conference reports

THANKS TO OUR SUMMARIZERS

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

Boston, MA
April 22–24, 2009

TRUST AND PRIVACY

Summarized by Michael Golightly (mgolight@princeton.edu)

- ***TrInc: Small Trusted Hardware for Large Distributed Systems***

Dave Levin, University of Maryland; John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda, Microsoft Research

Awarded Best Paper!

Dave described how equivocation, making conflicting statements to others, is a very common and powerful tool for selfish and malicious users in distributed systems. It occurs in the Byzantine general's problem, voting, and BitTorrent, where traditionally 3f+1 users are needed to tolerate f malicious users. By using trusted hardware, equivocation can be made impossible, and now only 2f+1 users are needed to reach consensus. To be practical, such trusted hardware needs to be small in order for it to be easily verifiable, ubiquitous via low cost, and tamper resilient. Dave then displayed a SmartCard that had TrInc, a trusted incrementer, implemented on it. TrInc consists only of a monotonically increasing counter and a key for signing attestations; a set of TrInc counters makes up what is called a trinket. There are two types of TrInc attestations: an *advance* attestation that increments a counter and *forever* binds a message to the counter's value, and a *status* attestation that allows peers to determine others' current counter values.

TrInc was used to implement trusted append-only logs that emulate attested append-only memory (A2M), which has been shown to solve Byzantine Fault Tolerance with fewer nodes. TrInc can also solve the problem of underreporting in BitTorrent. In this scenario, the counter represents the number of pieces the peer has received, and peers attest to what pieces they currently hold, along with the most recent piece they have received. Peers attest when they receive a piece and when they synchronize their counters with one another. With TrInc, users can tell if a peer is underreporting and can choose to stop communicating with that peer.

TrInc was also applied to PeerReview to drastically reduce communication overhead in the system, and it can be used to ensure fresh data in DHTs and to prevent Sybil attacks. The macro-benchmarks for the asymmetric performance of TrInc were shown to be around 200–225ms for advance and status attestations, while the equivalent symmetric attestations were about 100–150ms. These operations are slow because trusted hardware is typically designed to be used for bootstrap-

ping and not in the manner that TrInc wishes to use it, but the hardware can be made faster.

Someone asked how secure TrInc would be in highly sensitive applications such as voting, and the response was that more investment would be made to make the trusted hardware resilient against reverse engineering and similar attacks in such scenarios. Another question was if counters could overflow and if they could be reset. The response was that TrInc assigns each counter a unique identifier, and overflow and resetting are handled by creating a new counter with a new unique identifier.

- ***Sybil-Resilient Online Content Voting***
  Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian, New York University

Tran described how Sybil attacks pollute voting results in popular Web sites, such as Digg, by out-voting legitimate users. Sybil attacks are hard to defend against in such systems because it is easy to create user accounts that are not strongly connected to identity. Defenses against these attacks then need to be based on resources that cannot be easily acquired in abundance, such as links in a social network.

Tran presented SumUp, a Sybil-resilient vote aggregation system that leverages the trust network among users. Using a max flow algorithm to collect votes at a central collector in a social network, bogus votes become congested at attack edges. To avoid congesting honest votes, link capacity assignment is done through a ticket distribution method that iteratively adjusts the number of tickets issued until the final number of votes collected approximates the number of honest votes expected to exist in the system. This approach assigns greater capacity to those edges closest to the vote collector, limiting the number of bogus votes collected. If an attacker manages to create attack edges in the legitimate network close to the vote collector, SumUp can leverage user feedback to reduce capacity on them or possibly ignore them altogether.

Simulations were conducted of SumUp's performance on social networks and voting traces from YouTube, Flickr, and a synthetic model. In all networks, SumUp was able to collect greater than 90% of honest votes, and the average number of bogus votes per attack edge was close to one or very small, even when all nodes voted. To evaluate SumUp on Digg, the vote collector was designated to be Kevin Rose, the founder of Digg, and then SumUp was run for all votes cast before an article was marked popular. An article was considered normal if SumUp collected more than 70% of all votes; otherwise it was deemed to be suspicious. From manual inspection, some of the suspicious articles were found to be composed of advertisements or phishing articles, indicating that a Sybil attack had in fact taken place.

Someone asked if an attacker can manipulate voting results if he knows who the vote collector is. The response was that if the attacker can create attack edges closer to the collector, he can make his votes count more, but the feedback mechanism of SumUp can help alleviate this problem.

- ***Bunker: A Privacy-Oriented Platform for Network Tracing***
  Andrew G. Miklas, University of Toronto; Stefan Saroiu and Alec Wolman, Microsoft Research; Angela Demke Brown, University of Toronto

Stefan described how network tracing is indispensable in areas like traffic engineering and fault diagnoses, but that issues of data being lost, misused, stolen, or accidentally disclosed raise many security and privacy concerns. Data must then be anonymized in a way that preserves meaningful information but destroys anything that can be used to identify users. Performing this anonymization offline has high privacy risks, while an online approach requires high engineering costs to process packets at line speed.

To solve this problem, the authors presented Bunker, a network-tracing system that buffers raw data on disk, only allowing anonymized information out. The logical design of the system has capture hardware directly interfaced with a closed-box virtual machine (VM) that encrypts data and moves it to disk for offline analysis. A separate open-box VM then provides access to trace data using a separate network interface card. A debugging configuration enables all drivers and allows access to the closed-box VM, while a tracing configuration disables all unnecessary I/O and drivers from the kernel and uses firewalls to allow access only through the open-box VM. Bunker took two months to develop, and its code base is an order of magnitude smaller than previous online tracing tools, since analysis of anonymized data can now be done, offline, however the user wishes.

Bunker's trusted computing base and narrow interfaces provide high security. Resources are isolated between the open-box and closed-box VMs, and a safe-on-reboot feature protects against many hardware-based attacks. Bunker may be vulnerable to cold-boot attacks and bus monitoring, but secure co-processors could provide a defense against those attacks.

Someone asked what the authors have learned trying to sell Bunker, with its admitted vulnerabilities, to network operators and whether they are looking for proof that Bunker is secure. The response was that a proof would be great, but given that one does not exist, carefully explaining policies and documentation helps operators to support Bunker. Another question was asked about how Bunker protects against human errors in the anonymization of data. The response was that Bunker provides the tools for anonymization; it is still up to operators to inspect their code and policies to make sure data is anonymized correctly. Someone else asked how useful Bunker's security model was, given that once attackers have physical access to the machine they can install a network tap anyway. Stefan said that Bunker reduces liability but does not stop someone with a subpoena from installing a network tap.

STORAGE

Summarized by Evan Jones (evanj@mit.edu)

- ***Flexible, Wide-Area Storage for Distributed Systems with WheelFS***
  Jeremy Stribling, MIT CSAIL; Yair Sovran, New York University; Irene Zhang and Xavid Pretzer, MIT CSAIL; Jinyang Li, New York University; M. Frans Kaashoek and Robert Morris, MIT CSAIL

Jeremy Stribling presented WheelFS, a distributed file system designed to operate over wide area networks. Operating across a wide area network presents many challenges due to the fundamental latencies between sites and the higher probability of link failures. WheelFS's design is based on the observation that many applications each design their own distributed storage layer because they make different design choices for how to handle these challenges.

WheelFS provides a single file system namespace where these choices can be made on a per-file basis, by attaching what the authors call "semantic cues." These cues are special strings embedded in the file path—for example, /wfs/.MaxTime=200/url, which specifies that the system should take a maximum of 200 milliseconds to try to locate the most recent copy, then return an error or whatever latest version was found. These cues can be used to implement a variety of services with very different requirements, such as a traditional distributed file system with strong close-to-open consistency, or a distributed Web cache with much weaker consistency but lower latency.

