslett for feedback on an early draft of our paper. This research was funded in part by a grant from the Internet Services Research Center (ISRC) of Microsoft Research.

REFERENCES

[1] Adobe Flash Player settings manager: http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager.html.

[2] Adobe, Flash Player update available to address security vulnerabilities: http://www.adobe.com/support/security/bulletins/apsb07-12.html.

[3] AusCERT, Sun Java runtime environment vulnerability allows remote compromise: http://www.auscert.org.au/render.html?it=7664.

[4] S. Chen, D. Ross, and Y.-M. Wang, "An Analysis of Browser Domain-Isolation Bugs and a Light-weight Transparent Defense Mechanism," *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS),* 2007.

[5] C. Grier, S. Tang, and S.T. King, "Secure Web Browsing with the OP Web Browser," *Proceedings of the 2008 IEEE Symposium on Security and Privacy,* May 2008.

[6] C. Jackson, A. Bortz, D. Boneh, and J.C. Mitchell, "Protecting Browser State from Web Privacy Attacks," P*roceedings of the 15th International Conference on World Wide Web* (New York: ACM Press, 2006).

[7] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," *Proceedings of the 2001 USENIX Annual Technical Conference FREENIX Track*, June 2001.

[8] Microsoft, "ActiveX Security: Improvements and Best Practices": http://msdn2.microsoft.com/en-us/library/bb250471.aspx.

[9] A. Moshchuk, T. Bragin, S.D. Gribble, and H.M. Levy, "A Crawler-based Study of Spyware on the Web," *Proceedings of the 2006 Network and Distributed System Security Symposium (NDSS),* February 2006.

[10] P.D. Petrkov, Oday: PDF pwns Windows: http://www.gnucitizen.org/blog/0day-pdf-pwns-windows.

[11] P.D. Petrkov, Oday: QuickTime pwns Firefox: http://www.gnucitizen.org/blog/0day-quicktime-pwns-firefox.

[12] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The Ghost in the Browser Analysis of Web-based Malware," *Proceedings of the 2007 Workshop on Hot Topics in Understanding Botnets (HotBots),* April 2007.

[13] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing,* 2(1):33–38, January 1998.

[14] Sun, Java Security Architecture: http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc1.html.

[15] D. Turner, Symantec Internet Security Threat Report: Trends for January–June 07, Technical Report, Symantec Inc., 2007.

[16] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated Web Patrol with Strider Honeymonkeys: Finding Web Sites That Exploit Browser Vulnerabilities," *Proceedings of the 2006 Network and Distributed System Security Symposium (NDSS),* February 2006.

COLIN DIXON, THOMAS ANDERSON, AND ARVIND KRISHNAMURTHY

withstanding multimillion-node botnets

Colin Dixon is a graduate student at the University of Washington. While an undergraduate at the University of Maryland he worked on approximation algorithms and anonymous communication. His current research interests include computer security, network architecture, and distributed systems with a focus on deployable solutions for real-world problems.

ckd@cs.washington.edu

Tom Anderson is a Professor in the Department of Computer Science and Engineering at the University of Washington. He is an ACM Fellow and a winner of the ACM SIGOPS Mark Weiser Award, but he is perhaps best known as the author of the Nachos operating system.

tom@cs.washington.edu

Arvind Krishnamurthy is an Assistant Research Professor at the University of Washington, Seattle. His research interests are primarily at the boundary between the theory and practice of distributed systems. He has worked on automated mechanisms for managing overlay networks and distributed hash tables, network measurements, parallel computing, techniques to make low-latency RAID devices, and distributed storage systems that integrate the numerous ad hoc devices around the home.

arvind@cs.washington.edu

LARGE-SCALE DISTRIBUTED DENIAL OF service (DoS) attacks are an unfortunate everyday reality on the Internet. They are simple to execute and, with the growing size of botnets, more effective than ever. Although much progress has been made in developing techniques to address DoS attacks, no existing solution handles non-cacheable content, is unilaterally deployable, works with the Internet model of open access and dynamic routes, and copes with the large numbers of attackers typical of today's botnets. We believe we have created a practical solution.

Setting the Stage

The current Internet is often compared to the Wild West and not without merit. A combination of the lack of accountability, the complexities of multiple legal jurisdictions, and an ever-changing technological battlefield has created a situation where cyber-criminals can operate lucrative businesses with little risk of being caught or punished.

The most brazen example of this is the growth of botnets. Attackers write viruses that compromise end hosts and tie them into a command and control system that enables the attacker to issue commands, install software, and otherwise control compromised machines. These networks are the basis for a whole underground economy in stolen financial information, stolen identities, spam email, and DoS attacks.

The size of these botnets is large and growing. A variety of recent estimates put the total number of bots on the Internet well into the millions and some estimates go upward of hundreds of millions [3, 5]. Recent examples including the Storm and Kraken botnets have made headlines in mainstream media. To make matters worse, the number of critical operating system vulnerabilities discovered is increasing steadily every year [2], giving botnets an ample supply of new recruits, so the problem is unlikely to get better on its own.

DoS attacks launched from botnets number in the hundreds each day and threaten large swaths of the Internet. If compromised nodes are typical of end hosts participating in other large peer-to-peer systems [10], a multimillion-node botnet would be able to generate terabits of attack traffic per second, sourced from virtually every routable IP prefix on

the planet. This scale of attack could, at least temporarily, overwhelm any current core link or router. It is also capable of indefinitely disrupting service to all but the best provisioned services on the Internet today.

In May 2006, a sustained attack against the anti-spam company Blue Security forced the company to close down its services [12]. The LiveJournal community was even knocked offline when it was caught in the crossfire. Further, attacks are not just limited to security companies. In April 2007, a sustained attack on government and business Web sites in Estonia effectively knocked the country off the Web [9]. Even nations are not safe. More disturbing is that, despite the attacks going on for weeks, no effective countermeasures were deployed and service was restored only after the attacks petered out.

DoS Attacks

Although DoS attacks come in many flavors, we focus on resource exhaustion attacks. These attacks flood some bottleneck resource with more requests than can be handled, ensuring that only a small fraction of the legitimate requests are serviced.

In the past, syn floods and other techniques aimed to exhaust end-host resources such as memory and process table space, but as these vulnerabilities have been fixed, increasingly the trend is simply to exhaust the target's bandwidth. Attack packets can emulate normal user behavior, making them difficult to detect and drop. Attack traffic often crowds out legitimate traffic upstream from where the victim has power to filter traffic even if it could distinguish good packets from bad. Further, with the increasing size of botnets attackers can simply send normal traffic and make the attack indistinguishable from a flash crowd, forcing defenses to drop packets at random.

Without information about which requests are legitimate and with limited buffer space, the only strategy for a victim is to serve requests at random. If there are G legitimate requests and B spurious requests, on average, O([G/(B+G)]) of the available resources go to legitimate requests. But B is often much larger than G, since, with a massive botnet, attackers can pick their target and focus their fire. Addressing this asymmetry is a main goal of our work.

State of the Art

There are two sets of deployed solutions today. The first involves heuristic-based filters deployed in special-purpose boxes in the network with the goal of finding and dropping the bad traffic. The second set makes use of large-scale content distribution networks (CDNs), which aim to solve the problem by sheer over-provisioning.

Heuristic-based filtering relies on an arms race between attackers and defenders—one that we are unlikely to win. Attackers are faster on their feet and, in the end, have an easier goal. They only have to make their traffic indistinguishable from the legitimate clients, whereas the filters have to detect—in real time and at high data rates—the ever-shrinking differences between the attack traffic and legitimate traffic.

Additionally, because heuristics can cause collateral damage, they are only activated in response to an attack. This requires both detecting the attack and communicating that fact to the upstream filters. This communication itself can be interrupted by the attacker.

Large-scale content distribution networks, however, work remarkably well for read-only Web sites, by replicating data everywhere. Massive replication not only increases performance and resilience to flash crowds but also provides extra capacity to deal with a DoS attack. This approach is even available as a commercial anti-DoS service [1]. However, CDNs do not work for nonreplicable services, such as read/write back-end databases for e-commerce, modern AJAX applications, e-government, and multiplayer games, or for point-to-point communication services such as VoIP or IM—in other words, much of the Internet as we know it today. How can we ensure communication with a fixed endpoint when that endpoint is being flooded?

We address this problem with the key insight that we need a system as powerful as a botnet to defend against a botnet.

Phalanx Architecture

Intuitively, Phalanx aims to use a large swarm of nodes (like those of a large-scale CDN) as points of presence for a protected server. Provided that the nodes' resources exceed those of attackers, legitimate clients will have some functioning channels for communication despite a widespread attack. In practice, the implementation of this, which allows for a reasonable deployment path, good performance, and an open model of communication, is somewhat more complicated than simply using nodes as proxies.

There are two key problems with simply using CDN nodes as proxies. First, these nodes cannot simply forward all traffic onto the server; instead, they have to do some kind of filtering at the behest of the server. Second, it is only in aggregate that the CDN nodes are resistant to attack, so any given connection must leverage a large set of these nodes to be resilient.

To solve the first problem we use the nodes as packet mailboxes rather than simple forwarding proxies. A mailbox in Phalanx is a best-effort packet store and pick-up point. The protected server must explicitly request each packet it wishes to receive and therefore is fail-safe: If a server doesn't request a packet, the packet is not delivered. These mailboxes are further explained below.

To solve the second problem, we send each packet through a different randomly chosen mailbox, thus drawing in a large number of mailboxes to protect each connection. If any given mailbox fails or is attacked, only a small fraction (often only one packet) of the connection will be lost. Because the mailbox used by a given packet is chosen randomly according to a seed known only to the two endpoints, the attacker must attack widely to have any impact. The exact mechanisms for this can be found below, in the section "Swarms and Iterated Hash Sequences."

The observant reader will note two remaining problems. First, there is nothing to stop an attacker from ignoring the mailboxes and attacking the server directly. To deal with this, we capitalize upon the request-response framework and install filters at the edge of the server's upstream ISP. These filters (whose functionality is described later as the filtering ring) simply drop all unrequested packets.

Second, now that we have blocked all unrequested packets, how can we initiate connections? For this, we extend the request-response framework and additionally send requests for new connections rather than explicit packets. These requests are a valuable scarce resource and so we protect access to them via authentication and fair queuing (as laid out under "Connection Establishment").

AN EXAMPLE

Before launching into each mechanism in detail we will narrate an example of how Phalanx would be used to protect a standard Web server with dynamic content, such as an e-commerce site. The example can also be followed in Figure 1, where the numbered steps will be mentioned.

First, the client looks up the address of the server and subsequently requests the static, cacheable content of the page via any current CDN-style system with high availability, such as Akamai, CoDeeN, or Coral (1). As part of fetching this content, the client receives a static and cacheable Java applet, which then serves as a zero-installation client to allow for interaction with Phalanx mailboxes. At this point, the Java applet is responsible for rendering the dynamic, noncacheable portions of the page and speaking the Phalanx protocols.

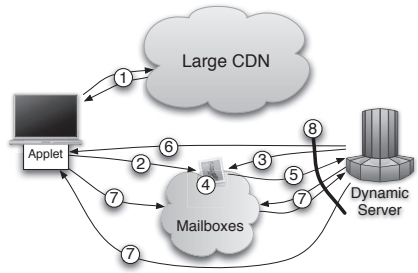


FIGURE 1: A DIAGRAM ILLUSTRATING A SIMPLE HTTP-STYLE REQUEST DONE WITH PHALANX. THE NUMBERS CORRESPOND TO THE ACCOMPANYING DESCRIPTION.

The applet begins by making a name request for the dynamic content server to the distributed name service (1). Again, because the naming information is static and cacheable, this service can be provided by any highly available name service, such as CoDoNs or Akamai's DNS service. The name service returns a list of "first-contact" mailboxes. These first-contact mailboxes hold the first packet requests that the server has issued to allow new connections to be made.

The applet requests a challenge from one of these first-contact mailboxes and replies with either a puzzle solution or an authentication token (2). In either case, the applet will resend the request, possibly with a more complex puzzle solution and/or a different mailbox if the connection is not established in a reasonable period of time.

At the mailbox, a steady stream of first packet requests has been arriving from the dynamic content server (3). One of these first packet requests is eventually assigned to the client's connection request (4), at which point the applet's request is forwarded to the server (5) to cross back through the filtering rings (8) without being dropped. This ensures that the rate of connection requests reaching the server is under the server's control.

Eventually, a response will come back from the server (6) containing a list of mailboxes to use for the remainder of the connection along with a shared secret allowing standard Phalanx communication to commence. At the same time, the server will send packet fetch requests to the first several mailboxes to be used in preparation for receiving further packets from the client.

The client uses the shared secret to determine the sequence of mailboxes to use and begins to send packets to these mailboxes. These data packets are paired with their corresponding requests and forwarded onto the server passing through the filtering ring (8) by virtue of the holes opened by the requests. This constitutes the normal behavior of the Phalanx connection (7).

If at any point in time the server decides that the connection is no longer desirable or it simply starts running low on resources, it can either decrease the rate at which it requests new data packets or simply stop requesting packets altogether.

MAILBOXES

We now proceed to describe each of these components in a bit more detail. The basic architecture of an established Phalanx connection is shown in Figure 2. A more complete description is available on the USENIX Web site in the proceedings of NSDI '08 [8].

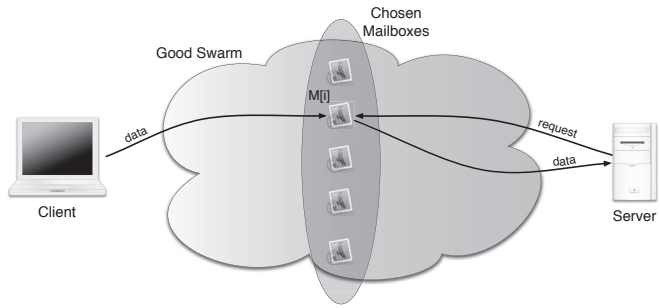


FIGURE 2: THE BASIC ARCHITECTURE OF AN ESTABLISHED PHALANX CONNECTION. EACH PACKET IS SENT THROUGH A ONE-HOP DETOUR VIA A RANDOMLY CHOSEN MAILBOX.

The basic building block in Phalanx is the packet mailbox. Mailboxes provide a simple abstraction that gives control to the destination instead of the source. Rather than packets being delivered directly to the specified destination as in previous anti-DoS overlays [4, 11, 17], traffic is first delivered to a mailbox, where it can either be "picked up" or ignored by the destination. Traffic that is ignored is eventually dropped from the buffers at packet mailboxes.

Mailboxes export two basic operations: `put` and `get`. A `put` inserts a packet into the mailbox's buffer, possibly evicting an old entry, and returns. A `get` installs a best-effort interrupt at the mailbox. If a matching packet is found before the request is bumped from the buffer, the packet is returned.

The mailbox abstraction puts the destination in complete control of which packets it receives. Flow policies can remain at the destination where the most information is available to make such decisions. These policies are implemented in the network via requests and the lack thereof. If no requests are sent, then no packets will come through. This behavior ensures that most errors are recoverable locally, rather than requiring cooperation and communication with the network control plane. This is in contrast to accidentally installing an overly permissive filter in the network and then being unable to correct the problem because the control channel can now be flooded.

SWARMS AND ITERATED HASH SEQUENCES

Individual flows are multiplexed over many mailboxes. Each packet in a flow is sent to a cryptographically random mailbox. Since each mailbox is secretly selected by the endpoints, an attacker cannot "follow" a flow by attacking each mailbox just before it is used.

We construct a pseudo-random sequence of mailboxes during connection setup by exchanging the set of mailboxes M and a shared secret x. The se-

quence of mailboxes is built by iterating a cryptographic hash function, such as SHA-1 or MD5, on the shared secret. Equipped with this shared sequence, both endpoints know in advance the precise mailbox to use for each packet in the connection.

To construct a sequence of mailboxes, we first define a sequence of nonces x_i based on the shared secret x and the cryptographic hash function h, as follows:

$$x_0 = h(x||x)$$

$$x_i = h(x_{i-1}||x)$$

Including x in every iteration prevents an attacker who sniffs one nonce from being able to calculate all future nonces by iterating the hash function themselves. Our current implementation uses MD5 [15] as the implementation of h and thus uses 16-byte nonces for simplicity. This sequence of nonces then determines a corresponding sequence of mailboxes $M[x_i]$ by modulo reducing the nonces, as follows:

$$M[x_i] = M_{x_i \bmod |M|}$$

Note that M need not be all mailboxes in the Phalanx deployment, as each flow can use a subset of the mailboxes. Indeed, a different set of mailboxes can be used for each half of the flow (client-to-server and server-to-client); both sets can be dynamically renegotiated within a flow.

Each nonce serves as a unique identifier for a packet and is included in the header to facilitate pairing each incoming packet with its corresponding request. Thus the receiver can know precisely which source sent which packet. Further, including a nonce in each packet simplifies the logic needed to drop unrequested packets. Lastly, nonces provide a limited form of authentication to requests; to subvert the system the attacker must snoop the nonce off the wire and then deliver a replacement packet to a mailbox before the correct packet arrives.

FILTERING RING

With Phalanx, a protected destination only receives those packets it explicitly requests from a mailbox. To enforce this, we drop all other packets for the destination at the edge of its upstream ISP.

Each request packet carries a unique nonce that allows a single data packet to return. In the simple case of symmetric routes, the border router records the nonce on the outgoing packet and matches each incoming packet to a recently stored nonce, discarding the packet if there is no match. Each nonce is single use, so once an incoming packet has matched a nonce, we remove that nonce.

To be effective, the filtering ring must be comprehensive enough to examine every packet destined for a protected destination, regardless of the source of the traffic. To prevent this an attacker might try to flood the border router (or, more precisely, the link immediately upstream from the border router). As we observed earlier, a massive botnet may be able to flood any single link or router in the network. However, this would disconnect only those mailboxes that used that specific router to access the destination; other mailboxes would continue to deliver packets unaffected.

Even a multimillion-node botnet would be unable to sustain enough traffic to completely disconnect a tier-1 ISP from the Internet. To have an effective defense against such a large-scale attack, a destination must either be a direct customer of a tier-1 that provides a filtering ring or be protected in-

directly, as a customer of an ISP that is a customer of that tier-1. Since each connection can spread its packets across a diverse set of mailboxes, connections might experience a higher packet loss rate during an attack, but otherwise would continue to make progress.

Deploying the filtering ring at a tier-1 has risks, though. Bots are everywhere—even inside corporate networks—and, as a result, it seems likely that filtering rings would be deployed in depth. Inner layers would provide protection against the limited number of potential attackers close to a server, while outer layers would provide the powerful filtering to deal with the brunt of larger attacks. Initially, small-scale ISPs close to the destination could offer a limited DoS protection service, capable of withstanding moderate-sized botnets. Moving outward, the cost of deploying the filtering ring would increase (as more border routers would need to be upgraded), but the value would also increase as the system would be able to withstand larger-scale botnets.

Our implementation of the filtering logic uses two lists of nonces, efficiently encoded using Bloom filters [7]. A whitelist contains a list of requested nonces, whereas a blacklist contains a list of nonces that have already entered the filtering ring. The whitelist ensures that only requested packets get through, and the blacklist ensures that at most one packet gets through per request. As request packets leave the ring, the router adds their nonces to the local whitelist. When data packets enter the ring, their nonces are verified by checking the whitelist and then are added to a blacklist. Bloom filters must be periodically flushed to work properly; to minimize the impact of these flushes, two copies of each list are maintained and they are alternately flushed.

We believe that the Phalanx filtering ring is efficient enough to be implemented even for high-speed links inside the core of the Internet, provided there is an incentive for ISPs to deploy the hardware, that is, provided that ISPs can charge their customers for DoS protection. (Note that ISPs that provide transit need to modify only their ingress routers and not all routers.) A 100-gigabit router line card would need about 50 MB of hash table space. For each delivered packet, six Bloom filter operations are required: The request packet places a nonce in the current copy of the whitelist, then when the actual packet is received it is checked against all four tables (the current and previous whitelist and the current and previous blacklist) and then added to the current blacklist. Both the storage and computation demands are small relative to those needed for core Internet routing tables and packet buffering.

Although the filtering ring will require either deploying new hardware in the network or upgrading the software running on existing routers, it does not require pervasive deployment. Upgrades need only be made at the border of ISPs looking to offer DoS protection. At first, filtering could be done by pairing a commodity server with each border router in an ISP and later moving the functionality into the routers if higher performance was needed and when appropriate software updates have been released.

Our discussion to this point has assumed routing symmetry. Of course, the real Internet has a substantial amount of routing asymmetry. A request packet sent to a mailbox may not leave the filtering ring at the same point as the corresponding data packet returns; if so, the Bloom filter at the return point will drop the packet. This problem becomes more likely as the nesting level increases.

To address this problem, we modify filtering ring nodes to stamp request packets as they pass through and allow mailboxes to loosely source route

data packets via IP-in-IP tunneling back through the filter ring nodes that are known to have been primed. This solves the problem of route asymmetry while only requiring cooperation from the nodes that are already being changed to do filtering.

CONNECTION ESTABLISHMENT

Thus far, we have described how to protect established connections but have yet to properly describe the details of connection establishment.

We allow for connection establishment by issuing periodic requests that ask for connection establishment packets rather than specific data packets. These general-purpose nonces are described above. Simply allowing for such first packets doesn't solve the problem, as they immediately become a scarce resource and this capability acquisition channel can be attacked [6]. To solve this problem, we require clients to meet some burden before giving them access to a general-purpose nonce. Clients can either present an authentication token signed by the server or present a cryptographic puzzle solution.

PASSING THROUGH THE FILTERING RING

Rather than invent new mechanisms to deal with allowing first packets through the filtering ring, we reuse the existing request packet framework to punch nonspecific holes in the filtering ring. Destinations send each mailbox a certain rate of general-purpose requests. Each request contains a nonce to be placed in such first packets. When a mailbox wishes to send a first packet, it places one of these general-purpose nonces into the packet, allowing it to pass through the filtering ring.

These general-purpose requests implement a form of admission control. Each general-purpose nonce announces the destination's willingness to admit another flow. This further increases the destination's control over the traffic it receives, allowing it to directly control the rate of new connections.

For the general-purpose nonce mechanism to be resilient to DoS attack, it is necessary to spread the nonces across a wide set of well-provisioned mailboxes; a particular client only needs to access one. Refreshing these general-purpose nonces can pose an unreasonable overhead for destinations that receive few connection requests; as a result, our prototype supports nonces issued for aggregates of IP addresses. Thus, an ISP can manage general-purpose nonces on behalf of an aggregate of users, at some loss in control over the rate of new connections being made to each address. Of course, the ISP must carefully assign aggregates based on their capacity to handle new connection requests; for example, Google should not be placed in the same aggregate as a small Web site, or else the attacker could use general-purpose nonces to flood the small site.

When a client wishes to contact some server, it first contacts a mailbox and asks that mailbox to insert a general-purpose nonce into its first packet and forward it to the destination. Because general-purpose nonces are a scarce resource, the mailbox needs rules governing which connections to give these nonces and in what order. The next two sections deal with those mechanisms.

AUTHENTICATION TOKENS

Each packet requesting to initiate a connection must either carry an authentication token or a solution to a cryptographic puzzle. These provide the

burden of proof necessary for a mailbox to allow access to general-purpose nonces. Authentication tokens provide support for pre-authenticated connections, allowing them to begin with no delay. For example, a popular e-commerce site such as Amazon might provide a cookie to allow quicker access to its Web site to its registered users or even just to users who had spent more than $1000. Cryptographic puzzles provide resource proofs to approximate fair queueing of requests, when no prior relationship exists between source and destination.

Authentication tokens are simply tokens signed by the server stating that the given client is allowed to contact that server. An additional message exchange is required to prove that the client is in fact the valid token holder.

CRYPTO-PUZZLES

The crypto-puzzle is designed to be a resource proof allowing hosts that spend more time solving the puzzle to get higher priority for the limited number of general-purpose nonces each mailbox possesses. Although there are many kinds of resource proofs, we opt for a computational resource proof rather than a bandwidth resource proof [18] because computation tends to be much cheaper and less disruptive when heavily used.

We borrow a solution from prior work [14, 16] where the crypto-puzzle is to find a partial second pre-image of a given random challenge string such that, when hashed, both strings match in the lower b bits. The goal for each client is then to find some string a given a challenge nonce N such that:

$$h(a||N) \circ h(N) \bmod 2^b$$

The random nonce is included in both strings to prevent attackers from building up tables of strings that cover all 2^b possible values of the lower b bits in advance. In effect, they need to cover $2^{b + |N|}$ possible values to find matches for all values of the lower b bits and for all possible nonces, whereas solving the puzzle online only requires searching 2^{b-1} strings on average. Because the length of the nonces is under the control of the mailboxes, it is possible to make the precomputing attack arbitrarily harder than waiting and solving puzzles online.

First packets are granted general-purpose nonces, with priority given first to those with valid authentication tokens and then in decreasing order of matching bits in the crypto-puzzle solution. This allows any source to get a first packet through against an attacker using only finite resources per first packet, albeit at an increase in latency.

Evaluation

Evaluating systems such as Phalanx at scale has always posed a problem because they are fundamentally intended to operate at scales well beyond what can be evaluated on a testbed. To address this issue, we built a simulator that captures the large-scale dynamics of Phalanx and allows us to simulate millions of hosts simultaneously.

The simulator uses a router-level topology gathered by having iPlane [13] probe a list of approximately 7200 known Akamai nodes from PlanetLab nodes. These Akamai nodes serve as stand-ins for appropriately located mailboxes. Each PlanetLab node serves as a stand-in for a server that is under attack.

We assume that attackers target the mailboxes, the server, and the links near the server. Traffic is assumed to flow from clients to mailboxes unmolested. We assign link capacities by assuming mailbox access links are 10 Mbps, the server access link is 200 Mbps, and link capacity increases to the next category of {10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 40 Gbps} as the links move from the edge to the core.

We assign attackers with attack rates according to end-host upload capacity information gathered in our previous work [10, 13] and assume that good clients communicate at a fixed rate of 160 kbps.

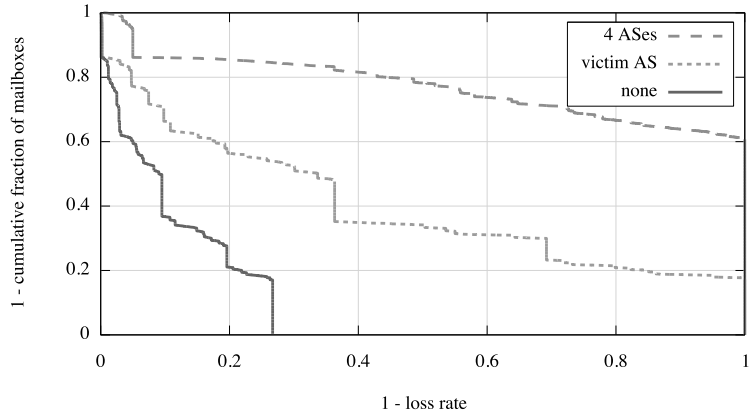


FIGURE 3: THE CUMULATIVE FRACTION OF MAILBOXES SEEING AT MOST A GIVEN FRACTION OF GOODPUT WHEN COMMUNICATING WITH THE SERVER

By using IP to AS mappings, we are able to simulate the behavior of the system under varying levels of deployment of the Phalanx filtering rings. Figure 3 shows the effect of increasing deployment of filtering rings for a server located at planetlab-01.kyushu.jgn2.jp. (The results are similar when we use other PlanetLab nodes as servers.) In this simulation, there are 100,000 attacking nodes and 1000 good clients all trying to reach the victim server. We simulate varying degrees of deployment by iteratively adding the largest adjacent AS to the current area of deployment.

As one might expect, even a little deployment helps quite a bit. Only deploying filters at the victim AS provides significant relief and allows some mailboxes to see lossless communication. Deploying in just four ASes (including the tier-1 AS NTT) results in the vast majority of mailboxes seeing lossless communication, effectively stopping the attack in its tracks if we assume that connections use any degree of redundancy to handle losses.

We next look at the scalability of Phalanx in handling attacks involving millions of bots. For this experiment we consider a somewhat stronger deployment: upgrading the mailboxes to 100 Mbps access links. Figure 4 examines the effect on mailbox loss rate as we increase the number of attackers. Most connections easily withstand the brunt of an attack involving one million nodes, and Phalanx still allows some (though severely degraded) communication through when facing 4 million nodes.

However, as the graph shows, increasing the capacity of the mailboxes by a factor of 5 to 500 Mbps is able to once again bring the attack into check. Thus, while any given deployment will have a breaking point, an increased deployment can bring increased protection to deal with even larger attacks.

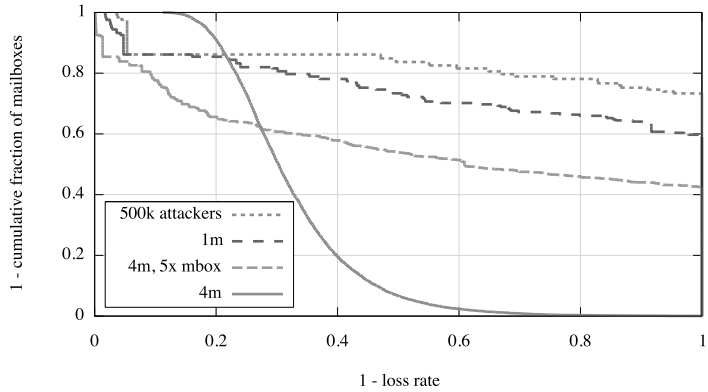


FIGURE 4: THE CUMULATIVE FRACTION OF MAILBOXES SEEING AT MOST A GIVEN LOSS RATE FOR A VARYING NUMBER OF ATTACKERS

Conclusion

In this article, we presented Phalanx, a system for addressing the emerging denial-of-service threat posed by multimillion-node botnets. Phalanx asks only for two primitives from the network. The first is a network of overlay nodes, each implementing a simple, but carefully engineered, packet forwarding mechanism; this network must be as massive as the botnet that it is defending against. Second, we require a filtering ring at the border routers of the destination's upstream tier-1 ISP; this filtering ring is designed to be simple enough to operate at the very high data rates typical of tier-1 border routers. We have implemented an initial prototype of Phalanx on PlanetLab and have used it to demonstrate its performance. We have further demonstrated Phalanx's ability to scale to million-node botnets through simulation.

ACKNOWLEDGMENTS

We would like to thank Arun Venkataramani for a set of conversations which helped us realize the need for more scalable DoS protection. We would also like to thank our NSDI shepherd, Sylvia Ratnasamy, as well as our anonymous reviewers, for help and valuable comments. This work was supported in part by National Science Foundation Grant No. CNS-0430304.

REFERENCES

[1] Akamai: http://www.akamai.com/.

[2] Microsoft's unabated patch flow: http://www.avertlabs.com/research/blog/index.php/category/security-bulletins/ (May 9, 2007).

[3] "Surge" in Hijacked PC Networks: http://news.bbc.co.uk/2/hi/technology/6465833.stm (March 2007).

[4] D.G. Andersen, "Mayday: Distributed Filtering for Internet Services." In *USITS*, 2003: http://www.usenix.org/events/usits03/tech/andersen.html.

[5] N. Anderson and Vint Cerf: One Quarter of All Computers Part of a Botnet: http://arstechnica.com/news.ars/post/20070125-8707.html (January 25, 2007).

[6] K. Argyraki and D. Cheriton, "Network Capabilities: The Good, the Bad and the Ugly." In *HotNets IV*, 2005.

[7] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, 13(7): 422–426 (1970).

[8] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: Withstanding Multi-million Node Botnets." In *NSDI*, 2008: http://www.usenix.org/events/nsdi08/tech/dixon.html.

[9] P. Finn, "Cyber Assaults on Estonia Typify a New Battle Tactic," *Washington Post*, May 19, 2007: http://www.washingtonpost.com/wp-dyn/content/article/2007/05/18/AR2007051802122.html.

[10] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Leveraging Bittorrent for End Host Measurements." In *PAM*, 2007.

[11] A.D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services." In *SIGCOMM*, 2002.

[12] B. Krebs, "Blue Security Kicked While It's Down," *Washington Post*, May, 2006: http://blog.washingtonpost.com/securityfix/2006/05/blue_security_surrenders_but_s.html.

[13] H.V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An Information Plane for Distributed Services." In *OSDI*, 2006: http://www.usenix.org/events/osdi06/tech/madhyastha.html.

[14] B. Parno, D. Wendlant, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks." In *SIGCOMM*, 2007.

[15] R. Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321 (Informational), April 1992.

[16] E. Shi, I. Stoica, D. Andersen, and A. Perrig, OverDoSe: A Generic DDoS Protection Service Using an Overlay Network, Technical report, Carnegie Mellon University, 2006: http://www.cs.cmu.edu/~dga/papers/CMU-CS-06-114.pdf.

[17] A. Stavrou and A.D. Keromytis, "Countering DoS Attacks with Stateless Multipath Overlays." In *CCS*, 2005.

[18] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense." In *SIGCOMM*, 2006.

DIANA SMETTERS, BRINDA DALAL, LES NELSON, NATHANIEL GOOD, AND AME ELLIOTT

ad hoc guesting: when exceptions are the rule

PEOPLE INCREASINGLY RELY ON THEIR ability to access and share data in order to get their jobs done and to enrich their personal lives, yet corporate security policies around sharing are rarely effective in enabling their users to achieve their goals. We offer our observations on how policy and practice often clash, as well as suggestions for improving the security of file sharing.

Diana K. Smetters is a senior security researcher at the Palo Alto Research Center. Her research interests include usability of security and applications of cryptography, particularly in social, mobile, and ubiquitous computing.

smetters@parc.com

Brinda Dalal is an anthropologist in the Ubiquitous Computing Area at the Palo Alto Research Center, focusing on sociotechnical design, usable security, and green technology innovation. Brinda received a PhD in Social Anthropology from the University of Cambridge, where she conducted research on nomadic identities and trade and barter in the Himalayas.

Brinda.Dalal@parc.com

Les Nelson is a senior research scientist at the Palo Alto Research Center. His research interests include social computing, mobile computing, ubiquitous computing, and in general the adaptation of sociotechnical systems in response to changing circumstances.

lesnelson@acm.org

Nathan Good is a research scientist at Palo Alto Research Center. His research interests include recommender systems, usable security, and mobile computing.

nathaniel.good@parc.com

Ame Elliott is a Senior Human Factors Researcher at IDEO, where she designs innovative technical solutions grounded in human need. She has a PhD in Architecture from the University of California, Berkeley, where she studied human-computer interaction. Prior to joining IDEO, she was a research scientist at PARC and Ricoh Innovations.

aelliott@ideo.com

We wanted to understand how users are sharing information and how their needs are or are not met by current tools, policies, and practices. We performed an ethnographic field study, interviewing a selected group of subjects about their practices around access control, security, and file sharing. Our intent was to understand three things: (1) Under what circumstances do people or companies share or restrict access to files? (2) What tools or behavioral practices do they use to accomplish that? (3) How are people's experiences, problems, and needs changing in regard to secure file sharing and access control, especially as they deal with geographically dispersed colleagues, clients, friends, and family members?

Background

Our research builds upon a growing body of literature on file sharing and access control. Previous studies have focused on personal file sharing, specifically, in the domains of photographs [1, 2] or music [3, 4], or professional collaborations in corporations [5, 6]. Overall, these studies concluded that current tools for managing sharing policies available in each domain were inadequate to meet their users' complex requirements [1, 2, 5, 6, 7].

In the corporate setting, email was routinely chosen as the preferred means of sharing files, even in the presence of other alternatives [5, 6]. However, both the studies of Voida et al. [5] and Whalen et al. [6] considered subjects selected from and predominantly sharing within single organizations, all of whom shared access to established file-sharing mechanisms (e.g., file servers). They did not consider the effect that these preexisting options had on the challenges users would face, or the choices they would make when sharing across organizational boundaries or operating in the absence of pre-existing shared infrastructure.

Our study focuses on the question of how users share content in the absence of existing shared infrastructure—for example, when sharing across

organizational boundaries. Based on interviews of users across various domains, we were able to explore access control and sharing issues across different types of organizations, such as those with stricter or more lax regulations and compliance policies. We examined in some depth how file-sharing and access controls were used, not used, or circumvented in order to get work done. From this analysis we identified key challenges faced by those using and choosing among current file-sharing technologies, including email.

Our Study

We conducted two-hour, in-depth interviews with ten subjects in places where they did at least some of their work—homes, home offices, or cafes. These interviews consisted of semi-structured and open-ended questions about file sharing and access control. We selected participants with file-sharing and access-control challenges, such as having to work with multiple clients from different organizations or having to share data with geographically dispersed teams or with those who need access to confidential data. Altogether we interviewed six men and four women between 23 and 53 years of age, working in a variety of fields. All used multiple digital devices and traveled frequently. Subjects were asked to describe examples of their professional and personal practices around security, privacy, and file sharing.

