OSDI '06: 7th USENIX Symposium on Operating Systems Design and Implementation

Sponsored by USENIX in cooperation with ACM SIGOPS

*Seattle, Washington
November 6–8, 2006*

Opening Remarks

Summarized by Rik Farrow

OSDI 2006 began with record rains in Seattle, but the rain and local flooding did nothing to dampen the mood in the conference. Jeff Mogul started out with the usual summary of the number of papers submitted versus those accepted (149/27), telling us that each paper was reviewed multiple times and that shepherds helped with each accepted paper. Papers that did not meet the format required by the CFP were rejected without review. As OSDI, together with SOSP, is the top venue for publishing refereed operating-system-related papers, hopeful authors are strongly motivated to do much more than adhere to formatting.

Jeff thanked the program committee members and the people and organizations who sponsored OSDI. He pointed out that registration fees do not begin to cover the cost of the conference, but through the work of Robbert van Renesse and USENIX, enough money was raised to pay for registration fees and travel expenses for 72 students, as well as two receptions. Jeff also told us that he had set up osdi2006.blogspot.com so that summaries and comments on papers could be posted in real time during the conference.

Brian Bershad announced the winner of the SIGOPS Hall of Fame award, "Safe Kernel Extensions without Runtime Checking," by George C. Necula and Peter Lee. The two Best Paper awards went to "Rethink the Sync" and "Bigtable: A Distributed Storage System for Structured Data."

I found most of the papers exciting and was busy emailing links to abstracts to friends and acquaintances who I thought would likely be interested (most were). OSDI certainly has become one of my favorite conferences, being full of great information and new ideas.

LOCAL STORAGE

Summarized by Anthony Nicholson

Rethink the Sync

Edmund B. Nightingale, Kaushik Veeraraghavan, Peter M. Chen, and Jason Flinn, University of Michigan

Awarded Best Paper

The authors note that asynchronous I/O provides good user-perceptible performance, but it does not provide reliable and timely safety of data on disk. Synchronous I/O provides data safety guarantees but incurs significant overhead. Ed Nightingale presented a new model of "externally synchronous" I/O that resolves this tension by approximating the performance of asynchronous I/O while providing the data safety of synchronous I/O. The main reason synchronous I/O is slow is that applications must block until data has been written safely to disk. The authors argue that only the user, not applications, should be considered "external" to the system. Therefore, under their model, applications need not block on I/O operations but can perform other work while writes are queued. The system only blocks when some output that depends on a pending write is about to be externalized to the screen, disk, or network—in other words, when any event occurs that would make the user think that the I/O has completed.

External synchrony preserves the same causal ordering of writes as synchronous I/O. The fact that unrelated I/O operations can be batched and overlapped without violating causal ordering is what enables the performance wins in their system. Their implementation leverages their prior work (Speculator, SOSP '05) to track causal dependencies across multiple applications and throughout the kernel. "Commit dependencies" inside the kernel track all the processes and objects that are causally dependent on a given pending write. Commit dependencies are forwarded to applications that become tainted by uncommitted data, to ensure preservation of causal ordering. They modified the Linux ext3 file system to support external synchrony, and they compared the performance of their system to native ext3 in both asynchronous and synchronous I/O mode and to ext3 synchronous with write barriers. Their evaluation results show that ext3, even using synchronous I/O, does not guarantee data durability across crashes or power failures, but ext3 with write barriers provides the same data safety of external synchrony, although at a severe performance cost. Their performance on various file system benchmarks shows performance close to that of asynchronous ext3, while providing superior security guarantees to that of synchronously mounted ext3. The performance of ext3 using synchronous I/O is also an order of magnitude slower than that of external synchrony. Their performance on the specweb99 benchmark also shows that external synchrony adds minimal latency overhead as compared to asynchronous file systems.

David Anderson from Carnegie Mellon asked whether there was a corner case where an application would do an asynchronous write, see it failed, and change behavior based on that. Ed responded that in their system, by the time an application discovers a write has failed it would have already moved on, complicating recovery. He argued, however, that "failure notification" usually means kernel panic and crashing owing to hardware failure, so the user has bigger problems in that case. George Candea from EPFL asked about network latency in the mySql benchmark from their paper. Ed said that the client and server were on the same box in their evaluation, because the SpecWeb benchmark was more concerned with network latency. George explained that he was more interested in group commit latency. In that case, Ed said, their system gets the benefit of group commit but wouldn't be able to commit multiple transactions from the same client because of the rules of external synchrony. Micah Brodsky from MIT asked whether performance would suffer under external synchrony for high bandwidth, sequential I/O operations. Ed said you would still see improvement, because the OS wouldn't be blocking between operations if they weren't dependent on each other. Micah wondered how their performance would compare to that of asynchronous I/O for such large sequential operations. Ed noted that asynchronous I/O is limited by the speed to write to memory, and external synchrony would be similarly limited.

■ *Type-Safe Disks*

Gopalan Sivathanu, Swaminathan Sundararaman, and Erez Zadok, Stony Brook University

Gopalan Sivathanu began by noting that data on disk consists of two things: data and pointers. The structure of pointers on disk implies how disk blocks are organized via higher-level abstractions into files and directories. Unfortunately, today's disks are pointer-oblivious. The file system knows all about the semantics of data and the disk knows about the hardware details. But because the interface between OS and disk is so constrained, little information is exchanged between the two. Everything is just reading and writing blocks. The disk doesn't know the high-level reason for a read/write. For example, it would be nice for a RAID system to prioritize the handling of metadata blocks, because those are more important. Type-safe disks try to bridge this semantic gap. The authors propose an extended interface between the kernel and disk, to enable both type-awareness (disks tracking pointers) and type-safety (disks using pointer information to enforce constraints).

Because type-safe disks are conscious of the relationship between data and pointers, they can do such things as automatically garbage-collect data blocks that are no longer referenced by any file or directory. Offloading these tasks to the disk lets the file system component of the operating system shrink in size and complexity. The authors added additional API calls between file system and disk, such as "allocate block," "create pointer," and "delete pointer." They have implemented a prototype in Linux as a pseudo-device driver and have ported the ext3 and vfat file systems to support TSD. The porting effort was minimal (approximately two person-weeks). Their case study is a security application (ACCESS: A capability conscious extended storage system), which is disk-enforced access control. To access data, an application must provide the disk with a valid capability (an encryption key). Thus the maximum amount of data that can be exposed is that which is currently in use or cached at a higher level. Traditionally, if the OS were compromised, then all data on disk would be compromised. ACCESS establishes a security perimeter on the disk itself instead.

Michael Scott from the University of Rochester asked how TSD would work with a file system that doesn't use the hierarchical pointer design, such as a file system that stores file data in a chain of blocks interconnected by pointers. Gopalan answered that they can support such file systems because the pointers tracked by type-safe disks need not be the same as those maintained by the file system in its own metadata. Margo Seltzer from Harvard asked how this was different from the semantically smart disk work. Gopalan responded that semantically smart disks need to do a lot of operations at the disk level to infer the sort of information that type-safe disks explicitly are given through the API. Margo concluded that the two solutions are basically the same but the implementation cost is different. Gopalan disagreed.

Another questioner asked where private keys for disk blocks can be stored, if we don't trust the operating system that can read application memory. That discussion was taken offline. Emin Gün Sirer from Cornell asked how they settled on the API between the kernel and disk. Why not just move the whole file system into the disk? Gopalan answered that because we often want to run multiple file systems (or none at all, for certain databases) at once, not all functionality can be pushed into the disk. He was not confident that their API was the most minimal interface possible.

Finally, Chad Verbowski from Microsoft Research asked how they would properly keep the parity blocks in a RAID system. Gopalan answered that if one wants to use type-safe disks, then all layers of the file system software stack must be modified, including the RAID software.

■ ***Stasis: Flexible Transactional Storage***

Russell Sears and Eric Brewer, University of California, Berkeley

Rusty Sears stated that systems researchers in particular often abandon off-the-shelf storage solutions because of their poor performance and reinvent the wheel on every project that juggles a large amount of data. The goal of their project was to provide a transactional storage framework that provides good performance off the shelf but is easily extensible to meet the needs of users without requiring that applications be rewritten because they are too tied into the underlying plumbing. This saves users from having to concern themselves with details of logging, recovery, etc., unless they really want to. Stasis has the following three design principles: (1) Provide simple, thin APIs to low-level components. (2) Ensure high-level semantics via local invariants. (3) Make all module interactions explicit—this lets them place policy decisions in replaceable modules.

Russell gave an example of implementing a concurrent hash table on top of Stasis. They wrap operations with Stasis calls so that the underlying layers know how to undo the operation that is being wrapped, in case a transaction needs to be rolled back. Russell also discussed their case study: persistent objects. In other words, these are systems that support transactional updates over a series of objects. To optimize these updates, Stasis can conserve log bandwidth by only logging diffs. Also, they halve memory usage by deferring page cache updates. Low-level modules implement log updates and defer writes to the page cache in order to save memory. They implemented group commit in the log manager, and evaluation shows good performance gains. Since the log manager sees all updates that are produced, it could be extended to transparently implement automatic replication, etc. Stasis can also ensure temporal ordering of certain writes (such as external synchrony). Similar to type-safe disks, the type-system of the disk could be coupled to a type-system of a higher-level application, since these modules are all extensible. The authors are interested in implementing zero-copy I/O for the page file or replicating the page file on multiple servers.

Compared to atop Berkeley DB and SQL, performance is reasonable, with the optimizations described here doubling throughput and halving required memory. There were no questions. The project Web page is http://www.cs.berkeley.edu/~sears/stasis/.

