COMPUTING edge

FACING THE WIRELESS CHALLENGE

Also in this issue:

> Architecture from a Developer's Perspective

> The Rise of Multimedia for Online Communication Startups

FEBRUARY 2016

www.computer.org

IEEE computer society

CELEBRATING 70 YEARS

Move Your Career Forward

IEEE Computer Society Membership

Explore These Wireless Technology Resources

Create Connections

LAN/MAN (IEEE 802) Standards Committee

Chartered by the IEEE Computer Society Standards Activities Board, the LAN/MAN committee is tasked to develop, maintain, and advocate for networking standards and recommended practices for local, metropolitan, and other area networks, using an open and accredited process. The most widely used standards are for:

• Ethernet • Bridging and virtual bridged LANs • Wireless LAN • Wireless PAN • Wireless MAN • Wireless coexistence • Media-independent handover services • Wireless RAN

Build Your Knowledge

WOWMOM

17th International Symposium on a World of Wireless, Mobile, and Multimedia Networks

21-24 June 2016 - Coimbra, Portugal

IEEE WoWMoM 2016 focuses on wireless networking technologies' evolution and key role in future Internet scenarios, offering an increasing wealth of opportunities for distributing multimedia over wireless networks and sharing user-generated content with mobile users.

MOWNET
WHERE MOBILITY MEETS INTELLIGENCE

International Conference on Selected Topics in Mobile and Wireless Networks

11-13 April 2016, Cairo, Egypt

MoWNet 2016 seeks to address the rising research issues that are requiring rethinking of current mobile technology solutions to meet the emerging needs of a broader and ever-growing base of smart-device users whose everyday lives are deeply influenced by wireless networks' ubiquitous availability.

FOR DIRECT LINKS TO THESE RESOURCES, VISIT

www.computer.org/edge-resources

IEEE computer society

CELEBRATING 70 YEARS

COMPUTING edge

IEEE COMPUTER SOCIETY http://computer.org • +1 714 821 8380

Printed with inks containing soy and/or vegetable oils

IEEE computer society
CELEBRATING 70 YEARS

STAFF

BPA WORLDWIDE

Circulation: ComputingEdge is published monthly by the IEEE Computer Society. IEEE Headquarters, Three Park Avenue, 17th Floor, New York, NY 10016-5997; IEEE Computer Society Publications Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; voice +1 714 821 8380; fax +1 714 821 4010; IEEE Computer Society Headquarters, 2001 L Street NW, Suite 700, Washington, DC 20036.

Postmaster: Send undelivered copies and address changes to ComputingEdge-IEEE Membership Processing Dept., 445 Hoes Lane, Piscataway, NJ 08855. Periodicals Postage Paid at New York, New York, and at additional mailing offices. Printed in USA.

IEEE Computer Society Magazine Editors in Chief

Computer
Sumi Helal, *University of Florida*

IEEE Software
Diomidis Spinellis, *Athens University of Economics and Business*

IEEE Internet Computing
M. Brian Blake, *University of Miami*

IT Professional
San Murugesan, *BRITE Professional Services*

IEEE Security & Privacy
Ahmad-Reza Sadeghi, *Technical University of Darmstadt*

IEEE Micro
Lieven Eeckhout, *Ghent University*

IEEE Computer Graphics and Applications
L. Miguel Encarnação, *ACT, Inc.*

IEEE Pervasive Computing
Maria Ebling, *IBM T.J. Watson Research Center*

Computing in Science & Engineering
George K. Thiruvathukal, *Loyola University Chicago*

IEEE Intelligent Systems
Daniel Zeng, *University of Arizona*

IEEE MultiMedia
Yong Rui, *Microsoft Research*

IEEE Annals of the History of Computing
Nathan Ensmenger, *Indiana University Bloomington*

IEEE Cloud Computing
Mazin Yousif, *T-Systems International*

FEBRUARY 2016 • VOLUME 2, NUMBER 2

COMPUTING edge

55

Technical Debt in Computational Science

Departments

Designing Effective Refreshable Braille Displays

Lynette A. Jones, MIT

This installment highlighting the work published in IEEE Computer Society journals comes from IEEE Transactions on Haptics.