Key Findings

From our data, we identified a number of key problems users face in sharing data:

Sharing with myself: Users are their own most common sharing partner, effortfully moving data among their own machines, accounts, and devices to ensure continued access.

Oversharing: Users grant more access than necessary when it is difficult to limit who has access to content or how much to share with others, or when pressed for time to extract information from larger data sets.

Transient data: Users often need to hold data only briefly while transporting it from one place or another, and that data may linger, be lost, and get forgotten.

Transient access: Users need to access data for only short periods of time—they intend only one-time access or to make data available in certain situations.

Impedance matching: Users spend considerable time and effort tailoring content for sharing based on their understanding of recipient needs or the demands of the sharing mechanisms in use.

Ad hoc sharing: Users often share content with groups of recipients they have not shared with before and may not again.

Based on these insights, we propose that the general nature of the problem faced by users is what we term *ad hoc guesting*: Users need to share data securely with unplanned sets of people with whom they have not previously shared. They may belong to another organization and thus cannot be "named" by traditional access control. These interactions are transitory and lightweight, often making it not worth the effort required to set up new sharing mechanisms or change administrative state.

In what follows, we quote directly from our respondents. Explanatory material (words garbled in transcription or context missing from a quote) is provided in square brackets ([]).

General Properties of Sharing

Our study highlights distinctions between personal and professional sharing. Professionally, 80% of respondents shared files with overseas collaborators or clients in Europe and the Asia-Pacific region, and 100% exchanged data with colleagues across the United States. When working from home, consultants and employees in larger corporations often shared files via distributed corporate servers, and, in three cases, on protected FTP sites. Predominantly, the data shared in professional settings revolved around project work: Shared documents included technical specifications, meeting minutes and action items, and proposals. One of the primary affordances of using a shared server within a company was the ability to reuse documents from one project to another. At the same time, people found it time-consuming to browse different versions of documents to find the proposal they wished to reuse and resorted instead to telephoning or emailing their colleagues to obtain the appropriate copy. On the whole, we found that individuals are deeply aware of and attempt to comply with security stipulations and privacy requirements for their clients and companies. However, compliance breaks down the instant that people perceive that they are unable to follow policy without compromising their accountability to clients or colleagues or their ability to complete a task.

In contrast, people's personal file sharing practices focused on ways experiences could be shared with others. The content being shared in this case—primarily multi-media—was relational in nature, such as sharing photographs of events with family members who live overseas. We also found that a surprising number of people shared the same personal account. For instance, relatives scattered across the United States used a photo sharing account that had a single login and password to ensure privacy. Another set of parents set up a "family email account" and used email messages within the same account to discuss homework with their children in the evening.

All respondents used email to share files. Fully 90% of subjects mentioned that they had multiple email accounts (largely personal accounts) and 80% said that they used personal email accounts for business.

A total of 80% of respondents, regardless of their demographics, also used a wide variety of social software, including wikis, blogs, social networking sites (including MySpace and Facebook), hosted services (such as Yahoo! Briefcase), public Web sites for sharing images and multimedia files (including Flickr and YouTube), and online forums and games.

People Are Their Own Best Friends

People are their own most common sharing partners. File sharing with oneself allows one to synchronize activities regardless of location (at work, while traveling, or at home) or what devices or network resources are accessible. For example, interviewees who did not have a printer at home often uploaded files to Yahoo! Briefcase, then downloaded and printed files out at their office. Eighty percent of the respondents used USB drives (rather than laptops) to download content at client sites, especially when policies required that they contact IT administrators before accessing electronic files.

Email is a convenient and preferred mechanism for sharing files with oneself. Often quicker and easier than accessing files via a VPN, it is often used to bridge home and work. Although this served a short-term need, people said they later ran into trouble trying to track source documents and different versions across their accounts: "I'll go home and look for a file and have to go through all the emails with no subjects and [ask], 'Oh, when did I email myself this file?'" Most respondents had multiple email accounts (some up to 12 or 15). Different types of content were filtered into different accounts—work, friends, dating services, rental businesses, family photographs, or spam. Professional and personal accounts bled into one another, opening avenues for significant security lapses. When email or corporate servers were inaccessible, people readily sent files to consultants using their personal email accounts.

Oversharing

Our subjects often found themselves forced to overshare—to share too much or to share inappropriate information with others or themselves. Oversharing often occurs when it is simply too difficult to share only the information needed. For example, a healthcare consultant noted that when she visits a client site, she lacks sufficient time to go through the database (which she is not permitted to access remotely) and extract only the records she needs. Instead she ends up downloading entire files, including social security numbers, onto USB drives. She remarked, "There are a lot of rules trying to get permission from state agencies [to access confidential data]. A lot of data really is protected, so a lot of times the only effective way for me to do the work really disturbs me. Like I can't get permissions, but I can dump huge amounts of data on flash drives that I can then [in theory] lose."

Our subjects also reported that although the initial decision of whether or not to share data was often well-considered and even heavily regulated, once that decision was made "everything relaxes"—in other words, sharing decisions are often all or nothing. For example, one respondent noted, "They say there's no way we can provide this information, HIPPA won't allow it—it's research, it's clearly protected, but then you get past that point and you have a data sharing agreement and they'll dump a bunch of stuff [people's social security numbers, date of birth] on a disk and mail it to a name they've never heard at an address on 29th Street, which strikes me as weird."

Managing Transient Data

Users frequently handle "transient data"—data useful for a single task or short period of time. Transient data is often copies created as a side effect of transporting data from one location to another (e.g., copies on USB flash drives or in emails to oneself or others). For example, one individual remarked that she had a shoebox full of USB drives. Other respondents reported having anywhere between 2 and 15 active or inactive flash drives stored in their cars or briefcases, at work, or at home. The "throwaway nature" of temporary storage and devices makes it difficult to remember where sensitive data has gone. Not only is it hard to find the most recent version when it is needed, but afterward it often languishes, unremembered, on such devices forever.

Users dealt with such "throwaway data" at different levels of granularity, up to and including entire accounts or identities. Respondents increasingly lacked the time to manage their many email accounts, and they tended to

shed entire accounts and open new ones rather than sort, archive, or destroy private data.

Transient Access

A number of individuals noted a need for transient access to data. Consultants, for example, were only supposed to have access to client data during the period of their contracts or while working in a certain environment. People also wanted to grant temporary access in the personal domain—for example, a landlord wanted to make rental property photos accessible only when the property was available, and some users shared their passwords for photo-sharing sites with others to whom they really only wanted to grant one-time access to pictures.

Users often made data available when temporary access was needed, even at the cost of security. Most commonly, respondents made data temporarily available via personal email accounts, when unable to access their work email because of VPN difficulties. One financial analyst noted, "You're not supposed to use 'unprotected email addresses,' like Yahoo!, Gmail, or whatever, but it's just a fact of life. Even our CEO has a Gmail account. There are work requirements but there is also an undeniable fact that people will be attracted to whatever is out in the market." A geotechnical engineer commented, "There are policies against sending clients electronic documents of any sort. But it's just so backassward that no one can possibly adhere to it."

Unfortunately, it can be difficult to go back and "fix" unwanted lingering access, as with one respondent: "But that pretty much is just a few phone calls, desperate phone calls saying, 'Delete from your servers; delete from your company; make sure it's completely clean.' You're at the mercy of hoping they follow your request."

Impedance Matching

Possibly the most interesting and significant challenge faced by our subjects was impedance matching—they were expending tremendous effort figuring out how, rather than what, to share.

People have varying degrees of technical skills required to use systems; consequently, there is a disparity in their ability to master the details involved in moving data around. Often those with greater need were forced to shoulder the work to obtain or share files. As one subject reported about a newly installed Web-based repository, "I think we have folks with very limited technical comfort. So for that reason I always have to upload my files [to the repository] and then email them around, so it's sort of another step rather than saving a step."

A major concern among respondents was preventing data sharing failures. The majority of our subjects spent time anticipating their own and their recipients' current state—the speed of their network connection, the sharing mechanisms available to all peers, and familiarity of the individuals involved with those systems—and changed their actions according to the (assumed) result. Sometimes this anticipatory work had to do with what the recipient was explicitly allowed to access: "I need to know other people's permissions, and I need to know because there's usually a lag between the permissions and the actual access; even if I have permission, it might take me six weeks to get my approval for the system, so I need to know where people are on the sort of really ridiculous timeline."

More often, users were concerned about working around constraints in bandwidth, network availability, or storage. Fully half of our interviewees expressed frustration in sending or receiving large files. Some specifically mentioned that their personal accounts or corporate email could not handle files over 10 MB. A design consultant who provides audio-visual material to his clients was exasperated by the effort it took to reformat content for clients: "It's absolutely absurd in this networked economy that we can't share [large] files without going to some extreme effort." People spent considerable time reformatting data for others, based on two parameters. First, they anticipated the constraints of their own or a recipient's system (such as capacity or bandwidth); second, they anticipated the recipient's sociotechnical knowledge regarding his or her ability to receive data. A software engineer explained the reasons he compressed photographs for his relatives: "A lot of my relatives are not very techie, so I'll just put photographs in an email attachment. I try to compress them so they are small jpeg sizes and then all people have to do is just click [on the images]." Another respondent drew a similar distinction: "When you're trying to share with family or friends the speed of the network really decides whether you can share five photos or just one."

The need for impedance matching means users are forced to decide between sharing modalities based on whether the sharing mechanisms will work with a particular user (do they have X or are they on Y?) or piece of content (is the file too big?) or what sharing mechanisms work best with that user (can they be counted on to log onto a separate system?). Equally importantly, how well can you gauge the accuracy of your assumptions about another's state (can they even receive your files?)? It is clear that the onus of work currently resides on users rather than on the systems they use.

The result of this impedance matching work is an overwhelming fear of failure. Users select the simplest, "safest" mode of sharing—email—because it is most likely to work in all settings. They only move beyond it when some constraint, such as file size or cultural pressure, forces them to.

Ad Hoc Sharing

We can divide the data sharing performed by our subjects into two types: repeated sharing, which occurs multiple times with the same set of participants, and ad hoc sharing, in effect sharing with strangers, which is sharing with people you haven't shared with before and whom you don't know whether you will share with again. If you are going to share repeatedly with a particular group of people, it might be worth doing some up-front work to improve the sharing experience—setting up a server or making sure everyone involved learns how to use a particular service. But it turns out to be harder than you might expect to know when sharing is going to be repeated: except when sharing with oneself or with a stable work group, or possibly with family or friends, sharing tends to be ad hoc. Even closeness or stability of real-world relationships does not serve as a good predictor of future sharing. Digital sharing among real-world connections is still not universal, so although you may expect your family and friends to continue to be connected to you, you may not know how often you are likely to share documents or media with them in the future.

For example, consultants in our subject pool were forced to establish new sharing mechanisms for each new customer engagement. Often prevented from using email by the file sizes involved, they were often expected by their clients to provide secure but provisional electronic sites on which to store interim data or final reports.

Implications for Design

Our findings led us to identify a number of common sharing tasks that are undersupported by the current tools available to users: sharing with themselves, managing transient data, providing transient access, and a general class of sharing problems we term *ad hoc guesting.* The latter refers to the problem commonly faced by our subjects of sharing data with new and unplanned sets of people (unplanned either by themselves or their respective organizations), often without assurance that they would ever share with that group of people again.

Currently, users preferentially and almost overwhelmingly turn to email to solve this problem, except when their impedance matching processes indicate that email is unlikely to be successful. To be successful, alternatives to email must reduce this impedance matching burden—they must be so universal and easy to use that it is worth using them even for ad hoc or one-time sharing.

Organizations wishing to move their users from email onto more secure forms of sharing might find the most effective approach is to enable them to easily share more things, rather than "locking things down" in an effort to get them to share less. In current and future work, we are focused on designing new technologies that effectively balance these tensions between usability and security.

REFERENCES

[1] S. Ahern, D. Eckles, N.S. Good, S. King, M. Naaman, and R. Nair, "Over-Exposed? Privacy Patterns and Considerations in Online and Mobile Photo Sharing," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* pp. 357–366, 2007.

[2] A.D. Miller and W.E. Edwards, "Give and Take: A Study of Consumer Photo-Sharing Culture and Practice," *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* pp. 347–356, 2007.

[3] B. Brown, A.J. Sellen, and E. Geelhoed, "Music Sharing as a Computer Supported Collaborative Application," *Proceedings of the Seventh European Conference on Computer Supported Cooperative Work,* pp. 179–198, 2001.

[4] A. Voida, R.E. Grinter, N. Ducheneaut, W.K. Edwards, and M.W. Newman, "Listening In: Practices Surrounding iTunes Music Sharing," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* pp. 191–200, 2005.

[5] S. Voida, W. Edwards, M.W. Newman, R.E. Grinter, and N. Ducheneaut, "Share and Share Alike: Exploring the User Interface Affordances of File Sharing," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* pp. 221–230, 2006.

[6] T. Whalen, D. Smetters, and E.F. Churchill, "User Experiences with Sharing and Access Control," *CHI '06 Extended Abstracts on Human Factors in Computing Systems,* pp. 1517–1522, 2006.

[7] J.S. Olson, J. Grudin, and E. Horvitz, "A Study of Preferences for Sharing and Privacy," *CHI '05 Extended Abstracts on Human Factors in Computing Systems,* pp. 1985–1988, 2005.

ROHAN MURTY, JITENDRA PADHYE,
RANVEER CHANDRA, ALEC WOLMAN,
AND BRIAN ZILL

designing high-performance enterprise Wi-Fi networks

Rohan Murty received his BS in Computer Science from Cornell University in 2005. He is currently a PhD student in the Computer Science Department (School of Engineering and Applied Sciences) at Harvard University.

rohan@eecs.harvard.edu

Jitendra Padhye received his BE in Computer Engineering from Victoria Jubilee Technical Institute, Mumbai, India, in 1992, his MS in Computer Science from Vanderbilt University in 1995, and his PhD in Computer Science from the University of Massachusetts, Amherst, in 2000. He then spent two years working at ACIRI, now called ICIR. He has been at Microsoft Research since April 2002.

padhye@microsoft.com

Ranveer Chandra is a researcher in the Networking Research Group at MSR. He completed his undergraduate studies at the Indian Institute of Technology, Kharagpur, and holds a PhD in Computer Science from Cornell University. He was the recipient of the Microsoft Graduate Research Fellowship during his PhD and his dissertation on VirtualWiFi was nominated by Cornell for the ACM Dissertation Award.

ranveer@microsoft.com

Alec Wolman is a researcher in the Networking group at Microsoft Research, Redmond. He received a PhD in Computer Science from the University of Washington in 2002. Before graduate school, he worked for Digital at the Cambridge Research Lab.

alecw@microsoft.com

Brian Zill is a Senior Research and Software Design Engineer at Microsoft Research, Redmond. Brian is the author of Microsoft's IPv6 stack as well as the Mesh Connectivity Layer (MCL).

bzill@microsoft.com

WE ARE STARTING TO SEE A SIGNIFI- cant increase in the use of mobile computing devices such as laptops, PDAs, and Wi-Fi enabled phones in the workplace. As the usage of corporate 802.11 wireless networks (WLANs) grows, network capacity is becoming a significant concern. In this paper, we propose DenseAP, a novel architecture for increasing the capacity of enterprise WLANs using a dense deployment of access points (APs). In sharp contrast with wired networks, one cannot automatically increase the capacity of a WLAN by simply deploying more equipment (APs). To succeed in increasing capacity, the APs must be assigned the appropriate channels and power levels, and the clients must make intelligent decisions about which AP to associate with. Furthermore, these decisions about channels, power assignment, and associations must be based on a global view of the entire WLAN, rather than the local viewpoint of an individual client or AP. Given the diversity of Wi-Fi devices in use today, another constraint on the design of DenseAP is that it must not require any modification to Wi-Fi clients. We outline the challenges faced in solving these problems and the novel ways in which DenseAP addresses them.

In a typical office environment, it is relatively easy to deploy a wired Ethernet network. These networks are generally well-engineered and overprovisioned. In contrast, deploying WLANs in enterprise environments is still a challenging and poorly understood problem. WLAN installers typically focus on ensuring coverage from all locations in the workplace, rather than the more difficult-to-measure properties such as capacity or quality of service. Thus, it is common for WLAN users to experience significant performance and reliability problems.

The usage model for enterprise WLANs is currently undergoing a significant transformation as the "culture of mobility" takes root. Many employees now prefer to use their laptops as their primary computing platform, in both conference rooms and offices [12]. A plethora of handheld Wi-Fi enabled

devices, such as PDAs, cell phones, VoIP-over-Wi-Fi phones, and personal multimedia devices, are becoming increasingly popular. These changes are leading enterprise network administrators to question the assumptions made during the design and deployment of their existing WLANs [1]. In addition to the scalability challenges that arise with increased WLAN usage, the applications for many of these new mobile devices require better QoS and mobility support.

The need to improve enterprise WLAN performance has been recognized by the research community [2, 13, 14] as well as by industry. Upgrades at the PHY layer, such as the transition from 802.11g to 802.11n, are important steps along the path to increasing WLAN capacity, but they are not enough. Deploying more APs has the potential to improve WLAN capacity, yet in sharp contrast to wired networks, one cannot automatically increase the capacity of a WLAN by simply deploying more equipment. To succeed in increasing capacity, intelligent software control of the WLAN devices is needed to deal with such issues as channel assignment, power management, and managing association decisions.

We present a new software architecture called DenseAP, with the goal of significantly improving the performance of corporate Wi-Fi networks. A key emphasis in our design of the DenseAP system is on practical deployability. For example, because of the incredibly wide diversity of existing Wi-Fi devices, DenseAP must provide significant performance benefits without requiring any modifications to existing Wi-Fi clients. Furthermore, as a consequence of these concerns, we do not consider changes that require hardware modifications or changes to Wi-Fi protocols. Although these constraints do limit the design space to a certain extent, we found they also open up a set of interesting research challenges.

DenseAP architecture and design challenge two fundamental characteristics of most current enterprise WLAN deployments. First, existing WLANs are designed with the assumption that there are far fewer APs than clients active in the network, whereas with the DenseAP architecture the common case will be that APs outnumber clients. Second, in conventional WLANs clients decide which AP to associate with, whereas the DenseAP systems use centralized control of the association process.

The scarcity of APs in conventional enterprise WLANs limits their performance in a variety of ways. For example, with a large number of nonoverlapping channels (e.g., 12 in 802.11a) but only a few APs, the WLAN is unable to fully utilize the available spectrum at each location. Because radio signals fade rapidly in indoor environments, adding extra radios to existing APs is not as effective as deploying a larger number of APs in different locations. If APs are densely deployed, each client can associate with a nearby AP and will thus see better performance. A dense deployment also ameliorates the "rate anomaly" problem [8] that hurts the performance of conventional WLANs.

To fully benefit from a dense deployment of APs, the clients must associate with the right AP. In conventional WLANs clients select which AP to associate with. Typically, the association policy is implemented within the device driver for most Wi-Fi clients, and it uses only locally available information. For example, most client drivers tend to use signal strength as the dominant factor in selecting an AP, yet it is well known that this behavior can lead to poor performance [9]. For example, when many clients congregate in a conference room, they all tend to choose the same AP. To improve performance, when multiple APs are available clients must associate with different APs. In the DenseAP architecture, the central controller gathers information from

all the APs and then determines which AP a particular client should associate with. Using a novel way to manipulate the 802.11 association process, the central controller ensures that a specific Wi-Fi client will only discover the AP that the controller has chosen, thus ensuring that the clients associate with this AP. Using a similar mechanism, the controller also carries out periodic load balancing by seamlessly moving clients from overloaded APs to nearby APs with significantly less load. The controller achieves all this without requiring any changes to the association control software that runs on the clients.

The DenseAP architecture is quite versatile and is capable of improving many aspects of performance of enterprise WLANs. In this paper, we focus on describing how DenseAP helps significantly improve the capacity of enterprise WLANs. We define *capacity* simply as the sum total of throughput all active clients in the network can potentially achieve. We also briefly discuss how the architecture can improve other aspects of performance, such as quality of service for delay- and jitter-sensitive applications such as VoIP.

We describe the DenseAP architecture, algorithms, interesting research challenges, and open issues that arise when deploying APs in a very dense manner. These challenges include the need for appropriate channel and power assignment, ensuring that clients associate with the appropriate AP, and the need to make the system self-managing and easy to deploy. We will point out how the performance could be further improved if we could modify the end clients or count on their cooperation in some manner. We view the current DenseAP architecture not as the final word but as a practical first step toward exploring many ways of improving the performance of Wi-Fi networks.

Architecture

Figure 1 presents a high-level illustration of the DenseAP system architecture. Broadly, the system consists of DenseAP nodes (DAPs) connected to the wired network and controlled by a central server. Each DAP has a programmable software AP running on it. The DAP sends periodic reports to the DenseAP Central Controller (DC). These reports consist of the list of clients associated with it and the amount of traffic sent to or received from each client. The DC aggregates reports received from all DAPs and uses this information to send commands to DAPs to control their behavior.

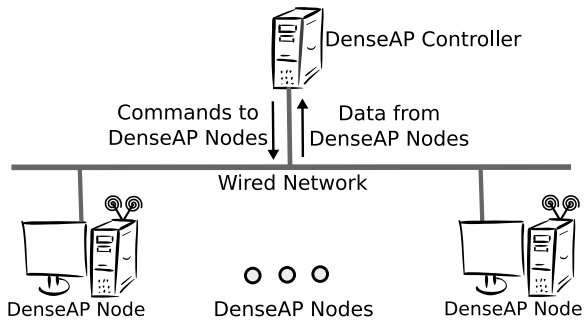


FIGURE 1: OVERALL ARCHITECTURE OF THE DENSEAP SYSTEM

In the context of this architecture, we need to answer the following three questions: First, what information does the DC need, and what "knobs" can it tune to improve the capacity of the system? Second, how densely should the DAPs be deployed? Third, is the architecture scalable and cost-effective?

Role of the DenseAP Central Controller

Given a set of DAPs and clients, the overall capacity of the WLAN depends on several factors. Since we do not wish to modify the clients, the set of performance "knobs" available to us is somewhat limited. In our current implementation, the DC attempts to improve capacity by controlling the channel each DAP operates on, the power with which each DAP transmits, and the DAP with which each client associates.

Two other knobs that can also affect the overall WLAN capacity are the Clear Channel Assessment (CCA) threshold used by each DAP and the autorate algorithm implemented on each DAP. The CCA threshold determines the level of background noise an 802.11 transmitter will consider acceptable before transmitting. If set to a high value, the transmitter is more likely to cause interference with other transmissions [13]. We do not modify the CCA threshold since most off-the-shelf wireless cards do not allow modifications to this value. The second knob is the autorate algorithm, which determines the transmission rate used by the DAP to communicate with the clients. Autorating algorithms have been studied extensively by prior research. DenseAP nodes use the autorate algorithm described in Wong et al. [17]. In the future, we plan to investigate whether the autorate algorithm can benefit from the network information gathered by the DC.

We now describe how the DC performs association and channel management in the system. We also address power control, mobility, and fault tolerance in the system.

ASSOCIATION CONTROL

In the DenseAP system, the DC decides which client associates with which DAP. This is achieved by limiting the visibility of DAPs to the clients. We first describe the mechanisms involved and then describe the algorithm used to decide which DAP is made visible to which client.

LIMITING DAP VISIBILITY TO CLIENTS

In conventional 802.11 networks, APs advertise their presence by sending out beacons, which include their SSID and BSSID. Prior to association, clients gather information about the APs by scanning the channels one by one and listening for beacons on each channel. This is called "passive scanning." The clients also perform "active scanning," whereby they send out a probe request message on each channel. This message is a request for APs to send out information about themselves. APs respond to a probe request message with a probe response message, the contents of which are similar to the beacon frame. Once the client gathers information about all APs, it decides which AP to associate with, and it carries out the association handshake.

The DC performs association control by limiting the visibility of DAPs to clients, exposing DAPs on a "need to know basis" to a particular client. This is achieved via two techniques. First, since 802.11 networks are identified by their SSIDs, DAPs are set to beacon with the SSID field set to NULL. Second, each DAP maintains a local access control list (ACL) of client MAC addresses that is solely managed by the DC. On receiving a probe request from a client, the DAP replies with a probe response message only if the client's MAC address is in its ACL (i.e., if the DC has previously added the MAC address to the ACL of this DAP). If a DAP receives a probe request from a client whose MAC address is not in its ACL, it sends a message to the DC, informing the controller that a client might be requesting service.

The reason to use hidden SSID beacons is to keep the network "hidden" and prevent clients from associating with any other DAP in its vicinity. Another alternative is for DAPs not to beacon at all; however, beacons are essential for clients to use power save mode. We have also found some client drivers that disconnect if they don't receive periodic beacons from the access point they are associated with.

By adding the MAC address of a client to only one DAP's ACL at a time, the DC ensures that, for the SSID associated with the DenseAP network, only one DAP is visible to the client at any given time.

Note that traditional MAC address filtering could not have achieved this. MAC address filtering only prevents association, not probe responses. With traditional MAC address filtering, a client would discover several DAPs, and it might not even try to associate with the one the DC has chosen for it.

This method of association control has two key advantages. First, it requires no changes to the client. Second, the DC has a comprehensive view of the traffic in the network and hence can make an informed decision when choosing a DAP for a client.

We have verified that most, if not all, wireless drivers and cards available on the market today perform active scanning and hence they are able to discover DAPs. DenseAP is also designed such that if a client fails to associate with the assigned DAP (say, because of interference near the client), the DC detects this since DAPs periodically report back information about associated clients. The DC then reassigns the client to a different DAP.

DAP SELECTION POLICY

We now consider how the DC determines which DAP a client should associate with. Our intuition is to take into account both the load on the DAP and the quality of the connection between the client and the DAP. We capture this in a metric called the available capacity, which is calculated as follows: Available Capacity = Free Air Time × Expected Data Rate. Free Air Time is the fraction of time during which the DAP and the channel it is on are not busy. Expected Data Rate is an estimate of the transmission rate the DAP and the client will achieve when communicating with each other. This is primarily determined by the quality of the connection between the client and a prospective candidate DAP. The client is assigned to the DAP with the most available capacity. In other words, the DAP with the most free capacity will allow the client to send the most data, while minimizing the impact on other clients.

Channel assignment is tied into the association scheme. Since we only have a limited number of channels, those DAPs not servicing clients do not need to beacon and hence we don't assign channels to them. Therefore a DAP is assigned a channel on an on-demand basis (i.e., only when at least one client is associated with it). When a DAP does not have a channel assigned, it scans all channels and estimates load on the various channels in its vicinity. This information is sent to the DC. When assigning a channel to a DAP, the DC picks the channel with the least load.

One could propose other association policies, depending on the end goal for which the system is optimizing. For example, it can take into account the number of clients associated with the DAP, or it can try to balance the load across all DAPs. It may be possible to anticipate and factor in the future load generated by the client (e.g., demand from VoIP clients is generally predictable). We are actively exploring this research space.

We now describe how we compute the free air time at the DAP and the expected data rate. We do not expect these calculations to be precise, particularly when it comes to estimating the expected data rate. However, our intention is to provide a reasonable ordering of DAPs and to recover from any mistakes via load balancing. Hence, even if a client were to be assigned an "incorrect" DAP, the system will, at some point, hand off the client to a more suitable DAP.

ESTIMATING FREE AIR TIME

We can estimate the free air time in the vicinity of a DAP with varying degrees of accuracy. The DAP could simply add up the air time used by all packets that it has sent and received, over a unit period of time. The remaining time is the free air time at the DAP. However, such an approach ignores the effects of interference. Part of the interference can be accounted for by adding up any traffic sent or received by nearby DAPs that are on the same channel. The DC can perform this calculation using the information submitted by each DAP. A much more accurate method of estimating the free air time is for each DAP to use the ProbeGap technique, as proposed by Lakshminarayan et al. [11], and report the information to the DC. This technique directly estimates the free air time by computing the delay experienced by small probe packets. We currently use a variant of this method in our implementation. Further details are provided in our paper [15].

ESTIMATING EXPECTED TRANSMISSION RATE

It is difficult to accurately predict the transmission rate a client will achieve when communicating with a DAP (or vice versa). The rate primarily depends on how well the DAP receives the client's signal. However, the rate also depends on a variety of other factors such as the autorate algorithm implemented by the client, power levels used by the client, and channel conditions near the client. Of these factors, we can only estimate how well the DAP receives a client's signal.

When attempting to associate, clients send out probe request messages, which are overheard by nearby DAPs, who then inform the central controller. We estimate the quality of the connection between the client and the various candidate DAPs using the signal strength (RSSI) of the received probe request frames at the various DAPs. We convert these observed signal strengths into estimates of expected transmission rate by using a mapping table. The mapping table drops RSSI values into fixed-size buckets and assigns an expected rate to each bucket. We assume that the same transmission rate will be used by both the client and the AP. We call this the rate-map approach. The mapping table is initially generated by manual profiling using a few clients at various locations. It can then be refined as actual data from more clients is gathered during live operation.

At first glance, it may appear that extrapolating the signal strength observed in the uplink direction to an expected transmission rate in both directions could result in inaccurate estimations and/or poor performance, especially given the other factors that are ignored. Yet, in our system, we find that it provides reasonable results for the following reasons. First, given the density of access points, a client generally associates with a nearby DAP. For such short distances, we find that signal strength measured in one direction is a good approximation of signal strength seen in the other direction. Second, because the client and the DAP are usually close to each other, we generally see good signal strength in both directions. Most commercial Wi-Fi cards

behave similarly in such conditions. Finally, note that we do not need the exact transmission rates used by either the client or the DAP. The conversion table is merely a way of ranking the relative importance of the observed signal strength. We present further details in our paper [15] on this approach as well as its efficacy.

A typical wireless network tends to be dynamic in that the available capacity of a DAP will change as clients enter or leave the network and as the traffic load changes. We aim to make a reasonably good and efficient choice when initially assigning the client to a DAP and then to adapt to the changes in the environment via load balancing as we now describe.

LOAD BALANCING VIA HANDOFFS

The goal of the load balancing algorithm is to detect and correct overload situations in the network. We expect that such situations will be rare in an environment with a dense deployment of access points and with numerous available orthogonal channels (e.g., 12 in 802.11a). However, it is important to watch for, and correct, the overload situations if and when they occur.

For example, an overload situation might occur if many clients congregate in a conference room and the network conditions are such that the algorithm used when associating new clients assigns several of them to a single DAP. In such a situation, all clients simultaneously transmitting or receiving data can cause an overload at the DAP.

The load balancing algorithm works as follows. Once every minute, the DC checks all DAPs to see if any are severely overloaded. Recall from earlier that the busy air time (load) calculation incorporates the impact of traffic/interference near the DAP and the downlink traffic generated by the DAP. We consider a DAP to be overloaded if it has at least one client associated with it and it reports free air time of less than 20%—if, in other words, the channel is more than 80% busy in the vicinity of this DAP. The DC considers the DAPs in decreasing order of load. If an overloaded DAP (A) is found, the DC considers the clients of A as potential candidates to move to another DAP. Recall that the DAPs send periodic summaries of client traffic to the DC. These summaries include, for each client, a smoothed average of the sum of uplink and downlink traffic load generated by the client during the previous interval. The load is reported in terms of air time consumed by the traffic of this client and the average transmission rate of the traffic.

For each client M at A, the DC attempts to find a DAP B such that the expected rate M will get at B is no less than the average transmission rate of the client at A, and the free air time at B is at least 25% more than the air time consumed by M at A. If such a DAP is found, M is moved to B by using a mechanism similar to the association mechanism described earlier [15]. Note that if B had no clients associated with it, the DC would also assign it a channel (the one B reported to have the most free air time on), just as it would do when associating a new client.

The load balancing algorithm moves at most one client that satisfies these criteria during each iteration. Furthermore, once a client M has been handed off from A to B, it is considered ineligible to participate in the next round of load balancing. These hysteresis techniques are intended to prevent oscillations.

We note a few things about the load balancing algorithm. (i) Our algorithm is conservative. Moving clients from one AP to another is a potentially disruptive event, and we try to minimize how often we force such reassocia-

tions to occur. (ii) The load balancing algorithm improves overall system throughput in two ways. First, the client that is moved to the less-loaded AP can ramp up and consume more bandwidth. Second, the clients that stayed with the previously overloaded AP now have one fewer client to contend with, and they can also increase their throughput. (iii) It is sometimes possible to do load balancing by changing the channel of the overloaded DAP. This technique is useful only if the background traffic/interference (potentially from other DAPs) on the channel is significantly higher compared to the traffic sent/received by the overloaded DAP itself. However, the drawback of this technique is that all clients associated with the DAP will have to reassociate. Since we consider client reassociations to be disruptive events, we do not use this technique.

POWER CONTROL

We have three options when it comes to power control in the DenseAP system. The first is to perform no power control at all, the second is to perform unilateral power control at the DAP, and the third option is to perform co-ordinated power control between clients and the DAPs. We rule out the third option at present, since it requires client modifications.

In our current testbed, we use the first option: We allow all DAPs to transmit at maximum power. This increases the coverage area, but it minimizes the potential for spatial channel reuse. We are also experimenting with unilateral power control at the DAP. The idea is that, given a set of clients associated with the DAP, the DAP transmits at the minimum power necessary to provide "good" service to all clients. The "goodness" of the service is defined in terms of loss rate and transmission rate. The problem, however, is that the clients are free to transmit at any power they choose, which reduces the potential for spatial reuse. Our preliminary experiments have uncovered other problems with this option. Our results show that unilateral power control at the DAPs results in increased instances of hidden terminal and capture-effect problems. We are investigating this issue further.

MOBILITY

To handle client mobility, the DC keeps track of a client's location, using the technique described in Chandra et al. [7]. When the location changes significantly, the system hands off the connection to a suitable DAP located nearby. The handoff is performed as previously described, and the DAP is selected using the available capacity metric. A client that undergoes handoff is considered ineligible to participate in load balancing for some period of time, to prevent oscillations. It is, however, eligible to participate in another, mobility-related handoff.

FAULT TOLERANCE

We want DenseAP to be a self-configuring and self-managing system. Hence, when a DAP goes offline or is rebooted, it no longer sends periodic load information to the DC. The DC detects this, flags the DAP as a possible failure, and does not assign any new clients to it. The clients associated with the failed DAP get disconnected. These clients immediately begin scanning for other DAPs in the vicinity by sending out probe request messages. Other DAPs in the vicinity pick up these probe messages and alert the DC, which assigns these clients to other DAPs, as per the association policy.

What Is the Desired Density of DAPs?

So far we have focused on the role of the central controller. The other key factor that affects the performance of the DenseAP system is the DAP deployment density. Several important questions need to be studied in this regard. For example: (i) Where should the DAPs be placed? (ii) Is there a point at which adding more DAPs to the system can hurt performance? (iii) How do we determine the minimum necessary density for a required level of service in a given environment? (iv) Since wired networks tend to have well-defined SLAs, how does one specify an acceptable level of service for WLAN?

Guidelines developed for traditional WLANs offer little help in answering these questions, since these guidelines are generally developed with the aim of using as few APs as possible while maximizing the coverage area. As such, the questions pertaining to density present interesting research challenges, and we are actively working to answer them. Mhatre and Papagiannaki [13] have explored a closed-form solution for optimal AP density by varying the CCA threshold, which in turn affects the throughput and coverage of the network.

In our current deployment, described earlier, we use ordinary user desktop machines to serve as DAPs. If this approach is followed, then question (i) need not be answered. Every desktop machine (or most of them) can serve as a DAP. However, this raises a different question: How do we know we have adequately covered the given area? Administrators of traditional WLANs use expensive site-surveying tools to determine the AP placements. Inspired by DAIR [3], we are exploring methods to automatically determine whether we have left any gaps in our coverage.

Scalability

Our architecture uses a central controller (the DC) to manage all DAPs. Each DAP sends out periodic reports to the DC. This raises scalability concerns. To address these concerns, we note that our DC was able to easily manage a network of 24 DAPs and 24 clients, without any special optimizations. The CPU load on the DC never exceeded 30%. We estimate that the amount of control traffic generated by each DAP was less than 20 kbps. Thus, we estimate that a slightly more powerful DC could easily handle a network of about 100 DAPs, without any optimizations. This should be enough to cover a floor of our office building.

We note here that it is not strictly necessary to use a single central controller. What is necessary is the use of global knowledge while making association and channel assignment decisions. In theory, the functionality of the central controller can either be replicated or even implemented in a fully distributed manner. The DAPs can exchange information with each other to gain a global view of the network and make appropriate decisions. However, this approach is more complex to implement and has its own set of scalability concerns.

Another issue we must address is the impact of several DAPs in close proximity, beaconing and sending probe packets. Our measurements show that, in the common case, the impact on performance is less than 1%. This is because only those DAPs servicing clients send beacons and because, when we use multiple channels, the number of DAPs on any one channel is smaller.

Implementation

Our implementation of the DenseAP system is deployed on a portion of our office floor. Our current testbed consists of 24 desktop-class PCs serving as DAPs. (See Figure 2.) The DAPs are deployed at a density of roughly one machine in every other office. The area is normally served by three of our corporate WLAN's APs. As illustrated in Figure 2, the current DenseAP deployment is nine times more dense than the corporate WLAN deployment.