RUNTIME RELIABILITY MECHANISMS

Summarized by Geoffrey Lefebvre

- ***SafeDrive: Safe and Recoverable Extensions Using Language-Based Techniques***

Feng Zhou, Jeremy Condit, Zachary Anderson, and Ilya Bagrak, University of California, Berkeley; Rob Ennals, Intel Research Berkeley; Matthew Harren, George Necula, and Eric Brewer, University of California, Berkeley

Feng Zhou began his talk by noting that many operating systems and applications run loadable extensions. These extensions are often buggier than their hosts and execute in the same protection domain. To address this issue, the authors present SafeDrive, a language-based approach to extension safety. SafeDrive can be decomposed into two principal components: the Deputy source-to-source compiler and the run-time recovery system. Although the principles presented could be applied to other systems, the talk and the paper focused on adding type-based checking and restart capability to existing Linux device drivers. The core idea is to transform code written in C into a safe variant, addressing issues such as out-of-bound array accesses, null terminated strings, and unions.

Previous approaches to retrofitting type safety to C, such as CCured, required the use of fat pointers, which contain both the pointer and bound information. This approach unfortunately requires changing the memory layout of data structures, making the modified extensions incompatible with their host's binaries. Instead of using fat pointers, Deputy relies on the programmer adding lightweight annotations to header files and extension source code. Since SafeDrive relies on run-time checks, it must provide mechanisms to deal with violations. SafeDrive enforces the invariant that no driver code will execute after a failure. The SafeDrive run-time system tracks all kernel resources used by a device driver, using wrappers around kernel API functions. Each tracked resource is paired with a compensation operation that performs an undo when a fault occurs in a driver. As an example, the compensation operation for a spinlock is to release the lock.

Compared to approaches using hardware memory protection such as Nooks, SafeDrive provides finer-grained memory protection. This allows it to catch more errors at compile time and exhibit less run-time overhead.

Andrew Baumann from the University of New South Wales stated that many bugs are concurrency-related and asked whether SafeDrive can detect and recover from such errors. Feng Zhou answered that SafeDrive does not currently address this issue. Brad Karp from University College London asked whether SafeDrive handled integer overflow, specifically the case where a signed integer overflow could result in a different memory allocation than the size requested. Feng Zhou answered that SafeDrive doesn't deal with integer overflow but that, in most cases, memory allocation routines should handle this issue. Rik Farrow asked what happens when programmers make errors writing the annotations. The answer is that the Deputy type system will catch some of the errors. Michael Swift from the University of Wisconsin asked how easy would it be to integrate this with Nooks to allow catching errors that only Nooks is able to catch. Feng Zhou said that it should be feasible to do so. Finally, someone from the University of California, Santa Cruz, asked whether SafeDrive was able to cope with complex data structures. Feng replied that by being able to deal directly with pointers, Deputy is able to do so.

- ***BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML***

Charles Reis, University of Washington; John Dunagan, Helen J. Wang, and Opher Dubrovsky, Microsoft; Saher Esmeir, Technion

Charles Reis began by pointing out that vulnerabilities in browsers are dealt with by patching, but there will often be a long delay between the time a patch is released and its application. This delay opens a dangerous time window when attackers can even use the patch as a blueprint to create exploits. A previous approach, Shield, aims to provide protection equivalent to patching but with easier deployment and roll-back. Shield allows filtering of malicious static content using vulnerability signatures. BrowserShield's goal is to provide similar protection for dynamic Web content by rewriting embedded scripts into safe equivalents on their way to the browser.

BrowserShield consists of a JavaScript library and a logic injector, which could be located at the server, at the firewall, or even as a proxy on the client. Both components are configured using flexible policies that can be tailored to address specific vulnerabilities. The injector performs a first translation which modifies the HTML to remove exploits and wraps all embedded scripts to force them to be invoked via the BrowserShield library. The library performs a second translation during page rendering by dynamically rewriting scripts to access the HTML document tree via an interposition layer. The evaluation shows that, combined with anti-virus and HTML filtering, BrowserShield provides patch-equivalent protection for all 19 vulnerabilities in IE for which patches were released in 2005.

George Candea from EPFL asked about the guarantees that can be provided that pages will be rendered correctly. The answer is that it's much easier to roll back a policy than an applied browser patch when something goes wrong. BrowserShield policy only affects pages, not the browser itself. George then asked how to evaluate the policies. Charles answered that the two metrics are how easy it is to write a policy and how to thoroughly test policies. Jason Flinn from University of Michigan asked whether the scripting was part of the TCB. Charles Reis said it is. Brad Chen from Google asked about specific things that the authors would like to see added or removed from JavaScript. Charles Reiss answered that a smaller API would make it easier to achieve complete interposition. Benjamin Reed from Yahoo! inquired how to debug a policy once deployed. Charles Reis answered that it's important to first distinguish whether a problem is due to BrowserShield or due to the page or browser. A solution would be to render the page in a unprotected browser running in a virtual machine. Benjamin further asked what the core dump should look like. Charles answered that BrowserShield could be enhanced with a set of debugging policies to generate the appropriate information.

Finally, Diwaker Gupta from UCSD asked how to prevent infinite loops in the script translation. Charles answered that if the script had an infinite loop then its interpretation would fall into an infinite loop, but he didn't think that it was possible for a script to force the translation step into an infinite loop.

■ *XFI: Software Guards for System Address Spaces*

Úlfar Erlingsson, Microsoft Research, Silicon Valley; Martín Abadi, Microsoft Research, Silicon Valley, and University of California, Santa Cruz; Michael Vrable, University of California, San Diego; Mihai Budiu, Microsoft Research, Silicon Valley; George C. Necula, University of California, Berkeley

Úlfar Erlingsson began by introducing XFI, a software-based protection system that provides safe system extensions. He proceeded with a demo of a JPEG image containing an exploit being rendered by both an unmodified JPEG library and an XFIed version of the same legacy library. The first case resulted in an application crash, whereas the XFIed version properly trapped and gracefully aborted the rendering.

The XFI implementation creates safe extensions from existing Windows x86 Portable Executables by performing binary rewriting. The rewriter adds inline guards or short machine-code sequences that perform checks at run-time. Guards are placed on computed control-flow transfers using unique labels as valid target identifiers. A guard verifies that the label is present at the target site before performing a computed jump or call. This mechanism ensures that all transfers remain within the control-flow graph. The rewriter also adds memory access guards to ensure that computed memory accesses lie within valid memory regions. The validation uses a fast path when the address lies within bounds established at load time. Memory accesses outside of these bounds fall through a slow path that validates the address using data structures similar to page tables. The binary is also modified to use two stacks, a scoped stack and an allocation stack. The scoped stack contains return address and variables that are accessed by name only. The allocation stack contains data that can be accessed using computed access. This mechanism allows the integrity of the scoped stack to be established through static verification.

Because rewriting binaries is a tricky process, XFI doesn't require this step to be trusted. Instead, XFI relies on a trusted verifier to parse the binary at load time, ensuring that binaries have the appropriate structure and the appropriate guards. The verifier is simple, fast, and consists of only 3000 lines of straightforward code, mostly x86 decoding tables, increasing confidence in its correctness.

Bryan Ford from MIT asked how XFI ensures that the heap is not incorrectly seen to contain matching identifiers. Úlfar answered that a guard checks whether computed control flows are to targets within the binary, in addition to matching identifiers and the verifier ensuring that identifiers are unique. Jim Lawson from MSB Associates asked whether XFI rejects self-modifying code. Úlfar acknowledged this. Rik Farrow asked about binary size increase. Úlfar answered that code size can increase by a factor of two or more but that most of the additional code is either in nonexecutable verification hints or in out-of-band trampolines, which are not invoked often. Therefore the increase in code size has a minimal impact on performance and i-cache behavior.

OS IMPLEMENTATION STRATEGIES

Summarized by Andrew Miklas

■ *Operating System Profiling via Latency Analysis*

Nikolai Joukov, Avishay Traeger, and Rakesh Iyer, Stony Brook University; Charles P. Wright, Stony Brook University and IBM T.J. Watson Research Center; Erez Zadok, Stony Brook University

Nikolai presented a new approach to operating system profiling. He showed that the logarithmic distributions of an OS's latencies can reveal most, if not all, aspects of the OS's internal operation. He presented several methods to analyze these profiles and provided interesting examples of the sorts of conclusions that can be drawn using the profiler data. Among others, the authors were able to diagnose a locking error in the Linux kernel without having to examine the source code. Also, because the latency can be measured entirely outside of the kernel, the method can be used to analyze operating systems even if the source is not available.

The basic technique involves repeatedly measuring the times required to complete OS requests. The latency of each request depends on the path taken through the code and interactions with other processes. Therefore, the distribution of these values can be used to make inferences about an operating system's inner workings. The latencies are used to generate histograms that can be analyzed either visually or by a provided automatic data-analysis toolset. Various events, such as being blocked on a lock, show up as spikes on the histogram. Correlations between different requests can reveal their contention on a shared resource. For example, similar spikes on the histograms of two different system calls that only appear when the two are run together suggest that they contend on the same lock.

Although the profiler can be used without any kernel changes whatsoever, it can take advantage of kernel instrumentation points if they are available. Loadable drivers can also be used as vantage points from which to measure latencies. The authors created a tool that can patch Linux file systems to automatically produce latency data. The presented profiler adds negligible overheads. The generated profiles are usually smaller than 1 kilobyte. The profiler adds less than 200 CPU cycles per profiled call, whereas existing profilers add overhead per profiled event. For example, if a system call is trying to acquire several semaphores and to perform I/O, existing profilers would add overhead for each such event. As a result, the presented profiler is efficient. As measured with the Postmark benchmark, the system's CPU time was affected by less than 4%.

Marcel Rosu of the IBM T.J. Watson Research Center asked whether the technique makes heavy use of CPU cycle counters in order to cheaply measure intervals of time and if they tested their system on CPUs that use variable clocks, such as those found in notebook computers. Although they didn't test on variable-clock CPUs, "it should be possible," Nikolai said, "to simply apply a scaling factor to handle the cases where the CPU isn't running at its maximum frequency. Also, remember that our system doesn't rely on using CPU cycle counters. We could use an off-CPU high-precision timer if the CPU lacked an appropriate method of measuring intervals." Brad Chen of Intel asked whether the profiler can be used to analyze anomalous cases, rather than just bad implementations of the main case. The paper describes a "sample profile," where instead of folding all the latencies for a given test together, these were separated based on fixed time intervals. Nikolai's group used this method to analyze ReiserFS's performance during the relatively infrequent periods of time when the Linux buffer flushing daemon bdflush was active.