Jeremy was asked if he had any big lessons after examining many different storage systems, and implementing WheelFS. His answer was that most applications need the same policy for both reads and writes, and a simple interface. The software and additional information can be found at http://pdos.csail.mit.edu/wheelfs.

- ***PADS: A Policy Architecture for Distributed Storage Systems***
  Nalini Belaramani, The University of Texas at Austin; Jiandan Zheng, Amazon.com Inc.; Amol Nayate, IBM T.J. Watson Research; Robert Soule, New York University; Mike Dahlin, The University of Texas at Austin; Robert Grimm, New York University

Nalini Belaramani presented PADS, a system to make it easy to build a custom distributed storage system. It grew out of the work on PRACTI, which took a microkernel approach, by providing a number of small building blocks that could be combined in interesting ways. However, Nalini found PRACTI too difficult to use to build a complete system. PADS addresses this problem by reducing the design of distributed storage systems to two parts: routing policy and blocking policy.

Routing policy specifies how data flows between nodes. The primary abstraction for routing is the subscription, which provides a flow of updates between nodes. Subscriptions propagate events that contain the updates to data objects. Triggers are points where the routing policy can make decisions, such as when a read blocks to obtain the most recent data. Routing policy is specified using Overlog, a domain-specific language for building peer-to-peer systems based on Datalog. Blocking policy specifies when operations should block in order to maintain the guarantees the storage system wants to provide. It is specified as a list of conditions for blocking points at data access. PADS provides built-in conditions as well as allowing system designers to implement custom conditions. The authors used PADS to build 12 different distributed storage systems, ranging from CODA to TierStore. Each one can be specified using fewer than 100 routing rules and 6 blocking conditions.

Nalini was asked what the division should be between a domain-specific language and a library in a system like this. Nalini answered that PADS' main contribution is the abstraction of routing and blocking policies. While Overlog helps, you could use Java with PADS if you wanted. It would still make the job easier. What about performance of the routing policies, since Datalog can be slow with large amounts of data? PADS does not maintain much state in their custom implementation, so it has not been an issue.

WIRELESS #1: SOFTWARE RADIOS

Summarized by Patrick Verkaik (pverkaik@cs.ucsd.edu)

- ***Sora: High Performance Software Radio Using General Purpose Multi-core Processors***
  Kun Tan and Jiansong Zhang, Microsoft Research Asia; Ji Fang, Beijing Jiaotong University; He Liu, Yusheng Ye, and Shen Wang, Tsinghua University; Yongguang Zhang, Haitao Wu, and Wei Wang, Microsoft Research Asia; Geoffrey M. Voelker, University of California, San Diego

  Awarded Best Paper!

Kun Tan presented Sora, an implementation of an 802.11a/g SDR (software-defined radio) on a commodity PC architecture. In SDR, the goal is to implement as much of the wireless protocol in software as possible, so that it is useful for research, development, and testing. However, achieving this goal is hard, because transferring and processing radio signals requires large I/O bandwidth (several Gbps) as well as a lot of computation. In addition, protocols such as 802.11 define very tight deadlines (microseconds) to generate responses. Therefore, up until now SDR has often made use of FPGAs, which can meet these performance requirements but are not very programmable, or sacrificed throughput to programmability when using a general-purpose processor.

Enter Sora, which is an SDR implementation based on a general-purpose processor architecture, yet can operate at the highest 802.11a/g MAC rates. Sora achieves this by making use of current commodity hardware (PCI express and multicore processors) combined with clever optimizations. The radios are located on a PCI-e card that contains a minimal amount of logic. As examples of optimizations, Sora trades memory for calculation using lookup tables that

still fit in an L2 cache, takes advantage of SIMD instructions to exploit PHY data parallelism, and carefully allocates tasks to multiple cores and schedules them at compile time. Together, these optimizations achieve a 10–30x speedup as well as an end-to-end throughput comparable to commercial (hardware-based) implementations. Taking advantage of SDR, the team experimented with several modifications to 802.11, such as a TDMA MAC and jumbo frames. Kun also showed a screenshot of a nice visualization tool.

Someone asked whether Sora could be used for power-constrained platforms. In Kun's view, SDR is currently useful mostly for experimentation rather than practical deployment, so energy use is not a concern. However, Sora could be deployed in base stations. Someone mentioned that a lot of the finer details of 802.11 deal with low-performance situations, such as weak signal strength and multipath. Kun observed that multipath is everywhere, and getting good throughput means that you must have handled it.

Not only did Sora win a Best Paper award, but Kun's demo at the reception also won Best Demo!

- ***Enabling MAC Protocol Implementations on Software-Defined Radios***
  George Nychis, Thibaud Hottelier, Zhuocheng Yang, Srinivasan Seshan, and Peter Steenkiste, Carnegie Mellon University

George Nychis presented their work on implementing software-defined radios (SDR) using a "split functionality" approach. The idea is to place a small, performance-critical part of the SDR on the radio hardware (small enough to allow low cost and complexity) and the remainder on the host, where it can be easily programmed, and connect the two through an API. In contrast with the previous talk, on Sora, this work can still use a USB-based USRP radio and is claimed to be more independent of specifics of the host architecture, such as the instruction set architecture and the latency from the radio to the software part.

What components should the high-performance toolbox consist of? George presented two of the components they developed: precision scheduling and fast packet detection. For precision scheduling, the host is in charge of scheduling an event (such as sending a packet) and specifies a time. The actual triggering at the given time, however, is performed by the hardware. The goal of fast packet detection is to detect whether an incoming signal is a packet *before* demodulating the signal, an optimization that saves processing power and allows faster turnaround time. In the split-functionality architecture, the host modulates the framing bits and passes that to the hardware. The hardware can then correlate an incoming signal with the modulated framing bits and detect an incoming packet. George described how they used their toolbox to implement 802.11- and Bluetooth-like protocols and compared their performance with host-based implementations of these. They found that while in terms of absolute performance both are limited by USRP, the split-functionality approach enables a throughput improvement of 2–4x over the host-based implementation.

An audience member asked how resilient the API is to protocols that have very specific features, such as virtual carrier sense in 802.11. George explained how they dealt with virtual carrier sense in particular, but he doesn't claim the split-functionality approach can handle everything. They are currently working on "fast ACKs": premodulating an ACK packet so that it can be sent quickly, yet allowing the ACK to contain the source address of the packet it is responding to. Someone else asked how generic the API is in considering new protocols, since in sensor networks it has turned out very hard to come up with a stable API. George answered that it is hard to say whether the toolbox set is ever complete. Instead, they try to make it so that the API can be tweaked easily to support such new protocols.

CONTENT DISTRIBUTION

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

- ***AntFarm: Efficient Content Distribution with Managed Swarms***
  Ryan S. Peterson and Emin Gün Sirer, Cornell University and United Networks, L.L.C.

Ryan S. Peterson presented AntFarm, a content distribution scheme that manages swarms of clients downloading a set of files. Instead of other approaches like the client/server model or traditional peer-to-peer networks (e.g., BitTorrent), an AntFarm coordinator actively manages content servers, seeds, and leechers by issuing tokens that clients can exchange for blocks of the files they desire.

The AntFarm coordinator uses an iterative algorithm to allocate bandwidth to target the highest aggregate bandwidth relative to seeder capacity. AntFarm significantly outperforms BitTorrent because it can optimize bandwidth use. Unlike BitTorrent's random unchoking, AntFarm specifically allocates seeders to new swarms. The coordinator algorithm scales linearly to more hosts, and a single machine can calculate allocations for 10 thousand swarms and 1 million peers in only 6 seconds.

- ***HashCache: Cache Storage for the Next Billion***
  Anirudh Badam, Princeton University; KyoungSoo Park, Princeton University and University of Pittsburgh; Vivek S. Pai and Larry L. Peterson, Princeton University

Anirudh Badam presented HashCache, a new algorithm for indexing a Web cache. In developing regions, Internet bandwidth is prohibitively expensive ($1500/Mbps/month), which makes Web caching very desirable. Although hard disks have been getting much cheaper ($100 for a 1TB drive), the memory required to index larger drives (10GB for a 1TB index) is expensive.