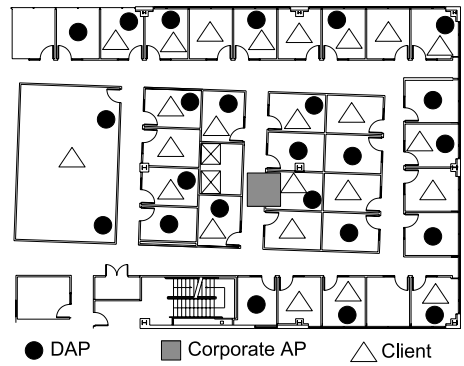


FIGURE 2: THE TESTBED. THE AREA IS ROUGHLY 32 × 35 M.

DAPs are constructed entirely from commodity hardware. We use off-the-shelf PCs. To each PC we add a Netgear JWAG511 wireless interface card. These are multiband 802.11 a/b/g radios using a RealTek chipset. The access point functionality is provided in software through a combination of a device driver and a system service. Having the AP functionality implemented in software was critical to our efforts, as it allowed us to easily modify the AP behavior to our specifications.

The DC also runs on an ordinary (but dedicated) desktop machine. All DAPs are connected to the same IP subnet on their wired Ethernet link. The DenseAP WLAN runs in the 5 GHz band (802.11a) on the lower 8 channels. The corporate network also operates in the same band.

Our paper [15] has further details regarding the evaluation of the system as well as an extensive discussion on various open questions and issues.

Related Work

Prior academic work on either improving capacity or managing dense deployments has focused on channel assignment, power control, and associations, most of which have required modifications to clients. Fundamentally, DenseAP differs from all prior work as follows:

- Practicality: Of the host of prior work in this area [4, 6, 10, 13, 14, 16], to our knowledge, DenseAP is the first system to be designed and deployed in a real-world scenario.
- Intelligent association and load balancing: To our knowledge, none of the prior proposals is capable of intelligent associations or able to deal with a dynamic operating wireless environment without requiring client modifications. In most such systems, associations tend to be static or solely driven by the clients.
- No modifications to clients necessary: Most approaches require modifications to the clients [4, 5, 6, 9, 16] or the clients to cooperate in some manner that breaks the prevalent 802.11 standards.

SMARTA [2], like DenseAP, addresses the problem of managing dense AP deployments to increase capacity or lower latency, without modifying clients. There are several key differentiators between it and DenseAP. First, SMARTA does not account for a dynamic operating environment. It lacks the ability to load-balance clients and hence the need to assign "correct" channels and power levels at the outset is greatly magnified. Second, unlike DenseAP, SMARTA relies entirely on the clients to make their own association decisions, which by prevalent standards are agnostic to network load. In a dense deployment, this approach can very easily lead to lower throughput for all clients [9]. Third, it is unclear how clients maintain persistent connections when SMARTA performs channel or power assignments. The system does not make an effort to sustain such connections at the client. Most of these differentiators arise from SMARTA having been studied almost completely in simulations.

Mhatre and Papagiannaki [13] propose varying the (CCA) threshold on APs to increase capacity in 802.11g mode. The system has been designed and studied within the confines of the Opnet simulator. Similarly, other proposals involve varying the receiver sensitivity as well as the CCA threshold [18].

A host of products by networking startup companies are designed to manage AP deployments in the enterprise. Although practical, most systems tend to ignore association control and load balancing, or they address such challenges by requiring users to install custom client drivers. Further references are provided in our paper [15].

Discussion and Conclusion

Use of the DenseAP system thus far has been focused on improving capacity in the enterprise. It can also improve other dimensions of WLAN performance such as lowering latency, separating voice and data traffic, and providing QoS. Each one of these goals entails tweaking the association and handoff policies. For example, in the case of handoffs, we could pick only those clients that appear to be experiencing low transmission rates to the DAP they are associated with. Another possibility is to try to determine client traffic patterns and aggregate all VoIP clients during the association process to a group of DAPs to provide better QoS. We are continuing to investigate these avenues with the overall goal of using DenseAP to improve WLAN performance along multiple dimensions.

REFERENCES

[1] R. Ahlawat and C. Canales, User Survey Report: Wireless LANs, North America and Europe, Gartner Report, 2005.

[2] N. Ahmed and S. Keshav, "SMARTA: A Self-Managing Architecture for Thin Access Points," *CoNEXT*, Lisboa, Portugal, December 2006.

[3] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill, "Enhancing the Security of Corporate Wi-Fi Networks Using DAIR," *MobiSys*, Uppsala, Sweden, June 2006.

[4] P. Bahl, M. Hajiaghayi, K. Jain, V. Mirrokni, L. Qiu, and A. Seberi, "Cell Breathing in Wireless LANs: Algorithms and Evaluation," *IEEE Transactions on Mobile Computing*, 2006.

[5] A. Balachandran, P. Bahl, and G. Voelker, "Hot-Spot Congestion Relief and Service Guarantees in Public-Area Wireless Networks," *SIGCOMM Computer Communication Review,* 32(1), 2002.

[6] Y. Bejerano and R.S. Bhatia, "MiFi: A Framework for Fairness and QoS Assurance in Current IEEE 802.11 Networks with Multiple Access Points," *Infocom*, Hong Kong, March 2004.

[7] R. Chandra, J. Padhye, A. Wolman, and B. Zill, "A Location-Based Management System for Enterprise Wireless LANs," *NSDI '07*, Cambridge, MA, April 2007: http://www.usenix.org/events/nsdi07/tech/chandra.html.

[8] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance Anomaly of 802.11b," *Infocom*, San Francisco, CA, March 2003.

[9] G. Judd and P. Steenkiste, "Fixing 802.11 Access Point Selection," *SIGCOMM Poster Session*, Pittsburgh, PA, July 2002.

[10] B. Kauffmann, F. Baccelli, A. Chaintreau, V. Mhatre, K. Papagiannaki, and C. Diot, "Measurement-Based Self Organization of Interfering 802.11 Wireless Access Networks," *Infocom*, Anchorage, Alaska, May 2007.

[11] K. Lakshminarayanan, V.N. Padmanabhan, and J. Padhye, "Bandwidth Estimation in Broadband Access Networks," *IMC*, Sicily, Italy, Taormina 2004.

[12] M. Lopez, The State of North American Enterprise Mobility in 2006, Forrester Research document, December 2006.

[13] V. Mhatre and K. Papagiannaki, "Optimal Design of High Density 802.11 WLANs," *CoNEXT*, Lisbon, Portugal, December 2006.

[14] V. Mhatre, K. Papagiannaki, and F. Baccelli, "Interference Mitigation through Power Control in High Density 802.11 WLANs," *Infocom*, Anchorage, Alaska, May 2007.

[15] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill, "Designing High Performance Enterprise Wi-Fi Networks," *NSDI '08*, San Francisco, CA, April 2008: http://www.usenix.org/events/nsdi08/tech/murty.html.

[16] A. Vasan, R. Ramjee, and T. Woo, "ECHOS—Enhanced Capacity 802.11 Hotspots," *Infocom*, Miami, Florida, March 2005.

[17] S. Wong, S. Lu, H. Yang, and V. Bharghavan, "Robust Rate Adaptation for 802.11 Wireless Networks," *MobiCom*, Los Angeles, California, September 2006.

[18] J. Zhu, B. Metzler, Y. Liu, and X. Guo, "Adaptive CSMA for Scalable Network Capacity in High-Density WLAN: A Hardware Prototyping Approach," *Infocom*, Barcelona, Spain, April 2006.

ALVA COUCH

"standard deviations" of the average system administrator

Alva Couch is an Associate Professor of Computer Science at Tufts University, where he and his students study the theory and practice of network and system administration. He served as Program Chair of LISA '02 and was a recipient of the 2003 SAGE Professional Service Award for contributions to the theory of system administration. He currently serves as Secretary of the USENIX Board of Directors.

couch@cs.tufts.edu

A FORMER SYSTEM ADMINISTRATOR, turned analyst, muses about standards in system administration and other professions and the profound social effects of establishing or ignoring standards.

System administrators have a surprising amount in common with electricians. Both professions require intensive training. Both professions are plagued by amateurs who believe (erroneously) that they can do as good a job as a professional. Both professions are based upon a shared body of knowledge.

But electricians can call upon several resources that system administrators lack. Electricians have a legally mandated mentorship/apprenticeship program for training novices. They have a well-defined and generally accepted progression of job grades, from apprentice to journeyman to master. They advance in grade partly through legally mandated apprenticeship and partly through legally mandated certifications. These certifications test for knowledge of a set of standards for practice—again, mandated by law. The regulations are almost universally accepted as essential to assuring quality workmanship, function, and safety.

In short, one electrician can leave a job and another can take over with minimal trouble and without any communications between the two, and one can be sure that the work will be completed in the same way and to the same standard. Can any two system administrators, working for different employers, be interchanged in such a fashion?

At present, system administrators are at a critical juncture. We have functioned largely as individuals and individualists, and we greatly value our independence. But the choices we make as individuals affect the profession as a whole. I think it is time for each of us to act for the good of the profession, and perhaps to sacrifice some of that independence for what promises to be a greater good. This will be a difficult sacrifice for some, and the benefits may be intangible and long-term rather than immediate. But I think it is time now for us to change the rules.

From standards for distributions (e.g., the Linux Standard Base) to standards for procedures (e.g., those upon which Microsoft Certified Engineers are tested), I believe that—although standards may annoy us as individuals— standards for our profession (and certification to those standards) help build respect for system administration as a profession. Compliance with standards gives us a new and objective way to measure the quality of man-

agement at a site. Standards not only make the task easier but also enforce desirable qualities of the work environment and help to justify appropriate practices to management. Adoption of standards also has a profound effect upon our ability to certify system administrators and even changes the meaning and form of such a certification.

Learning from Electricians

Is a system administrator accorded the same respect as an electrician? I think the answer is an emphatic "no," at least for those electricians who hold a master's license. There are two factors that engender respect for a master electrician: legally mandated standards linked closely to legally mandated apprenticeship and certification.

One difference between being a senior system administrator and being a master electrician is the existence of the National Electrical Code and related local codes. The Code is a set of standards for wiring that broadly defines how licensed electricians must do wiring, as well as how vendors of electrical appliances and devices must construct those devices. The code has a specific and powerful property that *compliance to the code may be checked and certified by an independent examiner.*

Another difference is that *electricians go through a legally mandated apprenticeship and certification* before being allowed to practice as a master. There are three levels of electrician: apprentice, journeyman, and master. Becoming a journeyman requires both a mandatory apprenticeship (to either a journeyman or master) and certification of knowledge of both the National Electrical Code and any local codes that may contradict or strengthen the national code. Becoming a master requires serving as a journeyman under another master and passing another, more stringent certification exam.

System administrators could benefit greatly from such a mentoring system. By contrast, a typical beginning system administrator receives little training, guidance, or supervision, and there is no clear and universal path into the profession. We acknowledge the need for mentoring but have difficulty finding mentors or even clearly defining what mentorship should comprise. The basis by which electricians obtain all of these desirable things—and the central component of their training and certification—is a set of externally verifiable standards, embodied as the Code.

The standards for electrical work are an inextricable part of the certification process for electricians. But there are many system administrators who find standards annoying. Many system administrators have complained to me that the answers they must give about "best practices" on certification exams are "wrong." We pride ourselves on individuality, and on coaxing the last ounce of performance out of any system, and in reacting faster than anyone else in repairing a problem. To improve our personal "reaction time," we construct systems "in our own images" and, upon coming to a new site, face an irresistible urge to force it to comply with our personal standards even before we force it to function properly. But the term "personal standards" is really an oxymoron, because anything that only one person considers to be a standard cannot—by definition—be one.

A Tale of Four Repositories

No one really cares where software is actually located in a filesystem, provided that it is in the user's path. But apparently system administrators do

care because at my site there are currently four competing systems for locating software in filesystems. These evolved over time as follows:

- I created a system based upon the pattern /loc/category/package-revision back in the early 1990s, documented in a 1996 LISA paper [1]. User software installed into /loc is mapped into user-space at /local. This scheme was used for many years, while I managed the repository between 1990 and 2000.
- When I hired my own replacement, he decided this scheme was "inelegant," and started installing packages in /loc/packages/package/revision. I did not stop him because I did not want that job back!
- When he hired a student assistant, the assistant did not like that scheme and started installing packages in /rel instead, using a format not easily explainable and whose description—mercifully—has not survived the ravages of time!
- When these folks left for greener pastures, the next sysadmin decided "all packages should be locally installed" and installed them in /var /local/.

And so it goes. What was the result of this? As I write, three of these four repository schemes are still active. Part of the reason for this is that the packages make reference to themselves at their installed locations. System administrators were quick to adopt new "personal standards" (oxymoron intended), but the work of making the system "comply" with those standards was never completed, and there are programs in my original repository that—even though it has not been updated since 2000—are still in use and were installed as early as 1994!

That Standard Is So Lame!

Each system administrator in this story thought he was doing a good thing by imposing "personal standards." The first one exclaimed to me that my "old standard is so lame!" The others made equivalent comments about the schemes of all of his predecessors. Each one believed sincerely that his new "personal standard" would be the one that would make the most sense and pass the test of time.

To me, looking at it from the outside, this attitude was something like a gambling addiction, as each new player entering the game thought his new scheme would "win" where other schemes had "lost." But—as surely as most gamblers lose at the casino—every "personal standard" ended up being "lame" in the end!

Each "personalization" seemed at the time to be a "solution" but turned over time into a "problem." The real thing that is "so lame" in this story is that there are three "personal standards" for doing the same thing. There is only one mechanism by which our site could escape the cycle: by adopting standards for shared package location and then assuring compliance to those standards. "Personal standards" just do not work.

Incidental Complexity

What standards could really help the profession of system administrator? I and my students have made a multi-year study of what we call the "incidental complexity" of system administration [2]. Incidental complexity arises from making arbitrary decisions without coordination and for no particularly good reason. The main cause of incidental complexity is that many management decisions have nothing to do with final system behavior. A pa-

rameter value is "incidental" if the choice of a value is a matter of style and preference and has no conceivable effect upon system function. My students and I have found that, on average, the values for most configuration parameters are incidental.

Incidental complexity includes choices for locations of files within a system. For example, the outward appearance of a Web server has nothing to do with where files for the Web server are stored, which led one of my students to propose that this choice be taken away from the administrator by automated mechanisms [3].

Based upon experiences like these, I believe that appropriate practice standards for system administration involve identifying incidental choices and standardizing those choices. This might seem hard for an unvirtualized system, but in a properly virtualized execution environment, there are even more incidental parameters than before. I believe the following choices are now completely arbitrary for a site of sufficient size:

- Names of home directories (use SAN volumes to enforce quotas and enforce barriers between user groups).
- Locations of published Web content (use SAN volumes and standardize locations of mount points).
- IP addresses of hosts (use IPV4/NAT and/or IPV6).
- Locations of remotely installed and locally installed software packages.

Through a combination of virtualization and standards, the values of these "parameters" do not matter, and every site can choose these in exactly the same way.

The benefits of this kind of "standard" are subtle but profound. If all repositories are named the same way, a system administrator won't have to read site documentation to fix a Web server, any more than an electrician has to stop—while wiring your house—to refer to the National Electrical Code. In the same way that no electrician will work on noncompliant wiring, a system administrator won't even try to learn what has been done in a nonstandard fashion, because such work is not sufficiently externally verifiable. No more "job security," but, rather, an increase in overall system administrator efficiency and a dramatic reduction in what every practitioner has to remember.

The Linux Standard Base

One example of eliminating incidental complexity is the Linux Standard Base (LSB) [4]. This is a set of standards for layout of Linux distributions, and it specifies the locations and versions of important files and libraries in a Linux distribution. The purpose of the LSB is to assure that vendor-supplied Linux applications will run in an LSB-certified environment. There are two levels of certification, both checked by software tools. First, the environment is certified as being compliant with the LSB standard, by running a script that checks that all files and versions are present and in the proper places. Then the application itself, as a binary file, is checked for accessing library functions properly, with the correct types of arguments.

The "theory" behind the LSB is "test once," "certify once," "works everywhere." If one tests an application in a certified environment, and the application works there, and the application is itself certified, then we have high (but not quite perfect) confidence that it will work in any certified environment (where uncertainty is due to esoteric technical limits of LSB beyond the scope of this article).

The LSB serves as one example of a practical standard. One can run the certification tools on an environment or application and get a "yes" or "no" answer, so the LSB meets the definition of external verifiability. A rather long document describes compliance measures in plain English, so the LSB meets all of the criteria for a standard.

Cost and Value of Standards

The LSB serves as a good case in point for considering the cost and value of standards. The cost of compliance to the LSB is actually severalfold and a bit subtle:

The LSB always lags behind current operating systems versions and distributions, because a distribution has to exist before it can be standardized, and the standardization process takes time. Thus, almost by definition, a compliant environment is "out of date" by some reckonings.

- The LSB is "a lot to remember" when managing a system. In particular, upgrading a system often inadvertently invalidates the base. There are inadequate tools for asking "what-if" questions about updates and making intelligent decisions. Compliance takes time to initially assure and time to maintain.
- The LSB does not standardize all aspects of application and system, and unstandardized aspects can cause an LSB-compliant application to fail in an LSB-compliant environment.
- But the value of compliance is a bit more subtle to itemize:
- Compliance only provides assurance that one configuration will work, but not that other configurations will not work. Thus it is possible to "do without" LSB compliance and not feel the pain.
- Compliance is thus not a guarantee of "point behavior," but of "lifecycle behavior"; it isn't "required" at any one time, but overall compliance—over time—increases software reliability, at the cost of being slightly "behind the curve" of software development.

In the same way, the National Electrical Code is important to electricians—not because houses could not be wired differently and still function—but because the standards therein ensure that, over the house's lifecycle, the wiring is unlikely to fail compliance with the Code and that any electrician who knows the Code can come to a compliant house and know what to expect and how to change it.

Certification to Standards

Adopting standards has a profound effect upon the meaning of certification. So far, certification efforts for system administrators have concentrated upon certifying skills; I have commented on the dubious value of this kind of testing and certification in a previous *;login:* article [5]. The license exam for a master electrician instead certifies knowledge of the Code; skills are tested instead in the context of apprenticeship, during which there are numerous opportunities to observe them. Likewise, an exam about system administration practice, rather than about knowledge, is much narrower and easier to create than an exam about the general knowledge required to function as a system administrator; the latter is also ideally tested during apprenticeship. In the context of practice standards, certification means that the system administrator has enough knowledge of the relevant standards to graduate from the apprenticeship and produce a compliant site.

Putting Lifecycle First

I propose, therefore, a profound change in strategy for the profession of system administration. I propose that we learn from other professionals and borrow some of their better ideas. I propose that we drop "personal standards" in favor of "professional standards" and strive for universal respect for those standards. I propose that we stop looking out for ourselves and start looking out for a profession that can take care of us as a collective group. Part of engendering that professionalism is a set of shared values that must trump the personal values of the past.

This is just part of the lifecycle of the profession. We learned in the early days that there were certain practices we use that distinguish us from amateurs. We progressed to define "best practices" as our first "standards" but realized that many of these are "personal." We progressed to understand the value of systems standards such as LSB. Now we are at a juncture where it is possible to move past personal professionalism to a definition of professionalism that is practice-wide. This requires giving up autonomy, very much as an electrician cannot function outside the bounds of the Code. What we gain, however, is something very precious, which is that the profession itself attains an enduring value that is—like a good standard—externally verifiable.

That is what "best practices" really means.

REFERENCES

[1] Alva L. Couch, "SLINK: Simple, Effective Filesystem Maintenance Abstractions for Community-Based Administration," *Proc. LISA X*, pp. 205–212, USENIX Association, 1996.

[2] Alva Couch, John Hart, Elizabeth G. Idhaw, and Dominic Kallas, "Seeking Closure in an Open World: A Behavioral Agent Approach to Configuration Management," *Proc. LISA XVII*, pp. 125–148, USENIX Association, 2003.

[3] Steven Schwartzberg and Alva Couch, "Experience Implementing an HTTP Service Closure," *Proc. LISA XVIII*, pp. 213–230, USENIX Association, 2004.

[4] The Linux Standard Base: http://www.linuxbase.org.

[5] Alva L. Couch, "Should the Root Prompt Require a Road Test?" *;login:*, Volume 32, Number 4 (August 2007).

DAVID N. BLANK-EDELMAN

practical Perl tools: Hi-ho the merry-o, debugging we will go

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 *Perl for System Administration*. He has spent the past 22+ 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.

dnb@ccs.neu

DEBUGGING CODE IS SUCH A NATURAL part of the software development process that it behooves us to do it as efficiently as possible. In my experience, many Perl programmers don't know about all of the many resources available that can make this process easier. We'll be taking a look at a grab bag of hints for debugging Perl.

Before we dive into the main subject, I feel compelled, some would say obsessively so, to mention that the most efficient way to get rid of bugs in your code is to avoid putting them there in the first place. Several of my previous columns have touted test-first and functional programming methodologies as two ways to work toward that goal, so I'll harp no more on them in this column.

Fun with the Perl Debugger

The Perl debugger is one of the best tools in your arsenal, so it is well worth your time to get to be best buddies with it. The output of `perldoc perldebug` and `perldoc perldebtut` plus the slim book *Perl Debugger Pocket Reference* by Richard Foley are required reading for this purpose. Here are a few tips surrounding the debugger that might not stand out for you on your first read through the material.

perl -de 0

Typing that command gives you a REPL (to use the computer-sciency term). A REPL is a Read-Eval-Print Loop. This basically provides a way to interactively type Perl code into a running Perl engine and receive the response back as fast as Perl can provide it. It can be very helpful to try out or hone small snippets of Perl code using this technique. As an aside, it should be noted that the REPL idea isn't even remotely new; several other languages (Python . . . cough . . . cough, etc.) even default to providing a REPL if you run their executable with no filename or other input specified.

$db::single

Another tip people often miss during a cursory read of the perldebug man page is the ability to write code that adds an automatic breakpoint for the Perl debugger when it is executed. If you set $DB::single to true (`$DB::single = 1` will do) and run the program under the debugger (`perl -d <filename>`), it will automatically stop at that

point in your program and wait for instructions. This is helpful if you know "here be dragons" at a certain point in your code. You can run the code until it hits this breakpoint and further scrutinize it from there. This is easier than having to search for that point in your program and set a manual breakpoint each time.

a [ln] cmd, w expr

These two debugger commands let the debugger do some work for you. For the first one, you can set an "action," which will fire when the specified line is reached in your program. That action is just a Perl expression. For example, if you type the following at the debugger prompt:

```
a 28 print "Contents of a flakey variable: $flakey\n"
```

the debugger will print that message, complete with the current value of some variable we might be concerned about (in this case, `$flakey`) every time it hits line 28 in the program.

If you don't know the specific line of the program that is messing with a variable you care about, you might want to know every time the contents of the variable changes. It is possible to set a global watch expression to look for these changes by using something like:

```
w $flakey
```

Now every time the contents of $flakey get modified, you'll see something like this in the debugger:

```
Watchpoint 0:    $flakey changed:
  old value:   ''
  new value:   '1'
```

x %something VS. x \%something

The x command dumps the contents of variables and entire data structures. You can ask it to dump a hash-based data structure by itself (e.g., `%something`), but for best results, give it a reference to that data structure (i.e., `x \%something`) instead. The output is much nicer. Here's an example:

```
DB<1> %s = ( 'fred' => 'protagonist', 'barney' => 'foil,' )
DB<2> x %s
0 'barney'
1 'foil'
2 'fred'
3 'protagonist'
DB<3> x \%s
0 HASH(0x3c24b0)
   'barney' => 'foil'
   'fred' => 'protagonist'
```

Finding Where You Are Using Devel:: Modules

There are a ton of modules to help the Perl programmer with the development process in the Devel:: namespace. Let me show you a few Devel:: modules that may be useful to you. The t command in the Perl debugger can show you a trace of what lines are being executed during your program's run, but a few Devel:: modules, such as Devel::Trace, Devel::Xray, and Devel::LineTrace, can improve on that basic idea.

Since we almost always call in another module with a use statement, you might have forgotten it is possible to bring another module to bear using the -d: switch. To use Devel::Trace, you would type perl -d:Trace {filename}. When you do this, you get something that resembled the output of the -x flag when using the (Bourne, etc.) shell. Here's the example output from the documentation:

```
>> ./test:4:   print "Statement 1 at line 4\n";
>> ./test:5:   print "Statement 2 at line 5\n";
>> ./test:6:   print "Call to sub x returns ", &x(), " at line 6.\n";
>> ./test:12:  print "In sub x at line 12.\n";
>> ./test:13:  return 13;
>> ./test:8:   exit 0;
```

The second module in our list, Devel::XRay, gets loaded in the conventional manner:

```
use Devel::XRay 'all'; # to trace everything, you can be more specific
```

The end result (again, an example from the docs) is something like this:

```
# Hi-res seconds          # package::sub
[1092265261.834574]   main::init
[1092265261.836732]   Example::Object::new
[1092265261.837563]   Example::Object::name
[1092265261.838245]   Example::Object::calc
[1092265261.839443]   main::cleanup
```

This shows which subroutines or methods are being executed, along with a hi-res counter so you can have some sense of how long the program spent in each part of your code.

Finally, Devel::LineTrace is a bit of a strange duck, because it requires you to have a separate configuration file (by default, perl-line-traces.txt) describing just how you'd like it to behave. That file contains a list of filenames with line numbers, along with code that should be run for each line specified. This is essentially the same as the a (action) command from the debugger I mentioned earlier but makes it easier to associate debugging code to specific lines in your program without having to actually add that code to the program in question or type it into the debugger.

Once you have a config file, you run it like Devel::Trace, that is:

```
perl -d:LineTrace {script filename}
```

Inspecting the Data

One time-tested method for debugging since the dawn of the modern computer age is Ye Olde Printf Methode. This is the technique whereby you attempt to understand the program's state, or at least the state of a particular variable in question, by liberally sprinkling printf or the nearest language-appropriate equivalent all over the code. It isn't particularly efficient but it still works for some debugging scenarios.

In fact, there are a number of Perl modules that I won't get into here that allow for more refined versions of that basic model. They allow you to put conditional debugging statements into or around your program that fire when in debug mode only. One particularly clever one, Devel::StealthDebug, places these constructs in comments within the program, thus keeping them from interfering with the program's logic.

The use of print statements or their equivalent in modules like these tends to be less helpful when one is dealing with more complex data structures (although some of the more complicated modules in the class of those I just mentioned can handle this as well). For instance, it is all well and good to be able to write:

```
print STDERR "fred: $fred\n";
```

when $fred is a scalar value, but if it is a reference to a different data structure, that command prints out something like this:

```
fred: HASH(0x919fec)
```

which is much less helpful. The standard way to show the full data structure is to load the Data::Dumper module and call its Dumper() routine:

```
use Data::Dumper;
my $fred = { bob => 1 };
print Dumper($fred);
```

This prints out:

```
$VAR1 = {
            'bob' => 1
          };
```

Data::Dumper ships with Perl, which makes it a good first choice, but many people don't know that it has some limitations. For example, it can't handle code references. The code:

```
use Data::Dumper;
my $sub = sub { print "Meet you in the yurt\n"};

print Dumper ($sub);
```

yields:

```
$VAR1 = sub { "DUMMY" };
```

Though it isn't included with Perl by default, Data::Dump::Streamer is well worth installing because it can often do a better job than Data::Dumper. For example, the first code example, when changed to this:

```
use Data::Dump::Streamer;
my $fred = { bob => 1 };
print Dump($fred);
```

prints something a little prettier and easier to understand:

```
$HASH1 = { bob => 1 };
```

The second example showing the code reference problem, when changed in a similar fashion, yields this output instead:

```
$CODE1 = sub {
             print "Meet you in the yurt\n";
         };
```

which is far more useful.

One final tip for dealing with complex data structures: If you are more of a visual person, you may find the modules that graph data structures mentioned in October 2007's column (GraphViz::Data::Grapher and GraphViz::Data::Structure) to be helpful in visualizing a more involved data structure.

Debugging Regular Expressions

One of Perl's strengths is its regular expression functionality. It is also an area that could use help when it comes to debugging. Regular expressions can be considered a language unto themselves (and come Perl 6 this will become even more apparent). The "code" we write in this language within a Perl program often itself requires some debugging.

There are a number of programs (both commercial and free) to help with this process. Let me mention two of the free ones. The command-line regex debugger rred (http://www.csn.ul.ie/~hannes/code/rred/) attempts to show you just how your regular expression matches against a given string. For example, let's match against the string "It was a dark and stormy night." We set that by entering the string with a `t` prepended:

```
rred> tIt was a dark and stormy night.
```

We can then specify a regular expression to match against it by using an `e` as the first character of the line:

```
rred> ea(n|r)
```

The output produced looks like this:

```
$text ="It was a dark and stormy night."; $text =~ m/a(n|r)/g;
  It was a dark and stormy night.
$&          ^^
$1            ^
  It was a dark and stormy night.
$&              ^^
$1                ^
```

rred echoes back the test text and the regexp (it adds the `_g_` flag by default to the regexp). After that, we can see that on the first pass `$&` will be set to the "a" and "r" in "dark" and `$1` is set to the "r" in that word. On the second pass, `$&` now points to "an" in "and" and `$1` now contains the "n" from "and".

The second tool I want to mention, and this is the last for the column, is the re_graph utility found at http://www.oualline.com/sw/. This utility makes it easy to spot some common mistakes. Here's an example of a mistake I find a fair number of beginners make when I first introduce them to regular expressions in the context of email header parsing:

```
/^From|To:/
```

The student who writes this is trying to find either the From: or the To: header but instead has constructed a regexp that will match "From" at the beginning of a line and "To:" any place in the text. The person probably meant:

```
/^(From|To):/
```

(As an aside, I'll mention that the person very likely would want to use a non-capturing indicator (`?:`), but I'll leave that out of this example for simplicity's sake.)

Finding this mistake becomes really easy when using re_graph. The first regular expression yields the graph in Figure 1.

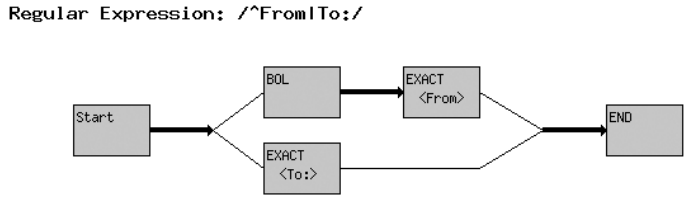


FIGURE 1: GRAPHING /^FROM|TO:/

The second regular expression produces the graph found in Figure 2.

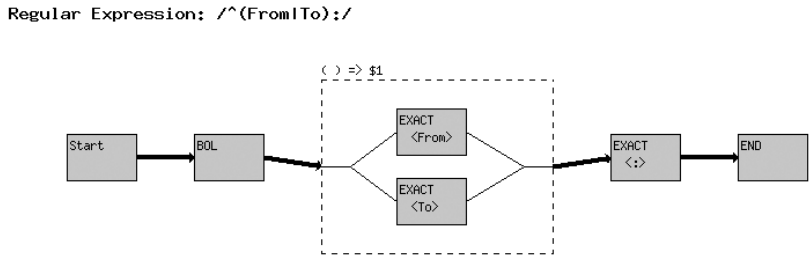


FIGURE 2: GRAPHING /^(FROM|TO):/

Even a cursory glance at the second graph makes it apparent that the second version has the "From" or "To" correctly preceded by a Beginning of Line (BOL). After either of the two headers is matched in an equal fashion a colon is required. This is definitely more correct than the first graph, which shows a BOL requirement only before the "From" match.

We have to bring things to a close now. Debugging is one of those surprisingly deep topics, so perhaps we'll revisit it in the future. Take care, and I'll see you next time.

PETER BAER GALVIN

Pete's all things Sun: Solaris System Analysis 101

Peter Baer Galvin is the Chief Technologist 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 coauthor 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.

pbg@cptech.com

AIRPLANE PILOTS MUST EXECUTE A pre-flight checklist before taking off. This list ensures that no steps are missed as the pilot prepares for flight. Over time, these checklists have been standardized and edited by many pilots and aircraft designers to the point that they are complete, logical, useful, and indispensable. Systems administrators are lacking such consensus documents, for the most part. Rather, some sysadmins have no useful checklists, doing all work ad hoc. Others have their own lists or work with groups that have documented methodologies that they follow. Frequently these lists have limited scope or assume site-specific details.

This month I'm unveiling one of my lists, "System Analysis 101." This list has been built over time via experience and trial-and-error. It is constantly expanding as new problems are encountered and solutions determined. Hopefully this list will grow into a consensus set of steps that both new and experienced sysadmins can use to save time and to avoid missing important aspects of a system that could be causing a problem. Input is welcome, and USENIX has set up a wiki to allow collaboration on this living, expanding checklist. Please visit it at http://wiki.sage.org/pbg and contribute.

This is the list that I used to diagnose "it's slow" or "it's not working right" kinds of problems. For more specific problems the list can be abbreviated, but carefully. Each of the entries comes from personal experience (or second-hand examples) in which that step wasn't taken and time was wasted. When a system or facility is not working right, time to resolution of the problem is of the essence, and counter-intuitively, this long list of steps can quickly lead to the diagnosis of the problem, the first step in getting it resolved.

This list may seem long and in some steps overly basic. When I approach a broken system I try to step through the list religiously, just as pilots execute their pre-flight checklist. Usually, lives are not at risk, but certainly time, and frequently money, gets lost when problems occur, and a systematic approach is the best way to resolving the issue. When I have been tempted into skipping steps, frequently I've regretted it as the steps skipped sometimes would have been useful in solving the problem.

Many of the steps here are general system-administration tasks that could be used on any system. Some of them are UNIX-specific, some are Solaris-specific, and some are Solaris 10–specific. If the problem is not occurring on a Solaris 10 system then other, equivalent (where possible) steps should be substituted.

Finally, the order of the steps need not be exactly as listed here, but the overall flow should be preserved, going from general to specific and from data gathering through making system changes.

Phase 0: Prepare for Problems

The time to learn how to use tools, to understand your facility architecture and performance, and to learn administration and debugging techniques is not when in the midst of a production problem. Rather, these things need to be part of your DNA, ready for when they are needed. To prepare for the inevitable problem, consider doing the following:

- Join helpful organizations, especially local user groups, to both learn and build your network.
- Take classes and tutorials (from organizations such as USENIX, of course).
- Read books on system administration and practice what they preach. My personal favorites include *UNIX System Administration Handbook* (latest version) by Nemeth et al., *The TCP/IP Guide* by Kozierok, *Solaris Performance and Tools* by McDougall et al., and *Solaris Internals,* 2nd ed., by McDougall and Mauro.
- Execute phase 4 when the facility is running normally (assuming there is such a thing) or at least at steady state. One of the easiest ways to determine problem details is to compare the state of the broken facility against the state when it was working better.
- Practice all of the other steps in this document to ensure that tools and documents are in place for when they are needed, that they work in your environment, and that you understand how to use them, what they do, and what their results mean.

Phase 1: Capture the Problem Definition as Succinctly as Possible

Capturing the problem definition as succinctly as possible helps keep focus on the problem and helps communicate the problem as needed. It also helps avoid the "death spiral," in which while exploring one problem other (potential) problems, or red herrings, are found.

Areas to capture include:

- When did the problem start?
- What invokes the problem?
- What avoids the problem?
- What is the problem, exactly?
- What changes were made before the problem started? (This is usually the key question!)
- What debugging/analyzing/testing changes have been done since the problem started? (Answering this can avoid wasting time and following those red herrings.)
- What existing diagnostics are available? These can include performance trends, performance monitoring tools, errors in log files, core dumps, angry users, and so on.

Phase 2: Phone a Friend

The timing of this phase is variable and generally occurs throughout the other phases. If more than one person is working on the problem, this phase can be delegated—it can be time-consuming.

- Place service calls on the problem components, including, for example, the application that is having a problem, the operating system software, and any hardware it is running on. Perform this early in the project if support contracts are in place and so the service calls are at no cost.
- Get in touch with whoever sold you the facility that is having the problem (for example, a reseller).
- Get in touch with anyone (if other than you) who was active in the implementation of the facility.
- Search for similar problems that other people have written about (even better, solved). General, technical, and product search engines are useful. For example, for a Solaris 10 on Sun hardware problem you could search at http://www.opensolaris.org/os/discussions/ and at http://sunsolve.sun.com/show.do?target=home. Looking for and exploring resources is a great thing to have done before the problem occurred, in all your "spare time."
- Get in touch with experienced and helpful sysadmins to whom you have been helpful in the past. (This is one of the many reasons to be helpful to other sysadmins.)