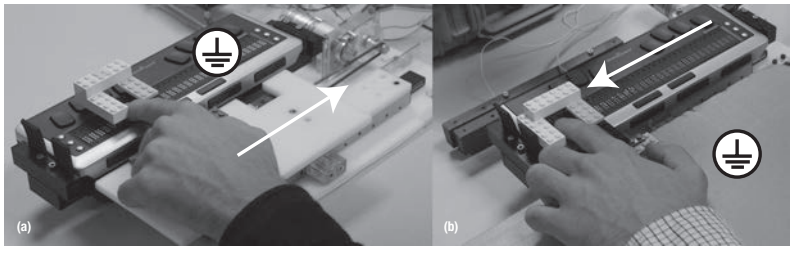


Figure 1. Braille display (a) mounted to a moving platform so that the display moves beneath a stationary finger (static condition) or (b) mounted to ground the finger mounted to the moving platform (sliding condition).

It's been shown that non–visually impaired readers comprehend text better when they actively read it rather than passively listen to it. Active modes of processing text such as reading braille—instead of more passive modes like speech—would likely confer a similar advantage to visually impaired readers, particularly for material that's highly technical or that requires spatial processing, such as maps and charts.

In active braille reading, readers move their hands over the text, with continuous slippage between the fingertip and the surface providing the cues necessary to render the braille image. It's thought that larger-scale features, such as the characters' spatial orientation, arise from proprioceptive cues as the hand moves across the display. Although skilled braille readers prefer active movement over the surface of the braille text, a low-cost display that incorporates sliding contact between the fingertip and the reading surface might suffice in many contexts.

Electronic braille displays present both text and graphics, but current refreshable braille displays are expensive, ranging from approximately $2,000 for an 18-character display to $50,000 for a half-page of braille. Low-cost refreshable braille displays that use newer actuator technologies and deliver both text and graphics content would significantly expand access to braille for individuals with visual impairments. In their *IEEE Transactions on Haptics* article "Refreshing Refreshable Braille Displays" (vol. 8, no. 3, 2015, pp. 287–297), Alexander Russomanno and his colleagues examined the features necessary for effective braille reading using a refreshable display.

They quantified the contributions of lateral motion and proprioception to experienced braille readers' recognition of braille letters. The study participants identified letters under static and sliding conditions as their hands moved to read braille at varying presentation speeds. The 32-character braille display illustrated in Figure 1 was either mounted to a moving platform so that the display moved beneath a stationary finger (static condition) or mounted to ground the finger mounted to the moving platform (sliding condition). The authors found that the second condition's relative motion—movement between the fingertip and the braille surface—resulted in more accurate letter recognition, particularly with faster presentation speeds. The passively moving hand's proprioceptive inputs didn't affect error rates.

Collectively, Russomanno and his colleagues' findings suggest that refreshable braille displays that allow sliding contact are more successful than displays that update in place and thus don't permit sliding contact between the finger and the braille letter.

LYNETTE A. JONES is a Senior Research Scientist in MIT's Department of Mechanical Engineering. Contact her at ljones@mit.edu.

 Published by the IEEE Computer Society

Magazine Roundup

The IEEE Computer Society's lineup of 13 peer-reviewed technical magazines covers cutting-edge topics ranging from software design and computer graphics to Internet computing and security, from scientific applications and machine intelligence to cloud migration and microchip manufacturing. Here are highlights from recent issues.

Computer

Computer's January 2016 special **outlook** issue peers into the future of computing, with articles exploring topics such as collective computing, an innovative software-engineering paradigm, high-confidence medical-device software, and new OS approaches.

IEEE Software

IEEE Software's January/February 2016 special issue on **software engineering's future** offers various perspectives from professionals around the world. The content ranges from detailed technical articles about the research areas behind today's trends to shorter essays and opinion pieces by authors sharpening their visions of the future.

IEEE Internet Computing

The advent of the Internet of Things has created the need for more stream-processing flexibility. The authors of "Elastic Stream Processing for Distributed Environments," from *IEEE Internet Computing*'s November/December 2015 issue, propose **elastic stream processing** to meet this challenge. Their proposal builds on cloud computing and allows more scalability and flexibility than traditional approaches.

Computing in Science & Engineering

CiSE's January/February 2016 special issue features articles on the US Department of Defense (DoD) High Performance Computing Modernization Program's **Computational Research and Engineering Acquisition Tools and Environments** (CREATE) program. The department launched CREATE in 2006 to design and deploy high-performance computing applications to help the DoD and its contractors develop innovative military equipment.