The solution, HashCache, calculates the hash of a URL and organizes the file system as the hash space. The basic version of HashCache stores metadata in the first block of the disk, and therefore is optimized for a single disk seek per URL lookup. A more advanced version uses a configurable amount of memory for the cache index, uses 20–50x less

memory than Squid (an open source Web cache) and 6–10x less memory than Tiger (a commercial Web cache) while maintaining comparable performance.

Someone asked about the need for larger disk caches, since the advantage of a cache drops off quickly as the size of the cache grows. Anirudh replied that a larger cache allows for additional applications such as WAN acceleration and prefetching.

- ***iPlane Nano: Path Prediction for Peer-to-Peer Applications***
  Harsha V. Madhyastha, University of California, San Diego; Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy, University of Washington; Arun Venkataramani, University of Massachusetts Amherst

Harsha V. Madhyastha presented iPlane Nano. Rather than P2P applications each trying to measure Internet paths independently, iPlane Nano provides a shared solution for other applications to use. iNano's approach is similar to the previous iPlane in that it predicts the AS-level paths between end hosts, but instead of keeping a large database of paths, iNano uses a compact atlas of measured links. By choosing two links that intersect, the iNano algorithm can infer the AS-level path correctly 70% of the time, while using three orders of magnitude less storage space for the atlas (7MB versus 2GB). The atlas itself is updated daily, with 80% of the links staying the same between updates.

Someone asked what happens when the prediction is incorrect. Harsha replied that it does help applications choose peers, even if incorrect some of the time. Why download the atlas, as opposed to simply querying a server? For applications such as BitTorrent, the load on a query server would be very high. What is the overhead of the iNano measurements? They are simply traceroutes, so they are low-cost (100KB of bandwidth per day).

BFT

Summarized by Wyatt Lloyd (wlloyd@cs.princeton.edu)

- ***Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults***
  Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mike Dahlin, The University of Texas at Austin; Mirco Marchetti, The University of Modena and Reggio Emilia

Allen Clement noted that, contradictorily, all existing Byzantine Fault Tolerant (BFT) systems perform poorly or crash in the presence of Byzantine faults. In the quest for higher and higher throughput numbers, BFT system designers have adopted frailer optimizations that increase best-case performance but decrease worse-case performance. These fragile optimizations also introduce new corner cases that designers can easily overlook when implementing their protocols. Thus, the new goal for BFT research is to design robust systems that tolerate and even perform well under the Byzantine faults they were designed to tolerate.

Aardvark, the first system in the new spirit of robust BFT, challenges the conventional wisdom used in designing conventional BFT systems. It uses public-key cryptography to authenticate clients instead of MACs. It explicitly isolates its resource. For instance, it requires separate wires and separate NICs for each communication pathway. Finally, Aardvark regularly executes view-changes to continue rotating which replica is the primary. These design decisions, especially to use public-key cryptography, were traditionally considered too computationally expensive. However, Aardvark achieves a peak throughput of 38667 ops/sec compared to PBFT's 61710 ops/sec and Zyzzyva's 65999 ops/sec.

One attendee asked why a MAC was being sent along with the client signature for requests. The speaker explained that misbehaving clients are blacklisted and that the primary can identify the client who sent a request using a MAC with significantly less computation than a signature. Another attendee noted that people in the real world don't think BFT is worth the performance hit and that systems like Aardvark increase this hit further. The speaker said there are always tradeoffs in designing a system. A third attendee asked if they were sure there are no attacks related to multiple processor speeds. The speaker replied that they focused on systems with homogeneous processors.

- ***Zeno: Eventually Consistent Byzantine-Fault Tolerance***
  Atul Singh, MPI-SWS and Rice University; Pedro Fonseca, MPI-SWS; Petr Kuznetsov, TU Berlin/Deutsche Telekom Laboratories; Rodrigo Rodrigues, MPI-SWS; Petros Maniatis, Intel Research Berkeley

Atul Singh said that availability has become king in the design of Web sites, with each hour of downtime costing major sites between $55,000 and $500,000. While some of these sites are designed to prevent crash faults, in practice Byzantine faults occur and have disastrous consequences. Combining these observations motivates Zeno, a Byzantine Fault Tolerant (BFT) system that strives to meet modern availability requirements.

All existing BFT protocols strive for strong consistency and will block if less than two-thirds of replicas are reachable. Zeno's key idea is relaxing consistency for availability: make the service available when other replicas are not reachable even though this will allow temporarily divergent state. Zeno implements eventual consistency, meaning clients will not always see the effects of other clients' operations immediately, though eventually they will all be coalesced.

Zeno has two types of operations: strong and weak. Strong operations function like normal BFT operations, require strong quorums of 2f+1 replicas, and have unique sequence numbers. Weak operations have eventual consistency semantics, only require weak quorums of f+1 replicas, and do not always have unique sequence numbers. When a network partition occurs that prevents strong quorums, strong operations cannot complete until the partition is healed and preceding weak operations that completed during the partition are merged. These merges require the partitions to roll back their state until they agree, agree on an order of operations, and then play forward those operations.

An attendee commented that it would be interesting to explore structured partitions instead of arbitrary partitions and the speaker concurred. Another attendee asked what happens to weak operations that complete before strong operations when state is rolled back for merges. Atul answered that the result seen by the weak operations may not be state that is actually represented in the final history of the system. Another attendee followed up by commenting that a client's future operations may be based on the inaccurate results of previous operations. Atul replied that if that was the case, strong operations should be used. A fourth attendee stated that a Byzantine node could always cause divergent views that would need to be merged. Atul said that if signatures were used, only the primary could do this.

EVALUATION/CORRECTNESS

Summarized by Evan Jones (evanj@mit.edu)

- ***SPLAY: Distributed Systems Evaluation Made Simple (or How to Turn Ideas into Live Systems in a Breeze)***
  Lorenzo Leonini, Étienne Rivière, and Pascal Felber, University of Neuchâtel, Switzerland

Étienne Rivière presented SPLAY, a system for building, deploying, and evaluating distributed systems. It is based on the observation that building large-scale systems is difficult, which leads many researchers to use simulations or small controlled deployments. SPLAY tries to make it easier to build real systems. It is designed to help users with all parts of the development process: implementation, deployment, and evaluation.

SPLAY exposes a Lua programming language interface. Lua is a high-level, dynamic programming language. Its concise and clear syntax, combined with SPLAY's libraries, produces implementations that can look very similar to the pseudocode describing the algorithms. As an example, Étienne showed the SPLAY implementation of Chord beside the pseudocode from the original paper. To assist with deployment, SPLAY requires a single lightweight daemon to be installed on each machine that participates in the system. Multiple SPLAY applications can then be deployed using a command-line or Web-based interface. Each application runs in its own sandbox, providing resource isolation. When evaluating a system, the SPLAY controller collects log data from multiple systems, which are then combined back on the user's machine, making it as easy to collect data as with simulations. Additionally, the controllers can be used for reproducible churn experiments, where the same set of node joining and leaving events can be replayed.

Étienne was asked if SPLAY can assist in validating simulation results. He said that the end user still must do this work, as SPLAY only provides infrastructure for running systems and does not understand any high-level information about the application. While the SPLAY implementation can be run on different testbeds, such as multiple processes on the local machine, PlanetLab, Emulab, or a private network of workstations, it currently does not support simulators. Could they reproduce the strange transient behavior observed on PlanetLab? While SPLAY can reproduce churn, it does not record and replay other kinds of events. Does SPLAY provide tools to build systems that are topology-aware, such as choosing local peers? While SPLAY does not have any tools like that in its set of libraries, the raw APIs are accessible, so they could be built. SPLAY is available at http://splay-project.org/.