Phase 3: Determine Available Testing/Resolution Resources

- Are there any similar development, Q/A, or business continuance systems available? (Watch out that "similar" might be different enough that the problem cannot be reproduced there.)
- Can the problem be reproduced? If so, capture the steps to re-creation.
- Can we make changes in production? If so, capture the details (e.g., downtime windows, change limits such as validated system and production lockdown times and low-impact times).
- If the problem only occurs under load, do we have the ability to test under load and to generate a sufficient load to cause the problem? The worst problems are those that only occur under load and when the load cannot be generated artificially.
- What is the change deployment method and cycle in case changes need to be made to resolve the problem?
- Is testing in production possible?
- Is having an impact on performance in production acceptable?
- Is the use of unsupported tools in the production environment allowable?

Phase 4: Capture the State of the Problem Environment

For each component in the problem environment (certainly computers, but this could also extend to storage, networking, and security components), do the following:

- Capture the state and configuration details with the "best" tools available. For Solaris, that means running explorer (if possible), which is now part of the Services Tools Bundle and is available for free from http://www.sun.com/download/products.xml?id=47c7250a.
- Capture state with GUDS, which is a tool similar to explorer but more complete; unfortunately, apparently it is only available from Sun Support on an as-needed basis.

- If using Solaris 10, use dexplorer while the problem is occurring, if possible and allowed. dexplorer is part of the indispensable DTrace-Toolkit available from http://opensolaris.org/os/community/dtrace/dtracetoolkit/. Note that it is free and that the tools it uses are supported, but the toolkit and its scripts themselves are not supported.

Some additional things to capture if not already recorded by these actions include the following:

- What are the operating system, firmware, and hardware versions and patch levels?
- What are the pertinent application release and patch level levels?
- Are the versions of the applications supported on the versions of the hardware and operating systems? If not an upgrade cycle is probably going to be necessary to bring all of the components into compliance before support organizations will help resolve the problem.

Note that support organizations are likely to push for changes even in supported configurations before escalating a problem. For example, a very common scenario is that technical support will encourage or try to require installation of the latest patches, upgrading to the latest firmware, or even upgrading operating system or application versions. This eliminates variables for them, but it's also how they try to get you off the phone. Such work can take hours or days and frequently is not necessary—the problem frequently still exists after the work. Push back on support, depending on the level of effort and time their recommendations would take and your level of support. Ask support if they have any evidence that the problem will be solved by the changes recommended. If they have no proof, try to force them to continue working on the problem without first making their suggested changes. Note that politeness, firmness, and mention of how much you pay them for support is much more effective than poor behavior. If satisfaction is not received, then (politely) try to get to the next level of support management. Also, whoever sold you the facility has a vested interest in having you as a happy customer, so have them help increase the priority of the problem within the support organizations.

Phase 5: Track Down the Problem

Tracking the problem down is, of course, the meat of the project and the most difficult part. But with the preparation done as outlined here, many variables have been eliminated and plenty of information is now readily available to help diagnose the problem. This phase can vary immensely depending on the kind of problem being worked on, the scale of the problem, and all of the details determined in the previous phases. Some first steps for Solaris 10 systems are listed here, but other lists for other operating systems and devices should be compiled for your site (or be found to have already been published).

Generally, compare the results attained during the problem against the same information from when the system was healthy (see Phase 0).

- Scan through log files such as /var/adm/messages and via dmesg. Don't ignore anything odd—it could be the canary indicating the problem.
- Run `svcs -a` to check for services that have failed or are disabled.
- Check for full disks or changed mount information via `df` and `mount`.
- Run `ifconfig -a` and look for any errors; run `kstat` and grep through the output for the network interface names (such as e1000g0) to check network parameters such as duplex and speed.

Read through /etc/system and look for settings copied from other systems or left behind during an upgrade. Note that /etc/system should never be copied or left intact between operating system or application upgrades; such events should cause an audit of the file for entries to remove or update. Check the *Solaris Tunable Parameters Reference Manual* at http://docs.sun.com/app/docs/doc/817-0404. This document is updated for every Solaris release.

- Check /etc/projects for any resource management settings that could be affecting system or application performance.
- Check the `stat` commands and look for anomalies:
  - `iostat -x 10`—are there large response times?
  - `mpstat 10`—how many threads are in which states?
  - `vmstat 10`—do the thread counts and scan rate indicate memory shortage?
  - `vmstat -p 10`—look for systemwide memory operations.
  - `prstat`—are there any resource hogs?
  - `prstat -Lmp <pid>`—look at detailed state information about a specific process.
  - `pmap -x <pid>`—explore the memory map of a problem process.
  - DTraceToolKit and DTrace scripts—look at specific suspect aspects of the system.

Some general areas to consider, especially on Solaris:

- Are you running the most appropriate scheduler for each system in your environment?
- Are you using the best-fit page sizes?
- Is your I/O well balanced and spread across enough devices (disks, network ports, etc.)?
- Are you using the best CPU for the workload? (Are there few fast threads or many slower threads?)
- If processes are contending with each other, implement resource management (e.g., containers, project sets, and dynamic resource pools).
- Since rebooting the system can reset the state to a known starting point and remove variables, consider it as appropriate after current state information is captured.
- Finally, if the code is your own, did you use the best current compiler to generate the machine code?

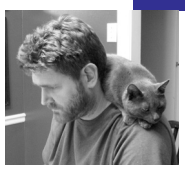
Next Time

My list might be a bit controversial. Don't like it? Have a better one? Have a checklist for some other sysadmin activity? System administration is a difficult activity because not much of it can be learned in school or from books. It's a journeyman's trade. Help your fellow sysadmins and help make the world a better place by documenting your accomplishments and making them available. Contribute to the SAGE Web site, create your own blog, contribute to the wiki of this column at http://wiki.sage.org/pbg, join (and attend!) a user group, and get sharing!

Next month, in Solaris Systems Analysis 102, I'll dive deeper into Phase 5, showing DTrace and system command examples and how to tune areas such as the scheduler class and resource management. Until then, there should be plenty here to add to the sysadmin to-do list.

DAVE JOSEPHSEN

iVoyeur: hold the pixels



David Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and 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 coauthored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

"CHECK THIS OUT," HE SAYS WITH THE merest hint of pride in his voice. As admins go, he's one of the best I've had the pleasure of working with—clever, thorough, and tenacious, so when he has something to show off, it's usually pretty impressive. This particular hack happens to be a billing program based on customer bandwidth usage. As you can probably imagine, many things are involved: SNMP, data stores, and lots of arithmetic. His data structures are beautiful, and his design is modular and elegant, but there is a rather large problem, and it has nothing to do with his code. While I page through the file my subconscious nags me, "Let's see, you're getting data from routers, you're storing it in a database, you're performing math on it . . . yep, complete reinvention of the wheel." So after nodding appreciatively, and knowing the answer before I ask the question, I ask, "Why didn't you use RRDtool?"

"I didn't need to draw graphs," he replies. I sigh inwardly. I've seen this one before; heck, I've even been guilty of it myself. One of the traps good admins often fall into is under-scoping our tools. Given tools that do one thing well, we sometimes miss what that thing actually is, and "graphing stuff" certainly isn't RRDtool's one thing (that'd be gnuplot). What RRDtool [1] excels at is dealing with time-series data, whether you need pretty pictures or not.

In fact, just about all the graph-related options to RRDtool's graph command are optional, and this is by design. In practice RRDtool is a great tool from the command line, and in fact my friend's entire billing program sans the data collection portions could have been done using nothing but RRDtool in command-line mode. I know all of the gripes: Reverse polish notation is . . . well, weird, RRDtool's command-line syntax can't be committed to memory, the data storage aspects aren't very transparent, and on and on. I submit that RRDtool is still the best tool available for dealing with any sort of time-series data in general.

Let me show you what I mean. Since my friend put his data in a mysql database, he has to pro-

grammatically extract it, calculate usage information for the given time interval, and compute the cost of that bandwidth. If he had stored it in an RRDtool database (using a gauge RRA, or a compute RRA to compute and store raw byte counts) all of this is (arguably) four lines of code:

```
rrdtool graph foo.png --start='now -1month' \
DEF:a=routerA_customerinterface4.rrd:traffic_counter:MIN \
DEF:b=routerA_customerinterface4.rrd:traffic_counter:MAX \
CDEF:price=a,b,-,1024,/,.05,* \
PRINT:price:%8.2lf | tail -n1
```

It's not my aim in this article to teach you RRDtool syntax, so I'm mostly going to let these examples speak for themselves. Suffice to say that in the short example above, we extracted the data we needed, and used it to calculate the price of a customer's bandwidth usage over the last month, assuming a price of five cents a meg. I will point out the use of the print statement, which simply prints the answer to stdout, suitable for piping to other applications such as tail. PRINT makes RRDtool into a command-line app. Although we specified the name of an output png (this, for reasons unknown to me, is required), we specified no graph-related options, so RRDtool didn't draw one; its only output was the cost of customer4's bandwidth for the last month, in text, to STDOUT.

This implementation has obvious advantages over rolling your own solution in Perl, not the least of which is speed. RRDtool internally performs math really quickly. Using "time" to measure the execution time of this problem in RRDtool yields "real 0m0.004s" versus my friend's Perl solution at "real 0m0.103s." Your results will surely vary, but I've yet to find a faster way to process heaps of time-series data. RRDtool also scales really well. An aversion to RPN is no excuse; you can spend 30 minutes learning the syntax or an hour writing your own code to perform the math, and your code will almost certainly be slower.

RRDtool takes care of data compression and rotation, temporal interpolation, data glitches, conversions of counters to rate information, and I/O access, without you having to do anything. It does this remarkably efficiently and on tons of data. It is as efficient when processing a month of data starting five years ago as it is processing a month of data ending five minutes ago. To write an application that handled your particular time-series data more effectively you'd probably need to start by forking RRDtool.

Good admins might suspect that a tool this efficient was over-optimized for the solution set and couldn't possibly be very flexible. I might agree, but I'd also wager they hadn't played with some of RRDtools more esoteric features such as CDEF conditionals. For example, my friend's billing program doesn't actually bill the customer unless the customer uses more than 100 Mb of bandwidth. We can use CDEF conditionals to solve this problem:

```
CDEF:bandwidth=a,b,-,1024,/ \
CDEF:price=a,b,-,1024,/,100,-,.05,* \
CDEF:finalprice=bandwith,100,GT,0,price,IF \
```

In pseudo-code this comes out to "if (bandwidth > 100) then finalprice=price else finalprice=0." This doesn't actually require three separate CDEFs; it could have been accomplished with a single long, convoluted RPN expression if you, for example, hated those who will come after you and wanted to give them something painful to decrypt. The point is that RRDtool can do a surprising amount of the work you need done in-

ternally, and without involving any pixels. So if you ever find yourself writing code to process time-series data or find yourself using RRDtool fetch to export data so that you can process it in a general-purpose language, you're probably doing it wrong. I'm just saying.

Aberrant Behavior Detection

And while I'm at it, I should add that if you're exporting data so you can run statistical analysis on it or send alarms based on thresholds, you're probably doing it wrong here as well. A few years ago, Jake Brutlag added some really cool statistical forecasting and problem-detection software to RRDtool. He published his work at LISA 2000 [2]. His work, collectively referred to as "Aberrant Behavior Detection," is the coolest code I'm aware of that nobody *ever* uses. Nobody seems to use it mostly because everyone thinks "it's hard"; yes, what little documentation there is, is not fun to read. So allow me to take a crack at putting this down where the goats can reach it (to quote an old boss of mine).

The problem with predicting the future is that a lot of noise gets in the way. One of the oldest, easiest methods we have to predict future data points within a set of time-series data is the simple moving average. Every time we get a new data point we re-average the data set, and the next point should land somewhere near that. Maybe we drop values that are more than *x* days old. This obviously isn't very accurate, and it suffers from a number of other problems as well. The two big ones are that it's very compute-intensive and we have to keep a lot of data points around (especially if we're polling every few minutes).

In the late 1950s, C.C. Holt proposed a formula that was more accurate and only needed two data points to compute. It's pretty clever and even came with a constant that you can tweak to give more recent values greater weight. In fact, giving more recent values greater weight in general was sort of a design goal of the equation, and it's where it derives its greater accuracy. This first equation is called "exponential smoothing." Within the Holt-Winters Method it's also referred to as the "baseline" equation. As your data set changes, the equations' predictions do so as well. If your data rises rapidly, it will outpace the predictions and your data may therefore be considered statistically improbable. The constant controls how quickly the data can change before it is considered abnormal. I'm not going to get into the specific math here because of the aforementioned goats. If you want to get a look at the equations, take a look at Brutlag's paper [2] and/or his implementation notes [3].

The only problem with exponential smoothing was that it didn't account for longer-term trends in the data set. This is of course what you would expect when you design a formula that gives greater weight to more recent values, so a few years later, Holt proposed another formula to account for longer term trends in the data. This formula, called "Double Exponential Smoothing," can basically be thought of as exponential smoothing of exponential smoothing. This formula also uses a tweakable constant to allow fine-tuning for the extent to which data trends affect the final prediction. This equation is sometimes referred to as the "slope" equation in the Holt-Winters Method.

The final piece of noise that Holt's equations didn't account for was periodic fluctuations (for example, people shopping more at Christmas or load from machines being backed up every night at 2 a.m.). In 1965 Winters proposed an equation to absorb the periodic (or "seasonal") noise.

This final equation uses yet another constant to control the weight of periodic data in the final prediction. The three equations combine to form the Holt-Winters Method, which can make pretty accurate predictions in sets of time-series data, or at least predictions that are good enough for our purposes.

Jake Brutlag took Holt-Winters and bolted it onto RRDtool, giving RRDtool the power to make real-time predictions for any given data set. He also added an easy mechanism for graphing error bars around the predicted data points, and for determining when a real data point or range of data points falls outside the error bars. All of this is implemented in a series of special-purpose RRAs, but in practice you don't need to know much about the inner workings of Holt-Winters or any of the special RRAs save one to get this working for you. All you need to do is create the HWPREDICT RRA and the others are implicitly created for you. The syntax looks like this:

```
RRA:HWPREDICT:<array length>:<alpha>:<beta>:<period>
```

The array length is the number of predictions you want to keep. This number needs to be as many data points as you want to graph error bars for, or at least as many data points as the seasonal period you want to account for. The seasonal period is usually 24 hours, so for an RRD with a 5-minute polling interval, the array length should be at least 288, but again, if you plan on drawing graphs with error bars, you probably want this to be a good deal bigger (10,080 gives you a week of error bars).

Alpha is the exponential smoothing constant I mentioned above. It is a number between 0 and 1. You can calculate this given a small judgment call on your part. The formula (taken from Brutlag's paper and translated to Perl syntax by me) for calculating the value is:

```
$alpha = 1-(exp(log(1-$pct)/$points))
```

The question you have to ask yourself is how quickly you want the baseline forecasts to adapt to changes in the data. If for example, you wanted observations in the last 30 minutes to account for 90% of the baseline weight, you would plug in .90 for $pct, and 6 for $points (assuming a 5-minute polling interval (30/5)). This yields an alpha of about .32, which, in my experience is a pretty reasonable starting point.

The same formula can be used to calculate beta, which is the slope (or trending) equation constant. Since the idea of the slope equation is to account for long-term trends, you should use a value for $points that is at least as long as the seasonal period. The seasonal period is most often 24 hours, so for a polling interval of 5 minutes $points should be at least 288, if not higher. A starting value of .0024 for beta will ensure that observations in the last day will account for no more than 50% of the slope equation smoothing weight. This is the same value Brutlag mentions in his paper and it's the one I usually start out with.

Finally, "period" is simply the periodic or seasonal period you want to account for. As I've mentioned this is usually 24 hours (288 points for a 5-minute polling interval).

If you want to get fancy, there's a whole lot more tweaking you can do, but for most admins wanting a fairly good and scalable method for detecting abnormal patterns in their time-series data, that's pretty much all there is to it.

Once the RRD with HWPREDICT in it is created, you can ask RRDtool to give you the instances of aberrant behavior for a given time interval from

the command line via the implicitly created FAILURES RRA. This is what most admins are looking for when they create Nagios plug-ins to dump and search through RRD data. For example, if I wanted to know the number of times customer4's interface displayed abnormal traffic patterns in the last 5 minutes, I could do this:

```
rrdtool graph foo.png --start='now -5minutes' \
DEF:failures=routerA_customerinterface4.rrd:traffic_counter:FAILURES \
PRINT:failures:%8lf | tail -n1
```

Anyway, I hope I've made a good case for the wholesale abandonment of the use of <insert your goto language here> for processing time-series data. RRDtool really is a heck of a piece of software, so if you thought it was all about graphing I urge you to take another look.

Take it easy.

REFERENCES

[1] http://oss.oetiker.ch/rrdtool/.

[2] http://www.usenix.org/events/lisa00/full_papers/brutlag/brutlag_html/index.html.

[3] http://cricket.sourceforge.net/aberrant/rrd_hw.htm.

HEISON CHAK

Media Resource Control Protocol

Heison Chak is a system and network administrator at SOMA Networks. He focuses on network management and performance analysis of data and voice networks. Heison has been an active member of the Asterisk community since 2003.

heison@chak.ca

WITH TODAY'S TELEPHONY SERVERS and applications, speech capability and intelligence have been key differentiators between a superb Interactive Voice Response (IVR) system and its ordinary counterparts. Text to speech (TTS) and Automatic Speech Recognition (ASR) are means of assisting self-service IVR systems in directing calls to appropriate destinations via spoken speech. This is the most natural form of input and becomes especially convenient when callers do not have easy access to a keypad for DTMF input. Voice applications written in VoiceXML (VXML), such as address/ZIP code recognition, email collection, or alpha-numeric input, may ease the pain of performing these tedious and difficult tasks, achieving higher automation with improved accuracy. In this column we will examine how it is that clients (e.g. ,the application server, the PBX) make text requests to speech servers and get spoken speech in return.

Synthesizer/Recognizer, Then and Now

Text-to-speech (TTS) and automatic speech recognition (ASR) technologies have been around for quite some time. The first computer-based speech synthesis systems were created in the late 1950s, and the first complete text-to-speech system was built in 1968.

Research on ASR began in 1936, but the problems of speed and accuracy were not overcome until 1982, when Covox launched the first commercial product. Another company founded in the speech recognition market the same year was Dragon Systems. Dragon Systems was acquired by Scansoft Inc. in 1999. A company that had grown through acquisition, Scansoft acquired Speechworks in 2003, to become an enterprise speech solutions company, a direct competitor for Nuance, which had been established in 1994 and was by 2003 a leader in computer telephony applications and automated call steering. In 2005, a de facto acquisition of Nuance by ScanSoft took place and the combined company changed its name to Nuance. (See Figure 1.)

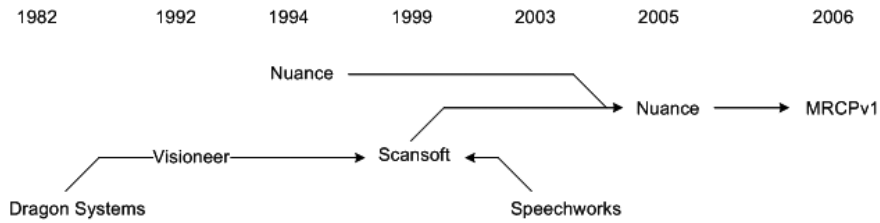


FIGURE 1: HISTORY OF MRCP AND NUANCE

RFC 4463: MRCPv1

While vendors (e.g., Nuance, IBM WebSphere Voice Server, AT&T Natural Voices) are competing for market share of TTS and ASR products, there are many proprietary systems for performing synthesis and recognition over the network. However, these nonstandard approaches have often caused migration problems and become major support issues as a result of vendor mergers and acquisitions. In 2006, Cisco, Nuance, and Speechworks jointly developed RFC 4463: Media Resource Control Protocol (MRCP), a protocol which controls media service resources such as speech synthesizers and recognizers over a network. MRCPv1 is an Informational RFC and therefore not a candidate to become an IETF Internet Standard (it was published for its historical value as an ancestor to the MRCPv2 standard). That said, MRCPv1 has been widely implemented by both speech server vendors and network equipment providers and is often viewed as a subset of MRCPv2.

A fundamental difference between MRCPv1 and MRCPv2 is seen in the mechanisms used for media session management and protocol transport. With MRCPv1, Real Time Streaming Protocol (RTSP) is used for session setup and RTP is used for media streaming. MRCPv2, in contrast, uses SIP instead of RTSP for session setup (RTP is still used for media streaming).

Consider the MRCPv2 architecture diagram (Figure 2) and the MRCPv2 message (Listing 1). When speech is sent from the client to the speech server via a SIP connection, the speech server will synthesize the speech using an installed voice (here, "Samantha") and return the speech ("Hello, my name is Samantha . . .") over an RTP stream negotiated during the SIP call setup phase.

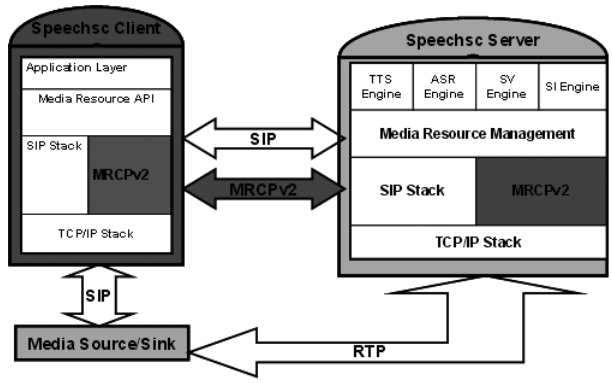


FIGURE 2: MRCPV2 ARCHITECTURE

```
MRCP/2.0 732 SPEAK 543257
Channel-Identifier:32AECB23433802@speechsynth
Voice-gender:neutral
Voice-Age:25
Voice-Name: Samantha
Prosody-volume:medium
```

```
Content-Type:application/ssml+xml
Content-Length:850
<?xml version="1.0"?>
<speak version="1.0"
  xmlns="http://www.w3.org/2001/10/synthesis"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
     http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
  xml:lang="en-US">
  <p>
  <s>Hello, my name is Samantha. The time is now
     <break/>
     <say-as type="time">0345p</say-as>.
  </s>
  </p>
</speak>
```

LISTING 1: CONTENTS OF MRCPV2 DEMO _ SPEECH.MSG

Voice- (e.g., Voice-gender, Voice-Name) parameters may be sent to define values for the entire session. These parameters may also be sent to affect a request in progress and change its behavior (from a woman's voice to a man's voice) on the fly, if that functionality is supported by the synthesizer.

OpenMRCP

Despite the lack of free commercial-grade speech servers, the open source community is striving for significant improvement over what is currently available (e.g., Festival, GVoice). With MRCP-enabled speech servers (e.g., Nuance, IBM WVS), the open source project OpenMRCP becomes a perfect tool to bridge the gap. Sponsored by Cepstral, a TTS voice provider, OpenMRCP provides a full stack of MRCP which can be used as the basis of either an MRCP server or an MRCP client. The MRCP client is available in Freeswitch as a module. It may also be built against a stand-alone Apache Portable Runtime (APR).

OpenMRCP Against Stand-Alone APR

To test OpenMRCP, I built three virtual machines under Xen 3.0.3:

- Nuance RealSpeak 4.5 with Nuance Speech Server 5.0 on CentOS 5
- Nuance Recognizer 9.0 with Nuance Speech Server 5.0 on CentOS 5
- OpenMRCP with apr, apr-util 1.2.12, and sofia-sip 1.12.8 on Debian Etch

In order to build OpenMRCP against a stand-alone APR (Apache Portable Runtime), several dependencies must be met. First, I need apr and apr-util version 1.2, which are available from the Apache Portable Runtime Project (http://apr.apache.org). For OpenMRCP to support MRCPv2, a SIP stack is required. This is where the Sofia-SIP library fits in. Sofia-SIP (http://sofia-sip.sourceforge.net) is used by OpenMRCP to set up MRCPv2 sessions.

Once all the dependencies have been built, OpenMRCP can be downloaded and built:

```
# cd /usr/src
# svn co http://svn.openmrcp.org/svn/openmrcp/trunk openmrcp
# cd /usr/src/openmrcp
# ./bootstrap
```

```
# ./configure     --with-apr=/usr/src/apr-1.2.12 \
          --with-apr-util=/usr/src/apr-util-1.2.12 \
          --with-sofia-sip=/usr/src/sofia-sip-1.12.8
# make && make install
```

To test the Nuance TTS synthesizer using the MRCP protocol, I launched OpenMRCP in client mode. From the OpenMRCP CLI, new sessions can be created, followed by allocation of TTS/ASR channels where MRCPv1/v2 commands can be sent. For example:

```
# openmrcpclient -c 10.155.1.29:5060 -s 10.155.1.27:5060
# Connect to server with a new session
> create 0
# Create a TTS channel (1st 0 - session slot, 2nd 0 - channel type of TTS)
> add 0 0
# Send MRCPv2 message (as in Listing 1)
> msg 0 /path_to/demo_speech.msg
# Tearing down session (when the synthesis completes)
> destroy 0
# Leaving OpenMRCP
> quit
```

For the duration of the speech, an output file, synth_result_${session}.pcm, will grow in size as RTP packets arrive from the speech server. To verify that it worked, the raw PCM can be converted to .wav for playback:

```
# sox -t raw -r 8000 -c 1 -w -s synth_result_0.pcm demo_speech.wav
```

Next Steps

The newly resampled .wav file can now be played by most media players and is ready for use in Asterisk. It may be used as a pre-recorded IVR menu file or greetings for voice-mail boxes.

Ideally, it would be best if real-time operation of MRCP could be integrated into Asterisk. Instead of hacking up an expect script or Python to maneuver the openmrcpclient CLI and passing the output file to Asterisk for playback, many would like to see OpenMRCP as an Asterisk module, as it is in FreeSwitch. Unfortunately, the author is currently tied up with a reimplementation of MRCP that will be free from corporate sponsorship, so integration of OpenMRCP into Asterisk may not happen anytime soon.

The new project, UniMRCP, the successor to OpenMRCP, will probably continue to depend on Sofia for SIP stack as well as APR (apr.apache.org) for low-level, cross-platform implementation of threads, mutexes, and sync objects. UniMRCP is currently available for Windows and UNIX via Subversion.

REFERENCES

http://en.wikipedia.org/wiki/Speech_synthesis

http://www.dragon-medical-transcription.com/historyspeechrecognition.html

http://tools.ietf.org/html/rfc4463

http://daveburke.org/downloads/SP-Appendix-A.pdf

http://wiki.freeswitch.org/wiki/OpenMRCP#Configure_OpenMRCP_against_standalone_APR

http://www.unimrcp.org/

http://www.cisco.com

ROBERT G. FERRELL

/dev/random

Robert G. Ferrell is an information security geek biding his time until that genius grant finally comes through.

rgferrell@gmail.com

SEVERAL YEARS AGO, IN A FIT OF WHAT might best be described as literary bulimia, I disgorged a speculative fiction novel titled *Tangent*, that appellation being more a nod to my writing style than indicative of any deeply salient plot point foreshadowing. It was not a particularly good novel, even from my own healthy ego's perspective, and it was received, when received at all, with more negative comments than positive. Part of the problem, I think, is that it's really only the first part of a trilogy. I did put a fair amount of effort into creating a volume that would stand alone, but knowing the entirety of the plot makes it difficult, especially for a first-time novelist, to make sound decisions about how to slip in pointers to future plot lines while keeping those teasers ancillary to the conflict and resolution of the current story. One of the wholly unsolicited critiques a kind soul felt compelled to make, in a public forum, on reading *Tangent* was that it appeared to have been constructed using a series of ideas discarded by Neal Stephenson during the writing of *Cryptonomicon*. Now, despite the fact that I am a Stephenson fan, I had always been intimidated by the sheer bulk of that particular tome and so had not read it at the time my own foray into the commercial fiction world took place; thus this critical observation remains itself forever in the speculative fictional realm.

I now count myself among those lucky, if hardy, souls who have undertaken the odyssey presented by this monumental work (which I loved, by the way), and therefore I finally have a cogent response to the aforementioned accusation:

Huh?

First and foremost, encryption has nothing to do with *Tangent*. It's about quantum metaphysics, hacking, and duct tape. It reveals my staggering lack of knowledge concerning geography and airports in Great Britain and a narrative style that many might feel could well do with a good sound thrashing, but the relation it exhibits (at least as far

as I can see) to anything covered, or to be accurate intentionally *not* covered, in *Cryptonomicon* could not be detected with even the most sensitive measurement device. Well, there are references to computers in both books.

Speaking of encryption, which of course is what prompted this little self-aggrandizing rant in the first place, I'm currently sitting on an MD-80 hurtling willy-nilly six miles above the Gulf of Mexico. In places like this a lot of the cultural fluff that ordinarily occupies my intellectual field of view is relegated to the musty stacks in the basement of my subconscious, there to fester like last week's discarded shrimp scampi. The fragrant vacuum this leaves is gradually filled by more esoteric cortical activities such as idle ruminations on the nature of cryptography.

Lest you fear I am now going to inundate you with elliptic curve arcana, be comforted by the fact that I don't understand that stuff very well. I can appreciate the elegance of the math the way a patron of the arts appreciates Van Gogh's energetic brush work, but trying it myself more often than not leads to numerical mayhem. In fact, my attempts at writing crypto algorithms have inadvertently created a whole new branch of mathematics I call "hermit theory" because they never go anywhere.

Cryptography is crazy stuff. A large number of very smart people have given a great deal of thought to creating ever more exotic processes for obfuscating human language. This wouldn't be quite so crazy except for the fact that human language is already about the most obfuscated means of communication imaginable. I think we could get some mileage out of melding the sciences of cryptology and molecular biology, though. Using DNA sequences for keys is hardly an original thought (although I did first think of it when I was in college in the 70s, so maybe it *is* original), but so far I don't see anyone doing it. Get with the program, people. While you're at it, where's my #@&! hoverboard?

One of the distinct inconveniences of encrypting communiqués is that for the result to be read by anyone, the intended recipient needs to have the same key (and complementary algorithm) used to encode them. This problem has been addressed in several novel and extremely clever ways, most notably PKI, with constantly increasing key lengths being one of the primary means of ratcheting up security as processing power continues its own inexorable advance.

I think it's time to think outside the packing crate here and come up with a new paradigm (I adore that word) using a radical form of key mechanism. Since we started off talking about Neal Stephenson, I'll borrow the kernel of an idea from him—for real this time—and suggest that we use subliminal memes based on person-specific cortical activity patterns as encoding keys. If you scan the message without the proper meme in place, it reads as gibberish. Since the interaction between a given person's neural cortex and the implanted meme will be unique to that individual, the odds that a computer could be used to replicate said key are absurdly remote, no matter the processing power involved. It wouldn't be a purely mathematical problem, for one thing, and computers don't yet deal well with the abstractions routinely taken in stride by the human brain.

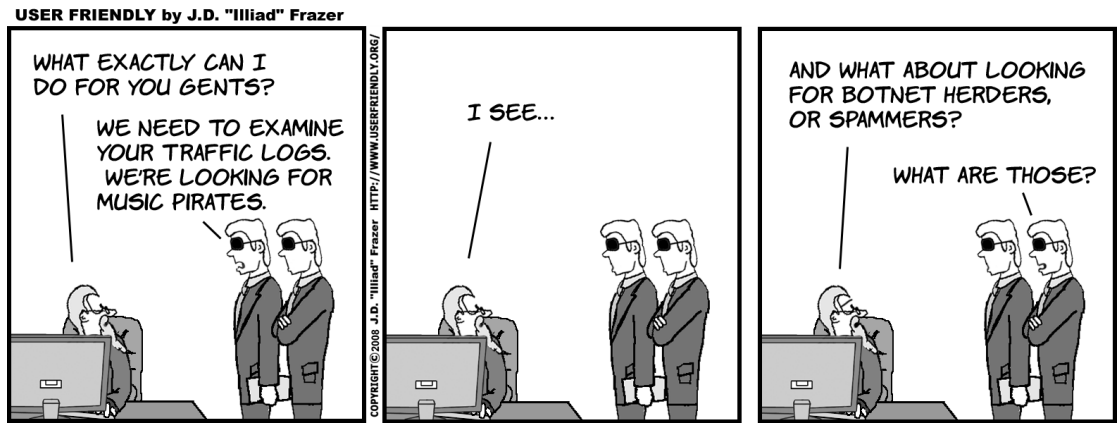
Implantation of the meme could take place using a device similar to the mind-erasing flashgun from *Men in Black*. The meme itself could be protected by a single-use safeguard, such that the implantation device will work once and only once, and would require a PIN or similar authentication to be activated. Any attempt to modify the device would render it permanently inoperable, like most of my cell phones. Thus, we have all the advantages of quantum cryptography without any of the logical contradictions.

Once the meme has been successfully implanted it is permanent, failing serious localized brain injury to the subject. The subject becomes, in effect, a living encryption key, able to decode encrypted messages merely by retyping them, as the neural processes involved in that mechanical act serve as the decryption mechanism.

If this sounds like far-fetched gobbledygook, consider quantum cryptography, the development of which is already well underway. In classical binary systems, the basic unit of information can have a value of 1 or 0. In a quantum system, the basic unit of information can have the value 1, 0, or *both*. That's what I call gobbledygook.

I've observed that the behavior of political candidates exhibits facets of all three systems: classical, relativistic, and quantum. The position a candidate takes on any given proposition is dependent on the observer (relativistic) and can be for, against, or both (quantum). Presidential candidates must belong to one of the two major parties in order to win (classical). American presidential politics, therefore, may well be the long-sought unifying theory of everything. The universe is composed neither of vibrating strings nor of oscillating toroids, but of annoying sound bites. Perhaps that should be sound *bytes*, given the pervasive nature of streaming audio/video.

Explains a lot, doesn't it? Do I really have to squeeze into a tuxedo to pick up the Nobel Prize, or will the dark suit I wear to weddings and funerals do?



book reviews

ELIZABETH ZWICKY, WITH SAM STOVER AND RIK FARROW

MAKING THINGS HAPPEN: MASTERING PROJECT MANAGEMENT

Scott Berkun

O'Reilly, 2008. 371 pages.
ISBN 978-0-596-51771-7

THE PRINCIPLES OF PROJECT MANAGEMENT

Meri Williams

Sitepoint, 2008. 192 pages.
ISBN 978-0-9802858-6-4

These are both good, accessible books on project management, written from a realistic point of view. They are about the nuts and bolts of managing real projects, not about theories. They're also very different books. If you are just learning to be a project manager, you could very happily buy both.

Making Things Happen is a big book. It's aimed at technical people who are going into project management without a lot of management experience, and it explicitly and carefully deals with the emotional and relational issues involved, as well as with the processes you need to design, plan, and manage. Most project management books carefully acknowledge person-to-person issues, declare them out of scope, and move on. That's a perfectly reasonable approach, but it's of limited effectiveness to somebody who doesn't understand exactly what those issues are. Sure, they could look in "Further References," but are they going to? *Making Things Happen* assumes that the human issues are the crux of the problem, and it tackles them straight on. Partly because of that, and partly because of the smart-aleck, eclectic voice, I love it a lot. It would be my first choice to shove into the hands of most struggling new project managers.

Which is sort of unfair, because I also like *The Principles of Project Management* a lot; there's a whole category of people who would be better off with it. *Making Things Happen* is great for somebody who likes reading, but *The Principles of Project Management* is a clear, no-nonsense book in big, friendly letters, suitable for people who want the largest amount of project management wisdom in the smallest number of words. It covers the bases solidly and realistically. It is also much closer to the system you would need for a PMP (Project Management Professional) certificate, and so it is a good place to start if you need a fast and purely practical way to get a handle on the terms and concepts used by PMP project managers. If you're OK with management, but you need some help with the whole "project" thing, *Principles* is a faster, more focused book.

HACKERTEEN: INTERNET BLACKOUT

Marcelo Marques and the Hackerteen Team

O'Reilly, 2008. 100 pages.
ISBN 978-0-596-51647-5

This is a graphic novel about computer security aimed at teenagers. My household doesn't contain any teenagers, or even any pre-teens (unless your definition of "pre-teen" includes anybody who has yet to become a teenager). But we all love graphic novels and two of the three of us (the two who are past pre-school) are interested in computer security. So we were ready to love it.

Unfortunately, we didn't. The plot is too adult for my daughter (it's about money and computers; there are no dogs, and very little that she perceives as action). Besides, it was frequently interrupted by cries of outrage from her father. Everybody but me gave up by page 30. Strike one for my husband was the kid who mystically knows how to program despite doing nothing on the computer but playing games. (I personally learned to use a debugger entirely to cheat at computer games—let me note that in my social circle that was considered part of the game—but that doesn't seem to be at play here.) Strike two was the idea that his parents would hand him over to a shadowy hacker organization because they'd heard of it on TV, and that's supposed to be a good thing. Strike three was the moment at which the kid tells the cute girl that the webcam she wants to buy is no good—because it doesn't have a Linux driver. "That isn't useful advice! That's knee-jerk prejudice! What's its low-light performance? How many pixels is it? Now *that's* technical! Who cares if it has a driver for an operating system she isn't running!"