IEEE Security & Privacy

IEEE S&P's editorial board members collectively have deep,

2376-113X/16/$33.00 © 2016 IEEE Published by the IEEE Computer Society

broad expertise and experience in all aspects of security and privacy, which helps them suggest, develop, and review articles for the magazine. In *S&P*'s November/December 2015 special issue on **lessons learned from the editorial board**, members share what they've learned in their careers.

IEEE Cloud Computing

Sensor networks and the Internet of Things will increase our ability to connect the cyber and physical worlds, and enable important new applications. The successful deployment of novel sensor-based applications requires the development of **cloud-based cyberinfrastructures** able to manage the sensors and data they collect. "Building Sensor-Based Big Data Cyberinfrastructures," from *IEEE Cloud Computing*'s September/October 2015 issue, elaborates on the key challenges this entails.

IEEE Computer Graphics and Applications

With the rise of massive open online courses (MOOCs), millions of learners can enroll in more than 1,000 courses via MOOC platforms. These systems have collected a huge amount of detailed data, including large quantities of information on learning behavior that researchers can analyze. "Visual Analytics for MOOC Data," which appears in *CG&A*'s November/December 2015 issue, discusses how **visual analytics** can help with this process.

IEEE Intelligent Systems

Nonoccurring behaviors (NOBs)—those that should happen but don't for some reason—widely occur in online, business, government, health, scientific, and social applications. Little research has examined NOBs because of the challenges posed by analyzing behaviors that don't actually occur. "Nonoccurring Behavior Analytics: A New Area," from *IEEE Intelligent Systems*' November/December 2015 issue, explores this topic.

IEEE MultiMedia

Multimedia is no longer confined to entertainment or personal media but instead is now an important general means of communication. In fact, it has become an integral part of the tools and systems that provide solutions to today's societal challenges, according to "**Multimedia Takes on Societal Challenges**," which appears in *IEEE MultiMedia*'s October–December 2015 issue.

IEEE Annals of the History of Computing

The history of computing in Latin America is the theme of *IEEE Annals*' October–December 2015 special issue.

IEEE Pervasive Computing

Smartphones generally handle and store sensitive data that users want protected. "**Smartphone Security**," from *IEEE Pervasive Computing*'s October–December 2015 issue, surveys various security issues and presents tools that can help users better protect sensitive data.

IT Professional

IT project failures are pervasive, and even though much has been written on the subject, IT project managers still must heed the lessons learned from such events. "**IT Project Failures: What Management Can Learn**," from *IT Pro*'s November/December 2015 issue, suggests ways that management can ensure IT project success.

IEEE Micro

Autonomous vehicles are an increasingly popular research topic. Despite the amount of attention the topic has received, details about experimental autonomous vehicles aren't being made accessible to researchers in general but instead are developed as proprietary assets. "An Open Approach to Autonomous Vehicles," which appears in *IEEE Micro*'s November/December 2015 issue, introduces an open platform using commodity vehicles and sensors, as well as a common interface. The authors say this platform could facilitate autonomous-vehicle development.

Computing Now

The Computing Now website (http://computingnow.computer.org) features **up-to-the-minute computing news** and blogs, along with articles ranging from peer-reviewed research to opinion pieces by industry leaders.

EDITOR'S NOTE

Facing the Wireless Challenge

Wireless technology is advancing on a regular basis. For example, mobile networks are faster and more reliable, enabling innovative new applications. And devices, including smartphones and sensors, are more capable.

However, new wireless capabilities have created new hurdles, with security being one of the biggest. This *ComputingEdge* issue looks at some of wireless technology's most important new developments, applications, and challenges.

IEEE Pervasive Computing's "Smartphone Security" presents a survey of smartphone issues, as well as tools that can help users better protect sensitive data.

Smartphones' and tablet computers' ownership model is more like that of game consoles than PCs. According to *IEEE Security & Privacy*'s "Possessing Mobile Devices," this leaves users vulnerable to significant security and privacy threats.

"Toward Mobile-Friendly Web Browsing," from *IEEE Internet Computing*, reveals why achieving mobile-friendly Web browsing requires joint efforts among organizations in the wireless ecosystem.

IT Professional's "Establishing and Maintaining Trust in a Mobile Device" looks at MobileRoT, a US Department of Homeland Security project investigating the use of trustworthy wireless-infrastructure components.