- ***Modeling and Emulation of Internet Paths***
  Pramod Sanaga, Jonathon Duerig, Robert Ricci, and Jay Lepreau, University of Utah

Jonathon Duerig presented his work on emulating Internet paths. When evaluating a system using a tool such as Emulab, users would like to be able to emulate behavior that is observed between two hosts on the Internet. Previous work provides ways to emulate the characteristics of single links. Emulating Internet behavior would require many links, each of which needs to be provided many specific parameters, such as queue sizes, delay, and data rate. Instead, this work attempts to provide accurate modeling of WAN paths using a single link, with some additional parameters. The techniques that Jonathon presented are tuning queue sizes, separating the effects of capacity and available bandwidth, and reactivity of cross traffic.

First, to emulate a WAN path the queue size must be set appropriately. In this work, both a lower and upper bound on the queue size are derived using both the desired bandwidth-delay product and the available bandwidth. This frequently leads to sizes which are not satisfiable, due to the lower bound being greater than the upper bound. To solve this, the authors observed that in real paths, the path capacity—the rate at which all packets are transmitted on the path—is different from the available bandwidth, the rate at which the application's packets are transmitted. Thus, the capacity can be adjusted until the queue sizes can be satisfied. Then constant bit-rate cross traffic is added to leave the desired available bandwidth on the path. Next, the cross traffic must react to the foreground traffic, as it would on the real Internet. This work adjusts the cross traffic as a function of the number of foreground flows. To evaluate this emulation, Jonathon presented results comparing measured performance on the Internet with emulated paths, showing that the bandwidth and latency are within 10% of the measured values.

Jonathon was asked about the distribution of round-trip times, which are more noisy on the Internet than in the emulation. His answer was that the model captures the high-level RTT behavior, but the individual RTT distribution will be different from the Internet RTTs. Why didn't they compare PlanetLab performance to their emulation for the BitTorrent experiments? From their previous work, they found that host contention on busy PlanetLab nodes makes it very difficult to measure the actual network conditions for typical applications under normal loads. Had they considered providing pre-defined scenarios, based on careful measurements? This would make it easier for researchers

to do experiments without setting hundreds of parameters. Jonathon said that is the ultimate goal of this research.

- ***MoDist: Transparent Model Checking of Unmodified Distributed Systems***
  Junfeng Yang, Columbia University and Microsoft Research Silicon Valley; Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, and Mao Yang, Microsoft Research Asia; Fan Long, Tsinghua University; Lintao Zhang and Lidong Zhou, Microsoft Research Asia and Microsoft Research Silicon Valley

Junfeng presented MoDist, a system for finding bugs in distributed systems implementations. The standard technique is stress testing or randomized testing using a synthetic workload. However, these tests do not trigger many of the rare corner cases. MoDist addresses this challenge through model checking techniques. Model checking exposes all possible actions at each state. To eliminate redundant sequences of actions, MoDist uses partial order reduction and remembers previously visited states. Unlike other systems, it runs unmodified applications on top of the operating system. A lightweight system call interposition layer makes executions deterministic and reproducible, as well as being capable of injecting errors. A static analysis technique is used to expose implicit timers as actions to the model checker. A set of default checks are performed at each state, and users can supply additional checks, including checks over the global state.

To use MoDist, the developer supplies a configuration file, telling it how to start the initial processes. MoDist runs the processes and explores the state space. When it finds a bug, it writes a trace file. This trace file can be fed back into MoDist to reproduce and debug the error. Junfeng presented a bug that was found in Berkeley DB after running for an hour. The authors used MoDist to test three systems: Berkeley DB; Microsoft's Paxos implementation, called MPS; and Pacifica, a distributed storage system. It found 35 bugs, 31 of which were confirmed by the original developers. Ten of those were serious protocol-level bugs.

Junfeng was asked how MoDist's implementation compares to work designed for checking multi-threaded systems. He said that MoDist's implementation handles threads as well as communication in distributed systems. There are different kinds of failures in distributed systems, so it is unclear how it could be used for multi-threaded systems.

- ***CrystalBall: Predicting and Preventing Inconsistencies in Deployed Distributed Systems***
  Maysam Yabandeh, Nikola Knežević, Dejan Kostić, and Viktor Kuncak, EPFL

Dejan presented CrystalBall, a system for finding and preventing bugs in distributed systems. CrystalBall can help find these bugs and prevent them from causing inconsistencies in deployed systems. The idea is to use model checking to see whether potential future actions can lead to inconsistencies or other errors. This can find bugs that typical model checking would not, since it examines states that are far from the initial conditions. These are relevant states because the search begins from a state observed in the real deployment. CrystalBall can, in most cases, prevent a bug it has found from violating safety properties. In order to model check the system at each node, it collects a consistent snapshot of a node's neighborhood, along with the normal messages. When an action arrives that CrystalBall has determined could lead to an inconsistency, it prevents it with a filter. It uses filters to cause events that could happen due to other reasons, such as breaking a TCP connection instead of delivering a message that triggers a bug.

CrystalBall is based on the MaceMC model checker, and thus systems are implemented in Mace. CrystalBall was evaluated using the Mace implementations of RandTree, Chord, and Bullet, using 6–100 participants on 25 machines. They found seven inconsistencies that were not found by MaceMC or by manual debugging. They also looked at a Paxos implementation where they injected two failures that were reported in previous research. Execution steering was able to avoid the inconsistencies in 95% of the random runs they examined. The performance impact was less than 5% for BulletPrime downloads, due to the additional overhead of transmitting checkpoints.

Dejan was asked to comment on the CPU overhead. CrystalBall fully utilizes one CPU on each node in order to model-check future states. What about systems that are multi-threaded and scale up with more CPUs? It would be possible to parallelize the model checker in order to explore states in parallel. What was the size of the state space? In order to explore eight levels, it takes approximately 600KB of RAM. Thus, this fits into the L2 cache of most CPUs.

WIDE-AREA SERVICES AND REPLICATION

Summarized by Wyatt Lloyd (wlloyd@cs.princeton.edu)

- ***Tolerating Latency in Replicated State Machines Through Client Speculation***
  Benjamin Wester, University of Michigan; James Cowling, MIT CSAIL; Edmund B. Nightingale, Microsoft Research; Peter M. Chen and Jason Flinn, University of Michigan; Barbara Liskov, MIT CSAIL

Benjamin Wester observed that replicated state machines (RSMs) are used to make services fault-tolerant. To truly achieve fault tolerance, the machines implementing the RSMs should be geographically distributed, but this can significantly increase latency. This latency can be hidden through client speculation.

Clients take a checkpoint of their state before issuing requests and then speculatively execute based on the first reply they receive. If consensus agrees with this first reply, the client continues its execution normally. If consensus disagrees with the first reply, the client rolls back its state to the checkpoint and executes based on the consensus reply. This new protocol changes the fast path of execution; now the latency of the first reply matters much more than the latency of the consensus reply.

When clients issue requests during speculative execution, this dependency must be made explicit. These dependencies can be expressed as predicates. For instance, if a client speculates it won the lottery and then issues a request to buy a car, that request should be "buy car if lottery=win." There may be a large list of these predicates—for instance, "buy insurance if lottery=win and car=bought." Client speculation was implemented on top of PBFT, with the primary sending a speculative reply as soon as it received a request. Evaluation showed that PBFT-CS was able to decrease latency under a variety of scenarios at the cost of decreasing peak throughput by 18%.

An audience member suggested using client speculation under low load and then switching it off for higher throughput under high load. Wester agreeed that the technique could work quite well and said it could be implemented easily by simply having the primary stop sending speculative replies when it was under high load. Was there a way to tether speculation across multiple RSMs? It would be possible if a distributed checkpoint and rollback mechanism was implemented, or the client could simply block before executing requests external to the system.