There's a lot of knee-jerk prejudice involved. The politicians are evil, the hackers are good, the bad guys are ugly. It's more of a comic book than a graphic novel. The computer security is comic book computer security, the equivalent of the booms and swooping punches that make up completely unrealistic fights in superhero comics. It may well appeal to teenagers, but I'm not sure I want more of those teenagers in computer security; there seems to be an oversupply of dramatic teenagers prone to see everything as a fight between Evil and Good to start with.

The book does provide a reasonably interesting moral story in the comic book mode that involves computers and computer security. It leans, sometimes preachily and sometimes more subtly, on the idea that everybody can do this (it literally says "every sex, race, and ethnic group" at one point). It makes a passionate case for the importance of open source. What it doesn't do is what it says on the back cover; you don't learn how Internet technologies work, how to protect yourself, or how people work together on the Internet, and while they do show people trying to hurt each other online, the online attacks that are shown are not particularly realistic. I must admit that the authors are not the only people who seem to believe that using a webcam to take nude video and demanding blackmail for it is a realistic threat for everybody; the same spammers who send me advertisements promising to increase my nonexistent manhood to the size of the Statue of Liberty also send me fake blackmail notes claiming to have done this.

On the whole, it's a perfectly OK teen comic being sold as something deeper and more interesting. If you're going to use it to try to hook your younger associates, try it out on them first; my daughter would rather read *Head First Object-Oriented Design*. (No, I'm not joking. Yes, she is 4. Yes, she is advanced. No, not that advanced. It has dogs in the pictures.)

VISUAL COMMUNICATION IN DIGITAL DESIGN

Ji Young Park

YoungJin.com, 2008. 217 pages.
ISBN 978-89-314-3434-7

Alas, this is another book I wanted to like and didn't. I want to be able to recommend a good, straightforward introduction to the principles of visual design, something up-to-date and computer-oriented but capable of getting across the importance of using white space, some basic facts about colors, and the primary tricks of doing a good layout.

My most recent references on this are aimed at print, because that's just how old they are. So I was happy to see something about design that used Web-based examples.

Unfortunately, this isn't going to help people who don't understand design already. It's written in design speak. Sample back-cover copy: "The task of the designer is to systematically interpret the audience's association with the information to be conveyed, and then translate those associations into visual designs using design principles and tools.") It does occasionally have good hints for Web design, but the basics of design aren't adequately explained. For instance, it talks about layout, and it says that when creating a layout, you should only use black and white. All the illustrations for this chapter are two-color, using black and brown, so they don't follow this rule. It's not clear whether the rule is one given for the specific exercise or is meant to hold for all layouts. (It's not a bad rule to follow, because it lets you think about visual balance without the distraction of color.)

As it is, I have to say my best advice for people who want to know the basics of visual design is to go buy Molly Bang's *Picture This: How Pictures Work*. You may have to look in the children's section, and it won't tell you anything about the Web. But you'll learn a lot about, well, how pictures work, which is the important part. I'm still waiting for something that does that and covers the computer stuff.

NO TECH HACKING

Johnny Long and Jack Wiles

Syngress, 2008. 480 pages.
ISBN: 978-1-59749-215-7

REVIEWED BY SAM STOVER (SAM.STOVER@GMAIL.COM)

I think this book is a must read for pretty much any tech-savvy reader, but not for the reasons you might think. Not because you'll learn the "Über-est of the Über," as quoted on the back of the book. Not because it will provide you with everything you ever wanted to know about physical penetration testing. No, the two reasons for reading this book are, one, it's an easy read, and, two, it's fun.

I told you that the reasons weren't what you would expect from a reviewer of security books, but let me explain my position. Let's start with it being an easy read. Obviously, being easy to read isn't high on your priority list for reasons to dive into a book. All too often "easy" means "technically lacking" and, hey, this is *;login:* after all—we want our meat and potatoes. But if it's easy for you to read, then it might be easy for someone else to read—such as

your boss, CISO, or maybe even (gulp) your CEO. How many other books on your shelf can make such a claim?

OK, OK, I can see it on your face right now—just because it's easy for someone else to read isn't enough to sell you on why *you* should read it. So let's talk about fun. Why is this book fun, and what does that have to do with technology? Well, the title *is No Tech Hacking*, so maybe there isn't much technology in the book? Well, there's plenty of technology, just not the kind you might be used to. It's technology of the mind, a.k.a. common sense, that fills the pages of this book, and that's what makes it so much fun. I would wager that a lot of people aren't going to learn from this book so much as they are going to *realize*. This book is so full of common sense that, by the time you are done, you're going to look at the world in a different way, which was precisely the intent. Now, that doesn't mean that you might not learn how to open a door with nothing more than a clothes hanger and a wet washcloth, because you just might. But the more important thing you'll walk away with is an appreciation of how other folks might just be using that kind of technology to compromise your organization.

As far as the layout of the book goes, there are 11 chapters plus an epilogue dedicated to suggestions for preventing all or most of the stuff you learn about in the book. As a heads-up, Chapter 6 is a reprint from Mr. Long's *Google Hacking for Penetration Testers*, Volume 2, a practice I normally despise. However, he's very open and honest about it being a reprint, and I think it plays well with the other chapters, so I'm cool with it. Chapter topics range from Dumpster Diving, Physical Security, Social Engineering, to People Watching, to name a few. There are number of spelling and grammatical errors, but nothing too crazy. The tone is light, conversational, humorous, and very engaging. Definitely not an easy book to put down.

So, now we know that it's easy and fun, and if you put those two things together, you have an extremely powerful weapon in your hands—powerful because your eyes will be opened to how social engineering and physical security are easier vectors than a 0day buffer overflow, and powerful because the knowledge is something that can easily be applied to everyday situations. Furthermore, you now have a book that you can put in the hands of just about everyone in your organization and they'll benefit from it. After all, everyone has some degree of common sense, right? Right?

SMALL FORM FACTOR PCS

Matthew Weaver and Duane Wessles

O'Reilly, 2008. 275 pages. ISBN 978-0-596-52076-2

REVIEWED BY RIK FARROW

I've been interested in do-it-yourself PCs for over 25 years and was excited about seeing a book dedicated to making things with small PCs. Weaver and Wessles do a good job of covering how to do things with some of the small, but Linux- and BSD-capable, devices that have appeared since about 2000. I've bought two Soekris boxes. I remember the struggles I had just getting started, as there is no documentation for installing an OS included with the boxes. If you ever wanted to play with Soekris boxes or build a MythTV on top of a Via M-1000, this is the book for you.

I found the authors' instructions on installing OpenBSD 3.7 on a Soekris net-4501 spot-on. I liked that they point out just how hard it is to set up MythTV, and they suggest that it will take at least a week of fussing to get it to work (which it will). Their writing is clear and their advice is good. (Pay attention to notes that appear in the page margins!) The biggest weakness of this book is that it appears to have been started in 2006—notice that they cover installing OpenBSD 3.7 instead of 4.3, the current version.

There are other issues. The authors decided to use the lowest-end Hauppage video capture card, yet a quick search of the MythTV wiki warns against doing this, promising jerky playback and dropped frames. A margin note does suggest that this was a mistake. The authors also have two projects that use laptop (2.5-inch) hard drives in systems designed to be always on, but I had learned from the Soekris mailing list that laptop hard drives are not designed for always-on use and will die within the first year most of the time.

As long as you are aware that you will want to go online and check hardware choices, the authors do a very good job of explaining the assembly of their projects, the installation of operating systems for Compact Flash-based systems, and how to set up the software for their projects, such as an LCD messaging sign using a gumstix processor with Bluetooth for wireless. If you have built small-form-factor PCs before, you will already have figured out a lot of what the authors cover. But if you are new to the field, this book will certainly help get you started.

SUBJECT TO CHANGE

Peter Merholz, Brandon Schauer, David Verba, and Todd Wilkens

O'Reilly, 2008. 186 pages. ISBN 978-0-596-51683-3

REVIEWED BY RIK FARROW

When I got this book, it came with a sticker that said "Galley Copy, Subject to Change." I immediately tossed in on a pile of books that were unlikely to get read: Why read a book that will be changed before it gets released? At some point, I found myself wondering about the silly sticker and discovered that *Subject to Change* was the snappy title, and it actually had some relevance to the topic of the book.

I can sum up this book by writing that the authors strongly suggest that you base your designs on the experience the users of anything you design have with your product. Your product can be a Web site or the latest consumer, must-have appliance, as the iPod once was. And I do agree with them. All too often, products are designed around a set of cool ideas and implemented by programmers, engineers, and marketing folk, and wind up failing badly. Instead of focusing on the end-user experience, they produced a product with more features than the competition's. The authors point to the Diamond Reo, a competent MP3 player that preceded the iPod and flopped. The iPod succeeded not because it had more features, but, rather, because it did just the minimum of what users needed and no more. By moving the management of music to a program on a computer (iTunes), Apple put the most complicated part of the experience on the device with the most capable interface for handling it.

This book is aimed more at corporations than individuals. And mention of the authors' company is a frequent appearance, making the book itself a subtle bit of advertising. But the information is worthwhile and the suggestions for determining what the experience of users will be and how to insert this knowledge into the design process, even when you face an entrenched marketing/engineering "axis of evil" that wants to build what they think will work, are priceless.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

FREE SUBSCRIPTION to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks, and operating systems, book reviews, and summaries of sessions at USENIX conferences.

ACCESS TO ;LOGIN: online from October 1997 to this month: www.usenix.org/publications/login/.

DISCOUNTS on registration fees for all USENIX conferences.

DISCOUNTS on the purchase of proceedings and CD-ROMs from USENIX conferences.

SPECIAL DISCOUNTS on a variety of products, books, software, and periodicals: www.usenix.org/membership/specialdisc.html.

THE RIGHT TO VOTE on matters affecting the Association, its bylaws, and election of its directors and officers.

FOR MORE INFORMATION regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

USENIX BOARD OF DIRECTORS

Communicate directly with the USENIX Board of Directors by writing to board@usenix.org.

PRESIDENT

Clem Cole, *Intel*
clem@usenix.org

VICE PRESIDENT

Margo Seltzer, *Harvard University*
margo@usenix.org

SECRETARY

Alva Couch, *Tufts University*
alva@usenix.org

TREASURER

Brian Noble, *University of Michigan*
brian@usenix.org

DIRECTORS

Matt Blaze, *University of Pennsylvania*
matt@usenix.org

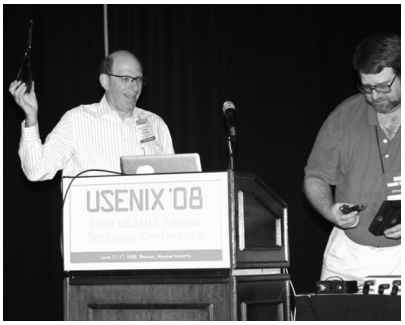
Gerald Carter, *Samba.org/Likewise Software*
jerry@usenix.org

Rémy Evard, *Novartis*
remy@usenix.org

Niels Provos, *Google*
niels@usenix.org

EXECUTIVE DIRECTOR

Ellie Young,
ellie@usenix.org



Andrew Tanenbaum and USENIX President Clem Cole assembling the Flame Award

USENIX LIFETIME ACHIEVEMENT AWARD

2008 FLAME AWARD WINNER:
ANDREW S. TANENBAUM

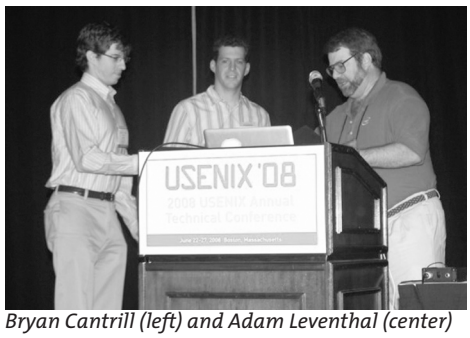
The 2008 USENIX Lifetime Achievement Award ("The Flame") went to Andrew S. Tanenbaum for his many contributions to systems design and to openness both in discussion and in source.

Andrew S. Tanenbaum has researched and written extensively on topics relevant to systems design and implementation. He has developed compilers, operating systems, and tools to help learn about, research, and teach important concepts. Multiple generations of engineers have read and learned from his books, which are to be found on the shelves of all serious computer science students and practitioners. We reference them often, since his lucid prose has clarified some of the most complicated ideas in systems. Andy has inspired healthy debate in our field, always focusing on what ideas are truly fundamental. He has long been a proponent of making everything open and available to all. Long before the terms "free" and "open source" were coined, he described computer science experiments as needing to be repeatable and transparent. He made his tools and technologies available in source, so we have been able to see what he and his team have done.

SOFTWARE TOOLS USER GROUP AWARD

2008 STUG AWARD WINNERS: BRYAN M. CANTRILL, MICHAEL W. SHAPIRO, AND ADAM H. LEVENTHAL

At the 2004 USENIX Annual Technical Conference, Bryan Cantrill, Michael Shapiro, and Adam Leventhal presented a paper on DTrace, which they described as "a new facility for dynamic instrumentation of production systems [which] fea-



Bryan Cantrill (left) and Adam Leventhal (center) receiving the STUG Award from Clem Cole

tures the ability to dynamically instrument both user-level and kernel-level software in a unified and absolutely safe fashion." The first component of the OpenSolaris project to have its source code released under the Common Development and Distribution License, DTrace was the Gold winner in the Wall Street Journal's 2006 Technology Innovation Awards contest. It is a positive demonstration of DTrace's design and implementation that this low-level performance tool originally written for OpenSolaris has already been ported to other operating systems, including many of the usual and popular ones.

USENIX is pleased to honor the DTrace Three for their powerful software tracing facility. It has greatly broadened our view into OS and application behavior both at the user level and inside the kernel, on live systems, with zero effect on the system except when it's explicitly enabled.

The winners generously donated the cash award to support USENIX pre-college student programs.

USENIX ASSOCIATION FINANCIAL REPORT FOR 2007

Ellie Young, Executive Director

The following information is provided as the annual report of the USENIX Association's finances. The accompanying statements have been reviewed by Michelle Suski, CPA, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public Accountants. The 2007 financial statements were also audited by McSweeney & Associates, CPAs. Accompanying the statements are several charts that illustrate where your USENIX and SAGE membership dues go. The Association's complete financial statements for the fiscal year ended December 31, 2007, are available on request.

USENIX continues to be a healthy organization. For fiscal year 2007, USENIX incurred a slight deficit in operations of $100K. The additional $356K in gains in investment income, interest, and dividend income from the Reserve Fund meant that we ended the fiscal year with a $245K increase in net assets.

USENIX MEMBERSHIP DUES AND EXPENSES

USENIX averaged 5,400 members in 2007, which is similar to the previous year. Of these, 2,400 opted for SAGE membership as well, and 500 people are SAGE-only members. Chart 1 shows the total USENIX membership dues revenue ($544K) for 2007, divided by membership type. Chart 2 presents how those dues were spent. Note that all costs for producing conferences, including staff, marketing, and sales and exhibits, are covered by revenue generated by the conferences. Chart 3 demonstrates how the money allocated to student programs, sponsorship of other conferences, and standards activities ($230K) was spent in 2007. Chart 4 shows how the USENIX administrative expenses were allocated. Chart 5 shows where the $165K to provide SAGE benefits and services was spent (note: SAGE member dues revenue was $128K).

USENIX ASSOCIATION
STATEMENTS OF FINANCIAL POSITION
As of December 31, 2007 and 2006

ASSETS	2007	2006
Current Assets		
Cash & cash equivalents	$ 1,165,165	$ 1,245,162
Receivables	37,694	63,087
Prepaid expenses	41,993	47,203
Inventory	3,925	4,388
Total current assets	1,248,777	1,359,840
Investments at fair market value	6,365,486	6,047,657
Property and Equipment		
Office furniture and equipment	555,393	503,596
Less: accumulated depreciation	(479,593)	(452,814)
Net property and equipment	75,800	50,782
Other assets	302,020	248,521
	$ 7,992,083	$ 7,706,800
LIABILITIES AND NET ASSETS		
Current Liabilities		
Accounts payable	$ 429,593	$ 588,561
Accrued expenses	80,484	83,360
Sponsorships for Linux Kernel '08	35,000	-
Contributions held for OpenAFS	-	176
Deferred revenue	168,715	55,290
Total current liabilities	713,792	727,387
Long-term Liabilities	302,020	248,521
Total liabilities	1,015,812	975,908
Net Assets		
Unrestricted net assets	6,976,271	6,730,892
Net Assets	6,976,271	6,730,892
	$ 7,992,083	$ 7,706,800

USENIX ASSOCIATION
STATEMENTS OF ACTIVITIES
For the Years Ended December 31, 2007 and 2006

	2007	2006
REVENUES		
Conference & workshop revenue	$ 3,206,716	$ 3,407,994
Membership dues	543,762	552,978
Product sales	5,155	8,017
SAGE dues & other revenue	135,885	131,290
General sponsorship	4,000	-
Total revenues	3,895,518	4,100,279
OPERATING EXPENSES		
Conferences & workshops	2,723,827	2,645,671
Membership; login:	391,690	394,439
Projects & Good Works	230,397	284,472
SAGE	165,248	224,313
Management and general	446,683	433,327
Fund raising	47,998	112,143
Total operating expenses	4,005,843	4,094,365
Net	(110,325)	5,914
NON-OPERATING ACTIVITY		
Donations - non-cash	15,140	-
Interest & dividend income	251,051	245,024
Gains & losses on marketable securities	157,508	556,471
Investment fees	(64,908)	(62,185)
Other non-operating	(3,087)	(3,305)
Net	355,704	736,005
Increase in net assets	245,379	741,919
Net assets, beginning of year	6,730,892	5,988,973
Net assets, end of year	$ 6,976,271	$ 6,730,892

USENIX ASSOCIATION
STATEMENTS OF FUNCTIONAL EXPENSES
For the Years Ended December 31, 2007 and 2006

		Conferences and Workshops	Programs and Membership	Student Programs, Good Works and Projects	SAGE	Total Program	Management and General	Fund Raising	Total Support	2007 Total
Operating Expenses										
Conference & workshop-direct	*	$ 1,815,318				$ 1,815,318				$ 1,815,318
Personnel and related benefits:		-								
Salaries		569,458	$ 121,280	$ 16,341	$ 91,876	777,470	$ 233,360	$ 24,145	$ 257,505	1,034,975
Payroll taxes		43,080	7,743	1,043	6,951	58,816	17,654	1,827	19,481	78,297
Employee benefits		105,352	-	-	16,997	122,349	43,173	4,467	47,640	191,475
Membership/products		-	3,324	-	-	3,324	-	-	-	3,324
Membership/login:		-	176,311	-	-	176,311	-	-	-	176,311
SAGE expenses	*	-	-	-	27,515	27,515	-	-	-	27,515
Student programs, Good . Works, and projects		-	-	200,483	-	200,483	-	-	-	200,483
General and administrative	*	190,618	83,032	12,530	21,909	308,089	152,496	17,560	170,056	478,145
		$ 2,723,826	$ 391,690	$ 230,397	$ 165,248	$ 3,511,161	$ 446,683	$ 47,999	$ 494,682	$ 4,005,843

		Conferences and Workshops	Programs and Membership	Student Programs, Good Works and Projects	SAGE	Total Program	Management and General	Fund Raising	Total Support	2006 Total
Operating Expenses										
Conference & workshop-direct	*	$ 1,747,016				$ 1,747,016				$ 1,747,016
Personnel and related benefits:		-								
Salaries		557,566	$ 112,292	$ 12,825	$ 62,924	745,607	$ 176,676	$ 56,728	$ 233,404	979,011
Payroll taxes		42,624	8,585	980	4,810	56,999	13,506	4,337	17,843	74,842
Employee benefits		121,927	24,557	2,805	13,760	163,049	38,634	12,405	51,039	214,088
Membership/products		-	6,241	-	-	6,241	-	-	-	6,241
Membership/login:		-	164,301	-	-	164,301	-	-	-	164,301
SAGE expenses	*	-	-	-	117,813	117,813	-	-	-	117,813
Student programs, Good . Works, and projects		-	-	262,250	-	262,250	-	-	-	262,250
General and administrative	*	176,538	78,463	5,612	25,006	285,619	204,511	38,673	243,184	528,803
		$ 2,645,671	$ 394,439	$ 284,472	$ 224,313	$ 3,548,895	$ 433,327	$ 112,143	$ 545,470	$ 4,094,365

USENIX ASSOCIATION
STATEMENTS OF CASH FLOW
For the Years Ended December 31, 2007 and 2006

	2007	2006
CASH FLOWS FROM OPERATING ACTIVITIES		
Change in net assets	$ 245,379	$ 741,919
Adjustments to reconcile increase in net assets to net cash provided by/(used for) operating activities:		
Donation - non cash	(15,140)	-
Depreciation	26,779	31,550
Decrease/(Increase) in receivables	25,393	(16,006)
Decrease in inventory	463	1,525
Decrease/(Increase) in prepaid expense	5,210	(8,200)
(Decrease)/Increase in accounts payable	(158,968)	81,107
Decrease in accrued expenses	31,948	215
Increase in accrued income taxes	-	(11,200)
Increase in deferred revenue	113,425	15,290
Total adjustments	29,110	94,281
Net cash provided by operating activities	274,489	836,200
CASH FLOWS PROVIDED BY/(USED FOR) INVESTING ACTIVITIES:		
Purchase of investments	4,808,743	(3,743,603)
Sale of investments	(4,808,743)	3,743,603
Net investment income designated for long-term purposes	(160,310)	(142,510)
Realized & unrealized gains on investments	(157,519)	(561,482)
Purchase of property & equipment	(36,657)	(25,872)
Net cash used for investing activities	(354,486)	(729,864)
Net change in cash & equivalents	(79,997)	106,336
Cash & equivalents, beginning of year	1,245,162	1,138,826
Cash & equivalents, end of year	$ 1,165,165	$ 1,245,162
Cash payments for:		
Interest	-	-
Taxes	-	$3,532

Chart 1: USENIX Member Revenue Sources 2007

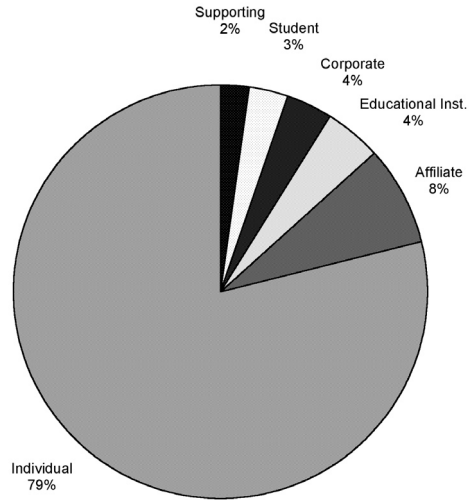


Chart 2: Where Your 2007 Membership Dues Went

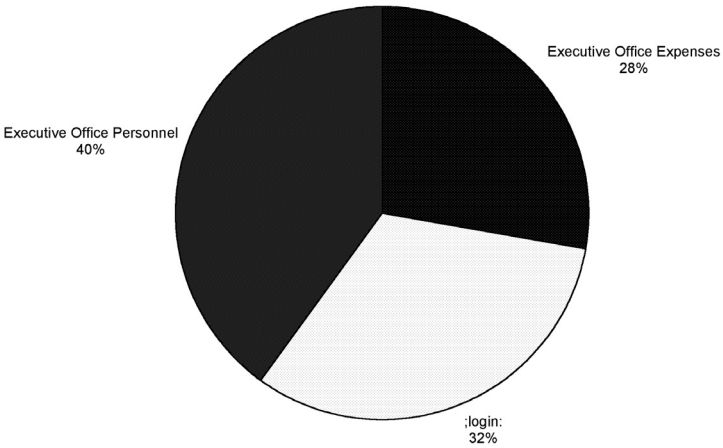


Chart 3: Good Works 2007

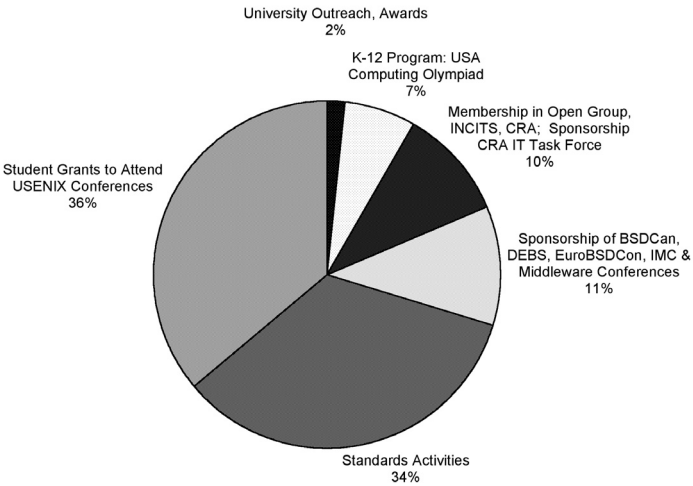


Chart 4: USENIX Administrative Expense 2007

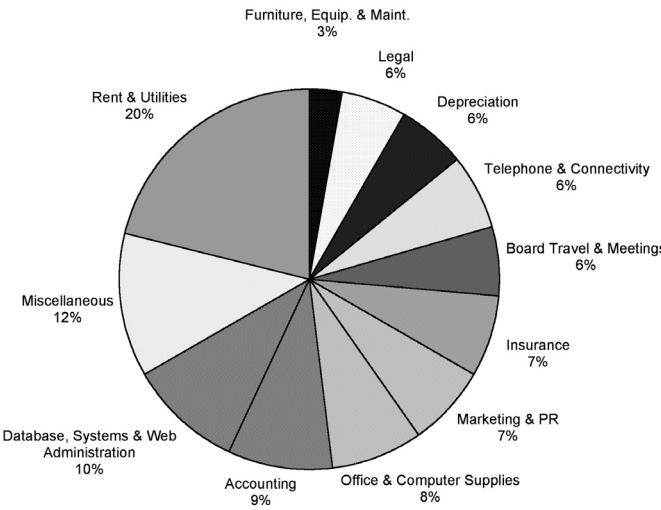
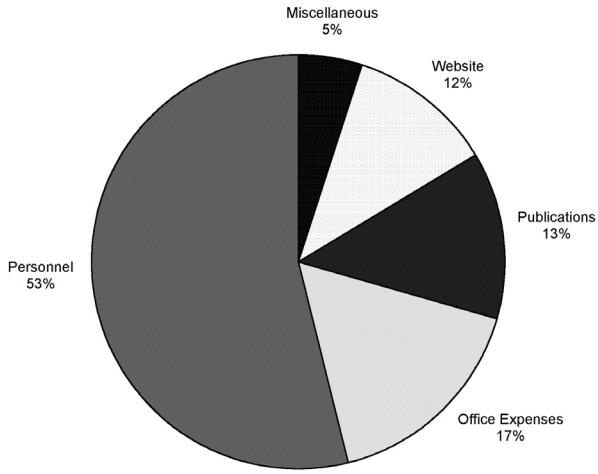


Chart 5: SAGE Expenses 2007



conference reports

THANKS TO OUR SUMMARIZERS

NSDI '08: 5th USENIX Symposium on Networked Systems Design & Implementation

San Francisco, CA
April 16–18, 2008

KEYNOTE: XEN AND THE ART OF VIRTUALIZATION REVISITED

Ian Pratt, Senior Lecturer, University of Cambridge Computer Laboratory, and Fellow, King's College Cambridge

Summarized by Geoffrey Werner-Allen (werner@eecs.harvard.edu)

Mr. Pratt presented a talk in three parts. He began by revisiting the Xen story, presenting lessons concerning doing relevant research in academia. Next he explored why virtualization is such a hot topic in research today. Finally, he explored the changes in Xen since the 2004 SOSP paper and emerging trends in hardware-software co-design.

Although Xen emerged naturally from the cloud-computing ethos and the needs of usage-based accounting, Mr. Pratt pointed out that there was a significant pause in its early days, primarily because the funding agencies had nothing to compare their project to. During this period, Xen was release via the GPL to "friends and family" and their team began to notice the differences in doing development not for an academic paper, that is, when their creations had to run for more than the 30 minutes required to produce the graphs for the latest paper. They kept working on the production-level aspects of their software.

The mission of Xen is to provide the industry-standard open-source hypervisor. Xen developers are interested in driving CPU development and showcasing new CPU features, as well as providing the type of security necessary for enterprise acceptance. As Xen has continued to develop it has found use in some interesting and perhaps unforeseen areas, such as cell phones. Mr. Pratt described plans that one cell-phone vendor has to run three separate hypervisors on its phone to isolate critical phone functionality, vendor-supplied software, and user-installed software from each other.

Why is virtualization hot at this particular moment? One reason is that it is driven by the "scale out" occurring at the enterprise level. Running each enterprise application on a single server leads to server sprawl, with CPU utilization of 5%–15% typical. Another reason involves the things that typical operating systems have failed to do well, including full configuration isolation, temporal isolation for performance, spatial isolation for security, and true backward compatibility. Virtualization has the potential to solve many of these problems. Moreover, the maintenance of a narrow interface to the hypervisor and the ability to hide machine-specific details behind

the hypervisor allows easier configuration. Other benefits of virtualization include reduced downtime during maintenance owing to the ability to migrate slices to other machines, the ability to rebalance load as workloads change, and hardware fault tolerance through checkpointing and replay.

Mr. Pratt discussed the issues concerning hypervisor security. Although the existence of a hypervisor does add to the attack surface, its small size should make it easier to defend. He discussed network control software that their group has been writing in OCaml, a language that allows certain guarantees to be made about the program's execution. In addition, there is additional complexity involved in device emulation, so to the degree that this is required this increases the complexity required in the hypervisor. He also discussed the possibility of performing a "measured launch" of the hypervisor to provide guarantees between boots. In addition, moving some of the OS administrative functionality outside the OS should help operating systems become more robust and harder to disable. Other tricks that can be used to improve hypervisor security include the use of immutable memory and taint tracking techniques (detailed elsewhere).

The next frontier in virtualization research should be making virtualized systems easier to administer. In particular, breaking the OS/HW bond should simplify the application certification process. Currently, many software vendors refuse to certify their applications on many hardware configurations or in the presence of other applications. Instead of certifying an application on top of many different hardware and operating system combinations, software makers can certify on a single operating system, which is then certified on top of Xen, meaning that the application can run on a variety of different hardware configurations. We are already starting to see application-specific operating systems being developed in the presence of virtualization techniques such as Xen. Virtual hardware also simplifies creating and modifying operating systems, as well as allowing hardware vendors to "light up" new features much faster.

In the final part of his talk Mr. Pratt discussed the process of paravirtualization through several examples, including MMU and network device virtualization. Interestingly, with regards to MMU virtualization Xen proved that some of the support that hardware vendors have added into their systems, in particular allowing "nested page tables," have actually not performed as well as the original direct and shadow page tables supported by Xen.

Network devices have proved extremely difficult to virtualize, but new NICs are emerging with features that make the process simpler and more effective. Xen developers have divided NICs into four levels, 0 through 3, each with increasing features allowing easier virtualization with better performance. Level-0 NICs are standard NICs, requiring significant hypervisor intervention to virtualize. Level-1 NICs have multiple receive queues, allowing these queues to be assigned to VMs making heavy use of the network. Level-2 NICs have enabled direct guest access, allowing the NIC to do traffic shaping, firewalling, filtering, and other processes. Level-3 NICs actually present to the system as multiple PCI devices, simplifying device management at the hypervisor level.

In conclusion, Mr. Pratt recommended the use of open source software to gain early and continuing impact while doing university research. In the future he sees hypervisors becoming ubiquitous and spawning a new golden age of operating system research.

Professor Birman from Cornell was curious about the tension between adding features, making the hypervisor more like larger buggier operating systems, and consolidation, possibly impacting performance but improving security. Mr. Pratt pointed out that many of the new features he described are actually being implemented outside of the core hypervisor, meaning that they can be isolated from it for the purposes of security. Greg Minshall from the University of Washington asked about Xen's impact on the previous optimization that had been done at the hardware/software boundary. Mr. Pratt responded that as machines get faster this sort of optimization is less important and that Xen does as little of this as possible. Professor Sirer from Cornell asked why operating systems are unable to provide "virtualization" at the POSIX level. Mr. Pratt replied that the interface is simply too broad and high level. Tomas Isdal asked whether Xen developers had a plan to scale to multiple cores; Mr. Pratt replied that they did.

TRUST

Summarized by Eric Hielscher (hielscher@cs.nyu.edu)

- ***One Hop Reputations for Peer to Peer File Sharing Workloads***
  Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson, University of Washington

Michael began by explaining that peer-to-peer scalability depends on user contributions but that users are often reluctant to contribute. Current peer-to-peer systems explicitly include contribution incentives such as tit-for-tat servicing. Next he presented results of a measurement study that show that increasing one's contribution to BitTorrent swarms has very little effect on one's download rates. Further, there is no fundamental lack of capacity in the swarms nor a lack of interest. Rather, results from another study show that the vast majority of bandwidth capacity in BitTorrent swarms is held by the top 10% of users (i.e., those with the least incentive to contribute).

This motivates the main point of the present work: to make incentives more effective at encouraging contributions. The typical popularity of a single swarm is shaped like a bell curve with a long right tail, as most users leave immediately

after downloading. Ideally, we would like users to be able instead to draw on all previous downloads for service. The problem with tit-for-tat in BitTorrent is that the incentives are applied only when users are actively downloading, and only in the context of a single swarm. The observation made is that peers should instead be rewarded for all contributions across all swarms.

A simple fix would be for peers to cache a local history of their interactions with other peers and to reward peers with whom they repeatedly interact who have contributed to them in the past. However, a study of peer interactions shows that repeat interactions are very rare. The proposed approach is to implement one-hop reputations with limited indirection. One hop is enough, since almost all peers are connected through a very small number of the most popular peers. The protocol involves key pairs as long-term IDs and intermediary nodes to maintain accounting information and provide signed verification receipts. Intermediaries are discovered by gossiping during connection setup, and in the default policy they are selected based on popularity. Intermediaries receive priority service, and thus peers have incentive to serve as one. In the evaluation of the protocol, it was shown that 97% of random peers shared an intermediary; on average 73% of intermediaries selected in a random interaction are selected again within 100 interactions.

The authors conclude by pointing out that for peer to peer to reach its full potential, persistent contribution incentives are necessary, and one-hop reputations leverage the popular minority of peers for this purpose. One questioner asked why anyone would ever want to send directly to peers, since traffic sent to an intermediary is far more valuable in terms of the reciprocation it will garner. The response was that peer traffic's value will be inflated in the induced reciprocation economy.

- ***Ostra: Leveraging Trust to Thwart Unwanted Communication***
  Alan Mislove and Ansley Post, Max Planck Institute for Software Systems and Rice University; Peter Druschel and Krishna P. Gummadi, Max Planck Institute for Software Systems

Alan motivated his talk and work by pointing out that since digital communication such as VoIP, email, and IM is so low-cost, it can easily be abused to send unwanted communications. This manifests itself in various forms including spam and mislabeled content on YouTube, and users are not easily held accountable for their actions since new IDs can be created for free. Previous approaches to solving this problem—such as filtering content, charging money for sending messages, or introducing strong IDs—all have shortcomings.

The authors present a solution to this problem called Ostra, based on an ancient system for transferring money in India called Hawala. The basic idea is to leverage existing offline social trust relationships, since these are expensive to create and maintain. Most communication systems have some sort of implicit or explicit social network, and Ostra assumes that links in this network are maintained by some trusted site. Recipients of messages classify messages (e.g., perhaps implicitly via deletion) as wanted or not. Messages between peers are sent directly, but a link to a peer is broken over time if that peer is on a path to a destination that is receiving unwanted messages. Each link has a credit balance, and the balance is adjusted in favor of a message recipient if the message was unwanted and vice versa otherwise. The credit balance is bounded with a certain range, and the process is iterated over intermediate peers in the event of no direct link between the sender and the recipient.

The system guarantees that no user can send more spam than the amount of spam he or she has received plus the lower bound on link credit times the number of in-links he or she has. Further, this holds for any subgraph, showing that collusion doesn't help attackers and neither does the creation of many Sybil identities. Credit is decayed by a fixed percentage daily to prevent a user from unfairly being blocked. The authors simulated Ostra using a social network taken from YouTube, as well as an email trace from the MPI, and found that even with 20% of users being attackers, only four spam messages were received by any good user per day. Further, very few messages were delayed from links reaching their credit limits. One person raised the point that the classifications may not always be black and white; for example, a message might be unimportant now but important later on. The response was that the system can work alongside other systems such as whitelisting and that finding the proper classification notions is a difficult issue.