■ *CRAMM: Virtual Memory Support for Garbage-Collected Applications*

Ting Yang and Emery D. Berger, University of Massachusetts Amherst; Scott F. Kaplan, Amherst College; J. Eliot B. Moss, University of Massachusetts Amherst

Ting Yang began by saying that the memory management component of an OS and the heap-management component of a garbage-collected (GC) run-time environment must cooperate with each other to ensure good performance under all memory loads. Today's operating systems don't provide enough VM sophistication to support GC run-times effectively when memory pressures are significant. As a result, current run-times are reactive; they only resize the heap once performance has degraded. In contrast, CRAMM is predictive; it can determine the best heap size and suggest that the run-time adjust it before the system begins to experience performance loss.

CRAMM consists of two components: an extension that sits in the kernel's VM subsystem, and a heap-size model that exists in the run-time environment. The kernel-mode component tracks each process's working-set size (WSS): the amount of memory that the process needs so that it does only a small amount of swapping. The model component computes the heap size that would cause the process's WSS to just fit within the maximum amount of memory the operating system is willing to allocate. If the current heap size and the computed optimal heap size differ, the model asks the run-time to adjust the size of the heap. The system can therefore quickly respond to changes in memory pressure by reducing the size of the heap before it is swapped out, avoiding reclamation-triggered thrashing. CRAMM adds only about 1–2.5% overhead during ordinary test runs where memory is plentiful. In exchange, CRAMM dramatically improves the performance in situations where memory pressure suddenly increases.

Chris Stewart, University of Rochester, asked whether the authors assumed that when using copying garbage collectors, the number of pages used to hold "copied survivors" does not change rapidly and wondered whether their system therefore would be able to handle flashloads when the run-time uses copying GCs. Ting explained that they keep track of the CS value from one invocation of the GC to the next and apply a smoothing function. The maximum value ever seen for the CS value is weighted more heavily to ensure that they don't underestimate this parameter.

- ***Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management***

Chad Verbowski, Emre Kıcıman, Arunvijay Kumar, and Brad Daniels, Microsoft Research; Shan Lu, University of Illinois at Urbana-Champaign; Juhan Lee, Microsoft MSN; Yi-Min Wang, Microsoft Research; Roussi Roussev, Florida Institute of Technology

Chad presented the Flight Data Recorder (FDR), a new tool which will ship with Windows Vista that allows all changes to the persistent state of a system to be logged for later analysis. He explained that various system management tasks that up until now have been something of a black art essentially reduce to queries over the logs gathered by the FDR. As a motivating example, Chad told of a server at Microsoft that would exhibit extremely poor performance every few weeks. A system administrator eventually determined that this was because the system's page file was being inappropriately shrunk. Unfortunately, he was unable to determine why this was happening. The best he could do was to send out email to the other admins asking them to make sure they weren't resizing the file. However, after running the FDR for a few weeks, the logs were used to quickly pinpoint the offending script.

Currently, system management tools break down into roughly three categories. Some use a similar logging approach but, owing to space constraints, activate only on demand. These types of tools are of limited usefulness when trying to determine why a particular piece of configuration data changed. Another class of tools uses signatures to look for known-bad configurations. However, creating signatures general enough to be useful across a wide variety of machines is a time-consuming process. Finally, manifest-based approaches require that applications provide the system with a list of all configuration dependencies. However, the authors point out that the uninstall tools included in many software packages leave behind files and configuration data, suggesting that building complete manifests is impractical.

The proposed approach simply logs all changes to the system's persistent state. The main contribution of the work is its novel method of encoding the activity logs. This method requires on average just 0.5–0.9 bytes per event. Since typical systems generate on the order of 10 million events per day, the resulting logs are small enough to be practically sent over the network, archived, correlated with other systems, and quickly queried. The authors report that they are able to execute common queries against a day's worth of stored data in as little as three seconds. They also note that it should be possible to serve as many as 5000 systems running the FDR with a single archive server.

The authors see the FDR as being useful not only for system management but also for ensuring that various security and management policies are being followed. For example, the logs captured by the FDR can be used to determine how often a locked-down production server is modified without proper approval. The FDR can also be used to assist in locating system "extensibility points": configuration settings that control the loading of extra system services or plug-ins. This has important implications for detecting and removing malware. In summary, the FDR has made it possible to know about everything that is happening on a system.

Q: Can the FDR be used to predict how a system will respond to a configuration change? A: We can search for another system that already has the configuration change but is otherwise similar to the machine in question. If we can find such a machine, we can use it to approximate how this system would behave with the change. Q: What is the right way to query the logs generated by the FDR? Should we be using SQL, or perhaps a customized query engine with a programmatic API, etc.? A: Right now, we just expose the raw tables as they are in the log file. Any special-built query engine should be optimized for queries of the form "What files have been changed since time T?" since the most common requests seem to be those that look for modified configuration entries. Q: Other types of events might be worth logging. For example, did you consider logging all socket activity? A: We did think about logging IPC activities, but the system doesn't currently implement this feature.

WORK-IN-PROGRESS REPORTS (WIPS)

Summarized by Andrew Miklas

■ *Taking the Trust out of Global-Scale Web Services*

Nikolaos Michalakis

If you were to contract out the hosting of a dynamic Web site to a number of content delivery networks, how could you be sure that every system serving your customers was running the exact software you supplied to the CDNs? This problem becomes even more severe for content dynamically generated by ordinary Internet users, where contract law might not be sufficient motivation to ensure that the distributors don't behave maliciously.

Nikolaos is researching ways to certify that dynamic content is served correctly in an environment where the delivery systems are not fully trusted by the content providers. The basic design has clients forward a fraction of the signed responses from one server to other replicas for verification. If the verifying replica computes a different result, it publishes the erroneous signed response so that other hosts can learn of the misbehaving server.

■ *Information Flow for the Masses*

Max Krohn, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, Robert Morris, and Alex Yip

Today's Web sites are serving an increasing amount of user-contributed content. They are no longer places where users go to passively download information, but instead they act as meeting points where people can exchange content. For example, consider Wikipedia, which is built on content contributed by its users.

However, sites today do not allow users to contribute to the applications running on them. Wikipedia does not allow anonymous users to patch up the MediaWiki software running on the live servers, as it does with its content. The main reason why this can't be allowed in today's hosting environments is security; a malicious patch could be used to leak ordinarily inaccessible information.

Traz is a novel Web-hosting environment that applies Asbestos-like reasoning to Web servers. User-contributed code may be allowed to manipulate data ordinarily inaccessible to the contributing user, but it will not be allowed to leak this information back to that user. Traz therefore allows Web sites to safely execute user-provided code against privileged information. Traz runs on ordinary commodity operating systems and allows developers to write their applications using any programming language.

■ *AutoBash: Hammering the Futz Out of System Management*

Ya-Yunn Su and Jason Flinn

We've all experienced it: a system that isn't performing quite the way we'd like. Maybe the video hardware doesn't set the appropriate resolution when plugging an external monitor into a notebook. Perhaps the wireless card doesn't reassociate with the nearest access point correctly on wake-up. No matter what the problem, we typically use the same approach to solve it: Type the symptoms into Google, find a page that describes a fix, apply the steps described in the fix, and check to see whether the problem is corrected. If not, back out the changes, and repeat. This process, which Ya-Yunn termed "futzing," requires a substantial amount of manual intervention and can lead to serious frustration.

AutoBash is a tool that automates the futzing process. When the user notices a configuration error, he or she describes the symptoms to the tool. AutoBash will search its database for scenarios where a user started with the current configuration, applied some changes, and ended up with a new configuration that satisfies the desired description. Once a record is found, the steps required to adjust the user's configuration will be automatically replayed. If the user is dissatisfied with the result, he or she will be given the opportunity to have the changes automatically rolled back and will possibly be presented with another solution. Finally, should the tool be unable to automatically correct the error, it will watch as the user does so manually, so that other users may benefit from the futzing done by this user.

■ *Dynamic Software Updating for the Linux Kernel*

Iulian Neamtiu and Michael Hicks

Software updates are a necessarily evil. In order to apply them, a service must typically be restarted. For many applications, the downtime that must be incurred in order to restart is undesirable. Worse, updates to system software such as the kernel can necessitate a full reboot of the machine, resulting in an even longer stretch of downtime.

Gingseng, a tool worked on by the presenter, can apply updates to running user-mode services. It does this by determining strategic locations in a service's execution where the code can be safely updated. Patching kernel code, however, is far more difficult, owing to its low-level and highly concurrent nature. Iulian described some of his work to make Ginseng able to update a live Linux kernel.

■ ***iTrustPage: Preventing Users from Filling Out Phishing Web Forms***

Troy Ronda and Stefan Saroiu

The U.S. economy loses billions of dollars each year to phishing attacks. Even worse, phishing erodes the public's trust in the Web as a platform for e-commerce. Troy claimed that many forms used to legitimately gather information originate from well-established Web sites, whereas phishing attacks are usually done from newly created sites. Fortunately, there are a number of services that can be used to estimate the popularity and thus the trustworthiness of a site. Phishing pages must also appear similar to their targets. Although it is difficult to design an algorithm to compare two Web pages, Troy mentioned that a person can usually determine with ease if one Web site is mimicking another.

These two key observations form the basis of iTrustPage, a Firefox extension that helps users avoid phishing attacks. When a user tries to fill out a form on a page that is not well established, iTrustPage stops the user from proceeding and asks the user to describe the task that he or she is trying to accomplish. Using this description, iTrustPage makes a query to Google and shows the user the Web sites associated with the first few hits. The user then indicates which Web site looks most similar to the page the user was expecting. Finally, the tool redirects the user to the organization's legitimate Web page and away from the phishing attempt. iTrustPage is currently available for download at http://www.cs.toronto.edu/~ronda/itrustpage/.