- ***Cimbiosys: A Platform for Content-based Partial Replication***
  Venugopalan Ramasubramanian, Thomas L. Rodeheffer, and Douglas B. Terry, Microsoft Research, Silicon Valley; Meg Walraed-Sullivan, University of California, San Diego; Ted Wobber and Catherine C. Marshall, Microsoft Research, Silicon Valley; Amin Vahdat, University of California, San Diego

Douglas Terry suggested considering a photo-sharing scenario where Alice uploads her pictures to her home PC and then tags and rates them. Then all of her photos tagged "family" should be replicated on her laptop and her Mom's computer. All her photos tagged "public" should be uploaded to her Flickr account, and all of her photos rated 5 stars should be put in her digital picture frame. This scenario leads to two observations. First, devices want to selectively replicate with each other data that they both are interested in. Second, there may not be a full mesh between all devices. For instance, Alice's Mom's computer may get photos from Alice's laptop when Alice is visiting but may have to get photos via Flickr at other times.

Cimbiosys aims to address this scenario by incorporating content-based filtering with eventual consistency. A filter selects which data items it is interested in, such as only photos with the family tag. Cimbiosys achieves what is termed "eventual filter consistency." Eventual filter consistency means that each device will eventually store the items that its filter would select from a set of all items in the entire distributed collection.

Devices synchronize to exchange items and metadata about items. Devices only transfer items and meta-data about items selected by their filter. Complications arise with this protocol when filters or items are updated so that items no longer belong to filters. For instance, a photo may be rerated to be 4 stars instead of 5. To deal with this situation, metadata about items that have fallen out of the filter is kept and propagated in synchronizations until certain conditions explained in the paper are met.

One attendee was confused about the semantic of the filter and asked if they could be composed. Terry replied that a filter's only requirement was being able to decide yes or no for every item. The same attendee asked how you could define a filter to be consistent. Terry replied that there is no notion of a filter being consistent. Another attendee asked how Cimbiosys dealt with failures. Terry replied that the system works for fail-stop faults but not for Byzantine faults. A third attendee asked what the trust model of the system was. Terry replied that they used an access control policy to govern the operations each device was allowed to perform on each item. Another attendee asked how this system is different from PRACTI. Terry replied that PRACTI provides a framework to build protocols and policies, so PRACTI could be used to implement Cimbiosys.

- ***RPC Chains: Efficient Client-Server Communication in Geodistributed Systems***
  Yee Jiun Song, Microsoft Research Silicon Valley and Cornell University; Marcos K. Aguilera, Ramakrishna Kotla, and Dahlia Malkhi, Microsoft Research Silicon Valley

When applications scale across heterogeneous and geographically diverse machines, Yee Jiun Song noted that remote procedure calls (RPCs) impose rigid and inefficient paths of communication. For instance, consider a webmail application where the front-end server communicates with an authentication server, a storage server, and an advertising server. Assuming these operations are not parallelizable, a more efficient communication path would go from the front-end server to the authentication server to the storage server to the advertising server and then back to the front-end server. RPC chains include logic along with RPCs that can be used to implement complex communication paths, such as the one described above.

The first step in creating RPC chains is embedding the chaining logic in the RPC call, by embedding C# static method names in the calls. These methods are stored at a central server so that servers may fetch them the first time they are encountered. The second step is maintaining a stack of chaining functions and state. This allows an RPC chain to spawn subchains that block its progress until they complete. The third step is allowing chaining functions to specify splits and merges so different parts of the chain can continue in parallel. With these three components, RPC chains can express complex communication paths that regular RPCs cannot. However, RPC chains make debugging, profiling, exceptions, and fault isolation more difficult.

One attendee asked about timeouts and noted that their optimizations of best-case performance would actually make worse-case performance much worse. The speaker replied that nodes along the chain are required to report back to the initiating node at every step, so liveness could be moni-

tored. Another attendee asked if the process of creating RPC chains was automated or if application developers had to do it themselves. The speaker replied that it wasn't automated but also wasn't too difficult; the webmail application chaining code was only 40–50 lines.

BOTNETS

Summarized by Patrick Verkaik (pverkaik@cs.ucsd.edu)

- ***Studying Spamming Botnets Using Botlab***
  John P. John, Alexander Moshchuk, Steven D. Gribble, and Arvind Krishnamurthy, University of Washington

John John described Botlab, which automates botnet analysis using a black-box approach (execute the bot and study its behavior). In particular they are interested in botnets that send spam. However, getting hold of such bots turns out to be tricky: running a simple honeypot did not catch any in over a month. The reason is that botnets these days expand mostly through social engineering techniques such as fake e-cards. Therefore Botlab enhances honeypots with a component that actively crawls spam emails (clicking "yes" on everything) from a spam feed from the University of Washington. Once Botlab has obtained a bot, it needs to figure out if it's a duplicate, which is challenging since bots obfuscate themselves. Botlab creates what is called a "network fingerprint" by running the bot inside a sandbox and observing what connections it creates. Botlab also uses these fingerprints to see if a bot detects whether it's running inside a virtual machine, by running the bot both inside a VM and on the bare metal and comparing its network fingerprints.

Botlab sends as many as six million spam emails per day to a wide variety of destinations (from just a dozen bots!), giving a local view of spam producers and a global view of spam produced. On the other hand, the University of Washington mail feed provides a local view of spam generated almost entirely by external producers. How do we map between these two complementary sources? The solution, as John explained, is to realize that different botnets tend to use different subjects in their spam. So Botlab identifies botnets based on email subjects. They found that 80% of spam comes from just six botnets, and most botnets contact only a small number of C&C servers. Additionally, they found a many-to-many relationship both between botnets and spam campaigns and between spam campaigns and Web hosting services.

An audience member asked how bots behave when a user is present, since the Botlab study shows that they can send very aggressively, which must surely inconvenience the user. According to John, some bots will back off when they detect mouse movement. However, they did not study this, since Botlab has no users. Another audience member observed that since the Botlab study shows that the Web hosting providers are so concentrated, they must have enormous bandwidth. John said that the bandwidth requirement depends on the click rate of users, which is pretty low after spam filtering.

- ***Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks***
  Ramakrishna Gummadi and Hari Balakrishnan, MIT CSAIL; Petros Maniatis and Sylvia Ratnasamy, Intel Research Berkeley

Ramki Gummadi presented their work on how to prove that human activity really is generated by a human rather than a bot. Currently, service availability suffers from over-zealous flagging of human activity as bot activity (preventing Ramki from sending email to his session chair!), and mail servers are getting overloaded with spam generated by bots. Ramki presented their solution, Not-a-Bot. The idea is based on having an "attester" built into each PC that checks whether some action generated by the PC (such as sending an email) was likely triggered by human activity. To do this, the attester monitors input peripherals such as the keyboard and "attests" the action if it was preceded by input device activity within some time window. The time window bounds the amount of malicious traffic a bot can generate. At the server end, a verifier is responsible for checking the attestation. For example, if a server is overloaded, it could choose to prioritize attested requests.

So where is the crypto to make it work? Many PCs today come with a Trusted Platform Module (TPM) chip. In Not-a-Bot, the TPM guards a certified key pair. On boot, the TPM verifies the integrity of the attester, after which the TPM releases its keys to it. Once everything is running, an application such as a mail client can request an attestation from the attester, which includes a signature of, say, an email and the certified public key. A nice aspect is that all this can be made to work even if the OS is compromised, so long as the attester is able to monitor peripherals without help from the OS. Ramki next described their Xen-based prototype implementation and their evaluation based on traces of clicks, spam, and DDoS. For these traces, Not-a-Bot would have removed around 90% of bot traffic.