- ***Detecting In-Flight Page Changes with Web Tripwires***
  Charles Reis, Steven D. Gribble, and Tadayoshi Kohno, University of Washington; Nicholas C. Weaver, International Computer Science Institute

Charles began his talk by discussing the recent phenomenon of ISPs injecting ads into the Web pages their users visit. The work discussed attempts to detect such in-flight changes to pages and measure them. The system is implemented as JavaScript code, which runs in the client's browser, finds changes in the HTML of the page, and reports them to the user and a central server. It works by fetching and rendering the original page while fetching the JavaScript code in the background from the page's server as well. This code contains a compressed version of the page's expected source code, and the JavaScript compares the two versions. In a study involving 50,000 unique IP addresses, 657 clients saw changes from client software, ISPs, firewalls, and malware.

Some of these changes inadvertently broke some pages by causing JavaScript errors or interfering with forum posting, and others introduced security vulnerabilities such as cross-site scripting vulnerabilities. A major concern with this problem is that it affects all Web pages—similar to a UNIX root exploit—and that the Web developers are powerless to fix the problem. Thus the authors caution users about

software such as client proxies, since they wield root-like power. Further, the Web Tripwires tool helped find vulnerabilities in such software, which have since been fixed.

A publisher of Web content could react in various ways to its pages being altered. First, it could simply use HTTPS to encrypt its pages. However, this is both costly and rigid in that it can't allow security checks or caching. Web Tripwires offers an alternative that allows publishers to easily and cheaply detect most changes, at the cost of somewhat lesser robustness to attacks. The performance of Web Tripwires is also much better than HTTPS, both in terms of latency and throughput. More information on Web Tripwires can be found at http://vancouver.cs.washington.edu.

- ***Phalanx: Withstanding Multimillion-Node Botnets***
  Colin Dixon, Thomas Anderson, and Arvind Krishnamurthy, University of Washington

Colin began with a list of major botnet attacks that have occurred in recent years, including the government of Estonia being shut down by an attack for three to four weeks. He then posed the question, "Why isn't the problem with botnets solved?" In one sense, it is solved for static content, in that we can simply replicate content and use large CDNs. A potential solution for dynamic content might involve replacing all routers in the Internet, but this is not feasible. The key ideas in the current work's solution involve tieing the fate of a server to a large part of the Internet in a way that is scalable and deployable in the current Internet.

The mechanisms in the solution include numerous hosts used as proxies to make packet filtering decisions, forwarding the unfiltered traffic to the server we wish to protect. The nodes are used as mailboxes and hold each packet while waiting for an explicit request from the server. Secure random multipathing is used to protect communication. Traffic is sent randomly among the mailboxes according to a shared secret, and thus the botnet can only take one link down while communication still continues. The mailboxes negotiate a secret at connection setup time and use a lightweight authenticator. This scheme necessitates a multipath congestion control algorithm.

The problem still remains that if attackers sent traffic to the server directly they could still bring it down. Thus a filtering ring is used to drop unrequested Web traffic and to allow only requested traffic to reach the server exactly once. This is implemented by installing blacklists and whitelists on the server's routers. The scheme so far still only protects established connections between a client and server. To initiate connections, the server sends the first packet requests. Access to these requests is mediated by computational puzzles or authentication tokens. The authors evaluated their system by simulating attacks on PlanetLab, with favorable results.

An article about Phalanx begins on page 22 of this issue.

WIRELESS

Summarized by Geoffrey Werner-Allen (werner@eecs.harvard.edu)

- ***Harnessing Exposed Terminals in Wireless Networks***
  Mythili Vutukuru, Kyle Jamieson, and Hari Balakrishnan, MIT Computer Science and Artificial Intelligence Laboratory

The high-level goal of a MAC protocol is to transmit as many packets as possible. Today, the dominant approach to MAC protocols is CSMA (Carrier-Sense Multiple Access). However, the problem with CSMA is that it prohibits many transmissions that would have succeeded, owing to its failure to address the exposed terminal problem. This is the case where, although transmissions might seem to the senders to conflict, the recipients are sufficiently separated that they would have been able to receive the packet correctly. Instead of simple heuristic approaches that attempt to generalize rules to each node in the face of fluctuating bandwidth and channel properties, this work attempts to use empirical evidence to determine when overlapping transmissions can proceed.

Identifying simultaneous transmissions that can proceed safely requires that each node maintain a conflict map that describes whether or not it can transmit safely to node X if it overhears node Y transmitting. The conflict map is built based on observation of the loss rates associated with transfers. Once they reach 50%, throughput would be higher if the transmissions were scheduled sequentially rather than in parallel, so this is the threshold for inclusion in the conflict map. ACKs and the backoff policy must also be adjusted in the face of concurrent transmissions. To allow the node to observe when transmissions conflict, the MAC layer must both be able to recover the node address from unsuccessful receptions, which is facilitated by its inclusion in both the packet header and trailer, and pass up the header before the rest of the packet, so that it can be accurately time-stamped.

A prototype implementation is tested to see whether it can produce no-CSMA behavior when the terminals are exposed and CSMA-like behavior when the terminals conflict. Indeed, experiments on a multi-node 802.11 testbed show that their prototype is able to improve performance overall by essentially acting like CSMA only when CSMA is actually needed.

Questions for the presenter included the choice of 50% as the cutoff point for inclusion in the conflict map, whether or not weighing the signal-to-noise ratio against the noise floor might allow a simpler approximation of this algorithm, and whether or not experiments in noisier environments had been performed. Ms. Vutukuru responded that performance is similar across a wide range of cutoff points near the middle (30% to 60%), and that more tests were needed in different environments to evaluate the impact of varying parameters not yet experimented with.

- ***Designing High Performance Enterprise Wi-Fi Networks***
  Rohan Murty, Harvard University; Jitendra Padhye, Ranveer Chandra, Alec Wolman, and Brian Zill, Microsoft Research

Murty began by stating that more and more wireless is being deployed in the enterprise and users are beginning to develop the same high-capacity expectations for wireless performance as they have for wired. However, currently deployed enterprise wireless networks have many limitations. Because of a phenomenon known as "rate anomaly," the performance of deployed access points is limited by their slowest client. DenseAP seeks to revisit some of the original assumptions surrounding enterprise wireless networks, specifically that the number of access points should be much lower than the number of clients. By deploying a large number of access points and carefully controlling client associations, load balancing, and channel usage, DenseAP seeks to deliver wired-like performance over wireless links.

The challenges this work faces are threefold. First, deciding which wireless access points a client should associate with (controlling association). Second, determining which channel each access point should be operating on (channel assignment). Finally, as clients enter and leave the network and their bandwidth demands change, it is likely that associations will need to be revisited to balance load among access points.

DenseAP controls client associations through a central server, which, when a client begins sending out probe requests, decides which access point is the best match for that client and only allows that access point to respond to the probe request. Association policy is dictated by the quality of the connection and the demand present on each access point. In general, available capacity is equal to the expected transmission rate times the free air time at that access point. To estimate the available capacity the authors use a mapping between RSSI and throughput driven by empirical observations. To estimate free air time they observe the queuing delay at each access point. Channel assignment between access points is done by simply assigning each new access point to the least-loaded channel. As clients move and their behavior changes, associations may need to be reevaluated. To do this, the central controller actively shifts load away from access points that are incurring high stress.

The testbed used for the experiments in the paper is a portion of a floor of a corporate office building. While this area was normally served by only one corporate wireless access point, during experiments up to 24 DenseAP nodes (or DAPs) were used to service up to 24 clients. The authors present results showing improvements in overall performance, as well as attempting to isolate the effects of channel assignment, DAP density, and their intelligent association policy.

During questions, one person wondered whether it would be possible to also allow clients to use multiple access points. Mr. Murty replied that although this would require changes to the client, which DenseAP avoids, it would be interesting if possible. Professor Birman from Cornell asked about what happens if the client associated with the particular DAP selected by the central controller fails. Mr. Murty replied that when the central controller observes such a failure it will choose a new DAP for the client to associate with.

An article about DenseAP begins on page 41 of this issue.

- ***FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput***
  Srikanth Kandula, Massachusetts Institute of Technology; Kate Ching-Ju Lin, National Taiwan University and Massachusetts Institute of Technology; Tural Badirkhanli and Dina Katabi, Massachusetts Institute of Technology

Mr. Kandula described FatVAP, which is designed to address several problems in current wireless 802.11 networks. The first is that the backhaul bandwidth capacity of a particular access point may be bottleneck limiting flows, meaning that there is spare bandwidth at the sender that could be used to send data through other access points. The second is that choosing access points based on proximity combined with a high density of clients leads to hotspots—overutilization of certain access points, leaving spare capacity at others that competing clients could be utilizing. Ideal performance can be obtained by aggregating all access points usable by a particular client or set of clients into one virtual access point, with wireless and backhaul bandwidth equal to the sum of its parts. However, this requires clients to be able to multiplex their connections across multiple access points, which is currently not possible. That said, their solution, once implemented on one or a set of clients, requires no changes to the access points themselves to increase client performance.

To determine how to divide time among APs, FatVAP must solve a scheduling problem. In general, if we have a set of access points, each with a different drain capacity e and available bandwidth w, then a client need not connect to that access point for more than e/w of its time, referred to as the useful fraction. This quantity subsumes link quality, contention, and backhaul capacity. As several examples given showed, no greedy solution for this scheduling problem exists, as the problem is equivalent to a bin-packing problem, with the bandwidth being the value and the time spent at each access point being the cost, bounded by the total time available.

This approach is difficult and presents many implementation challenges. First, to estimate wireless bandwidth, synchronous acks can be used to measure the queue drain rate on each access point; estimating backhaul bandwidth can be accomplished through observing back-to-back large packets. To allow reception from multiple hosts, FatVAP uses 802.11 power save mode to compel access points to cache packets for it while it rotates through others it is using. A large set of client-side changes are needed, includ-

ing allowing the kernel to rotate through multiple APs by spreading traffic through a number of different interfaces above the kernel level. "Soft-switch" between access points allows them to enable high-rate TCP through multiple access points on top of FatVAP.

In conclusion, the authors have shown that FatVAP can aggregate throughput, balance load, and adapt to changing network conditions. In questioning, one person was curious about why the authors focused on bandwidth while neglecting latency. Mr. Kandula replied that further experiments were necessary to assess the impact of FatVAP on latency.

- ***Efficiency Through Eavesdropping: Link-layer Packet Caching***
  Mikhail Afanasyev, University of California, San Diego; David G. Andersen, Carnegie Mellon University; Alex C. Snoeren, University of California, San Diego

In real networks, overhearing happens, meaning that even if a route from A to C normally passes through B, some of the time C may overhear the packet being transmitted from A to B directly. In this case, it is advantageous to avoid retransmitting the packet that C already has from B to C. This scenario can also lead to unnecessary acknowledgment messages. Earlier solutions to the overhearing and multiple transmission problem have used caching, which introduces an unacceptable amount of delay at each client between transmissions.

To reduce retransmissions without introducing latency, RTS-id embeds a packet identifier in the RTS/CTS 802.11 exchange. The packet IDs are based on a hash of the packet contents, although RTS-id is careful not to include portions of the packet that may change as it traverses multiple hops.

RTS-id was implemented on top of Cal Radio, using packet modifications designed to look normal on nonparticipating nodes. The testbed consisted of three Cal Radio nodes, although simulations were also performed on data gathered from the RoofNet outdoor testbed. A state machine was used to model packet forwarding behavior during the simulations. Results show that RTS-id reduces retransmissions in the face of overhearing, with savings naturally scaling with the number of hops that the packet traverses. Because of the way that RTS-id was implemented it can also work seamlessly alongside nodes not implementing the protocol.

Professor Levis from Stanford was curious about whether the authors had investigated possibilities for spatial use as a result of their work. Mr. Afanasyev replied that they were considering this. Professor Karp from CMU asked whether or not this could be combined with other forms of network coding. Mr. Afanasyev wasn't sure.

LARGE SCALE SYSTEMS

Summarized by Jeff Terrace (jterrace@cs.princeton.edu)

- ***Beyond Pilots: Keeping Rural Wireless Networks Alive***
  Sonesh Surana, Rabin Patra, and Sergiu Nedevschi, University of California, Berkeley; Manuel Ramos, University of the Philippines; Lakshminarayanan Subramanian, New York University; Yahel Ben-David, AirJaldi, Dharamsala, India; Eric Brewer, University of California, Berkeley, and Intel Research, Berkeley

There has been considerable research done on deploying network infrastructure into developing, rural areas around the world, but the problem that Sonesh Surana et al. were trying to solve is that, once an infrastructure is in place, it's very difficult to keep it maintained and sustainable over long periods of time. Two existing wireless networks were studied: the Aravind Eye Hospital's video-conferencing network in southern India and the AirJaldi network in northern India, which provides Internet access to rural users. The largest problems facing sustainability are poor-quality grid power, limited local expertise for maintenance and diagnosis, lack of full connectivity in the network for remote management, and the physical location of networks residing in remote locations that are difficult to reach.

Hardware faults in these two networks were dominated by power-related faults. The problem in developing countries is that instead of a steady, reliable voltage rating, grid power can result in a large range of voltages, which ends up damaging electronic equipment. A UPS does not help, because although it provides reliable power during an outage, it passes power directly to the device during normal operation, which still results in bad voltages. Because commercial products were too expensive and sensitive, one solution was a custom-built low-voltage disconnect circuit to guard against low-voltage situations combined with solar power to handle peaks and swells in the power grid. A push-based PhoneHome system was also implemented that uses the cell phone network to report node, link, and network properties every 3 hours to a central server. Satellite links were also used to provide additional entry points into the network to address software and link failures causing some nodes to be unreachable. A cheap hardware watchdog device was also used to reboot routers that fail.

The additional devices and methods used here eliminated the need for weekly reboots, reduced power failures, and reduced prolonged downtime in the two networks; as a result, both networks are now financially stable.

- ***UsenetDHT: A Low-Overhead Design for Usenet***
  Emil Sit, Robert Morris, and M. Frans Kaashoek, MIT CSAIL

Emil Sit began by stating that there are over 2 million articles and files that arrive on Usenet every day, which translates to 30 MB/s. Usenet was one of the first P2P systems created. Servers that store Usenet articles are distributed geographically and as an article is posted to a server, the article is passed to the server's peers until eventually

the article is held on all Usenet servers. This system makes it difficult to create a Usenet server because of the large volume of data that the server must be able to store. UsenetDHT is a shared Usenet server that allows multiple servers to cooperatively share Usenet articles in a DHT.

UsenetDHT combines the storage space of multiple servers by distributing a single copy of a Usenet article among them. It separates article headers from article contents, and it stores only a single copy of an article's contents in the DHT. All servers keep a copy of the article headers (which makes up less than one percent of the storage cost) to allow clients to see headers immediately. By leveraging UsenetDHT, several small sites can benefit from the resources of its peers and can cooperatively run a Usenet server. UsenetDHT requires high throughput and data durability, but current algorithms for DHTs are synchronization heavy. Each node in the network must sync with several other nodes to provide durability and replication, which is a slow process over a WAN. To solve these problems, UsenetDHT uses Passing Tone, an algorithm on top of DHash that balances minimizing the bandwidth used between nodes and minimizing the amount of state that needs to be stored on each server. Passing Tone only keeps local synchronization data, shares the responsibility of ensuring proper replication with its neighbors, and can make decisions about replicas by only communicating with its immediate neighbors. Passing Tone is a simple algorithm that minimizes overhead but still performs almost as well as previous algorithms.

A question was asked about what UsenetDHT provides for censorship resilience. In reply, the speaker stated that there is an advantage in replicas being distributed, but that he doesn't envision UsenetDHT replacing Usenet because the latency might be too high across servers. UsenetDHT does not affect censorship. In reply to another question, the speaker stated that there is no public information about how much of Usenet is spam.

- ***San Fermin: Aggregating Large Data Sets Using a Binomial Swap Forest***
  Justin Cappos and John H. Hartman, University of Arizona

Justin Cappos said that computing results of a computation over a large, distributed data set can be difficult. When the amount of data you need to process is large, aggregating the data at a single location can take too long to process, so distributed aggregation algorithms have been devised. An example when this type of algorithm is needed is when a programmer is trying to analyze end-user traces of a program's execution. The programmer is only interested in a total sum, not individual values, but trace files can be too large to transfer to a single point. Instead, the traces can be processed locally and just the aggregated sums can be transferred as the result. The goals of an algorithm for aggregating large data sets are to have complete coverage, have no duplicates in the answer, have no partial data, tolerate node failures, not overload any individual node, and produce a result fast.

San Fermin is an algorithm for large data aggregation that uses a binomial swap forest to calculate results. Each node is assigned a unique identifier to prevent duplicate and missed nodes. Each node then starts swapping data with other nodes by considering each bit in its ID value from right to left and choosing another node that has a different target bit and has the same prefix. Each node has its own view of the network, that is, as a binomial swap tree, and by aggregating data with the nodes it chooses to swap with, it will eventually have the aggregated result that is desired. Since every node runs the algorithm, the first node to complete the aggregation reports its result and all other nodes can stop. This method allows the aggregation to be robust to node failures since a node will usually swap its data with multiple other nodes before failing, so its information is already in the binomial swap tree of others. The prototype of San Fermin is built on top of Pastry, which provides IDs, failure detection, and routing. To evaluate San Fermin, the prototype was tested on 100 PlanetLab nodes and compared to SDIMS, which is also built on top of Pastry.

The evaluation showed that both algorithms perform well with small amounts of data, but SDIMS starts to fall apart with a large amount of data. San Fermin has a much lower variance of completion time than SDIMS, and it scales much better. Increasing the number of nodes from 32 to 1024 or the data size from 256 kB to 1 MB only increases the completion time by a factor of 4. When 50 failures occurred during aggregation, the final results were only missing 5–11 nodes. If there is a large variance in bandwidth capacity across nodes, the faster nodes tend to finish first, which is a desirable property.

The conclusion was that San Fermin performs aggregation better than previous algorithms for large data sets, scales well, and is robust to failures. In reply to questions, Cappos stated that San Fermin applies only when exact results are needed instead of trying to approximate, and that San Fermin differs from MapReduce because MapReduce focuses on distributed computation, whereas San Fermin may only be computing a very simple operation but wants to avoid centralizing the data.

FAULT TOLERANCE

Summarized by Petr Marchenko (p.marchenko@ucl.ac.uk)

Awarded Best Paper!

- ***Remus: High Availability via Asynchronous Virtual Machine Replication***
  Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, and Norm Hutchinson, University of British Columbia; Andrew Warfield, University of British Columbia and Citrix Systems, Inc.

Brendan Cully presented Remus, a system that allows unmodified software to be protected from the failures of the physical machine on which it runs. In case of a failure, a running system can continue its execution on an alternative physical host with only seconds of downtime while com-

pletely preserving its internal state. Remus uses virtualization, whereby protected software is encapsulated in a virtual machine, and its runtime state is propagated to a backup host at a high frequency, e.g., 40 times per second.

This state propagation is possible because of virtualization, which allows running VMs to migrate between physical hosts. Remus applies asynchronous whole-system replication at particular checkpoints, and the primary server remains productive between the checkpoints. Remus buffers output until a more convenient later time in order to delay the synchronization and perform payload computation. This technique is called speculative execution. It yields substantial performance benefits and allows checkpointing intervals on the order of tens of milliseconds. To keep the primary system and backup hosts consistent, Remus does checkpointing, which has to deal with CPU and memory replication, network buffering, and disk buffering. Memory and CPU replication are based on Xen's existing live migration mechanism. Network input and disk reads are applied to the system immediately; however, the network output and disk writes are buffered until a checkpoint is performed. Remus's protection overhead mainly exists from checkpointing and network delays introduced by network buffering.

Fred Douglis asked about the effects of network activity on high-throughput applications. Brendan said that the network-sensitive applications incur higher performance overhead. He added that the results of the SPECweb benchmark are presented in the paper and that Remus provides one-quarter of applications' native performance. Amin Bada questioned the large amount of data that was transmitted across the network, not all of which was strictly necessary for Remus's operation. Brendan acknowledged that they did not evaluate this in the work, but using more focused data might offer a significant performance improvement. Someone from the audience was interested in whether Brendan and his co-workers tried to experiment with hardware virtualization for their system. The speaker said that these sorts of experiments were not done because of the absence of appropriate equipment, but it would be an interesting direction for their future work.

- ***Nysiad: Practical Protocol Transformation to Tolerate Byzantine Failures***
  Chi Ho and Robbert van Renesse, Cornell University; Mark Bickford, ATC-NY; Danny Dolev, Hebrew University of Jerusalem

Distributed systems and protocols such as DNS, BGP, and OSPF are designed to tolerate only crash failures; however, it is crucial to have the ability to deal with Byzantine failures. Chi Ho discussed Nysiad, a technique for transforming a scalable distributed system or a network protocol designed to tolerate only crash failures to one that tolerates arbitrary failures. It uses a variant of Replicated State Machine (RSM) to translate Byzantine faults into crash faults.

The state machine of a host is replicated onto the guards of the host, together constituting an RSM. Nysiad's replication protocol, OARcast, ensures that the guards of the host remain synchronized. OARcast provides the following properties: All correct guards deliver a message if one correct guard does; the messages from a single origin are delivered in the same order; and a compromised host cannot forge a message of a correct host. When the communication graph is unknown, the common case, Nysiad has no good way of determining which hosts will be communicating with other hosts. In this case, the replication protocol will not work, as it relies on the trustworthiness of the sender's guards. The same problem arises when a host changes its guards or when reconfiguration takes place. To handle this problem, Nysiad introduces a logically centralized trusted certification service, Olympus. It is involved only when changing the communication and guard graphs. It produces signed certificates for hosts containing information that is sufficient for a receiver of a message to check its validity. Owing to the increased number of control messages sent per single end-to-end message, Nysiad's message latency is three times higher than the latency in the nonconverted system.

An attendee from Microsoft Research was curious how the system behaves when a host lies consistently. In response, Chi said that Nysiad includes additional protocols, which were not mentioned in the talk, that deal with this problem. An attestation protocol guarantees that messages delivered to the guards are a valid execution of the protocol and a credit protocol forces a host to either process all its input fairly or to ignore all input.

- ***BFT Protocols Under Fire***
  Atul Singh, Max Planck Institute for Software Systems and Rice University; Tathagata Das, IIT Kharagpur; Petros Maniatis, Intel Research Berkeley; Peter Druschel, Max Planck Institute for Software Systems; Timothy Roscoe, ETH Zürich

Byzantine Fault Tolerant (BFT) protocols for replicated systems have received considerable attention in the systems research community. However, it is hard to evaluate these protocols and distinguish the best one under certain conditions. This is because the BFT protocols are implemented in different languages, may require nontrivial libraries, and depend on particular systems. Thus, the implementation-based approach for comparison of BFT protocols is not always possible. Atul Singh presented BFTSim, a simulation environment for performance-modeling-based comparison of BFT protocols. The system includes a high-level protocol specification language, an execution environment, and a network simulator.

The protocol specification language allows one to capture the salient points of protocols without drowning in the implementation details (e.g., threads and cryptographic primitives). The network simulator provides the ability to explore protocols under different network conditions. The execution system runs the protocols and it emulates the execution overhead by introducing delays. Thus, a programmer has to specify the cost of a protocol's primitives, such as cryptographic operations.

Atul and his co-workers verified the correctness of BFTSim by comparing the evaluation of Zyzzyva, PBFT, and Q/U protocols under their simulator and the real evaluation presented in the literature. BFTSim was able to match the performance graph for the real protocols with an error less than 10%. BFTSim makes BFT protocols more accessible, as it offers a unified system for protocol performance comparison under certain network conditions.

There was a question about whether the protocol specification language captures the complexity of the protocol implementation such as lines of code. Atul explained that it does, as the amount of code in the specification language is proportional to the amount of code in the protocol's implementation.

MONITORING AND MEASUREMENT

Summarized by Eric Hielscher (hielscher@cs.nyu.edu)

- ***Uncovering Performance Differences Among Backbone ISPs with Netdiff***
  Ratul Mahajan and Ming Zhang, Microsoft Research; Lindsey Poole and Vivek Pai, Princeton University

Ming began by pointing out that there have been numerous studies done on evaluating and comparing systems such as file systems, databases, and Web servers but there has been little such work done on evaluating and comparing different ISPs. Thus customers don't have enough information to make a good decision as to which ISP is the best for their needs. The current state of the art involves service level agreements between customers and their ISPs in which the ISP guarantees some aggregate performance, something that doesn't easily translate into an assessment of perceived end-user experience.

The requirements the authors outlined to structure their study include that the ISP comparisons be both relevant to customers (by measuring end-to-end paths target destinations of interest and making comparisons based on workloads similar to their own) and useful to ISPs (by helping them to account for geographic presence and to identify bad Points of Presence [PoPs] or destinations). The ideal architecture of the comparison framework would involve deploying probes inside every PoP of the ISPs, and taking a probe from every PoP to every destination on the Internet. However, the overhead would be too high. In the Netdiff, probers are deployed at the edge of the network, and probes are sent from ends to various destinations on the Internet—for example, from an ingress PoP to an egress PoP of an ISP. A single centralized controller sends probe lists to all probers, which then send back their results.

The system is able to generate a complete snapshot of each of 18 backbone ISPs using between 5,000 and 23,000 probes in under 20 minutes, a significant improvement over another similar system called Keynote. The system is deployed on PlanetLab and has been generating such snapshots every 20 minutes for the past year. The comparison methodology involved using path stretch and grouping paths based on length and differentiating between paths to the destination on the Internet as well as internal paths, with ISPs ranking very differently on the various metrics. Detailed information on the data as well as the ability to generate individualized comparisons is available at http://netdiff.org.

- ***Effective Diagnosis of Routing Disruptions from End Systems***
  Ying Zhang and Z. Morley Mao, University of Michigan; Ming Zhang, Microsoft Research

Ying began by stating that the goal of his work is to diagnose routing disruptions purely by using end systems, a departure from existing approaches, since they are controlled by end users and needn't use ISPs' proprietary data. The desire for such diagnosis comes from the fact that such disruptions impact application performance as well as causing high loss and long delays. The approach taken only requires probing from end hosts with traceroute and can cover all PoPs of a target ISP as well as most destinations on the Internet. Disruptions are identified by comparing paths that are consecutively measured. Some challenges involved in this approach include limited probing resources, limited coverage of probed paths, and issues related to timing granularity and measurement noise.

The system's architecture involves collaborative probing by a set of distributed hosts, each of which sends traceroutes to different destinations on the Internet to learn routing state, improve coverage, and reduce overhead. Events are then classified according to ingress/egress changes into three types: the ingress PoP changes, the egress PoP changes, or neither does. Events are then correlated both spatially and temporally, since events happening close together in space or time are likely due to a few root causes. They employ an inference methodology by compiling pieces of evidence that support various causes such as an egress link being down. They then list all likely causes of each event of interest and build an evidence graph that maps evidence nodes to cause nodes close together in time. A conflict graph is also generated, with nodes that represent evidence that conflicts with a given event, to reduce cause candidate sets, and a greedy algorithm is used to search for a minimum set of causes while covering all evidence and having minimal conflicts.

Five large ISPs were monitored via a deployment on PlanetLab, covering all of the ISPs' PoPs, with refreshes occurring every 18 minutes. The results show that many events discovered were internal changes, something that BGP-based methods wouldn't find. The system was validated against existing BGP-based approaches as well, with somewhat high error rates owing to limited coverage, coarse-grained probing, and measurement noise. The system performed well enough to be usable for real-time diagnosis.

- ***Csamp: A System for Network-Wide Flow Monitoring***
  Vyas Sekar, Carnegie Mellon University; Michael K. Reiter, University of North Carolina, Chapel Hill; Walter Willinger, AT&T Labs—Research; Hui Zhang, Carnegie Mellon University; Ramana Rao Kompella, Purdue University; David G. Anderson, Carnegie Mellon University

Vyas began his talk by pointing out that the needs for network monitoring stem from things such as traffic monitoring, analysis of new user applications, and network forensics. In particular, good traffic measurements, including measurements of fine-grained traffic structure, are needed. Some design goals include limited resource consumption on routers, high flow coverage, ability to specify network-wide goals, and low data management overhead. Current systems for network monitoring employ uniform packet sampling on routers, with aggregation of individual router data into flow reports. This results in a bias toward large flows, coarse goal specifications, and redundant measurements.

The proposed system, cSAMP, randomly samples flows rather than packets to solve these issues. Each router hashes a tuple consisting of the network protocol and the source and destination IP addresses and ports to compute a FlowID. Each router in the network is configured to store a different subset of the hash function's range. This allows for global configuration (i.e., routers are not required to communicate during sampling). To allow for network-wide configuration, different hash ranges are configured per origin-destination pair in the network (e.g., NYC/PIT). A framework is provided for generating sampling manifests (the configuration files for the routers). This involves a linear programming problem, which takes as inputs origin-destination pair information and router resource constraints and outputs the optimal sampling strategy that maximizes traffic and coverage.

cSAMP was evaluated against fixed rate and maximal flow sampling as well as packet sampling. Its flow coverage was 2–3 times better than packet sampling and 30% better than maximal flow sampling. In addition, cSAMP is significantly better than the other methods at achieving minimal fractional coverage and network-wide goals. It is robust to traffic dynamics and scalable. A question was asked about whether cSAMP could be used for per-flow rate-limiting applications, and the response was that the current infrastructure is geared toward near-real-time analysis of flow reports rather than real-time monitoring for rate-limiting.

- ***Studying Black Holes on the Internet with Hubble***
  Ethan Katz-Bassett, Harsha V. Madhyastha, John P. John, and Arvind Krishnamurthy, University of Washington; David Wetherall, University of Washington and Intel Research; Thomas Anderson, University of Washington

Ethan started off his talk by pointing out that global reachability is a basic Internet goal. In use, the Internet seems usually to have such reachability, but there are numerous cases where this isn't true and there are transient reachability problems. The Hubble system aims to automatically identify persistent reachability problems. The algorithm it employs includes three steps. First, distributed ping monitors detect when a destination becomes unreachable. Second, reachability analysis is conducted using distributed traceroutes. Finally, the problem is classified. To detect whether the problem involves the forward or backward link between the source and the destination, Hubble employs IP address spoofing by having another source send a packet to the destination with the first source as its spoofed IP address. If the original source then hears a response, we can conclude that the problem was with the forward link; otherwise, it must be with the backward link.

Problems are detected by pinging destinations every two minutes. A destination is reported after a series of failed pings. A BGP table is maintained from RouteViews feeds, allowing for an IP-address-to-AS mapping. Next, the extent of the problem is assessed by using traceroutes to gather topological data, with probing continuing while the problem persists. Analysis is performed to determine which traceroutes reach the destination. Next, the problem is classified according to ISPs, routers, and destinations, in order to help operators diagnose and repair it.

The evaluation of Hubble presented in the talk focuses on two questions: How much of the Internet is monitored, and what percentage of paths is analyzed for each given prefix? The results show that, every two minutes, 89% of the Internet's edge space and 92% of ASes are monitored. In addition, for 60% of prefixes, Hubble monitored routes through all ASes on RIPE BGP paths to the prefix. Further results show that spoofing works well and that many Internet holes last for more than 10 hours and most were cases of partial reachability. An interesting result was that multihoming may not give resiliance to failure, since many multihomed prefixes had problems in which multiple traceroutes terminated in one provider while the prefix remained reachable through another provider. Hubble is running continuously, and a map of ongoing problems is available at http://hubble.cs.washington.edu.

PERFORMANCE

Summarized by Petr Marchenko (p.marchenko@ucl.ac.uk)

- ***Maelstrom: Transparent Error Correction for Lambda Networks***
  Mahesh Balakrishnan, Tudor Marian, Ken Birman, Hakim Weatherspoon, and Einar Vollset, Cornell University

Mahesh Balakrishnan started by explaining the problem that TCP/IP has when it is used in high-speed lambda networks. TCP/IP was designed to provide connectivity in congested networks; however, in networks such as TeraGrid with 40-Gbps links, there is no congestion but there are packet drops because of dirty fiber, misconfiguration, and switching contention. TCP/IP uses a feedback loop to recover lost packets, which results in dramatic throughput

reduction. A loss rate of 0.1% is sufficient to reduce TCP/IP's throughput by an order of magnitude.

As a solution for this problem, Mahesh proposed the Maelstrom Error Correction appliance, a rack of proxies residing between a sender and a receiver in a WAN link. These proxies apply Forward Error Correction (FEC) for the traffic being transmitted over the link. The sender proxy encodes every five data packets in three FEC packets, and the receiver proxy checks the correctness of data packets and uses FEC to recover lost or damaged data packets. This technique works well for random losses but not for burst losses. Therefore, Maelstrom uses a new encoding scheme called layer interleaving, which applies extra correction packets over the blocks of packets (layers), using three layers of different length.

Maelstrom was evaluated on the Emulab testbed. It is able to cope with loss rates up to 2% without significant throughput degradation, whereas TCP/IP's performance degrades dramatically when loss rate is increased from 0.01% to 1%. The overhead introduced by Maelstrom does not depend on the length of the links, but only on the data rate. The proposed solution is transparent, as it does not require modification of network infrastructure and software. Thus, TCP/IP can be run over Maelstrom.

Michael Walfish wondered why they rejected the possibility of using rateless code in their paper. Mahesh admitted that rateless code could be used in their system; however, he did not explain why they found it unusable but suggested taking this question offline. One attendee asked for a clarification of the difference between this work and the work presented at NSDI four years ago. Mahesh agreed that the work has some similarities, but the earlier one was doing error correction in the network; thus, their deployment models are completely different. Bob Read from Facebook asked whether Maelstrom supports $n + 1$ connectivity or whether there is always one-to-one mapping. Mahesh responded that if there are several connection points, they have to be paired; therefore, it is always one-to-one mapping.

- ***Swift: A Fast Dynamic Packet Filter***
  Zhenyu Wu, Mengjun Xie, and Haining Wang, The College of William and Mary

Zhenyu Wu addressed the problem of fast dynamic packet filtering. Dynamic filtering is essential for building network services, network engineering, and intrusion detection, where it is required to adjust the filter at runtime. When there is a dynamic filter update, the traditional filters such as BSD Packet Filter (BPF) requires three preprocessing phases: compilation of a new filter, user-kernel copying (as the filter runs in the kernel), and security checking to make sure that a new filter can be safely run in the kernel. These stages prolong filter update latency, which results in misses of hundreds or even thousands of packets. This gap can cause serious problems for critical applications such as intrusion detection systems.

Zhenyu presented SWIFT, a packet filter that takes an alternative approach to achieving high performance, especially for dynamic filtering tasks. Like BPF, SWIFT is based on a fixed set of instructions executed by the in-kernel interpreter. However, SWIFT is designed to optimize the filtering performance with powerful instructions and a simplified computational model. Powerful instructions allow SWIFT to accomplish common filtering tasks with a smaller number of instructions. This speeds up static filtering and allows removing the filter compilation stage in filter updates, which improves the dynamic filtering performance. SWIFT eliminates security checking during filter update; instead, it is banned from controlling the execution path and storing data. This prevents it from tampering with the kernel.

Simplifying the filter update procedure by removing the compilation and security checks allows SWIFT to achieve at least three orders of magnitude lower filter update latency in comparison with Linux Socket Filter (LSF). This reduces the number of missing packets per connection by about two orders of magnitude. The powerful instruction set and simplified computational model increase filtering speed; thus, SWIFT outperforms LSF by up to three times in terms of packet processing speed.

SECURITY

Summarized by Brendan Cully (brendan@cs.ubc.ca)

- ***Securing Distributed Systems with Information Flow Control***
  Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières, Stanford University

It is very hard to build secure distributed systems. One major reason is simply code size: Application code can run into millions of lines, much of which is unaudited third-party library code of uncertain provenance. Within these large applications, even tiny vulnerabilities can lead to catastrophic data exposure. Although it isn't feasible to fix every application bug, systems such as Asbestos, HiStar, and Flume demonstrate that it is possible to prevent untrusted code from seeing private data in the first place. They do this using decentralized information flow control (DIFC) to track data as it flows across applications and enforce access control rules on that data. For example, a database credit card query might be labeled according to the user credentials supplied with it, and data flow control could then ensure that the response is only visible to the same application path that provided the credentials.

Current DIFC systems are limited to applications running on a single host. Nickolai Zeldovich presented a system, called DStar, that allows DIFC to be enforced across a network of mutually distrusting applications. This is done by

delegating local labels to an export process on each physical host, which uses self-signed labels (in which the public key of the exporter is part of the label name) to transfer labels over the network, where they may be converted back to local labels. To support decentralized flow control, any process can create new labels, remove labels it owns, and grant the ability to remove labels to other processes.

Because DStar's trust model is decentralized, it is possible to use flow control even across multiple operating systems. For instance, highly sensitive data might be processed under the HiStar environment, but less sensitive data could be handed off to Linux systems or even completely untrusted cloud computing systems.

An audience member asked how this category system differs from a normal capability system. Nickolai responded that categories are strictly more general. For instance, they make it possible to assert negative access rules.

- ***Wedge: Splitting Applications into Reduced-Privilege Compartments***
  Andrea Bittau, Petr Marchenko, Mark Handley, and Brad Karp, University College London

The number of reported security vulnerabilities continues to increase every year. This is partly because programmers ignore the principle of least privilege. Andrea Birtau argued that they do this because the process-based privilege mechanisms commonly available now are not fine-grained enough to provide good protection. Wedge is a system designed to help with this problem in two ways: first, by providing a simple partitioning mechanism called sthreads, which only shares memory with other sthreads that have been explicitly tagged, and second, by introducing a tool called crowbar which helps to find good partition strategies for existing legacy applications.