■ ***Failures in the Real World***

Bianca Schroeder and Garth A. Gibson

A major challenge in running large-scale systems is that component failure is the norm rather than the exception. Unfortunately, most work on dealing with failures is based on simplistic assumptions rather than real failure data. Bianca has been collecting and analyzing failure data from several real-world installations. The initial results indicate that many commonly used failure models are not supported by real data. For example, the probability of a RAID failure can be an order of magnitude larger, based on actual observation, than one would expect given the standard model, which uses exponentially distributed intervals between failures.

Motivated by these initial results, Bianca is continuing to collect and analyze failure data from a large variety of real-world installations. By carefully grounding new failure models in real data, researchers will be able to more accurately model a system's response to component failure.

■ ***Dynamically Instrumenting Operating Systems with JIT Recompilations***

Marek Olszewski, Keir Mierle, Adam Czajkowski, and Angela Demke Brown

Operating systems, like applications, grow more complicated each year. Unfortunately, the techniques and tools used to instrument, trace, and debug kernel-level code have not advanced as quickly as their user-mode counterparts. For example, tools such as Valgrind have made it possible to inject instrumentation into running user-mode code using JIT recompilation techniques. Because these probes are dynamically compiled directly into the surrounding code, they perform much better than traditional patch-and-redirect techniques. Unfortunately, JIT instrumentation tools are currently unable to instrument kernel code.

Marek plans to create JIT instrumentation tools that can be used with kernel code. Given the time-sensitive nature of much of a kernel, this approach may make it possible to instrument code that previously could not be probed for performance reasons. By bringing the proven benefits of JIT instrumentation to the kernel, Marek will assist systems programmers in better understanding their operating systems and ultimately will help them to produce more efficient and correct kernels.

■ ***Pattern Mining Kernel Trace Data to Detect Systemic Problems***

Christopher LaRosa, Li Xiong, and Ken Mandelberg

Profilers, debuggers, and system call tracers can all be used to diagnose performance issues within a process. However, diagnosing performance problems that result from the interplay of two or more processes can be complicated. For example, determining why an X server is exhibiting poor performance can involve gathering and correlating traces from both the X server and any active X clients. Unfortunately, there are few tools to automatically correlate traces, and programmers must usually resort either to poring over the gathered data by hand or writing ad-hoc scripts.

Christopher plans to apply data-mining techniques to system-wide activity traces. Using these techniques, anomalous conditions that might impact system performance can be automatically detected and isolated, even if they span multiple processes. He provided an example involving a stock-ticker toolbar applet that unnecessarily flooded the X server with requests. His trace analyzer was able to automatically detect the excess of X calls and pinpoint their origin. He would appreciate it if DTrace or LTT users would share their hard-to-find bugs with him, so that he can test the effectiveness of his system's automatic detection.

■ ***Spectrum: Overlay Network Bandwidth Provisioning***

Dejan Kostić

Overlay networks are currently used to efficiently disseminate content. However, because of their decentralized nature, it can be difficult to ensure that there is enough outbound capacity to support all receivers as well as prevent other overlays from "stealing" bandwidth from higher-priority services. This presents a serious problem when overlays are used to transfer streaming media, where timely delivery of content is necessary for the system to operate correctly.

Dejan is working on algorithms to measure and disseminate bandwidth availability information throughout an overlay network. By doing so, the system can make globally optimal decisions about how much bandwidth to dedicate to a media stream. This is especially useful when the same overlay is carrying a variety of different content. For example, a BitTorrent–like transfer through the overlay might be permitted as long as it doesn't cause anyone's video stream to drop below a certain bit-rate.

■ ***An Infrastructure for Characterizing the Sensitivity of Parallel Applications to OS Noise***

Kurt B. Ferreira, Ron Brightwell, and Patrick Bridges

Many commodity operating systems do not scale well to the number of processors found in today's supercomputers. When running such operating systems, as much as 50% of the system's performance can be used by the operating system itself. For this reason, many of today's largest supercomputers run stripped-down operating systems that impose as small an overhead as possible.

Kurt's research seeks to understand exactly how this overhead, termed "OS noise," affects the bottom-line performance of various scientific computing applications running on large supercomputers. He is also interested in finding ways to reduce the overheads found in ordinary operating systems in order to make them more suitable for use on large supercomputers.

■ ***Limits of Power and Latency Reductions by Intelligent Grouping***

David Essary

Disk accesses are a very expensive operation; entire classes of applications are limited by the I/O capability of their hosts, rather than its raw processing power. Improving an I/O subsystem's ability to quickly respond to requests can greatly improve the overall efficiency of such systems.

David's research seeks to improve storage access time and throughput by carefully controlling how the data is physically laid out on disk. Data can even be stored on multiple drives in an array to give the reading process more flexibility when deciding how to optimally read the data back. These techniques can result in a 70% reduction in disk-related latencies and energy consumption. David also discussed some of his work on predictive retrieval algorithms and how he had explored theoretical limits. Finally, he pointed out that his work to reduce seek operations not only improves performance but also reduces drive power consumption.

■ ***Distributed Filename Look-up Using DNS***

Cristian Tapus, David Noblet, and Jason Hickey

One of the challenges in building a distributed file system is finding a way to locate the data and metadata associated with files. Usually, this information is replicated to provide reliability and thus might be distributed across a wide-area network. A centralized directory service is undesirable because it provides a single point of failure and may behave as a bottleneck for the system.

Cristian noted that many of the problems faced when serving file metadata and data are in fact the same as those solved by DNS. For example, both DNS and FS metadata are used to resolve names in a hierarchical name space to addresses. For these reasons, Cristian suggested using DNS itself as a localization service for the data and metadata of files in a distributed FS. Looking up a file would involve making a DNS query for a name such as "passwd.etc.mojavefs.caltech .edu." The query would return the addresses of the replica file servers that could serve the named file. Replication of the metadata is handled automatically by the caching mechanisms built into DNS.

■ ***Bounded Inconsistency BFT Protocols: Trading Consistency for Throughput***

Atul Singh, Petros Maniatis, Peter Druschel, and Timothy Roscoe

Many protocols exist for ensuring the high availability of a system despite the potential for byzantine failure of its components. However, these protocols have negative scaling properties: The more nodes added, the higher the performance penalties associated with keeping all of the components synchronized.

Atul proposed a solution where replicas return results that differ slightly from the correct result. The key is that the variability of the response is bounded; a client can be sure the true value is within some range of the returned quantity. By allowing the replicas to run slightly out of sync, the overall performance of the system can be improved. This approach can be useful for applications that don't require precise results. For example, it might be acceptable for a disk quota system to allow a user to consume at most 5% more disk space than the allotted quota.

■ ***EyesOn: A Secure File System That Supports Intelligent Version Creation and Management***

Yougang Song and Brett D. Fleisch

Versioning file systems are often used to enhance the security capabilities of an operating system. Since they preserve the change history for each file, they can help system administrators both detect intrusions and roll back unauthorized changes. However, maintaining a comprehensive change history can become overwhelmingly expensive in both disk space and performance overhead.

The EyesOn system aims to preserve normal file operations and existing file structures while leaving the complexity of recovery operations to the time they are requested. EyesOn extends the same strategy used by file system journaling to record its in-memory modified data in a log without significant additional processing. EyesOn uses these logs to create file versions that can be used to accelerate the retrieval of a file's change history. Versions are created based on user-supplied predicates that can make use of statistics stored in the log. For example, predicates can use the elapsed time since a file was last modified, the total size of the change, or whether the user has explicitly requested that a snapshot of the file be taken at this time. Two types of versions are created in EyesOn. Normal versions are created for quickly retrieving recent changes and will automatically be culled once they reach a certain age. Landmark versions are created for keeping valuable information for a longer time.

■ ***Robust Isolation of Browser-Based Applications***

Charles Reis

First it was online email. Next came online scheduling, photo archiving, and journaling. Today, companies have begun testing online word processors and spreadsheet applications. Eventually, it's possible that most applications will be run on racks of systems in far-away data centers and served over the Web.

If the future of applications is the Web, than in some sense the future of operating systems is the Web browser. Although they may never directly interact with hardware, they certainly will fulfill other roles traditionally handled by an operating system. For example, Web browsers should ensure that a buggy or malicious script on one site doesn't adversely affect the scripts of another. Browsers should also protect the locally stored data associated with one site from unauthorized access by scripts from another site. Charles is currently looking at ways to build these types of containment mechanisms into browsers such that changes to the server-side applications are kept to a minimum.

■ ***Stealth Attacks on Kernel Data***

Arati Baliga

Rootkits use an array of impressive techniques to hide themselves from detection. Some go so far as to rewrite portions of the in-memory kernel image to perfect the illusion. New system calls might be added to render the rootkit's processes invisible to ps. Others carefully manipulate the process lists and file system handlers to evade detection.

Arati is investigating all of the ways in which rootkits can tamper with the running kernel image by solely manipulating kernel data. She hopes to use her findings to develop monitoring systems that can't be easily fooled. In particular, she is looking at attacks that do not employ conventional hiding techniques yet are able to cause stealth damage to the system and evade detection from state-of-the-art integrity monitoring tools.

PROGRAM ANALYSIS TECHNIQUES

Summarized by Geoffrey Lefebvre

■ **EXPLODE*: A Lightweight, General System for Finding Serious Storage System Errors***

Junfeng Yang, Can Sar, and Dawson Engler, Stanford University

Junfeng Yang began his talk by noting that storage systems errors are the worst kind of error since they can result in corruption of persistent state, possibly leading to permanent loss of data. To address this issue, the authors presented EXPLODE, a system to find errors in storage systems. eXplode borrows ideas from model checking by being comprehensive, but instead of running the checked system inside a model checker, eXplode runs client-defined checkers inside the checked system. Running on a real system allows it to easily check storage stacks. A file system checker can be used to find errors in storage layers either above or below the checked system. Another advantage of this approach is that checkers are easy to write. A checker for a storage system can be written in less than 200 lines of C++ and often consists of a simple wrapping layer around existing utilities such as mkfs and fsck. The authors state that this work completely subsumes their previous work (FiSC).