Someone asked if it was possible to outsource TPMs similar to how captchas have been outsourced. Ramki answered that there would be little point: each outsourced TPM would only be able to generate a small amount of bad traffic. The next question was, How would Not-a-Bot cope with peripherals that require (updated) device drivers, and what about keystrokes generated by remote access? Ramki first clarified that the virtual machine implementation was just a prototype; the real thing would be using trusted hardware. Second, the input device must always be physically connected to the PC in some way, and that physical connection can be used to identify user input. At that point we ran out of time, so the remote access question did not get answered.

- ***BotGraph: Large Scale Spamming Botnet Detection***
  Yao Zhao, Northwestern University and Microsoft Research Silicon Valley; Yinglian Xie, Fang Yu, Qifa Ke, and Yuan Yu, Microsoft Research Silicon Valley; Yan Chen, Northwestern University; Eliot Gillum, Microsoft Corporation

Yao Zhao observed that Hotmail receives many signups from bots for accounts that are used to send spam. The goal of their work, BotGraph, is to mitigate such behavior based

solely on user activity logs (signups, logins, emails sent). This is a challenging problem, since each botnet may have access to many accounts and thus only needs to send a few emails from each to be effective. BotGraph introduces two new techniques. The simpler of the two looks at the number of account signups from each IP address over time and flags anomalies as malicious. This technique was able to detect 20 million malicious accounts in two months and can be executed in real-time. The main part of the talk, however, concerned the second technique, which examines the AS number of the IP address that a user connects from when logging into their email account. Human users typically share just one such AS number with other user accounts (for example, several users in the same home might share an IP address). Bots, on the other hand, work collaboratively, and their account logins tend to share multiple ASes. To distinguish between the two, BotGraph creates a graph of user accounts that weights an edge between two accounts with the number of shared ASes and subsequently considers the edges with weight greater than one. The problem then reduces to detecting a giant connected component formed by bot-controlled accounts.

The implementation of BotGraph is based on DryadLinq running on a 240-machine cluster. Yao presented a number of optimizations that reduce the runtime 5x. They performed validation of the results using a combination of manual checks on samples and a comparison with a list of Hotmail accounts known to be used by spammers. BotGraph detected 80% of known spammer accounts and discovered 54% more accounts than in the known spammer account list. In addition, BotGraph has a false positive rate of less than 0.5%. Yao claims that the only way to evade BotGraph is to be stealthy (send few emails) and bind an account through just one AS number. However, doing so would severely limit an attacker's spamming throughput.

The session chair observed that while the false-positive rate as a percentage is low, the absolute number is actually quite high. Yao answered that their estimates of the false-positive rate are conservative and probably over-estimate. In addition, a false positive doesn't mean the user account is immediately blocked. Instead, the user may be subject to an additional test to verify they are human.

NETWORK MANAGEMENT

Summarized by Eric Keller (ekeller@princeton.edu)

- ***Unraveling the Complexity of Network Management***
  Theophilus Benson and Aditya Akella, University of Wisconsin, Madison; David Maltz, Microsoft Research

High complexity in the design and configuration of enterprise networks leads to a lot of manual effort in managing the network. Theophilus Benson explained that there is currently no way to quantify how complex an enterprise configuration is. They found that complexity is unrelated to the size of the network or the line count of the configuration. Because of this, network operators cannot understand how changes they make now will affect the difficulty of future changes.

Based on a study of seven enterprise and campus networks, the authors defined three metrics which succinctly describe the design complexity, can be automatically calculated from configuration files, and are aligned with operators' mental models (i.e., they can predict difficulty of future changes). The first metric, referential complexity, is the number of references between the stanzas across all of the routers' configuration (e.g., a routing protocol references an interface, the interface stanza creates a reference to an ACL, and a separate configuration might have reference to a similar subnet). A greater number of links means higher complexity, because of the dependencies. The second metric, number of roles, was not discussed in the presentation. The third metric captures the inherent complexity of the network—identical or similar policies among all routers has low complexity; subtle distinctions across groups of users have higher complexity.

Someone asked if complexity was introduced for non-technical reasons (cost), did the network operators know what they were doing, and did the metrics help them since they knew it would be more complex? The operators did know what they were doing, so the metrics would not have helped. Why normalize by number of devices? It helps compare across networks of different sizes, but they do hope to further refine the metrics. Someone commented that the approach is pretty syntactic and asked whether they thought about the complexity of provisioning versus runtime (provisioning could be done by scripts, but runtime issues cannot)? This is a first step, so as they learn more, they'll explore that.

- ***NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge***
  Bhavish Aggarwal, Ranjita Bhagwan, and Tathagata Das, Microsoft Research India; Siddharth Eswaran, IIT Delhi; Venkata N. Padmanabhan, Microsoft Research India; Geoffrey M. Voelker, University of California, San Diego

Ranjita Bhagwan said that home networks consist of many components (router, firewall, servers, etc.). The setup is highly diverse from one home network to another and there is no network administrator. Misconfiguration of these components leads to application failures, of which there are a huge set of example problems: some are router misconfigurations, some are on end-hosts, and some are remote problems where local changes can work around the problem.

NetPrints, which stands for network problem fingerprinting, automates problem diagnosis using shared knowledge. Each network periodically sends configuration information of all devices to the NetPrints service, which builds a knowledge base of configurations and state (working/not working) tied directly to an application. Someone with a problem will send their configuration and report which application is not working correctly, and NetPrints will suggest a fix. In response to a user with a VPN client who has experienced

a failed connection, for example, NetPrints will provide instructions to set pptp_pass to 1 in the router's configuration, since NetPrints has seen that problem before. Different configurations can have different costs associated with them (setting pptp_pass to 1 is less costly than changing routers), and the recommendations take that into account.

Someone asked if they'd considered merging trees in cases where NetPrints couldn't find a solution? They are looking at that, but the challenge is finding an application that is similar enough. Are there any user-specified constraints (weights)? Not at the moment, but the server can respond with several choices. In the examples given, the trees were not too big; would they still be small if NetPrints went beyond connectivity management (VPN)? They haven't faced that in the examples they've tried. There are cases that are notoriously hard to debug (e.g., plugging into uplinks, running two home networks. Can NetPrints handle cases where the user fails to report something (because it wasn't captured or was non-deterministic)? No, the system is limited to the configuration that they can and do capture.

GREEN NETWORKED SYSTEMS

Summarized by Michael Golightly (mgolight@princeton.edu)

- ***Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage***
  Yuvraj Agarwal, University of California, San Diego; Steve Hodges, Ranveer Chandra, James Scott, and Paramvir Bahl, Microsoft Research; Rajesh Gupta, University of California, San Diego

Energy efficiency is a key driver in PCs today, and although sleep has solved the problem of maintaining application state, it does not maintain presence or allow occasional remote access. The goal is to reach a hybrid state, where the machine is in a sleep state but is perceived as awake and responsive across the entire protocol stack, with no changes to infrastructure or user behavior.

Yuvraj presented Somniloquy, which enables PCs to "talk in their sleep" by augmenting network devices with a low-power processor, memory, flash storage, and network stack that operates when the host is asleep. Stateless applications are supported by filters that can be specified at any layer of the network stack to wake the host under predefined conditions. Stateful applications are supported by application stubs that are specifically programmed to run on the limited resources of the low-power processor. Currently, these stubs have been generated manually for BitTorrent, Web downloads, and instant messaging.

The prototypes of Somniloquy were built using the gumstix platform with a USB connection to the host. The evaluation of network reachability found that a host was unresponsive to pings for the 4–5 second transition between sleep and awake states. Stateless applications were found to have 3–10 seconds of additional setup latency, a small proportion of the overall session length. In no case was the prototype solution consuming more power than the original unmodified host. Assuming a 45-hour work week, one could save $56 annually or reduce 10% of one's carbon footprint using Somniloquy on a desktop PC. Somniloquy also increased battery life from 6 to 60 hours for laptops. Using workload traces from 24 desktop PCs, energy savings ranged from 38% to 85%. Lastly, using the Web download application stub, Somniloquy was able to use 92% less energy than a host-only solution.