Smartphones' ubiquity means that criminal suspects probably use such devices and that their handsets likely contain incriminating data. Thus, law enforcement officers want a forensically sound methodology—like the one described in *IEEE Cloud Computing*'s "A Cloud-Focused Mobile Forensics Methodology"—to access such data remotely.

Computer's "Tracking Cows Wirelessly" introduces a prototype wireless network that a North Carolina State University student team designed and built to monitor the milking and weighing of cows.

In "Concurrency in Mobile Browser Engines," from *IEEE Pervasive Computing*, the authors discuss advances in browsers—required to run today's Web apps—that exploit multicore processing by using concurrency at different levels.

ComputingEdge articles on other subjects include the following:

- Architectural considerations' importance for software developers is the focus of "Architecture from a Developer's Perspective" from *IEEE Software*.
- *IEEE MultiMedia*'s "The Rise of Multimedia for Online Communication Startups" examines how online multimedia communications helped some recent start-ups succeed.
- "Technical Debt in Computational Science," from *Computing in Science & Engineering*, explores technical debt, a programming concept reflecting the extra development work that arises when code that's easy to implement in the short run is used in a software project instead of code that represents the best overall solution.

2376-113X/16/$33.00 © 2016 IEEE Published by the IEEE Computer Society

Smartphones

Editor: Nayeem Islam ■ Qualcomm ■ nayeem.islam@gmail.com

Smartphone Security

Lori Flynn and Will Klieber, CERT

Smartphones handle and store sensitive data that should be protected. The vast amount of private information stored on smartphones was even cited by the US Supreme Court, in *Riley v. California* (2014), as a factor in ruling that searches of these devices require a warrant. *Taint-flow analyzers* use static or dynamic analysis techniques to trace the flow of sensitive data to undesired locations.

If a user's location data, such as GPS coordinates or Wi-Fi access point information, is disclosed, it can compromise the user's privacy and, in extreme cases, put the user's physical safety at risk. Medical information is also increasingly an issue, given the increased popularity of wearable computing devices (such as health sensors) that communicate with users' smartphones. In addition, data from the phone's sensors or stored on the device (in emails, texts, or photos) could be used for theft (bank and credit card numbers), blackmail, stalking, unfair competition, public humiliation, and other abuses. Malware could surveil the smartphone user with microphone, video, and other sensors. Furthermore, privacy threats to users can come from many sources, including advertisers, hackers, and governments. Finally, employees often use their smartphones for both personal and business purposes; accordingly, technological measures should ensure that the employee's personal data is not leaked to the employer and that proprietary business data is kept secure.

If a user's location data, such as GPS coordinates or Wi-Fi access point information, is disclosed, it can compromise the user's privacy.

Here, we discuss in detail various smartphone security issues and present tools and strategies that can help us better protect sensitive data.

SECURITY ISSUES

Smartphones present a unique environment that comes with its own set of security concerns (see the "Desktops vs. Smartphones Security" sidebar for more information).

Operating System Vulnerabilities

Each smartphone operating system (OS) has security vulnerabilities particular to its system. For example, Apple and Microsoft have a mechanism to push out security updates to smartphones using their OSs, but Google can only push updates to pure-Android devices, such as Nexus phones.

Google provides fixes to original equipment manufacturers (OEMs) and service providers (SPs) that provide specialized versions of the Android OS, but OEMs and SPs often don't implement and distribute fixes, or take a long time to do so. Recent studies show Android OS updates permeate extremely slowly over Android phones. Only 0.7 percent of Android phones use the latest OS version, while widely fragmented large segments of Android users have old OS versions.[1] Missing security fixes hits lower-cost Android phones the hardest: many receive no updates and others only rarely. This issue recently has been highlighted by the public disclosure of Android Stagefright vulnerabilities, a severe problem that might allow a remote attacker to execute code on

DESKTOPS VS. SMARTPHONE SECURITY

Under popular desktop operating systems (including Windows, Mac OS X, and Linux), programs usually execute with all permissions of the user. Smartphone apps are more tightly constrained. Apps must request and be granted permission to do things, such as reading from the microphone or accessing the phone's general file system. Apps are sandboxed from each other more tightly than on desktop OSs. On Android, each app has private storage that other apps can't read or write. Unlike desktop programs, which can be run with root privileges via the `su` command or the Windows User Account Control, third-party apps on Android and iOS smartphones can't be run as root unless the user has unlocked the phone's bootloader. App stores perform some checks on apps to try to prevent malicious apps from being released on the app store.