Because bugs are often triggered by corner cases, eXplode's core idea is to explore all choices. eXplode provides choose(N), an N-way fork that allows checkers to fork at every decision point during testing and explore every possible operation. Before exploring a decision point, eXplode checkpoints the state of the system. A checkpoint is simply the recorded sequence of return values from choose(). To restore a checkpoint, eXplode deterministically replays this sequence from the initial state. These two mechanisms allow eXplode to perform an exhaustive state exploration. At any point during testing, a checker has the ability to force crashes. Upon a forced crash, eXplode generates crash disks based on all possible orderings of dirty buffers. A checker-supplied routine then tests all disks for specific invariants.

A challenge faced by the authors was dealing with nondeterminism. The choose() primitive could be called by nonchecking code such as interrupt handler. eXplode solves this problem by filtering on thread ID. eXplode must deterministically schedule all threads involved with the checked system. This includes the checker's thread but also all threads belonging to the checked storage system. By playing with thread's priorities, eXplode is able to enforce deterministic scheduling most of the time and can detect when it fails to do so.

Junfeng then presented an evaluation of eXplode. The authors tested various file and storage systems such as ext2, ext3, JFS, the VFS layer, NFS, Berkeley DB, and VMware with eXplode and found bugs in all of them. Someone from UCSD asked about the option not to replay instrumented kernel functions and if doing so could lead to nondeterminism. Junfeng answered that short traces were deterministic, since calls to these kernel functions succeed most of the time, but that long traces had nondeterminism. Someone asked whether it was possible to test subcomponents of file systems. Junfeng answered that normally filesystem implementations were not structured that cleanly, so it makes more sense to check a file system as a whole. Another person asked how eXplode handled nondeterminism created by disk actually performing an operation out of order internally. Junfeng answered that eXplode uses a RAM disk, which avoids this problem.

■ *Securing Software by Enforcing Data-Flow Integrity*

Miguel Castro, Microsoft Research; Manuel Costa, Microsoft Research Cambridge; Tim Harris, Microsoft Research

Manuel Costa began his presentation by noting that most of the software in use today is written in C++. This body of software has a large number of defects and there exist many ways to exploit these defects, such as corrupting control data. He presented some of the various approaches to securing software. He noted that removing or avoiding all defects is hard and that although it is possible to prevent attacks based on control-data exploits, certain attacks can succeed without compromising control flow. Approaches based on tainting can prevent noncontrol-data exploits, but they may lead to false positives and have a high overhead.

To address these issues, the authors present a new approach to secure software based on enforcing Data-Flow Integrity (DFI). Their approach uses reaching definition analysis to compute a data-flow graph at compile time. For every load, compute the set of stores that may produce the loaded data. An ID is assigned to every store operation and, for each load, the set of allowed IDs is computed. The results of the analysis is used to add run-time checks that will enforce data-flow integrity. Stores are instrumented to write their ID into the run-time definition table (RDT). The RDT keeps track of the last store to write to each memory location. Loads are instrumented to check whether the store in the RDT is in their set of allowed writes. If a store ID is not in the set during a check, an exception is raised. Because the analysis is conservative, an exception guarantees the presence of an error. There are no false positives. Manuel also noted that control-flow attacks are a form of data-flow attacks. It is possible to use the same mechanism to protect control-flow data.

Manuel described a set of optimizations to improve the run-time performance. The most important optimization is to rename store IDs so that they appear as a continuous range in the definition set. The membership test can be replaced with a subtraction and a compare. The evaluation presented demonstrates that this optimization is fundamental to the performance of DFI. On average, DFI imposes a space overhead of around 50%, and the run-time overhead ranges from 44% to 103%.

Brad Karp from University College London asked whether DFI handles function pointers and other complex program constructs. Manuel answered that DFI handles function pointers but has problems with certain pieces of software written in assembly. These problems can result in not detecting certain DFI violations. An alternative would be to use CFI in this case. It is important to understand that these limitations are a property of the program to instrument, not the attack. Bill Bolosky from Microsoft Research asked whether every store had to be instrumented. Manuel acknowledged that this was the case.

■ ***From Uncertainty to Belief: Inferring the Specification Within***

Ted Kremenek and Paul Twohey, Stanford University; Godmar Back, Virginia Polytechnic Institute and State University; Andrew Ng and Dawson Engler, Stanford University

Ted Kremenek began by saying that all systems have correctness rules, such as not to leak memory, or to acquire some lock before accessing data. We can check these rules using program analysis, but the problem is that missed rules lead to missed bugs. The specification of a system is the set of these rules and invariants. The problem addressed in this talk is how to find errors that violate these rules when we don't know what the rules are or, more precisely, how we can infer the specification. The talk presented a general framework to do so and the technique is described using an example: finding resource leaks by inferring allocators and deallocators.

There are many sources of knowledge that can be used to infer system rules, such as behavioral tendencies (i.e., programs are generally correct) and function names, but there isn't a way to bind all of this information together. To address this issue, the authors present an approach based on the Annotation Factor Graph (AFG), a form of probabilistic graphical modeling. This approach reduces all forms of knowledge, either from evidence or intuitions, to probabilities. The idea is to express program properties to infer as annotation variables and infer these annotations by combining scores obtained from factors. Factors represent models of domain-specific knowledge and are used to score assignments of values to annotation variables based on the belief that an assignment is relatively more or less likely to be correct.

The talk focused on how to infer resource ownership using AFGs. First, functions return values and parameters are annotated. The domain for a return value annotation variable is to return or not return ownership, and the domain for a function parameter is to claim or not claim ownership. Ted Kremenek then described some of the factors used to infer resource ownership. Some use static analysis based on common programming axioms such as that a resource should never be claimed twice; others are based on ad-hoc knowledge such as function names.

The evaluation was based on inferring annotations on five projects: SDL, OpenSSH, GIMP, XNU, and the Linux kernel. The authors' technique obtained a 90%+ accuracy on the top 20 ranked annotations. It was also able to infer allocators unknown or misclassified by Coverity Prevent. Using their technique, the authors found memory leaks in all five projects. Ted Kremenek described a complex memory leak they were able to find in the GIMP library.

Someone from Yahoo! asked how well the tool performs when allocators that are simply wrappers around malloc are factored out. Ted answered that although many allocators call malloc, the tool doesn't use this knowledge. Brad Chen from Google asked whether the tool would be confused by realloc. Ted answered that the tool did infer that realloc was both an allocator and a deallocator. Corner cases, such as this one, are detected automatically but have to be dealt with separately. They had a similar issue with certain functions in the Linux kernel. Such outliers have to be modeled explicitly but are fairly rare. Micah Brodsky from MIT asked whether AFGs are an instance of Bayesian Net. Ted answered that AFGs and Bayes Nets belong to the same family of probability modeling. The authors actually started with Bayes Net but found the class to be too rigid. AFGs were much easier to work with. Someone asked what other things could be modeled. Ted answered that locks are very behavioral, so their approach would work well. He stated that the temporal relationship could be expressed as a grammar.

DISTRIBUTED SYSTEM INFRASTRUCTURE

Summarized by Anthony Nicholson

■ ***HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance***

James Cowling, Daniel Myers, and Barbara Liskov, MIT CSAIL; Rodrigo Rodrigues, INESC-ID and Instituto Superior Técnico; Liuba Shrira, Brandeis University

The authors address the problem of building reliable client-server distributed systems. They note that the current state of the art either requires a small number of replicas (3f+1) but has high communication overhead or reduces communication complexity by requiring a much larger number of replicas (5f+1). The second case still suffers from degraded performance in cases of write contention, however. They propose a hybrid scheme that combines the best aspects of both schemes to achieve a low number of replicas (3f+1) while bounding communication overhead.

Their system uses a two-phase write protocol. First, a client obtains a timestamp grant from each replica. This grant is essentially a promise to execute the given operation at a given sequence number, assuming agreement from a quorum of replicas. In the second phase, the client forms a certificate from 2f+1 matching grants and sends this certificate to all the replicas, which then complete the write operation. A certificate proves that a quorum of replicas has agreed to a given ordering of operations. Importantly, the existence of a certificate precludes existence of conflicting certificate. Replicas are forbidden to have two outstanding grants in progress, and they return the currently outstanding grant to clients while a grant is in progress, as proof that it is busy. The authors have deployed both their Hybrid Quorum (HQ) and BFT prototypes on Emulab. Their results show that HQ performs better than BFT up until around 25% contention.

Atul Adya from Microsoft Research noted that their protocol allows clients to commit operations on behalf of other clients, and asked whether this was a security hole. James said that since certificates are free-standing and cryptographically signed, any client can send a certificate to a replica and it will commit faithfully on behalf of the originating client. Bill Blaskey, also from MSR, noted that commit could be painful because potentially thousands of operations occur per second. James noted that all data lives completely in RAM in their experiments, and a reboot is considered a failure. Petros Maniatis from Intel Research asked why the authors chose to do a two-phase protocol with 3f+1 replicas, rather than a one-phase protocol with 5f+1 replicas. James noted that they could have done so but decided that 5f+1 is just too many. The last question involved whether colluding replicas would just always tell clients that they had an outstanding grant, to force the slow path of the protocol. James responded yes, in the worst case this would happen.

■ ***BAR Gossip***

Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin, University of Texas at Austin

The motivation scenario for talk is a live, peer-to-peer streaming media application. Such applications can fall apart, however, in the presence of malicious nodes, if it is not in a node's self-interest to forward a data packet on to its peers. The authors introduce the concept of BAR Gossip, a gossip protocol that makes it in the interest of selfish nodes to act for the benefit of the overall system, while detecting and punishing byzantine (malicious) nodes. BAR Gossip is in fact two protocols: balanced exchange and optimistic push.

During balanced exchange, at each round every node selects one partner, the partners exchange their transaction histories, and then they trade equal numbers of data updates. Clients must commit to their histories before discovering their partner's history; similarly, they must send the data updates first in encrypted form before subsequently sending the key. This makes bandwidth a "sunk cost," so it is illogical for selfish nodes to withhold the data key from their partner later on. The authors also introduce the notion of verifiable partner selection to prevent clients from talking to more than one client per round (to accumulate more updates).