Someone asked how this differed from Windows Sideshow. Yuvraj answered that Windows Sideshow does not keep the network active and that Somniloquy could augment this technology. Why are only clients augmented rather than other points in the network? Somniloquy works well for individual users; it might be better from a cost perspective in the enterprise setting to focus elsewhere in the network, but there would be huge overheads in implementation and security. How difficult would it be to integrate Somniloquy into a motherboard? Somniloquy could be implemented anywhere; the prototype is an initial solution.

- ***Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems***
  Sergiu Nedevschi, International Computer Science Institute and Intel Research; Jaideep Chandrashekar, Intel Research; Junda Liu, University of California, Berkeley, and International Computer Science Institute; Bruce Nordman, Lawrence Berkeley National Laboratories; Sylvia Ratnasamy and Nina Taft, Intel Research

The authors' work is a trace-driven evaluation of the benefits and design tradeoffs for energy savings that can be obtained with simpler, more adoptive techniques. Sergiu presented results from a four-week trace of 250 Intel hosts, 90% laptops and 10% desktops, in both an office and a home setting. Desktops were found to be idle greater than 50% of the time, wasting upwards of 60% of their energy. Given that there are 170 million desktop PCs in the US, this translates into 60 terawatt hours per year wasted, or $6 billion.

Incoming host traffic was found to be high but bursty, making it infeasible to wake for every packet. Packets then need to be handled transparently, by waking the host, or non-transparently, by ignoring them. Key multicast and broadcast offenders of sleep deprivation whose packets could be ignored were found to be NBDGM, IPX, HSRP, and PIM. ARP, NBNS, IGMP, and SSDP were also found to be key offenders, but could be handled simply. For unicast, key offenders were TCP and UDP, but by looking at port numbers, it was found that some can be handled simply, while others such as DCE/RPC and SMB/CIFS cannot.

A general proxy architecture should consist of rules, triggers, and actions. A trigger is a regular expression on incoming packets, and actions define whether to wake the host or to drop, respond, or redirect the packet. The authors implemented a proxy in Click as a stand-alone machine on the same LAN as hosts. It masqueraded as sleeping machines, waking them when necessary. It used a simple, non-transparent set of rules and learned hosts' state by sniffing

traffic. This approach required no modification to end systems and could be sold as a separate network product; it is agnostic to whether the proxy runs on the NIC, server, router, or elsewhere.

Someone asked about the tradeoff between using idleness for prefetching purposes and saving power. Sergiu replied that a host should wake up periodically and do work in batches at a higher utilization rate. Why was there such high background traffic touching idle hosts? This traffic was mainly caused by background services that would probably not be seen if hosts could enter sleep states. Could the problem be completely solved by proxy or would application and protocol support be a better approach? Proxy-friendly applications and protocols would help, but it is uncertain whether they could solve the problem alone.

WIRELESS #2: PROGRAMMING AND TRANSPORT

Summarized by Devesh Agrawal (dagrawal@cs.umass.edu)

- ***Wishbone: Profile-based Partitioning for Sensornet Applications***
  Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden, MIT CSAIL

There is an important class of sensing applications that use high-data-volume sensors. These also require significant computation and processing. Examples include animal localization using acoustic sensors and pothole detection using vibration sensors. Ryan presented Wishbone, a system providing two key benefits to the design of such applications, First, it optimally partitions the sensing application across the embedded and back-end servers, subject to the CPU, bandwidth, and energy constraints. Second, it enables the application to be automatically deployed across a range of hardware, including TinyOS-based motes, JavaME-based smartphones, and full-blown embedded Linux microservers.

Wishbone is built on top of the WaveScope system. The application is specified in the WaveScript language. WaveScope converts this high-level representation into a dataflow graph. Nodes of the graph represent stream processing operators, and the edges represent the dataflow across the operators. Sensing data is fed into this graph and the resulting processed output is either stored or visualized at a base station. The Wishbone system first optimally partitions this graph across the sensor network and the base station. It then compiles and loads the partitions onto the embedded nodes and the server.

Offline profile-based partitioning is at the heart of the Wishbone system. This partitioning assumes that the input data rates are fairly stable and a representative data trace is easily obtained. This representative trace is used to profile the dataflow graph to measure the CPU time taken by each node and the flow rate across each edge. Along with the available network bandwidth, this information is fed into an integer linear program that finds an optimal (offline) partition subject to the CPU and network constraints.

Ryan presented two case studies to evaluate Wishbone: a speaker-identification application and a seizure-detection application. The speaker identification had a linear pipeline of eight steps, while the EEG application had more than 1400 nodes. In both cases, Wishbone correctly identified the optimal partitioning point if feasible or the partition having the highest throughput otherwise. Particularly noteworthy was the example that in the speaker identification application many partitions resulted in zero data throughput, while the best partition was more than 20 times better than the worst partition, thereby highlighting the crucial importance of correct partitioning.

During Q&A, Ryan clarified that the static offline partitioning scheme does not work for dynamic operators that adapt to the offered load. He also conceded that while the current implementation only works with homogeneous embedded devices, they are working toward supporting a fully heterogeneous network having a variety of embedded platforms.

- ***Softspeak: Making VoIP Play Well in Existing 802.11 Deployments***
  Patrick Verkaik, Yuvraj Agarwal, Rajesh Gupta, and Alex C. Snoeren, University of California, San Diego

VoIP over WiFi is becoming increasingly popular with the advent of 802.11-enabled mobile handsets. Hence it is important to understand the impact of VoIP users on 802.11 deployments. Patrick presented Softspeak, a system that dramatically improves VoIP call quality and its impact on data transfers. There are two main reasons why VoIP makes inefficient use of WiFi. First, VoIP packets are just tens of bytes long and hence incur significant framing and header overheads. Second, VoIP has a high packet rate, which causes excessive contention at the AP. This significantly hurts data transfers and impacts call quality.

Softspeak employs TDMA in the uplink direction (from clients to the AP). In contrast to the usual DCF of 802.11, the TDMA schedule does not suffer backoff and collision overheads and hence improves the VoIP channel utilization. However, data packets do not know about this TDMA schedule, which raises two key implementation issues. First, VoIP packets contend with data packets and may miss their slots. Softspeak addresses this by changing the 802.11 carrier sense time for VoIP packets such that VoIP packets can grab the channel ahead of the data packets. Second, a late VoIP station may miss its assigned slot and contend with another station in the following slot. This is also addressed by letting the late VoIP station proceed first. Softspeak uses downlink aggregation to amortize framing and header overheads (from AP to the clients). It batches multiple VoIP packets, possibly addressed to different client nodes, into a single IP packet and unicasts it to one of the intended recipients. Other intended recipients overhear this packet and extract the relevant VoIP packets for themselves.

Patrick demonstrated that Softspeak significantly improves call quality as well as throughput of data flows compared to the status quo in both 802.11b and 802.11g networks. For

example, he showed that without Softspeak, a voice call in the presence of ten competing VoIP stations was extremely choppy and barely audible, whereas with Softspeak the same voice call was as good as when there were no contending VoIP stations.

In response to a question, he said that Softspeak can also handle multiple collision domains and multiple APs, as is the case in most enterprise WLANs. He conceded that reserving a special channel for VoIP traffic might obviate the need for Softspeak, but reserving a channel is seldom possible, due to the very narrow WiFi spectrum available. Softspeak is available at http://sysnet.ucsd.edu/wireless/softspeak/.