 Published by the IEEE Computer Society

Android devices.[2] An estimated 950 million Android phones are still vulnerable,[3] over three months after a security researcher disclosed the vulnerability to Google along with code patches, even though Google applied the patches to internal code branches within 48 hours.

Additional OS-specific issues include the following.

iOS security issues. Widespread vulnerabilities have recently been shown in iOS app-to-app and app-to-operating-system communications,[4] involving scheme hijacking and possibly WebSocket abuses. These vulnerabilities are due to a lack of authentication for multiple reasons: iOS doesn't provide some types of authentication APIs, enforce some authentication, or advise developers to check for particular authentications. The Xavus tool found many of these exploitable vulnerabilities in popular iOS apps.[4]

Android security issues. Android has a complex inter-app communication system that can be used in attacks. An *intent* is a message sent to a component of an app. An intent might explicitly designate its recipient by name, or it might rely on the OS to find a suitable recipient by matching properties of the intent to potential recipients' intent filters. The latter type of intent, an *implicit intent*, poses the greatest security concerns.

Intents can be used to make it difficult to statically analyze the flow of sensitive data between apps in a precise manner (that is, with few false negatives and few false positives). *Intent hijacking* occurs when a malicious app receives an intent that was intended for (but not explicitly designated for) another app. If two apps have *activity* components that can handle an implicit intent, then the user is presented with a choice of which app to use. A malicious app can try to trick the user into choosing it by using a confusing name. Also, an inattentive user might not give much thought to the choice. Furthermore, the touchscreen might register a tap for the malicious app that the user did not intend.

Beyond inter-app communication, intents are also used for intra-app communication between different components of a single app. It is easy for a developer to mistakenly make app interfaces public when they should be private, allowing malicious apps to eavesdrop or hijack data. Epicc is a static-analysis tool that analyzes inter-component communication vulnerabilities.[5]

Memory Corruption Attacks

Memory corruption attacks (such as buffer overflows) commonly exploited on desktop systems are also applicable to mobile devices. In Android, many apps are written purely in Java, a memory-safe language, which limits the attack surface to

- apps that employ native code;
- vulnerabilities in the Java virtual machine and the Java runtime environment; and
- vulnerabilities in the underlying OS.

It is easy for a developer to mistakenly make app interfaces public when they should be private, allowing malicious apps to eavesdrop or hijack data.

Mitigations include address-space layout randomization (ASLR) and data execution protection (DEP).

DEP allows regions of memory (such as the stack) to be marked with a "non-executable" (NX) bit, which the CPU checks before executing code from the memory region. Partial ASLR support has been present on Android since 4.0 and on iOS since 4.3; however, even small libraries unprotected by ASLR have been shown to offer sufficient gadgets for return-oriented programming (ROP) exploits.[6] Modern Android and iOS versions use DEP on supporting hardware. ROP is a technique to exploit memory corruption even in the presence of DEP. Rather than writing new executable code onto the stack, the exploit takes advantage of existing *gadgets* (small sequences of machine code that typically end with a `RET` instruction) that can be effectively chained together. A ROP exploit is used by the Evasi0n jailbreaking tool for iOS 6.0.

PROTECTIVE MEASURES (AND SOME FAILURES)

Just as each OS has its own vulnerabilities, each also has security measures specific to its system. Also, some protective security measures need to be applied (and researched), regardless of the OS.

OS-Specific Security

Different smartphone OSs allow varying levels of user control (and protection) over sensitive dataflow. The smartphone OS with the largest worldwide market share, Android, currently offers only limited control by users over their data, requiring all permissions requested to be granted before an app is installed. The public release of the Android M software developer's kit (SDK) is scheduled for the third quarter of 2015 (https://developer.android.com/preview/overview.html), and it changes the Android permissions model, so permissions won't need to be requested during installation, can be asked for during use as needed, and can be revoked by users without removing the app.