The standard approach to privilege separation is to fork when changing privilege levels, using file descriptors between processes to share data. Unfortunately, it's easy to accidentally leak information to child processes, because memory is copied into them by default, and so it must be manually scrubbed before the child begins executing. It's also hard to share information, because it must be serialized across a file descriptor. Sthreads avoid the latter problem because, like standard threads, they run in the same address space. They also avoid the former problem because they can only see memory in other sthreads for which they have been granted a capability. Sthreads associate a tag with all memory they allocate and may grant tags to other sthreads.

It would be desirable to apply this mechanism to existing code, but ad hoc analysis of how data is shared among tasks in an existing monolithic application is impractical. For example, the Apache Web server accesses over 600 different memory objects. Manually tagging each of them would be both painful and hard to get exactly right. Static analysis may also fail (e.g., because of function pointer usage). Crowbar attempts to discover partitions by observing the actual access patterns of running applications, using the Pin dynamic instrumentation system.

Andrea was asked about his experience designing applications using Wedge. He told the audience that he wrote a DNS server from scratch with sthreads and didn't feel that explicit memory tagging required significant extra effort. Converting legacy code was of course much more difficult.

ENERGY

Summarized by Geoffrey Werner-Allen (werner@eecs.harvard.edu)

- ***Reducing Network Energy Consumption via Sleeping and Rate-Adaptation***
  Sergiu Nedevschi and Lucian Popa, University of California, Berkeley, and Intel Research, Berkeley; Gianluca Iannaccone and Sylvia Ratnasamy, Intel Research, Berkeley; David Wetherall, University of Washington and Intel Research, Seattle

The rising energy consumption of networking-related equipment is a pressing issue, given rising energy costs and the increased recognition of the impact of CO_2 emissions on the global climate. Given that most network equipment is provisioned for maximum load, which is rarely reached, network devices provide a great opportunity for power savings. Power consumption should reflect utilization, not capacity. This work explores two techniques to reduce power consumption: sleeping, that is, disabling routers for periods of time, and frequency scaling, that is, reducing the processing speed of the router itself. Combined, these techniques should reduce both the active and the idle power consumption of routers. Given that new routers are beginning to be shipped with the ability to sleep and rate-adapt, these techniques are a promising way to reduce power consumption while protecting performance.

In general, sleep states consume much less power than even idle ones; however, the transition time to awaken the router when data arrives is a concern. The authors assume that the router can be awakened by either a timer or link activity. To create periods in which a link can sleep, they buffer packets at the link and then transmit them through in a burst. Coordinating these buffer and burst periods throughout a network can ensure that the introduced latency does not increase as data traverses multiple hops. However, it turns out that their experiments show that the benefits of coordinating sleeping are minimal compared to uncoordinated sleeping, which captures most of the available energy savings.

Rate adaptation involves lowering the processing rate of the router to just the point necessary to keep up with link traffic. In general, the perfect link adaptation algorithm is not implementable, as it requires future knowledge of link activity. Instead, the heuristic algorithm the authors implement observes the local queue depth and current rates in order to choose future rates. As their analysis shows, uniformly

spaced variable rates are better for capturing the benefits of rate adaptation then the exponentially spaced rates available on many routers currently shipping. The authors evaluated their approach through simulations using power states and transitions from an Intel NIC. They found that, on this particular card, sleeping produced better results than rate adaptation, but they point out that this card, like many others, was not really designed for power savings. They then present a complete model of power savings that will allow them to evaluate future cards. Finally, they showed results indicating that, when comparing rate adaptation and sleeping, there is a utilization threshold that serves as a crossover point: Under it, sleeping performs better; after it, rate adaptation performs better.

Brian Zill from Microsoft Research asked whether the hardware used for this purpose in typical networking is optimized for energy consumption, and whether many of the benefits they described may be achieved simply through better hardware design. Mr. Nedevschi responded that today's networking hardware is indeed not particularly power-aware, but that this is beginning to change. Another questioner asked what percentage of the total energy consumption of networked equipment was tied up in the switching hardware that they are improving. Mr. Nedevschi wasn't sure. Finally, another questioner asked how this would affect TCP congestion control, and Mr. Nedevschi pointed out that they were careful that their changes produced no impact on TCP performance.

- ***Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services***
  Gong Chen, University of California, Los Angeles; Wenbo He, University of Illinois at Urbana-Champaign; Jie Liu and Suman Nath, Microsoft Research; Leonidas Rigas, Microsoft; Lin Xiao and Feng Zhao, Microsoft Research

Jie Liu stated that IT servers are the energy hog of the IT industry. The speaker pointed out that the increase in server energy consumption between 2000 and 2006 was enough to power 5.8 million American homes. Obviously, there is a chance to save a significant amount of energy if server energy consumption can be better managed. And the opportunity exists, because server load fluctuates for a variety of reasons throughout the day. This work focuses on adjusting the number of servers needed to serve MSN Messenger, a connection-intensive application. Because it is costly to migrate connections between servers, the authors focus on predictive techniques to identify the number of servers to have active at any given moment, combined with different approaches to load balancing connections across the servers active at any given moment. In addition, because shutting down servers conserves the most energy, the authors focus on ways to completely shut down servers when not in use.

A brief overview of the MSN Messenger architecture was presented. The servers targeted for power savings are the connection servers, which are in charge of maintaining persistent client connections but not storing a great deal of client state. The authors identify three metrics: service availability, service continuity, and service latency. Each server has a bounded number of connections and a bounded connection rate at which it can accept new connections. For the servers and application studied, the number of connections is on the order of 100k, whereas the server can only add around 70 new connections per second, meaning that they can be modeled by leaky buckets with tiny input pipes. The slow speed with which connections can be added also makes forward-looking provisioning all the more important.

The first step is load forecasting, in which regression models incorporating daily and seasonal fluctuations, along with the current state of the system, are used to predict load. In the experiments they performed, the system was trained on five weeks of data and then tried to predict a single week. Load dispatching is another key part of the system since the rates with which users can be added are limited and the distribution of users affects which machines can be shut down and how many users will be affected. The balancing approach assigns users to all available machines roughly evenly, whereas the skewing approach assigns users to fill one machine at a time.

Their evaluation looked at several different combinations of these approaches (e.g., skewing versus balancing and forecasting versus no forecasting). What they found is that skewing plus forecasting performs the best, with a 30% reduction in energy used. A number of graphs pictorially demonstrating the impact of different load balancing policies were shown. Finally, the authors identify a number of alternate approaches, including TCP state migration as well and building support into the client, allowing it to handle requests to move to a different server.

Rik Farrow of USENIX asked about some sharp spikes in the load graphs that had been shown. These turned out to be due to code rollouts or the effect of shutting down machines and having a bunch of clients reconnect all at once. Professor Vahdat from UCSD asked about using virtual machines to assist in the state migration, to which Mr. Liu responded that their servers don't actually maintain much state.

ROUTING

Summarized by Petr Marchenko (p.marchenko@ucl.ac.uk)

Awarded Best Paper!

- ***Consensus Routing: The Internet as a Distributed System***
  John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, and Thomas Anderson, University of Washington; Arun Venkataramani, University of Massachusetts Amherst

Internet protocols have traditionally favored responsiveness (a liveness property) over consistency (a safety property). Thus, they apply routing updates immediately to its forwarding tables before propagating them to other routers.

This causes routing loops and blackholes. Fully 10%–15% of BGP updates cause loops and 30% packet loss.

John P. John presented consensus routing, where he proposed to separate safety and liveness properties using two models of packet delivery: stable and transient. A stable mode ensures that a route is adopted only after all dependent routers have agreed upon a consistent view of the global state. This is achieved by a distributed snapshot and a consensus protocol. Transient mode ensures availability when a packet encounters a router that does not possess a stable route because of a link failure or an incompletion of consensus protocol. In this case, the router makes forwarding decisions based on transient heuristics such as backup routes, deflections, and detours. Consensus routing, which resides as a layer on top of BGP, does not require changes to BGP and does not disclose any more information regarding its routing policies than BGP does.

Comparison of BGP's connectivity and consensus routing connectivity in case of AS traffic engineering (prefix withdrawing) showed that BGP maintains connectivity only in 40% of the test cases, whereas consensus routing does so in 99% of the test cases. The loss of connectivity happens because of the transient loops. Consensus routing was able to converge from one consistent state to another, thereby avoiding transient loops in all test cases. Consensus routing adds traffic overhead, as it requires 30% more update bytes than BGP.

There was a question about whether a policy is more or less opaque with consensus routing than with BGP. John answered that since the entire policy is not known to all, it is as opaque as with BGP. Another question concerned whether it is possible to bundle updates. The answer was that bundling would cause inconsistency as a single update propagate policy that affects individual ASes. Someone wondered about the downtime in traffic forwarding that is caused by performing updates. John said that the effect of applying updates is covered by detour routing in the transient stage.

- ***Passport: Secure and Adoptable Source Authentication***
  Xin Liu, Ang Li, and Xiaowei Yang, University of California, Irvine; David Wetherall, Intel Research Seattle and University of Washington

Xin Liu addressed the problem of source address spoofing, since it damages the Internet in a variety of ways. Address spoofing significantly mitigates the effectiveness of DoS defense mechanisms. It also makes possible reflector attacks and makes source address filtering untrustworthy.

Xin Liu proposed Passport, a novel network-layer source authentication system. Passport treats an AS as a trusted and fate-sharing unit, and it authenticates the source of a packet to the granularity of the origin AS. It uses symmetric-key cryptography and checks packets only at administrative boundaries. When a packet leaves its source AS, the border router stamps one Message Authentication Code (MAC) for each AS on the path into the packet's Passport header. When the packet enters an AS on the path, the border router verifies the corresponding MAC value, using the secret key shared with the source AS. The correct MAC can only be produced by the source AS that also knows the key.

Passport relies on the routing system to efficiently manage keys using Diffie-Hellman key exchange on routing advertisements. Source address spoofing within a single AS is considered to be an internal issue for an AS. This solution can be incrementally deployed, as it is interoperable with legacy ASes.

An attendee asked about the routing assumptions. Xin stated that Passport requires routers having complete routing tables. Another question was asked about MTU because Passport adds MACs. Xin said that this is not an issue for routers; they can increase the size of the packet and they do it anyway for things such as VPNs.

- ***Context-based Routing: Technique, Applications, and Experience***
  Saumitra Das, Purdue University; Yunnan Wu and Ranveer Chandra, Microsoft Research, Redmond; Y. Charlie Hu, Purdue University

Saumitra Das discussed the effects of new lower-layer technologies such as multiple radios and link layer network coding on the routing path in wireless mesh networks. As he pointed out, conventional routing frameworks do not allow taking advantage of the new lower-layer technologies, since the costs of the links are examined in isolation from each other. Thus, multiple radios and network coding are not considered by conventional routing mechanisms.

Saumitra suggested a framework for routing in the presence of inherent link interdependencies, called context-based routing. It includes a new context-based path metric and route selection method that leverage the advantages of network coding and multiple radios. This context-based framework uses conditional link metrics: the Expected Resource Consumption (ERC), which models the cost saving from network coding, and a Self-Interference–aware Metric (SIM) for multiple radio systems. A context-based path pruning method uses these metrics to identify a preferable path. Based on these primitives, Saumitra and his colleagues implemented a Context Routing Protocol (CRP) and conducted experiments on two testbeds, demonstrating significant throughput gains.

An attendee asked whether the links advertised by CRP would already be congested. Saumitra said that the lower cost is advertised based on the throughput gain that you would get. One attendee wanted to know how much predictability you need in the flows to calculate the correct cost of the flow. The response was that they have a mechanism in the paper to ensure that equilibrium is reached.

UNDERSTANDING SYSTEMS

Summarized by Brendan Cully (brendan@cs.ubc.ca)

- ***NetComplex: A Complexity Metric for Networked System Designs***
  Byung-Gon Chun, ICSI; Sylvia Ratnasamy, Intel Research Berkeley; Eddie Kohler, University of California, Los Angeles

Simplicity has always been valued very highly in system design, but it is hard to measure quantitatively. Crude metrics like number of messages or total state size can be very misleading; for example, flooding is simple but produces many messages. Byung-Gon Chun presented a new metric, called NetComplex, to better reflect our intuition about the complexity of the algorithmic component of networked systems. It is based on the observation that these systems center on distributed state, and this state is dependent on the messages that communicate it. NetComplex uses a dependency graph in which discrete elements of single-host state form the nodes of the graph and messages that change that state form the edges.

NetComplex divides complexity into two levels. The most basic level is state complexity, which is the number of state changes that occur across the dependency graph as a result of changes to each variable. Operation complexity is a higher-level metric which aggregates the total state complexity resulting from an operation as defined by the system API. This is the metric by which Byung-Gon proposed that alternative algorithms be compared.

The rest of the presentation attempted to demonstrate the accuracy of the metric, first by using it on several different routing protocols, where it was determined that compact routing was the most complex protocol in spite of the fact that it had both the least state and the fewest messages (because it was designed for scalability). The metric was also applied to a number of classical distributed systems and then compared to the complexity rankings assigned by a survey of 19 graduate students in a distributed systems class; they matched closely.

There were a number of interesting questions. One attendee observed that Ethernet was a wildly successful algorithm, but according to this metric it would be classified as extremely complex (owing to exponential backoff). Another attendee pointed out that conventional metrics apply to resources, so that a system with limited bandwidth might optimize for fewer messages at the expense of more state. He wondered how NetComplex was intended to be used to select systems given that it did not apply to particular resources. Byung-Gon replied that, in general, the simplest algorithm was the best choice for producing robust systems.

- ***DieCast: Testing Distributed Systems with an Accurate Scale Model***
  Diwaker Gupta, Kashi V. Vishwanath, and Amin Vahdat, University of California, San Diego

A recurring problem for application developers is that they simply do not have the resources to test their applications in all of the different environments in which they will eventually be deployed. DieCast is a system that attempts to replicate large systems with a high degree of fidelity on a much smaller number of machines, while also providing reproducibility and making efficient use of the available hardware. Its approach is to use virtualization to multiplex many logical machines onto a single physical host, and then to carefully manipulate perceived time within the VMs to adjust for the reduced CPU available to them. This allows CPU to scale to large numbers of logical systems, but it does not scale either RAM or disk capacity.

Within a single virtual machine, time dilation (presented at NSDI '06 by the same group) can be used to hide increased runtime from the running operating system. But in such an environment, unmodified I/O would appear correspondingly faster. For example, in a VM in which virtual time progresses at one-tenth the speed of real time, a 1-Gbps network link would appear to run at 10 Gbps. Therefore, DieCast interposes on network and disk devices to scale them according to the time dilation factor in effect, so that perceived latency and throughput match those of the real devices.

The accuracy and utility of DieCast were evaluated in two ways. First, the RUBiS Web application benchmark was run natively on 40 nodes, and under DieCast on 4 nodes of 10 VMs each. The resulting throughput and response time scores matched very closely. A case study was also provided in which a scalable storage company reported good results from testing changes to their high-performance computing application.

- ***D^3S: Debugging Deployed Distributed Systems***
  Xuezheng Liu and Zhenyu Guo, Microsoft Research Asia; Xi Wang, Tsinghua University; Feibo Chen, Fudan University; Xiaochen Lian, Shanghai Jiaotong University; Jian Tang and Ming Wu, Microsoft Research Asia; M. Frans Kaashoek, MIT CSAIL; Zheng Zhang, Microsoft Research Asia

It is difficult to debug distributed systems, in particular because it is hard to reproduce error conditions. Machines run concurrently at varying speeds, and network conditions change dynamically. For example, a distributed lock manager may provide exclusive or shared locks with the invariant that only one client can hold an exclusive lock. Optimizations such as local state caching can make it tricky to reason about whether the invariant always holds. Simulation and model checking can help, but only to a degree. Eventually, runtime checking is likely to be necessary.

The most common approach to runtime checking is to add logging to an existing system and then to attempt to replay

from the logs. This can entail considerable developer effort, and getting just the right level of logging can require many iterations: Too much logging can produce unacceptable overhead, but too little will miss key state changes. And even after the logs are captured, analysis remains challenging. D^{3}S attempts to simplify the process of runtime assertion checking, by letting developers add distributed assertions to running systems on the fly. The primary contributions of D^{3}S are a simple language for distributed predicates, the ability to inject predicates into running systems, and tolerance of host or network failures. D^{3}S injects code into running systems by rewriting the running binary at specified hook points to collect assertions. These are sent to a set of assertion-checking servers using messages tagged with a Lamport clock to form globally consistent snapshots. In order to tolerate failure, each node provides a heartbeat, the loss of which removes it from the snapshot set.

The authors used D^{3}S on five real systems (all third-party applications) to evaluate whether it helped to find bugs. They found that it was easy to write predicates for these systems and that they were able to discover bugs that required runtime checking. Because only assertion state was logged and checked, the overhead on running systems was low (between 3% and 8%).

One audience member wondered how one could specify a predicate that could be used to find performance problems. Xuezheng acknowledged that this was a tricky problem, but argued that being able to add and remove probes on the fly would still be very helpful. Another attendee asked how probes could be written for applications written in higher-level languages other than C or C++. Xuezheng claimed that most real applications are written in C/C++ and that higher-level languages often provided better debugging facilities directly.

LEET '08: First USENIX Workshop on Large-Scale Exploits and Emergent Threats

San Francisco, CA
April 15, 2008

ATTACKER BEHAVIOR

Summarized by Joshua Mason (josh@jhu.edu)

- ***On the Spam Campaign Trail***
  Christian Kreibich, International Computer Science Institute; Chris Kanich, Kirill Levchenko, Brandon Enright, and Geoffrey M. Voelker, University of California, San Diego; Vern Paxson, International Computer Science Institute; Stefan Savage, University of California, San Diego

Christian Kreibich presented data he and his collaborators gathered about the Storm botnet. The data was collected by first reverse engineering and subsequently infiltrating the botnet with the intention of discerning email address harvesting properties, spam delivery efficacy, and the size of individual spam campaigns. Data capture was accomplished by running 16 virtual machines infected by Storm and situating the nodes at several levels in the Storm hierarchy while disallowing malicious activity such as actually sending spam.

Running live instances of the Storm botnet led to several interesting discoveries. First, Christian discussed the spam templating functionality, which allows spammers to craft messages using a variety of macros. These macros can then be substituted with random data to make emails containing the same general message difficult to cluster. They observed 14 different macros used during their deployment and discovered 10 more with experimentation. The team also discovered dictionaries for use in macro values (e.g., subject lines and domain names) and various lists of email addresses (hit lists) used in different spam campaigns.

Kreibuch went on to give a myriad of different statistics on the spam traffic they observed. They saw over 100,000 command and control connections for worker nodes of the Storm network and were able to collect 172,000 spam templates. They also observed 272,546 harvest reports that contained information gathered from worker nodes. Perhaps the most staggering statistic was the number of targeted email addresses, coming in at 66.7 million. A survey of these addresses revealed some fairly comical addresses such as "first.lady@whitehouse.gov" and "stalin@kremlin.ru."

Someone asked about what led to the discovery that one of the largest lists collected contained domains for use in randomizing spam by way of templates. This led an audience member to inquire as to whether templates were linked to dictionary lists so as to better convince the receivers of the spam's legitimacy. Christian's group did not observe the behavior, but he admitted that it is an interesting possibility. Other questions related to the encrypted communication present in Storm and about the ease of infiltrating the network. The speaker noted that infiltration was surprisingly easy and encrypted communication is subject to man-in-the-middle attacks. Niels Provos wondered whether they'd tried to inject error messages to the bot master. They did not, but the question led to a discussion of how easy it would have been for the bot master(s) to detect their presence. The bot master could have asked Kreibuch's worker bots to send spam to certain addresses and then checked whether the spam was actually sent, but this did not happen.

- ***Characterizing Botnets from Email Spam Records***
  Li Zhuang, University of California, Berkeley; John Dunagan, Daniel R. Simon, Helen J. Wang, Ivan Osipkov, and Geoff Hulten, Microsoft Research; J.D. Tygar, University of California, Berkeley

John Dunagan presented a work led by Li Zhuang at UC Berkeley that used trace information present in spam messages to correlate spam campaigns. Their spam corpora

was gathered from the "junk" folder of Hotmail users over 9 days. Using this data, they discovered that 50% of spam botnets have more than 1,000 bots and 80% of botnets use less than half of their bots in each spam campaign. The last statistic begs the somewhat depressing question: Have spam botnets reached the point where they don't need as many bots as they have? In addition, Dunagan indicated that 60% of botnet-related spam is from long-lived botnets.

To associate spam bots with botnets, they attempted to link these bots to individual spam campaigns, in the hope that the same spam campaigns are perpetrated by individual botnets. This was accomplished by using three separate techniques. First, the same spam campaigns tend to use the same target URLs (i.e., ask the spammed user to visit the same site). The target URLs had to match exactly for this metric to work, which seems to be a somewhat defeated spam campaign correlation mechanism based on the randomization of URLs discussed in Kreibuch's presentation. Their second technique to link spam campaigns, then, used the similar body content present in messages. Finally, they also attempted to link bots to botnets based on whether the same bots are participating in the same campaigns.

Once they associated a spam campaign to an individual botnet, they tried to estimate the number of individual machines present in the botnet. This becomes difficult because of the prevalence of dynamic IP addresses among compromised machines. So, they used MSN data containing login events to link machines across dynamic IP addresses and thus to establish the variation pattern on subnets. Because users could easily be logging in from home and then from work, they define an upper bound on the potential variability present on subnets.

The first questioner asked how overlapping content in spam messages was used, given that the messages are often designed to defeat such correlation techniques. Dunagan said they used Rabin fingerprints and that currently used spam obfuscation techniques do not achieve enough polymorphism to make correlation impossible or even difficult. Another audience member asked whether the team notified MSN users found to be infected. Dunagan noted that their MSN data was not from the same 9-day period as their spam data; while they might be able to notify a user that they were infected a month ago, they didn't have the clearance to do so.

- ***Peeking into Spammer Behavior from a Unique Vantage Point***
  Abhinav Pathak and Y. Charlie Hu, Purdue University; Z. Morley Mao, University of Michigan

Abhinav Pathak presented the third and final spam talk at LEET. His research observed spam from the vantage point of open SMTP relays. To collect data, they set up an open relay that sent only those messages that test for open relays. All other email was blocked. Spammers attempting to locate open relays send messages containing the IP address of the relay they are testing to email addresses the spammers control. Thus, to fool the spammers into thinking the relay is functional, Pathak's team allows sending these messages. This methodology for convincing spammers of an open relay also leads to the relay being blacklisted by projects such as Spamhaus. To counteract this, emails containing the strings DNSBL, ORDB, and a few others are not relayed.

Their open relay data collection approach identified two types of spammers: low-volume spammers (LVS), which appear in large numbers and use coordinated spamming at a low rate and low volume, and high-volume spammers (HVS), which have fewer nodes and send uncoordinated/disorganized spam at a very high rate of throughput. The LVS are considered more interesting because of their coordinated approach. They perform open relay scanning and distribute the open relays identified. The list of email addresses is also split into chunks and processed so as to avoid sending the same message to the same address multiple times. The chunking they observed is done alphabetically and is thus easily identifiable.

Perhaps the most interesting portion of the talk came in the discussion of a graph of email list chunk number versus time. This graph allows a systematic distinction to be made between the LVS and HVS types. The LVS spam increases linearly over time whereas HVS spam happens in one burst. Also, based on the observation that list chunking happens alphabetically, the graph also allows the separation of spam into spam campaigns.

Some interesting questions centered on the effectiveness of spam blacklisting. One audience member inquired as to the effect on observed spam when Pathak did happen to get the relay blacklisted. Pathak replied that upon blacklisting their open relay, spam stopped entirely, indicating that either spam blacklists are checked by spammers or that spammers constantly test open relays for efficacy. Other audience members inquired as to the amount of spam that is actually sent using open relays, given the automatic open relay blocking by Hotmail and other large email hosts. These questions couldn't really be answered, but work is being conducted now to better grasp how much spam employs open relays.

- ***Behind Phishing: An Examination of Phisher Modi Operandi***
  D. Kevin McGrath and Minaxi Gupta, Indiana University, Bloomington

Kevin McGrath presented his measurement study on phishing. His intention was to determine whether phishing URLs have differing characteristics when it comes to URL composition, registration, and cycle. He had two data sources: Mark Monitor, which is a list of phishing sites obtained from large ISPs that are verified by hand and updated every 5 minutes, and PhishTank, which has a list of community submitted and verified phishing URLs updated once every

hour. McGrath also obtained the zone files for the com, net, info, and biz top-level domains.

Their methodology for information gathering begins by obtaining a thin whois of the domain upon the domain's first occurrence. Then when the feed is updated, they fetch the DNS records for every domain seen to date to establish domain life cycle. They also perform geolocation via the IP2location service. Collecting these pieces of information over a period of 211 days allowed McGrath to establish several patterns in phishing domain characteristics. He gave details of the composition of phishing URLs. For example, over 30% of phishing domains are 8 characters in length, and the relative letter frequencies between phishing and nonphishing domains differ considerably. McGrath notes that the characters a, c, and e tend to appear with the same frequencies in phishing domains, whereas nonphishing domains follow the typical English frequency table. The more interesting observation is in the lifetime of a phishing domain, lasting approximately 3 days on average.

Someone inquired as to whether this study was really a characterization of phishing domains or whether it was simply characterizing data present in MarkMonitor and PhishTank. The answer is of course unknown as there is no global list of phishing sites, but it is an important point. An audience member also inquired about the incentive of domain name registrars to fix this problem, given that they receive money for these registrations. McGrath responded that registrars do not profit from typical phishing sites because of the 5-day registration grace period. If a domain lasts less than 5 days, no money is exchanged. This fact also yields a deeper understanding as to why the average lifespan of a phishing site is under 5 days.

NEW THREATS AND RELATED CHALLENGES

Summarized by Rik Farrow (rik@usenix.org)

Awarded Best Paper!

- ***Designing and Implementing Malicious Hardware***
  Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou, University of Illinois at Urbana Champaign

Sam King began with some history of similar attacks, as well as a mention of the recent sales of Chinese-made, bogus Cisco gear by contractors to U.S. DoD customers. King and his co-authors have designed the Illinois Malicious Processor (IMP), a SPARC v8 processor that runs Linux with a twist.

They implemented the IMP by adding a small number of gates (.05 and .08% of the total gates) in two different attacks, using a FPGA (Field Programmable Gate Array) programmed using a modified version of VHDL (Very High Speed Integrated Circuit Hardware Description Language) for the Leon3 implementation of the SPARC processor.

In one attack, a local attacker runs code that includes a sequence of bytes that gets detected by additional code in the logic of the data cache controller. When this trigger is seen, other added logic loads code and data into the L1 caches, executes this code, and elevates the privilege level of the process that sent the sequence of bytes as the trigger (instant root).

King also presented a second design, called shadow mode, where the trigger sequence appears in a dropped network packet, and the code to execute gets copied from the data portion of this packet. King described two attacks, one where login as any user is permitted with the password "letmein" after the trigger and a second that hooks read and write system calls and captures possible passwords. The login backdoor exits immediately after use, disappearing from cache, whereas the password capture code remains resident. The login attack has a small impact on performance (barely more than that of a local attacker logging in as root), but the password capture attack results in 13% loss in performance. King then demonstrated the login attack using the embedded system with the IMP version of the SPARC he had set up.

The first questions related to how easy it might be to discover this attack. Sergey Bratus mentioned that in the USSR, chips were routinely reverse engineered specifically to address this attack, and King countered by mentioning the CIA pipeline control software that was acquired by the Russians and caused a catastrophe when used. Another person wondered whether multicore processors would make this trick more difficult. King responded that the same changes could be used in all processors. Kevin McGrath suggested that special-purpose multicore systems might even make this attack simpler if you just target the one core you are interested in. Brandon Enright pointed out that the MMU or some other device might work as well, but King stated that the CPU got to see the entire dynamic instruction stream, making it better suited as a target for this attack.

- ***Catching Instant Messaging Worms with Change-Point Detection Techniques***
  Guanhua Yan, Los Alamos National Laboratory; Zhen Xiao, Peking University; Stephan Eidenbenz, Los Alamos National Laboratory

Guanhua Yan begin by explaining the issues with IM worms. Instant Messaging relies on servers for transferring messages, but the protocols permit file transfer directly between clients that a worm can use to infect another system without passing through any server. IM worms can also use a URL that points to a malware download site, also resulting in potential infection without passing through a central server.

The authors propose a statistical method that watches for the change-point in frequency of file-transfer requests or URLs being sent. They designed and tested, using simulated infections, two algorithms based on CUSUM, a sequential

analysis technique used for monitoring change detection. In their simulation, their algorithms were able to detect the presence of both aggressive spreading and self-restraining IM worms. The self-restraining worms would be designed specifically to avoid detection by throttling infection attempts below a threshold.

Niels Provos asked how computationally expensive their algorithms are. Yan answered that the performance scales linearly because you can keep track of past values for total file transfers or URLs included. Provos also asked about the computational complexity, and Yan said that their algorithm is $O(n^2)$ and is practical for up to 100 internal users. Someone else observed that social intimacy in IM is very skewed, with most conversations with 1.9 buddies over a month in AIM, and 5.5 in MSN, so worm propagation could be detected more simply by noticing abrupt changes in social intimacy. Someone else asked whether all clients could become infected during the five-minute window used in the experiment, and Yan responded that only a fraction of clients were infected in five minutes. Angelos Keromytis and Niels Provos wondered whether network intrusion detection that watched for patterns in data would work as well. Yan pointed out that this approach is statistics-based. The session chair ended the discussion at this point.

- ***Exploiting Machine Learning to Subvert Your Spam Filter***
  Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I.P. Rubinstein, Udam Saini, Charles Sutton, J.D. Tygar, and Kai Xia, University of California, Berkeley

Blain Nelson proposed techniques for preventing spammers from poisoning Bayesian spam filters. Bayesian filters must be taught the difference between ham (good email) and spam. The spammers do this by creating emails that will be classified as spam, for example, by including the words "replica Rolex" in the subject, then including a large number of nonspam words into their message. The goal is to cause the spam filter to misclassify ham (nonspam), and thus force the adjustment of the spam threshold so that more spam gets through the filter. Another possible goal would be for an attacker to cause an email, for example a bid, to be misclassified as spam. For example, sending the most common 90,000 tokens from Usenet postings (a set that includes both common misspellings and slang) increases the misclassification rate to 36% when just 1% of the mail is used for training the SpamBayes to recognize spam.

The authors suggest the Reject On Negative Impact (RONI) defense, where any email message that causes the SpamBayes filter to begin to reject a set of known ham messages must not be included in the spam learning set. This approach works well against dictionary attacks, but not against focused attacks. The authors also used a second technique, in which the thresholds for ham and spam get adjusted dynamically.

Jaeyeon Jung asked about the spammer sending multiple messages instead of just one, and Nelson responded that that method results in less impact, so many more messages are required. Someone else asked whether this was why spammers were including blocks of valid text in past spam, and Nelson answered that it is not clear why spammers were doing this in the past, but if you have enough tokens, the effect would be one of poisoning SpamBayes. Another person asked about excluding just some tokens instead of entire messages, and Nelson said they hadn't looked into that, leading to some discussion. Brandon Enright suggested defending against this attack by using bigram (word pairs). Nelson answered that they were looking into doing that. Sergey Bratus wondered whether they check if the message is actually read or not in deciding on using it in training; Nelson responded that they did consider some work like this. As the workshop broke up for lunch, a small crowd gathered around Nelson.

MEASUREMENTS, UNCERTAINTIES, AND LEGAL ISSUES

Summarized by Rik Farrow (rik@usenix.org)

- ***Conducting Cybersecurity Research Legally and Ethically***
  Aaron J. Burstein, University of California, Berkeley, School of Law

Aaron Burstein began his talk with a disclaimer. "Nothing in this presentation constitutes legal advice. If this was legal advice, it would be followed by a bill." He then went on to explain the U.S. legal landscape that impacts security researchers. The DMCA (Digital Millennium Copyright Act) has no research exception, for example. For researchers interested in capturing network traffic, the relevant laws are:

- Wiretap Act: Prohibits real-time interception of the content of electronic communications; the distinction between content and noncontent is vague, with the To and From lines being noncontent, but the Subject line of email is considered content.
- Pen Register/Trap and Trace statute: Prohibits real-time interception of noncontent portions of electronic communications.
- Stored Communications Act (SCA): Prohibits providers of "electronic communications service to the public" from knowingly disclosing the contents of customers' communications.

All three of these acts include loopholes that allow the providers of a service to monitor and capture network data. In the cases of Wiretap and Pen/Trap acts, providers may capture whatever content or noncontent they want as needed to protect the "rights and property" of the operator. In the case of the SCA, the operator can use stored data within the organization however they want. But in all of these cases, handing over this data to a researcher could be illegal.

The Wiretap Act and the SCA both came before widespread computer networks, and the Electronic Communication and Privacy Act (ECPA) and Computer Fraud and Abuse Act (CFAA) were written later. Burstein then presented two

scenarios. In the first, a researcher approaches a commercial ISP and asks for packet traces. Burstein points out that this would be covered by the ECPA and that there are no research exceptions. At this point, I asked about ISPs who share content and noncontent data with advertisers so the ISP can insert ads into email and Web browsing. Burstein said that this is allowed under the law. Someone else asked about having a student who works for an ISP during the summer. Burstein thought this would work, as long as the student did not remove the data from the ISP. Even continuing to use a login account to view logs later appeared to be okay.

In the second scenario, a researcher is capturing malware, allowing it to infect a sandbox, then watching what the malware does on the network. Note that this is similar to what Polychronakis et al. did in their paper, except that they prevent the malware from infecting other machines and captured all communications. Burstein said that if the researcher permits the malware to send out code and or data that infects systems not under the researcher's control, that would be in violation of the CFAA. He noted that the CFAA does not ban malware, that communicating with any external system was problematical, and sending out malware or even certain data (the CFAA specifically prohibits the sending of stolen passwords and financial data) runs afoul of the law.

Burstein concluded by saying that researchers should work closely with their own network administrators, as they can then work to help protect the rights and properties of the network owner while having legal access to network content and noncontent. He suggested both legal fixes, as well as working toward best practices and a code of conduct.

Someone asked whether a researcher has a duty to report certain content, and Burstein pointed out that the ECPA does allow you to report certain things. In some cases, such as discovering child pornography, you have an obligation to report, and running crawlers can put you into serious jeopardy.

- ***Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm***
  Thorsten Holz, University of Mannheim; Moritz Steiner, University of Mannheim and Institut Eurécom; Frederic Dahl, University of Mannheim; Ernst Biersack, Eurécom; Felix Freiling, University of Mannheim

Thorsten Holz presented more work related to Storm, and as he did so, it quickly became apparent that groups of researchers had actually been interacting via the Storm in an unexpected manner that has inflated the reported size of Storm botnets. Storm uses P2P for commands and updates, but it also communicates with a list of servers, so it is a hybrid. The P2P portion uses Overnet, and by crawling Overnet, Holz and his co-authors discovered 45,000–80,000 Storm bots at different times. They send out probes every 30 minutes, whereas the UCSD group (Kanich et al.) sends probes every 15 seconds.

Holz reported that Storm infections tripled over the Christmas to New Year week of 2007 because of successful social engineering attacks. Fabian Monrose asked why the numbers go down sometimes, and Holz replied that events such as MSRT sending out a patch can result in systems becoming clean. Then Holz stated that they introduce 2^{24} hashes (16 million) to the P2P system (the hashes being used to locate bots), and Niels Provos immediately asked whether this could inflate the number of discovered Storm bots. Holz said this certainly could, and someone else said "That's you!" Holz went on to mention that they had also experimented with disrupting Storm. One method relies on introducing sybils, malicious peers under the control of the researchers, that can be used to spy on traffic and abuse the network in other ways.

Through their crawling of P2P and their sybils, Holz claims to have seen between a minimum of 5,000–6,000 and a maximum of 80,000 Storm bots per day. David Dagon, the session chair, suggested that perhaps researchers need to set up a Storm users list. Someone else asked why they don't see the 16 million nodes represented by the hashes Holz injects into the network. Holz responded by saying they are using only two IP addresses. Someone else mentioned that researchers need to be consistent in their methods, so they aren't tripping over one another while researching Storm. Brandon Enright of UCSD (another Storm researcher) expressed concern that the Storm authors might stop using Overnet (the P2P network that Storm relies on), and Holz agreed. You can learn more about Storm from previously published articles in the December 2007 issue of *;login:*.

- ***The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff***
  Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, and Stefan Savage, University of California, San Diego

Chris Kanich described the UCSD team's work in determining the number of active Storm participants and succeeding in outing another researcher active in crawling/poisoning the Storm botnet. Kanich pointed out that the number of claimed Storm bots is extremely high, with MSRT reporting a lower bound of 275,000. Kanich reported that research groups, as well as competitors to the Storm botnet, have been very active, and that this has inflated the number of nodes.