- ***Block-switched Networks: A New Paradigm for Wireless Transport***
  Ming Li, Devesh Agrawal, Deepak Ganesan, and Arun Venkataramani, University of Massachusetts Amherst

Ming presented Hop, a high throughput wireless transport protocol that achieves orders of magnitude better performance than TCP. Decades of wireless transport research have provided two main insights into TCP's poor performance: First, TCP's end-to-end congestion control is error-prone and fails to effectively utilize the available wireless capacity. Second, there is a significant per-packet overhead due to the lossy and broadcast-nature of wireless links. Hop recognizes that most of TCP's problems stem from its legacy as a transport protocol for the wired Internet, where losses were rare, links quite stable, and storage expensive. Recognizing this, Hop advocates a clean-slate re-design: End-to-end becomes hop-by-hop, and packets change to blocks.

The main building block of Hop is reliable per-hop block transfer, in which a node reliably sends a large block (for example, up to 1MB) of data to its next hop. Blocks significantly reduce control overhead, as the sender requires only one handshake for the entire block of data, as opposed to doing ARQ for each packet. Further, Hop leverages existing 802.11e features, such as burst mode transfer and disabling link layer ARQ, to exploit the available wireless bandwidth. Hop's end-to-end loss recovery mechanism uses in-network caching to only transfer data to nodes that do not have the data cached. This strategy prevents wasteful retransmissions. It uses back pressure–based congestion control, wherein each node limits the number of outstanding blocks per flow. Two key benefits of this simple scheme are that the source stops sending if the downstream path is congested, and network utilization is improved by allocating bandwidth to good links over bad ones. Hop also addresses hidden terminals by serializing the data transfers to a common receiver and, finally, employs several optimizations to improve the delay performance of small blocks.

Ming demonstrated that Hop achieves significant gains over TCP over one hop, over multiple hops, and in a WLAN setting. But the most impressive result was that Hop achieved more than two orders-of-magnitude improvement under a highly loaded mesh network scenario. He showed that in such high-load conditions, TCP allocates almost the entire bandwidth to a couple of flows while starving the rest. By contrast, Hop distributes the network bandwidth almost equitably, thereby improving fairness.

During Q&A, Ming discussed a simple proxy-based solution to bridge a Hop connection on the wireless side with TCP on the wired side, while conceding the possibility of more sophisticated proxy-based solutions. He also clarified that the back-pressure mechanism is on a per-flow basis and there is no explicit rate allocation across different flows. Hop can be downloaded from http://hop.cs.umass.edu/.

ROUTING

Summarized by Eric Keller (ekeller@princeton.edu)

- ***NetReview: Detecting When Interdomain Routing Goes Wrong***
  Andreas Haeberlen, MPI-SWS and Rice University; Ioannis Avramopoulos, Deutsche Telekom Laboratories; Jennifer Rexford, Princeton University; Peter Druschel, MPI-SWS

Andreas Haeberlen pointed out that the Internet's interdomain routing is vulnerable to errors: misconfigurations, buggy software, failing equipment. Rather than attempt to prevent specific problems, with NetReview the approach is to detect problems and identify the offending party. This leads to greater coverage and easier deployment than previous approaches.

To do this, one could enable full logging at all routers and upload each log to a central entity that inspects them for problems. However, this has privacy concerns (logs contain sensitive info), has reliability issues (logs inaccurate, bugs, hackers), has impacts on automation (lots of data to inspect), and is difficult to deploy (can't assume global deployment). Instead, in NetReview, all border routers maintain logs of all BGP messages (both sent and received). These logs are tamper-evident: one can reliably detect and obtain proof if faulty routers omit, forge, or modify entries. This is done through the use of hash chains.

A neighbor can audit the AS by requesting the logs from each border router (note that the auditor can be a server). The auditor can then talk to the neighbors of the auditee to see if any entries are missing or modified. The auditor locally replays the logs to get a series of routing states and evaluates the rules over the routing state to see if any have been violated. From this the auditor can extract evidence from logs.

In the evaluation, they found that there were few rules needed, low processing requirements, a manageable storage requirement, and an insignificant bandwidth requirement.

Someone asked how, without a public key infrastructure, one can tell a log hasn't been tampered with. No PKI is needed, because you know who is on the other end of the link and therefore can certify the identity of that AS. Does their sys-

tem handle collusion among ASes? Colluding ASes cannot hide bad behavior or create evidence against a good AS; they can only hide the messages between those two ASes.

- ***Making Routers Last Longer with ViAggre***
  Hitesh Ballani, Paul Francis, and Tuan Cao, Cornell University; Jia Wang, AT&T Labs—Research

Hitesh Ballani said that routing-table sizes are increasing rapidly. As the IPv4 address space runs out, this problem will become even worse as hierarchical aggregation deteriorates, and switching to IPv6 would cause very large tables. These routing tables need to be in *fast memory* in the forwarding information base (FIB). Throwing more RAM at the problem has technical and cost issues.

Rather than each router having an entire table, ViAggre splits prefix space into virtual prefixes (not necessarily of the same size) and assigns each split to a particular router. For the control plane, an external router peers with a route reflector which then sends only a subset of routes to each router. For the data plane, when a router receives a packet for which it does not have a route, it has an entry for the virtual prefix that says the *next hop* is the aggregate router that was assigned that prefix space (the packet traverses an MPLS tunnel to get to that router). As an optimization, since 95% of all traffic goes to only 5% of the prefixes, they maintain this 5% on all routers. The choice of aggregation points leaves room for tradeoffs: the more aggregation points you have, the less stretch there is, but the bigger the FIB size.

An audience member asked if IP–in-IP tunneling was done on slow path. They use MPLS, which is on fast path. The underlying premise is that routing tables are growing faster than traffic: why is that? That is not necessarily true. Bigger ISPs have large pipes and may have to upgrade only to address memory concerns. Why not only maintain popular prefixes and ship the rest to a default upstream router? One approach for this is route cache (hierarchy of memory). This hasn't worked in the past: unpredictable performance. Plus, for medium ISPs, you may have multiple upstreams (or peers), so you don't know where it would go. Can you apply this to data centers (switch tables)? SEATTLE from SIGCOMM did that last year—ViAggre works at layer 3. Why is it expensive to do route suppression from the RIB to the FIB? They achieved this through the use of ACLs, which on Juniper and Cisco are heavyweight mechanisms today.

- ***Symbiotic Relationships in Internet Routing Overlays***
  Cristian Lumezanu, Randy Baden, Dave Levin, Neil Spring, and Bobby Bhattacharjee, University of Maryland

Two nodes are in symbiosis when they can benefit from one another (i.e., there is mutual advantage). Examples include file sharing (BitTorrent), backup systems (Samsara), AS relationships—no tragedy of the commons, no free riding.

Cristian Lumezanu presented PeerWise, a latency-reducing routing overlay based on this concept of mutual advantage. Suppose node A in Maryland wants to talk to node C in Seattle. The direct path takes longer than going through node B in Boston. In PeerWise, B wouldn't let this happen unless B wants to communicate with D in San Diego, which happens to be faster if packets go through A first.

In their measurement study, they collected two sets of latency data and found that 21% and 51% of all node pairs, respectively, would benefit from detours, with half being eliminated due to PeerWise's restriction of mutual advantage. To test if user-level applications can benefit, they used wget to download 500 popular Web sites using direct and PeerWise detour and found 58% were faster (if delay due to PlanetLab was removed, 80% would be faster).

Since using network coordinates seems counter-intuitive, an attendee wondered, why not use more topological information? Network coordinates give pretty good results, so they haven't looked elsewhere. Had they considered going beyond bilateral agreements into more complicated situations (e.g., A helps B, B helps C, so C will help A)? Not yet. Had they looked at including the load of the nodes in the weighting (to account for PlanetLab overhead)? No, they had not. Since this work used TCP relays, which has benefits on its own, had they separated the benefits of splitting the TCP connection from the benefits of going through a detour? Not sure how they would.