The M SDK also introduces *App Links*, which enable a website to designate an official app, which, if installed, will automatically be chosen as the default handler for links to that website. This helps mitigate intent hijacking if a malicious third-party app also tries to register itself to handle

SMARTPHONES

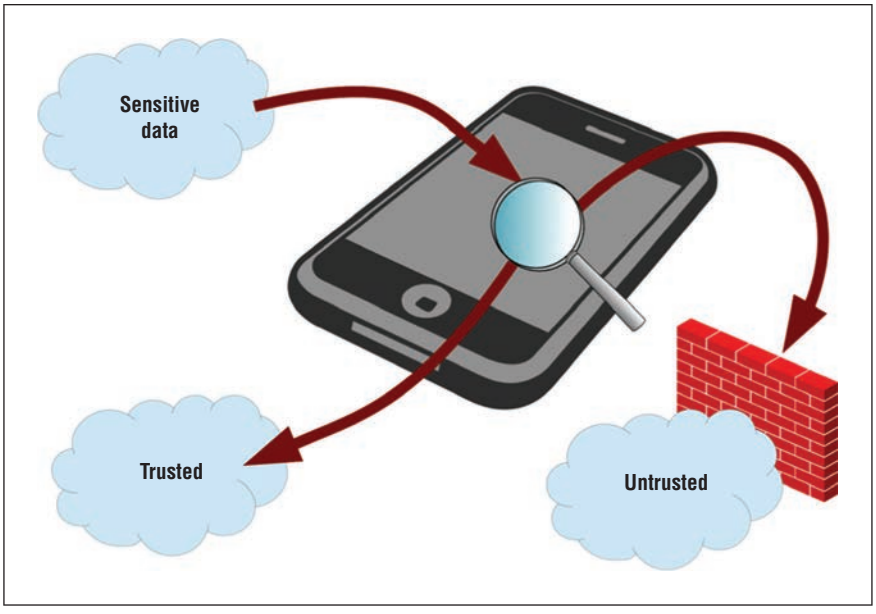


Figure 1. Taint-flow analysis can be used in protecting against the flow of sensitive data to undesired locations.

those links. The M SDK will increase Android security in additional ways, including Wi-Fi, Android application package (APK) validation, camera use, and more.

The second-highest market share smartphone is iOS. In iOS 8, users can install apps and control permissions afterward, although with limited granularity. In contrast to the current Android permissions model, iOS prompts the user to grant permissions only when the app is actually about to use the permission.

The worldwide third-highest-selling smartphone OS consistently (from 2012 through 2015) is the Windows Phone, which in Q1 2015 is estimated at almost three percent of worldwide smartphone sales (see www.idc.com/prodserv/smartphone-os-market-share.jsp). As opposed to iOS and Android, Microsoft provides developers five different application models for building Windows Phone apps. This adds to the complexity of app analysis, as well as to the analysis of dataflow and control (both app-to-app and app-to-system). Microsoft provides a tool for profiling and monitoring some behaviors of apps, and researchers have created some app analysis tools, but the Windows Phone lacks the number and depth of dynamic and static analysis frameworks and tools that exist for Android and iOS apps.

Analysis Tools

Many Android app analysis tools are built on the Soot[7] and T.J. Watson Libraries for Analysis (WALA) static analysis frameworks, and there are many standard dynamic analyzers (such as DroidScope[8]) and fuzzers (such as DroidFuzzer[9]) for Android apps. There are many analysis tools for iOS, including the PiOS[10] and Xavus[11] static analyzers and the PSiOS policy enforcement framework.[12] Static and dynamic (including fuzzing) analysis of potential dataflows and control flows are vital for understanding potential security issues in each smartphone system, including apps.

Moreover, vulnerabilities inherent to programming languages used for the systems should be examined, along with the security of the system in general (including encryption, deletion, password handling, and communications protocols used). Systems with small market shares tend to have fewer analytical tools. Figure 1 shows a high-level view of taint-flow analysis, which can be done with static tools (such as DidFail[13]) and dynamic tools (such TaintDroid[11]).

Smaller-Market Phones

CyanogenMod is an open-source firmware distribution based on Android that lets users install apps without granting all requested permissions. It also lets users substitute fake data instead of real data (for example, in place of real location data). Blackphone has an OS that is based on a fork of Android. It uses peer-to-peer encrypted calling and video, and it can use a privacy-focused enterprise management system. Silent Circle (the maker of Blackphone) has a privacy-focused app store, including Android and iOS apps with full call and text encryption (https://www.eff.org/secure-messaging-scorecard). Additional smartphone OSs with much smaller market shares include Blackberry, Symbian, Ubuntu, and China Operating System (COS).