The second part of BAR Gossip is optimistic push, which handles bootstrapping for new nodes. In this case, unequal numbers of updates may be exchanged, but the lesser peer must sacrifice the same amount of energy and bandwidth by sending junk to even out the exchange. Their simulation results show that following the protocol was the most beneficial strategy for clients in all cases.

Rob Sherwood from Maryland asked about cases where clients consider different weights for inbound and outbound traffic. Harry argued that such asymmetry can be handled by sizing key requests accordingly. Rob responded that in cases where one is downloading illegal content, it might be overwhelmingly important never to upload anything. Harry countered that in such extreme cases BAR Gossip might not work. Jim Liang from UIUC noted that the authors assume that all clients know the complete membership list. Harry said that they are currently looking at handling cases where clients have only partial membership information. Jim further asked how the BAR Gossip protocol achieves low latency for streaming media. Harry said their experiments showed an average latency of 20 seconds for a live video stream. This compares favorably with existing streaming media applications (such as the free NCAA Final Four video feed). Another questioner asked how their protocol handles collusion. Harry replied that they have no explicit mechanism for this, because handling collusion in game theory is difficult. Their results showed, however, that BAR Gossip is robust for small colluding groups (with up to 30% nodes colluding). The last question concerned the scalability of BAR Gossip. Harry cited three limitations in scaling their system: (1) handling dynamic membership churn, (2) handling nodes with only partial membership information, and (3) locality-aware partner selection. They are currently looking at all three issues.

■ ***Bigtable: A Distributed Storage System for Structured Data***

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, Google, Inc.

Awarded Best Paper

Mike Burrows described Bigtable, a storage system designed for in-house use at Google. Bigtable developed in response to a need to store information on over 10 billion URLs, with wide variety in the size of objects associated with a given URL, and variety in usage patterns. Commercial databases are obviously unsuitable owing to the scale (petabytes of data on billions of objects, with thousands of clients and servers). Bigtable stores data in a three-dimensional, sparse sorted map. Data values are located by their row, column, and timestamp.

Since these tables can be quite large, Bigtable allows dynamic partitions by row, called tablets, which are distributed over many Bigtable servers. Clients can manage locality by choosing row keys in such a way that data that should be grouped together achieves spatial locality. A given tablet is owned by one server, but load balancing can shift tablets around. Similarly to tablets, locality groups are partitions by column instead of row. These locality groups, however, segregate data within a tablet. All data is stored as files in the Google File System (GFS), with different locality groups stored as different files in GFS. One master Bigtable server controls the many tablet servers. Clients access a tablet by requesting a handle from the Chubby lock service (described in another OSDI talk), then doing read/write directly with the tablet server that owns a given tablet. The client only talks to the master when it needs to manipulate metadata (e.g., create a table or manipulate ACLs). Currently, Bigtable is deployed on over 24,000 machines in approximately 400 clusters and is used by over 60 projects at Google.

The first question concerned the poor random read performance described in the paper and noted that the authors attributed this to shared network links. The questioner added that it seems an easy optimization to move tablet servers closer to the GFS storage that the tablets actually reside on. Mike answered that, by default, if a GFS server is collocated with the tablet server, the tablet's data will be stored on that GFS server. Atul Adya from Microsoft Research asked how they deal with hierarchical data—does this generate thousands of columns in their table paradigm? Mike noted that Bigtable is not a database, and such hierarchical data situations are not suited for Bigtable. He noted that their clients need to conform to how Bigtable works, not the other way around. George Candea from EPFL asked whether they had any technical insights to offer based on their experiences. Mike said that if you have the freedom to do so, build something with a custom API that matches the needs of both clients and users.

DISTRIBUTED SYSTEMS OF LITTLE THINGS

Summarized by Anthony Nicholson

■ ***EnsemBlue: Integrating Distributed Storage and Consumer Electronics***

Daniel Peek and Jason Flinn, University of Michigan

Daniel Peek described EnsemBlue, a framework for integrating consumer electronic devices (CEDs) into commodity distributed file systems (DFSes). This has been difficult because of the closed nature of CEDs and because such devices cannot simply run the DFS's client software to integrate its storage with all the user's other computing devices. Instead, Dan described how EnsemBlue leverages the user's general-purpose computers (such as desktops and laptops) to act as a bridge between the DFS and each CED that connects to the computer (e.g., when an iPod syncs with a user's desktop machine).

A key challenge here involves namespace conflicts between the DFS and the proprietary naming structures found on CEDs. EnsemBlue handles this by tracking mappings between the name of an object in the DFS and its name on each given CED. Since the CEDs comprise a closed system, the general-purpose computers in the system must execute all custom code in the system. These computers therefore need to know when data is updated in the system, to take certain actions such as updating custom indexes on CEDs. The authors leverage the fact that every DFS has a distributed notification protocol already—the cache consistency mechanism. Therefore, the authors introduce the concept of a "persistent query," which is an object in the DFS that indicates what the query is looking for, such as new mp3 files added to the DFS. The authors presented an example of how a persistent query for all new m4a files could be used to implement a transcoder from m4a to mp3 files. Finally, the authors described how they handle disconnected devices that cannot speak with the general file server. In such situations, several of the user's devices might be able to contact each other but not the remote file server. One of the devices becomes a "pseudo-file server," acting as a file server to the best of its ability, serving to the other devices those files that it happens to have at the time.

One questioner ask how this work would fit into the universal Plug-and-Play (uPnP) initiative. Dan responded that currently, EnsemBlue requires read and write access to the device (through USB, for example). Their future work will allow them to work with arbitrary protocols such as uPnP. Jawwad Shamsi from Wayne State University asked whether they require a separate general-purpose device for each mobile device. Dan answered that any number of CEDs can connect to any number of general-purpose computers. Christopher Stewart from the University of Rochester asked whether they had encountered any performance tradeoffs in building the protocol that interacts with the dedicated host machine. Dan responded that they hadn't measured the performance of integrating data back to the DFS through the general-purpose computer, but since their system is weakly consistent anyway, such performance would not be that important.

■ *Persistent Personal Names for Globally Connected Mobile Devices*

Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris, Massachusetts Institute of Technology

Currently, locating users' personal devices over the Internet is difficult. Local discovery protocols such as Bonjour don't work over long distances, and requesting DNS names for all of one's devices is impractical at best. Bryan Ford argued that people should be able to name their devices in a personal fashion and have global connectivity with each other's devices, without having to keep changing the way the devices locate each other. Their proposed solution is called UIA (Unmanaged Internet Architecture).

In UIA, users managed their own personal namespaces. When they acquire a new device (such as a phone, laptop, or camera) they assign a name to the device and "introduce" it to their existing devices. Each device has a unique endpoint identifier (EID)—just a hash of its public key. All clients running UIA belong to an overlay that lets this namespace exist atop IP. Users assign short personal names to identify their friends. Other users' devices can then be named by the combination of friend name and device name. Devices then gossip to propagate their name records. Friendship is transitive, so if Alice knows Bob directly, and Bob knows Charlie, Alice can name Charlie's phone as Phone.Charlie.Bob. The authors have implemented UIA on Linux, Mac OS X, and the Nokia 770 tablet. A UIA name daemon and router run atop the TCP/IP stack in userspace, with some GUI controls as well. Bryan also showed an entertaining demo video that highlighted the ease of use of the UIA paradigm. Their code is available for download (in a rough state!) on their project Web page: http://pdos.csail.mit.edu/uia/.

Mark Aiken from Microsoft Research asked how this system manages connectivity to devices that have moved while being communicated with. Bryan answered that in UIA's routing protocol, devices track other devices in their local social neighborhood and then hope to find a few devices that are stable enough to be rendezvous points to disseminate current IP address information. These stationary nodes help bootstrap what is essentially a distributed DNS scheme. Another questioner asked whether the authors had conducted any usability studies of their system. Bryan answered that they hadn't had any users outside their research group. Mahur Shah from HP noted that all the examples in the talk deal with devices that are fully owned by one user. He wondered how this would work for shared devices—in a family setting, for example. Bryan noted that they discuss this in the paper. Each physical device can have multiple EIDs and go by different names.

■ ***A Modular Network Layer for Sensornets***

Cheng Tien Ee, Rodrigo Fonseca, Sukun Kim, Daekyeong Moon, and Arsalan Tavakoli, University of California, Berkeley; David Culler, University of California, Berkeley, and Arch Rock Corporation; Scott Shenker, University of California, Berkeley, and International Computer Science Institute (ICSI); Ion Stoica, University of California, Berkeley

Cheng Tien Ee described work at Berkeley on a modular network layer for sensor networks. The authors recognize that sensor nets software from different organizations does not interoperate easily. The specific problem they address in this work is monolithic, vertically integrated network stacks. Intuitively, one would expect that if such network layers were modularized there would be a good deal of overlay among different implementations. Since the authors argue that we are probably stuck with multiple network protocols, they focus on making it as efficient as possible to run multiple network protocols at once on one system.

Their solution decomposes the network layer into modules. First, they break the network layers into the data plane and the control plane. Each of these is further subdivided into many components, such as an output queue and forwarding engine in the data layer, and a routing engine and routing topology in the control plane. They show how diverse protocols can actually share components, such as output queues, resulting in run-time benefits and code reuse. Their evaluation of several common protocols showed that protocol-specific code made up a small fraction of the total code base for their implemented examples.