Storm uses Overnet, a P2P file-sharing network based on the Kademlia DHT algorithm. The UCSD team discovered an error in the generation of unique object IDs (OIDs) used by Storm, limiting the total number of OIDs to 32k (2^{15}). This does not place an upper bound on the number of nodes, as not all nodes will communicate, but it does make the OID itself into an oracle that can identify a true Storm infection as opposed to a file-sharing client or another research crawler. The UCSD team built a tool named Storm-

drain that implements a state machine for categorizing Overnet nodes. Potential Storm nodes are only considered Active when they actually respond, and nonresponding systems are moved into a Removed state, then quickly into a Dead state, over a short period of time.

Someone asked about dynamic IP address, and Kanich replied that they don't care about this, as they are only interested in the instantaneous number of nodes. Someone else pointed out that Kademlia should time out old peers, but Kanich reported that Storm's implementation is broken, and its K buckets are not recycled every four hours as they should be. David Dagon noticed a spike in a graph and asked when that occurred. Kanich replied, "March 10," to which Dagon said, "I owe you a drink." Kanich described improvements in Stormdrain, such as advertising OID hashes that are "close" to recently advertised peers, and this increased the proportion of nodes considered Active rather than just Live. Gary Warner wondered whether the Storm nodes could be distinguished from Overnet nodes based on the command set used, and Kanich replied that although they didn't do that, it should work.

During three weeks of Stormdrain crawling in March 2008, the number of active nodes varied between 8,000 and 23,000 Active nodes. David Dagon asked whether the UCSD group would be willing to coordinate with his groups in probing, and both Kanich and Brandon Enright said they would be willing to communicate with other researchers.

An article on Storm begins on page 6 of this issue.

- ***Ghost Turns Zombie: Exploring the Life Cycle of Web-based Malware***
  Michalis Polychronakis, FORTH-ICS; Panayiotis Mavrommatis and Niels Provos, Google Inc.

Michalis Polychronakis presented this paper, which expands on work published last year at HotBots about drive-by downloads. Drive-by downloads involve Web pages that have been modified to include script or iFrame sections, resulting in the installation of malware on systems, currently focused on Windows. In this work, the researchers monitored attempted communications after infection, analyzing over 448,000 responder sessions. Polychronakis said that they found that malware reports information about the infected system, address books, browser history files, stored passwords captured by keyloggers or browser hooks, and attempts to join botnets.

Their setup used Windows systems running within VMs that were passed a URL suspected of causing drive-by downloads. To capture outgoing connections, they set up a number of proxies for known protocols, as well as generic responders that often worked, even though the actual protocol was unknown. The generic responder looks for hints to the protocol when a nonstandard port is used, then emulates that protocol if known. If unknown, a generic banner gets sent to the malware if there is no activity after a number of seconds, and this often resulted in a useful response. Besides connecting to servers that collect stolen data, malware often portscanned local networks, looking for Windows services such as SMB, NetBIOS, MSSQL, and DCOM.

McGrath asked whether some requests to nonstandard ports were using HTTPS, and Polychronakis replied that they generally were not using that protocol. Jaeyeon Jung asked how many types of malware were seen; Polychronakis responded that they didn't analyze which malware family was sending traffic as they couldn't perform analysis on so many infections. Someone else asked about the capacity of their system, and Polychronakis said they could check about a million pages a day using their setup. John Ramsey mentioned they had developed Caffeine Monkey, which does some URL analysis. Then he asked whether the malware was encrypted or packed. Niels Provos answered, that most is at least obfuscated and a lot of the Javascript is encrypted. David Dagon asked whether the malware tests to see whether it is running in a VM or in an emulated environment. Provos responded that malware download servers won't even respond to requests from IP source addresses known to belong to researchers' networks. But they have not seen malware that appears to be aware that it is running within a VM.

WORK-IN-PROGRESS REPORTS

Summarized by Joshua Mason (josh@jhu.edu)

Will Drewry presented a methodology for fuzzing regular expressions. Although the methodology was not discussed in detail, their results were quite impressive. Their fuzzer has so far led to 15 security advisories, with 3 or 4 causing code execution. The impact of the methodology is intriguing because of the number of applications affected by the regular expression engines they broke. Their advisories affect applications such as Adobe's Flash Player, Apple's Safari browser, Adobe Acrobat Reader, and Postgres SQL. Adobe Flash alone is one of the most prevalent pieces of client-side software on the Internet today, with over 98% market penetration. They intend to publish the source for the fuzzer, which will hopefully lead to more secure regular expression engines in the future.

Gary Warner from the University of Alabama at Birmingham presented an ongoing work aiming to gather an unprecedented amount of spam. He presented techniques he is currently employing, such as asking for the MX record for popular domains without a mail server and voluntarily submitting their email addresses to email address farming malware. Warner's team is also attempting to get an "opt-in" plug-in for SpamAssassin that would, if a user agrees, have all the user's spam sent to their spam collection project. The intended uses for the captured spam are numerous; he briefly discussed using some data-mining algorithms to attempt day-to-day spam campaign tracking.

Rick Wesson from Support Intelligence presented an ongoing Internet mapping project. They use software from measurementfactory.com to map live portions of the Internet. The point of the project is to employ visualization tools to establish trends present online. Data gathered can potentially be used for a variety of applications, such as establishing malicious segments of the Internet.

David Dagon presented a project he's working on that he calls "memory dumpster diving." He intends to use his technique to perform automated memory analysis on malware. This would spare malware analysts from performing the arduous task of constantly having to reverse engineer new instantiations of the same general bot software to obtain required information such as encryption keys or connected hosts. His platform would perform run-time analysis to dump what seem to be relevant portions of memory, so the analyst can simply take the information he wants out of the memory trace.

Thorsten Holz presented a measurement study he and Frederic Dahl are working on that gathers data on DDoS attacks launched by the Storm worm. So far, it seems the Storm worm's attacks last an average of 90 minutes at 61 packets per second and are typically against either individual users or anti-spam/anti-spyware companies. He also very briefly covered some new reverse engineering they were able to do on the Storm networks' encrypted communication. They obtained the RSA key and can now encrypt messages to Storm nodes to make them connect to arbitrary hosts.

BSDCan: The BSD Conference

Ottawa, Canada
May 16–17, 2008

OPENING SESSION

Summarized by Bjoern A. Zeeb (bz@FreeBSD.org)

"BSDCan 2008, welcome back" was Dan Langille's first slide. But before telling you about all the conference talks let's go forward to the closing session to tell you one reason why these summaries were written.

Dan's Rules of Conference:

1. You do talk about conference.
2. You DO talk about conference.
3. You shall not stand in a direct line between TV and Dan during an NHL game at conference.

In case you ignore rule #3 you'll find out about #4, but that is left as an exercise to the reader.

So the opening talk started with a screensaver of lots of Nigeria s(c|p)am asking for letters of invitation. Would you have imagined this happens to an organizer of a conference? Dan continued thanking all the sponsors, talked about the organizational things, and gave his talk, a summary of what happened to him during the past year. It is the personal touch that makes this special every year.

FREEBSD/MIPS, EMBEDDING FREEBSD

Summarized by Bjoern A. Zeeb (bz@FreeBSD.org)

Warner Losh

Warner Losh began talking about the long history of FreeBSD/mips starting in the late 1990s with FreeBSD 3.x. The second try to bring it into the mainstream started in 2002 and the third one at BSDCan 2006, which led to more community success in getting to single users on real hardware. In 2007 Juniper released code that was later merged with the mips branch and gets to multiusers. FreeBSD/mips is self-hosting now. Today mips32/r2 and mips64/r2 are supported and FreeBSD runs on at least four SoC families: ADMTek ADM5120, IDT RC32432, Broadcomm MIPS, and the MIPS 4Ke core. More are to come soon. Currently the work is merged from the Perforce repository into mainline CVS.

Warner Losh then gave an update on the embedded FreeBSD world. Two Google Summer of Code students will work on a PowerPC port and on further reducing the footprint size of embedded FreeBSD. Both the PowerPC and ARM support develop well and there is more and more support in the repositories. He closed with an outline of future projects.

RESOURCE-LIMITING ON THE VIRTUAL PRIVATE SERVER

Summarized by Mathieu Arnold (mat@FreeBSD.org)

Fred Clift, Verio

NTT-Verio uses FreeBSD extensively on its virtual hosting services. They started at about the same time as jail(8), but they added a lot of things to it, such as limits, similar to rlimits, with io, network, mail inject, and syscall rate limit. Those limits were needed to have numerous jails without one taking all the resources. All those limits have clever algorithms that allow bursts, so that the virtual servers feel responsive when needed. They're waiting on the lawyers to release the code, which they have on FreeBSD 4, 6, and 7. Slides with nice graphs are available at http://clift.org/fred/bsdcan2008.pdf.

A CLOSER LOOK AT THE ZFS FILE SYSTEM/ZFS, THE INTERNALS

Summarized by Mathieu Arnold (mat@FreeBSD.org)

Pawel Jakub Dawidek

Pawel started by peeling ZFS like an onion, explaining how the components talk to each other and what they do. The best way to get an idea is to look at his slides at http://www.bsdcan.org/2008/schedule/attachments/58_BSDCan2008-ZFSInternals.pdf. He then explained

how RAID-Z is similar to RAID5/RAID6 but so different. The problem is that ZFS is both the filesystem and the RAID system, so it knows what to write and in what order. It's also self-healing because everything on disk is checksummed, so if it reads parity that's bad or data that's bad, it will rewrite what should be back to the disk. Also, every write to the disk has a transaction number, so a hard drive that's only been temporarily unavailable will only need to synchronize changes. As for snapshots, well, ZFS does copy on write; so it already does snapshots behind your back: taking a snapshot is O(1). Pawel fielded several questions, mainly about "how do I do that and get it all right," stating that adding disks is easy and replacing disks with bigger ones is almost as easy.

MEASURED (ALMOST) DOES AIR TRAFFIC CONTROL

Summarized by Mathieu Arnold (mat@FreeBSD.org)

Poul-Henning Kamp

Poul-Henning Kamp explains how he helped the Danish Air Traffic Control monitor 30-year-old navigation devices (DME, VOR, etc.). He set up devices based on NanoBSD (stripped down FreeBSD configured to run on a compact flash card) to run the MeasureD daemon. The daemon gets information in many different ways, mostly through obscure serial protocols (almost reverse engineered) from all the devices, transmitters, fuel gauges, battery voltage levels, and so on. MeasureD multiplexes all it gets and incorporates a small HTTP daemon in which you can get all the data you want. (Another MeasureD can also get the data, say, from a central location.) There is also a bsnmp plug-in so you can get MeasureD data from SNMP. All that data has to be logged somewhere, and with the storage device being a flash card it has to be done in a sensible manner. Kamp wrote fifolog, which time-stamps and multiplexes its input, compresses it, and stores it regularly (with padding if the compressed size does not fill a data cell). It can be grepped through pretty easily with the fiforead command. See http://www.bsdcan.org/2008/schedule/attachments/64_BSDCan2008-AirTrafficControl.pdf for more details.

SCTP

Summarized by Florent Parent (florent.parent@beon.ca)

Randall Stewart

SCTP, or Stream Control Transmission Protocol, is a new transport protocol that sits above IPv4 and IPv6, at the same layer as TCP and UDP. SCTP has been standardized by the IETF in 2000 and is available in FreeBSD 7.0.

In this talk, Randall Stewart presented an overview of SCTP. New features offered by SCTP are four-way handshakes (which help reduce DoS attacks), framing (which is used to preserve message boundaries), multistreaming, multihoming, and reachability.

SCTP multistreaming allows logical separation of data within an association. Stewart demonstrated video multistreaming using a Web page download containing multiple pictures. The download was done over a path exhibiting packet loss. SCTP took about half the time that TCP took to do the same thing.

The SCTP socket API was also presented. The API offers two socket types: one-to-one (STREAM) and one-to-many (SEQPACKET). The one-to-one interface offers backward compatibility with TCP sockets. The one-to-many offers a UDP-like interface and is the type of interface a peer-to-peer application would use.

Overall, this talk was an excellent introduction to the benefits of SCTP. The Web site http://www.sctp.org is SCTP equipped.

PORTING FREEBSD/ARM TO MARVELL ORION SYSTEM-ON-A-CHIP

Summarized by Mathieu Arnold <mat@FreeBSD.org>

Rafal Jaworowski, Semihalf

System-on-a-Chip (SoC) is an integrated chip with many things integrated, such as the main CPU, GPIO (General Purpose I/O), Ethernet, UART, PCI, USB, SATA, Crypto, and SPI. The FreeBSD/ARM port to that SoC family is working pretty well, with new chips being added every now and then and much work still not committed to the FreeBSD source tree. For more information see http://www.bsdcan.org/2008/schedule/attachments/50_2008_marvell_freebsd.pdf.

GOOGLE SUMMER OF CODE

Summarized by Mathieu Arnold (mat@FreeBSD.org)

Leslie Hawthorne, Google

Leslie Hawthorne is the Project Manager of the Google Summer of Code program. She came here to explain how the idea came into being at Google—"Here, we have too much money; let's spend some"—and also why they're not evil—"We're giving you money and ask nothing in return." Since the beginning, the Summer of Code project has involved participation from 1500 students, 2000 mentors, 175+ open source projects, and 98 countries, and more than 10 million U.S. dollars have been spent on students and projects. Details can be found at http://www.bsdcan.org/2008/schedule/attachments/52_LeslieHawthorn_bsdcan2008.pdf.

UP CLOSE AND PERSONAL WITH TCP IN FREEBSD

Summarized by Florent Parent (florent.parent@beon.ca)

Lawrence Stewart

Lawrence Stewart presented a look at a new modular TCP congestion control framework. The talk started with a review of the current state of TCP. (RFC4616 is a good summary of TCP-related RFCs.) Today, NewReno TCP is the de facto standard and is used in BSD and many other OSes. But there are still open issues in high-speed networks, where improvements to current congestion control protocols are required.

Many new TCP congestion control protocols have been proposed, and some of them are being used in Linux (CUBIC) and Windows Vista (CTCP). A new modular framework for TCP congestion control (CC) protocols has been developed in FreeBSD. This framework allows the support for new CC protocols in addition to NewReno.

Lawrence Stewart showed a demo where the TCP congestion control protocol selection was done using the sysctl command.

The tool SIFTR (Statistical Information For TCP Research) was presented. SIFTR is a kernel module that logs to a file different statistics on TCP connections. This tool helps the development and testing of new protocols. Many other tools are used, such as dummynet, iperf, tcpstat, R, and ns-2.

There is still work and testing to be completed, and there is a plan to share the congestion control protocols between TCP and SCTP. The home page for this work (including code) is found at http://caia.swin.edu.au/urp/newtcp/.

OPENBSD HARDWARE SENSORS FRAMEWORK

Summarized by Constantine A. Murenin (cnst@openbsd.org)

Constantine A. Murenin

Constantine is a math graduate student at the University of Waterloo, a committer at OpenBSD, and a Google Summer of Code 2007 (SoC2007) student at FreeBSD. During this Invited Talk, an overview of the sensors framework was presented, including the recent developments in the driver arena and the Google SoC2007 experience of the speaker.

The talk started with a brief introduction to the main ideas behind a unified sensor framework and followed with some numbers representing the pervasiveness of the framework within OpenBSD: At the end of March 2008, a 64th driver utilizing the interface was committed into the code tree, representing an anniversary of some kind, and at the time of the presentation the number of drivers calling sensordev_install(9) was as high as 67. Some of these drivers are unique to OpenBSD and are not yet available elsewhere; for example, Theo de Raadt has recently added support for the JEDEC JC-42.4 SO-DIMM temperature sensors, and Constantine has provided support for the temperature sensors embedded in AMD Phenom and Opteron Barcelona processors (neither of which is yet available in the Linux lm-sensors package).

The rationale behind the framework design was explained, with the primary objective of the API being "simple, secure, and usable." An example was given on how the voltage sensors work in the hardware monitoring modules of most popular Super I/O solutions, where it is often the case that it is impossible to know the true relationship between the sensors and the power lines of the power supply unit, so an overengineered framework isn't likely to be beneficial for most simple drivers aimed at being usable by default. An overview of the API and of the userland utilities was presented. Userland utilities include sysctl, sensorsd, ntpd, systat, snmpd, and ports/sysutils/symon.

Constantine then proceeded to describe his experience with porting the framework to FreeBSD, sponsored by the Google SoC2007 program. Most popular parts of the framework were ported, and a complete patchset was publicly released on September 13, 2007, but the FreeBSD CVS HEAD tree was frozen at the time because of the then-upcoming RELENG_7 branching, limiting the integration of the new components into the tree.

In the meantime, the framework was committed to DragonFly BSD by Hasso Tepper, who within a few days of the posting adapted Constantine's patch for inclusion into DragonFly. A few days later, Constantine's patch was approved by the FreeBSD Release Engineering team to be committed into FreeBSD after RELENG_7 was branched, and on October 14, 2007, it was committed into the FreeBSD CVS repository by Alexander Leidinger. The commit has generated a lot of attention in the FreeBSD community, and some people suddenly felt that the framework itself was designed only with the OpenBSD architecture in mind and didn't have a FreeBSD feel. The framework was then backed out from the FreeBSD CVS tree within a few days upon a request from Poul-Henning Kamp, who perceived it as not being architecturally fit for FreeBSD, even if it may be appraised in OpenBSD, as FreeBSD and OpenBSD have different architectural goals in mind, although Poul-Henning specifically clarified on the mailing lists that the SoC2007 porting itself was done to his satisfaction.

Poul-Henning was present in the audience, and during the comments-and-questions portion of the talk he clarified that he doesn't want the code being imported into FreeBSD so that FreeBSD could have a clear space in the area and someone could implement a framework more suitable for FreeBSD sometime in the future. Given that the framework in question was based on a framework that was available in NetBSD since 1999, Constantine and Poul-Henning agreed that designing and implementing a sensors framework perfect for FreeBSD with usable drivers may not be easy (especially considering the often-inadequate hardware specifications in the area).

X.ORG

Summarized by Constantine A. Murenin (cnst@openbsd.org)

Matthieu Herrb

Matthieu Herrb is an OpenBSD X Window System developer and a member of the Board of Directors of the X.Org Foundation. In this Invited Talk, Matthieu provided some historical insights into the development of the X Window System, from the late 1980s and the time of the monochrome and 8-bit color framebuffers, to the present time of Direct Rendering Infrastructure (DRI) and anti-aliasing support.

Major administrative milestones were briefly covered (e.g., the XFree86 license fiasco and the creation of the X.Org Foundation). Since the establishment of the foundation in 2004, the two most visible changes to X were the conversion of the build system to a modularized design and the embracement of git, the distributed version control system.

The interaction between various components of the windowing system were described, both the direct and indirect rendering modes, as well as plans for the future regarding DRI2 and Gallium 3D, which aim at making the Direct Rendering Manager simpler and more in line with the architectural characteristics of modern 3D hardware.

Matthieu also summarized major changes in X.Org 7.3, which was released in September 2007, and future plans for the upcoming 7.4 release. For example, in late 2007 and early 2008, AMD has released free programming documentation that made the new radeonhd driver possible, and 7.4 will be the first release to include this new driver, supporting ATI Radeon HD chipsets.

Last but not least, the situation with X around the BSD systems was covered. OpenBSD 4.3 includes X.Org 7.3, and work on the DR is underway based on the code from NetBSD. However, NetBSD is the only significant system that continues to ship XFree86 releases in its base system, although X.Org is also available through pkgsrc, a managing facility for third-party packages. One interesting problem that OpenBSD and NetBSD face is on the legacy architectures front, where some architectures have limited processor and memory resources and sometimes lack shared library support, making it increasingly difficult to support newer X releases on such machines.

BSD LICENSED C++ COMPILER

Summarized by Constantine A. Murenin (cnst@openbsd.org)

Chris Lattner, Apple, Inc.

Chris Lattner is the chief architect of the Low Level Virtual Machine Compiler Infrastructure, currently managing the LLVM group at Apple Inc. In this Invited Talk, Chris provided an outline of the technological advances of the BSD-like licensed LLVM compiler suite.

In a nutshell, the LLVM project consists of the language-independent optimizer and code generator, llvm-gcc 4.2 front-end, and the clang front-end. In the introduction to the talk, Chris described why there is a need for a new compiler technology, which, for someone somewhat familiar with the GNU Compiler Collection, wasn't all that surprising: GCC keeps getting slower with every release, cannot be easily reused in other applications, and is bloated to the point where it is quite difficult to read and modify the code. LLVM, however, takes a modular approach, where more components can be shared across different compilers and processor architectures.

The GCC 4.x design was highlighted, and this was followed by an explanation of how the llvm-gcc 4.2 is designed to work: llvm-gcc is a drop-in replacement for the gcc and uses the GCC front-end with the LLVM optimizer and code generator. Chris has reported that not only are the LLVM optimizer and code generator faster in such a GCC compound (30% improvement at -O3) but they also produce better code (5% to 10% improvement on an x86/x86-64); moreover, they allow some interesting applications, such as just-in-time compiling, optimization of the C/C++ code, and generation of the executable code at the install time.

Special attention was devoted to one other topic, namely, LLVM on the OpenGL front. Mac OS X 10.5 provides mechanisms for colorspace conversion, code for which has hundreds of conversion combinations among the color formats, and patterns can be applied to the input and output in the "case" statements inside a couple of "switch" statements inside a "for" loop for every pixel. For such an example, run-time optimization can greatly improve the performance, with 5.4x being the average improvement and 19.3x being the maximum speed-up, depending on the source and destination color formats. Some insights were also given regarding the Mac OS OpenGL state before LLVM came about. Chris was happy to note that no polygon can get onto the screen in Mac OS X without LLVM.

The next big part of the talk was a presentation on clang, a front-end for C, C++, and Objective-C. In comparison, GCC, apart from being slow and memory-hungry, doesn't serve the needs of various IDE applications, such as indexing of the code for "jump to the definition" features or "smart-editing." One of the most significant problems with GCC front-end is, however, the limited information that is usually provided when the compiler encounters errors in the source code. Since clang always keeps the information about the columns where the errors occur, error messages explicitly contain not only the line but also the column, as well as providing ASCII graphics of the exact point on the line where the errors occur, accompanied by a more meaningful error message than the GCC usually offers. This feature was very well received by the audience.

INTRODUCTION TO DEBUGGING THE FREEBSD KERNEL

Summarized by Bjoern A. Zeeb (bz@FreeBSD.org)

John H. Baldwin, Yahoo! Inc.
John Baldwin started with an overview of the various places to find documentation on the subject. He continued by showing how to use the interactive kernel debugger for investigating deadlocks. For the developers in the audience he talked about how to enhance ddb(4) by adding new commands. The next section was on the kgdb(1) debugging kernel modules and scripting by adding user-defined commands. He closed with a summary on the different strategies for debugging kernel crashes versus system hangs. Slides and a paper with more information are available online at http://people.freebsd.org/~jhb/papers/bsdcan/2008/.

WIPS

Summarized by Mathieu Arnold (mat@FreeBSD.org)

* Ivan Voras on mdcached: It is a caching daemon, much better than memcached, optimized for multicore servers, and very fast; you can put tags on data and search with tags, which seems to be a handy idea.

* Frank Pikelner on Versiera: This multi-OS server management software, developed by Netcraft, seems nice.

* Philip Paeps on syscons and other scary things: Philip says he came into device drivers because of a touchpad that was working rather strangely. He's also going to break syscons in the upcoming months by taking it apart and separating it from the framebuffer and ttys.

* Peter Losher on IPv6 and the root name servers: We'll be out of IPv4 in two years, so get used to it, but 6 of the 13 root servers have had IPv6 for quite some time now, and those IPs have been added to the root zone file.

* Bjoern Zeeb on multi-IPv4/IPv6/no-ip jails: There have been multiple patches for multiple IPv4, vimage, IPv4+IPv6, and jailv6, and some things are moving along nicely. Ultimately, we'll have DDB and SCTP support. The current system is pretty light, and it works in production. Things to do include source-routing; cpuset selection; adding a name to the jail so that it can be put in ps, for instance; support for bsnmpd; and adding resource limits.

* Zachary Loafman on FreeBSD at Isilon: They have a distributed filesystem called OneFS, sponsored the work to have NFSv3 locks working, and have tons of other things (as shown in too many interesting slides with no time to take notes), but they lack time to contribute them and are hiring and willing to sponsor projects.

* Mark Linimon on Bugathons + BugBusting BoF: Bugathons bring volunteers, and it really helps to categorize PRs; volunteers bring in lots of fresh blood and also many fresh interesting ideas, so if you want, you can help too.

* Julian Elisher on Vimage, MRT: The kernel modules will be virtualized one after another.

* George Neville-Neil on network testing: TCP is king, in general, and peasants like multicast and UDP get much less testing. The network test utility mctest, with sources and sinks, is in src/tools/tools/mctest. PCS is another network tester and has been improved; this year it's a Google SoC project.

* George Neville-Neil on XEN: HEAD works with Xen 3.1 and 3.2 in perforce, and Xen 3.0.3 is in the pipe for Amazon EC2. It'll happen right after FreeBSD switches to svn, and it will support 64-bit architectures.

* Julian Elisher on multiple routing tables: He showed two big schemas with lots of structures, pointers, and other stuff; one is bad because the API changes too much; the other is good because it does not change that much.

THE VERY END

Summarized by Bjoern A. Zeeb (bz@FreeBSD.org)

If you want to find out how Dan felt about the conference, go to his blog at http:/dan.langille.org/ and check for "Three Weeks in the Life of a Conference Organizer." Thank you, Dan, for another fantastic BSDCan! Is this the end? No. Do you want to know what else happened those days? You had better come and find out yourself next year. See you at BSDCan 2009, when it will be the biggest OS(S) conference in town.

BSDCan 2008 FreeBSD Developer Summit

Ottawa, Canada
May 14–15, 2008

Summarized by Bjoern A. Zeeb (bz@FreeBSD.org) and Marshall Kirk McKusick (McKusick@McKusick.com)

INTRODUCTION

A developer summit is a get-together of some FreeBSD developers to present recently finished work and to talk about ongoing work or future plans. Although this is not a place to make decisions concerning the entire project, there are usually enough of the key developers present that a lot of design issues get hashed out.

Developer summits are often aligned with a BSD conference, as many developers are going to be there anyway. The summit also provides an opportunity for people from companies using FreeBSD or other FreeBSD-derived projects to attend. This mix of developers and users presents a great opportunity for the developers to get direct feedback on their work and for the users to gain an understanding of likely future developments.

This report gives our thoughts on the developer summit events that we attended. For more complete details and often the slides used by the presenter, see the Developer Summit wiki page, http://wiki.FreeBSD.org/200805DevSummit.

The format of the summit was to have formal talks in the morning, with afternoons devoted to free-for-all discussions by smaller, self-selected groups on various specialized topics largely selected at the conclusion of the morning talks. Most of this report will focus on the morning sessions, as their content is more easily summarized.

SUMMIT DAY 1

The day opened with a welcome message from Robert Watson, the main organizer for the summit.

Poul-Henning Kamp started with a general discussion about GreenBSD. This project will not be a fork of FreeBSD, but it might become a new, large-scale project encompassing large parts of the system. The main thrust of GreenBSD is "green computing," that is, reducing power consumption whenever and wherever possible. Examples include turning down gigabit interfaces to 100 Mbit/s if there is no need for more speed, entirely shutting down unused NICs, or reducing clock speeds and voltages on otherwise idle CPUs. The key is to figure out the correct abstraction and support functionality so that it is not necessary to endlessly duplicate code in each device driver.

Next, Ivan Voras spoke about his Google Summer of Code 2007 project, finstall, a graphical installer that provides more automated decision making and provides interfaces to more parts of the systems such as geom, networking options, ZFS, and gjournal. It is split into a front end and a back end and is written in GTK and Python. The split allows finstall to implement a remote installation console. The back end can be used for non-base-system installs as well. At the moment it is still lacking a graphical partition editor.

During the question period Ivan was asked how much work would be required to replace the Python back end with a C implementation. Concerns were also raised about the front end using GTK because of it being LGPL. In particular, the discussion migrated to whether the front end could be replaced with a text-mode-only implementation in C and whether that would really just drop back to the current capabilities of the curses-based front end.

After a short break Alexey Tarasov spoke on his Google Summer of Code 2007 project, which involved kernel netbooting via HTTP. The work consists of a PXE API, a tiny TCP/IP stack, PXE sockets, etc. Unfortunately, there is no IPv6 support with PXE. There are basic filters to restrict IP/ports. The most important but also complicated task had been the user mode implementation of the tiny TCP/IP stack along with memory constraints (buffering issues). It uses httpfs provided by the boot loader including HTTP 1.1 support. DHCP, DNS client, ICMP echo, ARP, and boot console commands are already implemented. Work in progress includes a telnet client, socketfs, IPv6 support, and a possible parsing of HTTP server index pages.

Rafal Jaworowski was up next with an embedded-architecture status report: arm, MIPS, and PowerPC. The arm port is "almost" tier-1, as of the FreeBSD 7.0 release. More arm functions and features are in the pipeline. Unlike the other architectures, it is nearly impossible to have a single arm GENERIC kernel as there are so many incompatible variations of the arm architecture. Juniper Networks has been providing a lot of the MIPS support. FreeBSD 7.0 supports both the 32-bit and 64-bit MIPS architectures. As with the arm architecture, much additional MIPS functionality is in the pipeline.

The PowerPC is more of a work in progress, also being supported by Juniper. At the moment most of the PowerPC support is for the high-end chipsets and SMP support. Bridge mode is going toward 64-bit PowerPC support.

There are two Google Summer of Code 2008 embedded-systems projects. The first involves optimizing the build system for embedded systems. The second is to port to Efika, a cheap platform. Other embedded projects on the wish list are improving support for a flash-memory-based filesystem, an improved build system to support cross-building from Linux or Windows, and better system/kernel configuration for creating a smaller footprint.

Ed Schouten talked about reimplementing FreeBSD's tty layer. The tty system is the one part of the system that has not been rewritten to support SMP so still needs to run under the Giant lock. Ed started with a design overview, which involved the removal of the fragile clists buffer mechanism from tty. Among other things he is now destroying ptys when unused so that they do not clutter up /dev and consume kernel resources. He multi-threaded the transmit path buffering and eliminated the global buffer list. Best of all, he managed to keep all but sgetty ABI compatible. Still to be done is the (fairly mechanical) task of adding multi-threading support to all the serial devices.

This concluded the first day's formal presentations. After a lunch of pizza everyone broke up into smaller discussion groups. The ``Network Cabal'' started with Jeff Roberson, Julian Elischer, and Kip Macy on a redesign for mbufs. Jeff has a new concept that ref-counts mbufs so that they require less copying on forks. In addition he combines a clustered mbuf with a small mbuf header to increase efficiency when a copy of the mbuf needs to be retained, for example with TCP. The ongoing discussion was mostly with Sam Leffler and Robert Watson debating mbuf layout, mbuf tags, mbuf back traces, and techniques for tracking mbuf leaks.

Lawrence Steward led a discussion about TCP bug forensics. After introducing himself and a bit of TCP jargon and history he delved into congestion control algorithms. He explained which OS is using which algorithm and talked

about what parameters to look at when comparing the various different algorithms for high-speed connections.

Next he turned his attention to tools. Dummynet has problems. SIFTR is their tool to generate CSV-like information for later analysis. He showed how to use the data of the tool in three interesting case studies of problems they had found.

The next slot was debugging and profiling tools. John Birrell led the discussion by opening with a demo on DTrace. One of the important things that he stressed was that DTrace is not a debugger. One key thing to understand about DTrace is that it can always be compiled into the kernel as it has almost no overhead when it is not being used. Indeed the (only) overhead is NULL pointer checks for the DTrace modules. If the modules are loaded, there will be an additional bitwise AND to do a mask check. So, one can expect to have DTrace available even on production systems. The other key point that came up is that DTrace does not replace other tools such as ktrace and gprof.

The last big section of the first day was a presentation on network stack virtualization by Marco Zec and Julian Elischer. They explained how they had implemented loadable network-stack support. In short, all the formerly global data structures and variables had been gathered together into a dynamically allocated structure. In this way, multiple copies of the network stack could run in isolation; for example, each jail could have its own network stack on which to operate. This functionality allows each jail to run its own packet filter, raw sockets, ICMP, ALTQ, etc.

SUMMIT DAY 2

Adrian Chadd started off the second day of presentations by talking about TCP content- and service-provider hijacking. He discussed both malicious and deliberate hijacking and described the different methods and technologies used. He explained the various problems that arise when deliberate hijacking is done. The issues involved TCP options, screw-ups in MTU discovery, failures in properly setting TCP options, and the effects of an older implementation such as TPROXY.

Next Doug Rabson talked about his work on replacing the error-prone userland NFS Lock Manager with a kernel-based one; this led to a lot of cheering. He started with a basic overview of the different NFS versions (2/3) and undocumented newer stuff. FreeBSD locking used to be done in the old userland rcp.lockd, which lacked proper client-side locking. The new kernel-based rcp.lockd supports everything but DOS shares. It has kernel-mode implementations for both client and server. Local locking now supports asynchronous operation and there is a graph-based deadlock detection. He also implemented fairness for contested locks so that locks are handed out first come, first served. He even added regression tests to ensure that it works and continues to work. The kernel-mode implementation is now the default, but there are options in GENERIC so that you can opt out of the kernel-mode lock manager and fall back to the "old rpc.lockd."

Justin Gibbs provided a break in the onslaught of technical information by giving an update on the FreeBSD Foundation. He explained that the foundation was created to provide a way to channel money to fund development of unpopular parts of the system, as a way to build long-standing relationships with vendors and to provide an organization that can negotiate legal contracts. By providing a standalone organization rather than affiliating with an existing corporation they avoided any conflicts of interest. Thus, the FreeBSD Foundation is an independent corporation with management that is internally elected, and its activities are guided by its charter, which states its role as improving, nurturing, protecting, and evangelizing FreeBSD. In short, it is the "tie that binds" FreeBSD.

The FreeBSD Foundation is providing travel grants, event sponsorship, funds development, IP protection, legal and contract negotiation, and management of hardware donations. Challenges include knowing the user base, maintaining critical mass, finding funding for the FreeBSD platform, helping to define and set milestones, and growing the capabilities of FreeBSD.

Returning to the technical theme, Erwin Lansing talked about the task of the FreeBSD port manager. He gave some numbers and statistics and went into the details of the port monitoring software. As there are now over 18,000 actively maintained ports, many manual tasks have to be handled automatically. There is software for tracking problem reports, detecting maintainer-timeouts, ensuring that packages compile on all platforms, determining package dependencies, etc. This summer three Google Summer of Code students will work on the ports infrastructure.

Robert Watson stepped up the pace of discussion with his talk on TCP scalability in the presence of 16-core SMP systems. He started with the big picture: MPSAFEness, Giant free, and improving multi-threaded workloads. He went on to describe UDP problems: throttling by exclusive write locks, and excessive overhead from the socket buffer code. Once these were fixed, the bottleneck moved to the routing code, which has no parallelism. Streamlining the routing code led to the transmit queues, which need to be serialized to preserve ordering. The trick seems to be to serialize them only per connection and not for all UDP traffic, which is easier said than done.

The TCP stack has even more bottlenecks, including one lock for all incoming packets, serialized access to look up the connection for which a packet is destined, along with socket buffer send/receive, routing, and the transmit queues noted in UDP. He talked more about stack parallelism, direct dispatch versus input queuing, and maintaining per-

connection ordering when running with multireceive and multisend queues.

The last formal talk was by Peter Wemm on Version Control. Whereas most agree that CVS has hit a wall, it was much less clear what should replace it. After much investigation Peter concluded that Subversion would be the best replacement. The command-line interface is nearly the same as that of CVS except that it uses URLs instead of paths in some places.

After the initial changeover, all changes into Subversion will be reflected into the CVS tree. That means cvsup will still work and there will be almost no visible changes for the world apart from minor things such as slightly different commit messages. That also means that cvsweb would still work and that there would be a backup plan in case Subversion does not work out. The project could just switch back to CVS.

The first after-lunch discussion group was on system-trust issues. Topics included the mandatory-access framework, auditing activities within jails, and increasing the granularity of privilege: getting away from all (root) or nothing (all other users).

Jeff Robinson led a discussion of an overhaul of the buffer cache. Much of the functionality formerly provided by the buffer cache (caching, identity, clustering, etc.) is now provided by the virtual-memory system. Jeff talked about which functionality remained in the buffer cache and how to push that functionality elsewhere so that the remains of the buffer-cache interface can be eliminated. Details are at http://wiki.FreeBSD.org/Buf0x.

The day ended with a long reprise of the network-stack virtualization discussion, getting down to the specifics of defining a set of steps toward its realization, ordering those steps, and setting a timeframe for each step.

;login:

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