Vulnerability Coordination

Despite the Blackphone's focus on security, a data-type confusion vulnerability in its code was disclosed and fixed in January 2015. The vulnerability could have allowed remote attackers to execute arbitrary code on Blackphones. This is a good example of how difficult it can be to secure smartphone communications and data, and of the importance of vulnerability report management. Blackphone's website has a secure form for reporting vulnerabilities. OS providers and app creators should have a way for the public to report security vulnerabilities and should work quickly to address them. Bug bounties are incentives to motivate vulnerability

disclosures and coordination with developers.

If the reporting method is insecure, a report could be intercepted by a third party, who could use it to exploit the vulnerability.[14] Google Android, Apple iOS, and Microsoft Phone have secure vulnerability reporting, coordination, and rewards programs. App developers might not respond to vulnerability disclosures, so to protect users, reporting should be coordinated by the app stores. CERT also handles vulnerability coordination between reporters and vendors/developers as a free public service.

App Permissions and Languages

Most users do not understand the full implications of allowing app permissions. A study in 2011 by Adrienne Porter Felt and her colleagues found evidence that even many developers don't fully understand permissions.[15] They found that many apps request extraneous permissions that aren't needed by any of the API calls that the app makes. They also found that, in many cases, the Android documentation about permissions was missing or incorrect.

User-experience researchers[16] work to understand effective (and ineffective) methods of conveying information to users who are not technical experts. Similar research projects strive to effectively support secure coding of apps with integrated development environment (IDE) assistance, secure coding standards, and other tools to analyze and improve app security during development. Developer education helps, including secure coding training for particular programming languages and OSs.

Undefined behavior in programming language standards leads to security vulnerabilities. Developers should follow secure coding standards for the programming languages and for the mobile OS, which impose rules and recommendations for coding securely that mitigate problems due to officially undefined behaviors. The smartphone's OS, drivers, application framework, virtual machine environment, and apps can be written in a variety of languages. For example, the Android OS is written mostly in C, runtime libraries are written in C/C++ except the Java Core libraries, and Android apps are written in Java but can incorporate native code (such as C or C++).

Hybrid Apps

Although hybrid Web/mobile application frameworks make development of cross-platform apps possible, recent research has shown serious vulnerabilities that expose sensitive local resources to malicious Web domains,[17] affecting all hybrid frameworks and smartphone platforms that deploy the frameworks.

Although hybrid Web/mobile application frameworks make development of cross-platform apps possible, recent research has shown serious vulnerabilities.

Cyber-Hygiene

Other factors in smartphone security could be helped by public-education programs similar to public-health education (such as campaigns to promote covering your mouth when sneezing) but for cyber-hygiene. Some users do not have a password login for their phone or a timed lockout, much less security afforded by phone encryption. These basic data protections should be used by everyone, given that devices are often lost or misplaced.

The above basic protections adequately protect data in many cases, but they are not fool-proof. A password-locked phone can be attacked by analyzing the smudges left when entering the password.[18] Sophisticated adversaries might be able to recover encryption keys from a powered-on Android phone's RAM[19] by a method involving physically chilling the phone.

USB power plugs could be abused as a data-channel attack vector against users who think they are simply charging their phone; a mitigation is to use a USB condom when connecting to an untrusted charging outlet. All personal data in the phone should be securely deleted before a user disposes of their phone. Backing up data by syncing it to a local machine or cloud protects the user's access to data even if a device is destroyed or lost, but privacy of the backed-up data depends on the backup system's privacy protections. A cyber-hygiene campaign could make more users aware of these risks and mitigations.

Women's clothing in particular presents a smartphone security issue, because most women's slacks and skirts do not come with front pockets even close to large enough to fit a smartphone. (However, you can have a tailor extend your front pockets to securely carry a smartphone.) Carrying a phone in a purse, backpack, or jacket pocket increases the likelihood of theft or loss, plus the risk of tampering (such as inserting a key logger), compared to carrying it in pants pockets.