Matt Welsh from Harvard was concerned about the interplay among different network protocols with regard to such things as packet scheduling and memory usage. Cheng responded that memory management is indeed a cross-layer issue and that they are currently looking at dealing with such effects. Matt also asked whether the code was available, and Cheng said they are currently attempting to integrate their work into TinyOS. Eddie Kohler from UCLA asked about the types of protocols that would fit this model less well than the examples the authors chose. Cheng answered that they can decompose any class of protocols, but the main difficulty they have with more complex protocols is the decomposition of REs (Routing Engines) and FEs (Forwarding Engines) into smaller ones that can be better reused. An example of such a protocol is one with multiple phases during the forwarding of a packet along its path. For such protocols, it is not immediately clear how the further decomposition can be done. An indication of an improperly decomposed network protocol would be multiple similar functions, such as packet forwarding methods, being implemented within a type of component. The last question noted that there are protocols they can't support, but this is due to SP (Sensornet Protocol); for example, anything with rate limiting can't be represented to SP. To the question of whether there is anything the authors would have wanted from the lower-level abstractions that they don't currently supply, Cheng replied that he didn't need anything else from SP and, on the contrary, found it often provided more info than necessary.

OPERATING SYSTEM STRUCTURE

Summarized by Leonid Ryzhyk

■ ***Making Information Flow Explicit in HiStar***

Nickolai Zeldovich and Silas Boyd-Wickizer, Stanford University; Eddie Kohler, University of California, Los Angeles; David Mazières, Stanford University

Nickolai Zeldovich stated that the HiStar operating system aims to prevent malicious and buggy software from leaking sensitive user data by making all information flow within the system explicit. The HiStar kernel implements six types of objects used as building blocks for user-level software: containers, segments, address spaces, threads, gates, and devices. Each object is assigned a security label that controls how the object can be modified or observed and can be thought of as a taint. Data in a tainted object can only be accessed by other tainted objects. Any data that flows outside the system has to be untainted first. Untainting can only be performed by a thread that has an untaint label.

Coming up with the right design of taint tracking can be difficult, if one wants to avoid covert channels. In a naive design, malicious applications could communicate by modifying and observing taint levels of different objects. HiStar closes this covert channel by making all nonthread object labels immutable. To avoid covert channels arising from resource allocation, HiStar provides a specialized IPC abstraction where the client donates initial resources to the server.

Flexibility is achieved by introducing multiple categories of taint. For example, UNIX users can be emulated by assigning a taint category to each user. A superuser can then be implemented as a thread holding untaint privilege for all user categories. As a result, root has no special privileges in the system and is not fundamentally trusted by the kernel.

Nickolai illustrated the HiStar architecture using two case studies. First, a running example of a virus scanner was used to introduce the taint-based access control model and demonstrate how HiStar allowed encapsulating application-specific security policies in a separate component. Second, an implementation of the UNIX authentication process based on an untrusted authentication service was described.

The main group of questions related to the HiStar resource management policy and dealing with different types of covert channels. Nickolai explained that static preallocation of resources is preferred in cases where you really want tight control over information flow, but in less-sensitive applications that is likely to be too restrictive. With regard to covert channels, someone asked how HiStar dealt with latency-based covert channels. Nickolai replied that although they were working on some ideas, the current implementation did not prevent such channels.

Another question was whether the authentication service could leak the password by denying and allowing login requests. The answer was that this could not happen, since every request is essentially handled by a newly forked instance of the authentication service, which is not allowed to communicate any data back to the original instance of the service. Another interesting question was whether HiStar could accommodate legacy applications requiring communication with the outside world. Nickolai said that HiStar could accommodate most legacy applications, but it may not be able to provide any added security if the application is monolithic and requires frequent network interaction.

■ *Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable*

Richard Ta-Min, Lionel Litty, and David Lie, University of Toronto

In the modern OS, the trusted computing base of an application includes not only the kernel but also all the privileged user-level services that run under the root account. By compromising one of these services, the attacker gets access to all sensitive data in the system.

Proxos aims to reduce the TCB by isolating sensitive applications inside their own private instances of the OS running under control of a virtual machine. All other applications run inside the public "commodity" OS. The commodity OS is also used for communication among secure applications running inside private OSes. This is achieved by selectively routing some system calls issued by secure applications to the commodity OS. The developer specifies which calls should be routed by using the Proxos routing language. To benefit from the Proxos architecture, secure applications need to be split into components running inside the commodity OS and those running inside the private OS.

The main performance overhead comes from context switching between the private and the commodity OS, which turns out to be an order of magnitude slower than Linux kernel call. However, at least for the proof-of-concept applications that have been ported to Proxos (Web browser, SSH authentication server, and the Apache Web server with SSL certificate service), this did not prove to be a problem, as the resulting end-to-end overhead was negligible.

Q: The private OS can become as complex as Linux itself. So how does this help reduce the TCB? A: Yes, security-sensitive applications still have to trust the entire Linux kernel, but not the privileged processes running on top. Q: Is it necessary to write proxy code for each ioctl to the commodity OS? A: Yes, but in our experience things you want to isolate do not require this. Q: So, are you claiming that context-switch time is irrelevant? A: This is the case for applications that we have ported so far. Of course, for applications that do more kernel calls the performance impact would be greater. Q: In your performance evaluation you compare overhead of Proxos against Linux running on top of Xen. What would be the overhead compared to Linux running on hardware? A: We haven't done such experiments but the overhead can be estimated based on available performance data for Xen.

■ *Connection Handoff Policies for TCP Offload Network Interfaces*

Hyong-youb Kim and Scott Rixner, Rice University

Hyong-youb Kim focused on efficient utilization of TCP offload capabilities available in some modern NICs. Whereas offload-capable NICs can potentially improve the performance of network-intensive applications by taking over some of the TCP processing load, it turns out that if used without care this feature can easily saturate the NIC and degrade the overall system performance.

Three techniques for optimizing connection handoff policy were proposed:

1. Prioritize packet processing on the NIC by giving packets handled by the host processor higher priority than packets processed by the NIC.
2. Dynamically adapt the number of TCP connections handled by the NIC based on the length of packet queues.
3. Compensate for the handoff cost by offloading long-lived connections to the NIC.

The proposed techniques were evaluated using a system simulator modeling the NIC as a MIPS processor with 32 MB of RAM.

Q: If there is a sudden change in number of received packets, would there be a lot of overhead in switching? For example, what happens if someone wants to DoS by starting to send lots of long packets and then stopping? A: The NIC currently does not switch connections back to the host, so there's no overhead involved in reducing the number of connections. If the host wants to hand a connection to the NIC it has to transfer a small buffer, but it's cheap. Q: You simulate the NIC as a single general-purpose processor; however, actual network processors are more complex and have multiple specialized cores. Are your results representative of what would be observed on real hardware? A: Network processors are not good at handing NIC workloads, so they should not be used for NICs. Network processors are built for switching packets. NIC workloads are different, and network processors do not work well.

DISTRIBUTED STORAGE AND LOCKING

Summarized by Leonid Ryzhyk

■ *Ceph: A Scalable, High-Performance Distributed File System*

Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, and Carlos Maltzahn, University of California, Santa Cruz

Sage A. Weil described Ceph as a high-performance distributed file system designed to accommodate hundreds of petabytes of data. The main principles underlying the Ceph architecture are separation of data and metadata, use of intelligent storage devices, and adaptable dynamic metadata management.

In his talk, Sage described the two main components of Ceph, the metadata store called MDS and the distributed object store called RADOS. MDS achieves excellent scalability by dynamically partitioning the directory tree among metadata servers based on metadata access frequencies. In addition, MDS simplifies the metadata structure by replacing file allocation data with seeds to a system-wide well-known hashing function called CRUSH, which is very stable in the face of storage device failures and other storage capacity changes.

The RADOS object store provides scalability and reliability through replication, failure detection, and recovery. RADOS achieves scalability by using intelligent object store nodes that take advantage of a very compact representation of the cluster state (made possible by CRUSH), enabling them to efficiently communicate with local peers to quickly recover from failures or any other changes to the cluster. To enable the use of Ceph for efficient communication, while guaranteeing data safety, RADOS implements a two-phase write acknowledgment protocol. The first acknowledgment confirms that the update has been propagated to all replicas, while the second confirms that the update has been physically committed to disk. RADOS uses the EBOFS object file system for local data storage. EBOFS implements a non-POSIX interface that supports features such as atomic transactions and asynchronous commit notifications.

Q: Does Ceph support directory move/rename? A: Yes. The only thing that moves is the inode, not the directory. Q: What is the effect of network latency on performance numbers? A: One of the assumptions is that we are deploying in a data center environment. In principle, however, you could deploy it in a wider scenario if you have sufficient bandwidth. Q: In your model, metadata lookups are done on the server. What are the characteristics of your workload that make this more appropriate for scalability than doing the lookup on the client? A: High-performance workloads with high contention for metadata require consistency to be managed within the metadata server. Q: What is the recovery strategy for metadata? A: The short-term log doubles as a journal, so if a metadata server fails, another node can rescan its journal.

■ *Distributed Directory Service in the Farsite File System*

John R. Douceur and Jon Howell, Microsoft Research

Jon Howell described Farsite as a distributed serverless file system for networks of workstations. The focus of the current talk was on the design and implementation of a distributed metadata service for Farsite. The design goals included support for fully functional directory move operations, including moves across partitions, support for atomic moves, support for load balancing, and hotspot mitigation.

The authors observe that the conventional approach to metadata partitioning based on path names precludes load balancing. Instead they suggest implementing partitioning based on hierarchical immutable file identifiers. Since the identifier hierarchy is not tied to the path hierarchy, load balancing and directory (re)naming become completely orthogonal. File identifiers are compactly represented using a variant of the Elias y coding. Efficient lookup is achieved by storing the file map in a data structure similar to the Lampson prefix table.

To implement atomic rename operations, the recursive path leases mechanism is introduced. It enables safe locking of the chain of file identifiers from the file-system root to a file, without putting excessive pressure on the root server. Finally, the Farsite metadata service minimizes false sharing by implementing a fine-grained locking scheme, where a client acquires locks for individual file metadata fields required for the requested access mode, rather than for the entire file.

Q: Does the repeated load rebalancing have an impact on the efficiency? A: Yes, one of the design decisions we made was that when we delegate load we can never coalesce it again. In practice it seems to work well. Q: Why aren't we already all using Farsite-like things on our mostly empty disks? Is there a drawback to this approach? A: The greatest limitation is that byzantine fault tolerance depends on assumptions about how many machines may fail, but if they are all running homogeneous software and have a similar vulnerability, they can all fail/misbehave in the same way. Q: Suppose we adopt a less pure approach and put some stuff inside protected infrastructure. How does that change things? A: Without having to care about byzantine fault tolerance, things would be much simpler. However, metadata load is a huge factor, which we would like to distribute among multiple machines.

■ *The Chubby Lock Service for Loosely-Coupled Distributed Systems*

Mike Burrows, Google Inc.

Chubby is a large-scale distributed lock service used in several Google products, including GFS and Bigtable. Mike focused mainly on introducing the Chubby API and the motivation behind it and describing the ways Chubby has been used, rather than on how it was implemented.

The main purpose of Chubby is to provide distributed-systems developers with a reliable and scalable implementation of the distributed consensus protocol. However, experience has shown that even if implemented as a library, the consensus protocol is still difficult to use for developers. Therefore, Chubby encapsulates it inside the familiar lock service API.

In addition to providing lock and unlock operations, Chubby allows associating small data records with locks and adopts a UNIX-like naming scheme for them, which makes it look and feel like a file system. However, it is not well suited for storing large amounts of data and lacks a number of filesystem features such as file renaming, atomic multifile operations, and partial-file reads and writes. This lack of features helps simplify the Chubby design and prevents developers from misusing it as a distributed file system.

Lock clients can be notified of certain types of events, including file content changes, file creation/deletion, and lock acquisition. Chubby is designed to support large numbers of clients per lock. Although changes of the lock ownership are typically infrequent, clients tend to periodically poll the lock, creating a lot of read traffic. To reduce this traffic, a consistent write-through client-side cache is used.

Q: Are there any examples of interactions that you had with the user community that led to the file-system abstraction? A: No, the design decision happened before the user base. I would put the main reason down to sharing an office with Rob Pike and Sean Quinlan (Plan 9 people), so everything looked like an FS. Q: Chubby allows developers to easily get reliability guarantees by using a lock server instead of a state machine. Were there other projects that had to go off and implement a state machine? A: Well, we did. There is a state machine library that we use; at present there are no other users of it. Q: It seems like large-scale tools are becoming increasingly more integrated. Have you thought about bad interactions where one misbehaving application of a tool can cause cascading failure in other applications? A: Yes, it happens all the time. My system has managed to bring down many others. What you do is analyze exactly what happened, fix your programming steps, and fix your code so that every single problem can't happen again, and of course something else happens next time.

LARGE DISTRIBUTED SYSTEMS

Summarized by Prashanth Radhakrishnan

■ *Experiences Building PlanetLab*

Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir, Princeton University

This talk, given by Larry Peterson, was about the authors' experience building PlanetLab (PL). PL is a global platform for deploying and evaluating planetary-scale network services. PL has machines spread around the world, with users' services running in a slice of PL's global resources.

The PL design was a synthesis of existing ideas to produce a fundamentally new system: It was experience- and conflict-driven.

Larry listed the requirements identified at the time PL was conceived and the design challenges they faced. Given its scale, PL had to rely on site autonomy and decentralized control for sustainability, while also managing the trust relationships between the users of PL and the owners of the machines. Next, it had to balance the need for resource isolation while coping with support for many users with minimal resources. Finally, PL had to be a stable, usable system, supporting long-running services and short experiments, while continuously evolving based on feedback.

PL's management architecture has the following key features to address the design challenges. PlanetLab Control (PLC), a centralized front-end, acts as the trusted intermediary between PL users and node owners. To support long-lived slices and accommodate scarce resources, PL decouples slice creation from resource allocation. Node-owner autonomy is achieved by making sure that only owners generate resources on their nodes and that they can directly allocate a fraction of their node's resources to virtual machines (VMs) of a specific slice. To support slice management through third-party services, PLC allows delegation of slice-creation by granting tickets to such services. For scalability, PL was designed so that multiple PL-like systems can coexist and federate with each other. As per the principle of least privilege, management functionality has been factored into self-contained services, isolated into their own VMs and granted minimal privileges. To address the resource allocation issues, PL provides fair sharing of CPU and network bandwidth and simple mechanisms to protect against thrashing and overuse. Finally, keeping PL's control plane orthogonal from the VMM, leveraging existing software, and rolling out upgrades incrementally helped PL evolve while also being operational.

Larry concluded with lessons learned from their experience. Key among them was the observation that decentralization follows centralization; that is, a centralized model is important for a system to achieve critical mass, and it is only by federation that the system can scale.

During the Q&A session, Sean Rhea of Intel Research Berkeley asked about Larry's comments on the proposal to set aside physical boxes for measurements. Larry said he was not convinced about reserving physical resources, but rather thought that logical isolation was sufficient. David Anderson of CMU noted that Larry's talk presented a rosy picture of PL, in contrast to the PL panel in WORLDS '06 that discussed problems with PL. David asked about the observed problems with running latency-sensitive services, disk thrashing, and scheduling. Larry said that there was room for improvement in scheduling. He also noted that since the PL code is available, the community was welcome to track down bugs that hamper their research and report patches. He said that there was a known kernel bug that could cause problems with latency-sensitive slices and that things would improve when the next kernel upgrade is rolled out.

■ *iPlane: An Information Plane for Distributed Services*

Harsha Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, and Arvind Krishnamurthy, University of Washington; Arun Venkataramani, University of Massachusetts Amherst

Harsha Madhyastha presented iPlane, a service that provides accurate predictions of end-to-end Internet path performance. He started off with the observation that large-scale distributed services, such as BitTorrent, depend on information about the state of the network for good performance. But most current Internet measurement efforts, such as GNP and Vivaldi, provide only latency predictions between a pair of nodes. In contrast, iPlane measures a richer set of metrics, such as latency, loss rate, and available bandwidth.

iPlane continuously performs measurements to generate and maintain an atlas of the Internet by doing traceroutes from a few distributed vantage points. For scalability, targets are clustered on the basis of BGP atoms and a representative target from each atom is used to approximate the performance of targets in the atom.

iPlane uses structural information such as the router-level topology and autonomous system (AS) topology to predict paths between arbitrary nodes in the Internet. This prediction is made by composing partial segments of known Internet paths so as to exploit the similarity of Internet routes. Next, iPlane measures the link properties in the Internet core and edge. In the Internet core, the special vantage points measure the link attributes. Link properties at the Internet edges are obtained by participating in BitTorrent swarms and measuring the links to the end-hosts while interacting with them.

Thus, to measure path properties between any two hosts, first the path between them is predicted. Then, iPlane composes the measured properties of the constituent path segments to predict the performance of the composite path.

iPlane has been demonstrated to improve the overlay performance of several representative overlay services such as content distribution networks, swarming peer-to-peer filesharing, and VoIP.

During the Q&A session, Buck Krasic of the University of British Columbia observed that participating in BitTorrent swarms to measure Internet edge link properties might result in conservative estimates; for example, BitTorrent clients may have multiple connections open and the bandwidth that iPlane observes might just be a fraction. He asked whether iPlane had some technique to compensate for that. Harsha replied that they were using BitTorrent to measure bandwidth capacity, not the available bandwidth, and that bandwidth capacity can be measured using a pair of back-to-back packets. And since measurements from BitTorrent are based on passive monitoring of a TCP connection of several packets, it is likely that at least one pair of back-to-back packets will be observed.

■ *Fidelity and Yield in a Volcano Monitoring Sensor Network*

Geoff Werner-Allen and Konrad Lorincz, Harvard University; Jeff Johnson, University of New Hampshire; Jonathan Lees, University of North Carolina; Matt Welsh, Harvard University

Geoff Werner-Allen presented this science-centric evaluation of a 19-day sensor network deployment at Reventador, an active volcano in Ecuador. The data collected by the sensor network deployment was evaluated based on five metrics, namely, robustness, event detection accuracy, data transfer performance, timing accuracy, and data fidelity. Of these, Geoff dealt with robustness, timing accuracy, and data fidelity in his talk.

The sensor hardware they used for volcano monitoring was small and provided real-time data acquisition, unlike conventional standalone dataloggers, which are unwieldy, and it logged data to a local flash drive. Their sensor network contained 16 sensor nodes, each equipped with a seismometer, a microphone, and an antenna. These nodes continuously sample seismic and acoustic signals and log the data to local flash memory. They also run an event-detection algorithm that transmits a time-stamped report to the base station upon detection of a seismic event. The base station, located 4.6 km away from the sensor deployment, initiates data collection if it receives triggers from more than 30% of the sensor nodes within a certain time.

The overall robustness of the system was limited by power outages at the base station and a single three-day software failure. Discounting these, the mean node uptime exceeded 96%, indicating that the sensor nodes themselves were reliable. Flooding Time Synchronization Protocol (FTSP) was used for time synchronization between sensor nodes. Although predeployment results with FTSP were good, during deployment they ran into stability issues, leading to occasional incorrect time-stamp reports. They developed a time rectification approach that filters and remaps recorded time stamps to accurately recover timing despite the incorrect time stamps. They evaluated the fidelity of the collected data by performing an analysis of the seismic and acoustic signals from a seismological perspective. Their results indicate that the collected signal quality and timing match the expectations of the infrasonic and seismic activity produced by the volcano.

Geoff concluded his talk with the three lessons they learned from this deployment: Ground truth and self-validation are critical, network infrastructure is more brittle than sensor nodes, and it's important to build confidence with domain scientists.

During the Q&A session, Geoff was asked whether they look at sensor networks as a tool that scientists in other domains could use without requiring the computer scientist's presence. Geoff responded by stating that this was a great observation and that they wanted sensor networks to eventually be used like that. Someone from Stony Brook University asked whether it was possible to simply broadcast time from the base station. Geoff replied that such a broadcast would work only with single-hop networks, which isn't the case with the Reventador deployment. Mehul Shah of HP Labs asked about the fidelity of the measurements with respect to its relevance to the domain scientists. Geoff said that the scientists are still working on the results obtained and that their initial observations are encouraging.