Encryption

SSL, if used correctly, promises to provide secure end-to-end communication over an insecure channel. A comprehensive research project, which analyzed Google Play apps that use cryptographic APIs, showed that 88 percent used SSL incorrectly.[20] Tools such as mallodroid and CERT Tapioca find SSL vulnerabilities in apps. Furthermore, a standard Android, iOS, or Windows Phone and browser are vulnerable to a *compelled certificate creation attack*, in which government authorities would compel a

SMARTPHONES

certificate authority to issue false SSL certificates for covertly intercepting and hijacking secure Web-based communications.[21]

Cell phones encrypt voice data using keys in SIM cards. However, if an attacker obtains these SIM keys, decryption of phone communications using those SIMs is trivial. Gemalto, which manufactures approximately 2 billion SIM cards annually, was reportedly hacked and its SIM cards' encryption keys were stolen.[22]

Baseband

The *baseband OS* provides another attack surface. Most smartphones include two operating systems on two different processors: the general-purpose applications processor runs the main OS (for example, Android or iOS) and a processor that executes a proprietary real-time OS and manages all radio functions (the baseband OS). Stingray technology uses vulnerabilities in baseband technologies, such as knocking phones off a 3G network and onto an insecure 2G network with a fake base station, to intercept cellphone communications.[23]

Baseband software is currently poorly understood, because it is closed-source. Tools available to the public for analyzing baseband software are limited, and baseband is a promising area for vulnerability research and mitigation. OpenBTS, OsmoBTS, and OpenLTE are open source software that enables software-defined radio communications, making research on mobile baseband security more affordable. Most baseband processors are ARM processors, which the widely used IDA Pro disassembler supports. Google's BinDiff tool has also been used by baseband researchers to identify and match functions in binaries. Increasingly, research publications detail baseband vulnerabilities and potential attacks that have been researched using OpenBTS with software-defined radios, IDA Pro, and BinDiff.

The security landscape of mobile devices is far from ideal, and there are many problem areas ripe for further research. Exciting, high-impact topics for research include better user interfaces, improved encryption, finding and securing baseband OS vulnerabilities, and many more. Non-research work needed includes public cyber-hygiene educational campaigns and improved distribution for security updates.

REFERENCES

1. L. Armasu, "Google Can't Ignore the Android Update Problem Any Longer," *Tom's Hardware*, 5 May 2015; www.tomshardware.com/news/google-android-update-problem-fix,29042.html.
2. G. Wassermann, *CERT Vulnerability Note VU#924951*, Vulnerability Notes Database, July 2015; www.kb.cert.org/vuls/id/924951#sthash.2Z6iNXBT.dpuf.
3. J. Minor, "There's (Almost) Nothing You Can Do About Stagefright," *PC Magazine*, 30 July 2015; www.pcmag.com/article2/0,2817,2488772,00.asp.
4. L. Xing et al., "Unauthorized Cross-App Resource Access on MAC OS X and iOS," 2015; http://arxiv.org/abs/1505.06836.
5. D. Octeau et al., "Effective Inter-Component Communication Mapping in Android: An Essential Step Towards Holistic Security Analysis," *Proc. 22nd USENIX Conf. Security* (SEC), 2013, pp. 543–558; http://dl.acm.org/citation.cfm?id=2534813.
6. E. Schwartz et al., "Q: Exploit Hardening Made Easy," *Proc. 20th USENIX Conf. Security* (SEC), 2011, p. 25; http://dl.acm.org/citation.cfm?id=2028092.
7. R. Vallée-Rai et al., "Soot—A Java Bytecode Optimization Framework," *Proc. 1999 Conf. Centre for Advanced Studies on Collaborative Research* (CASCON), 1999, p. 13; http://dl.acm.org/citation.cfm?id=782008.
8. L.K. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," *Proc. 21st USENIX Conf. Security Symp*, 2012, pp. 569–584.
9. H. Ye et al., "DroidFuzzer: Fuzzing the Android Apps with Intent-Filter Tag," *Proc. Int'l Conf. Advances in Mobile Computing & Multimedia* (MoMM), 2013, p. 68; http://dl.acm.org/citation.cfm?id=2536881.
10. M. Egele et al., "PiOS: Detecting Privacy Leaks in iOS Applications," *Proc. 18th Ann. Network and Distributed System Security Symp*., 2011; https://iseclab.org/papers/egele-ndss11.pdf.
11. W. Enck et al., "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Trans. Computer Systems*, vol. 32, no. 2, 2014, article no. 5; http://dl.acm.org/citation.cfm?id=2619091.
12. T. Werthmann et al., "PSiOS: Bring Your Own Privacy & Security to iOS Devices," *Proc. 8th ACM SIGSAC Symp. Information, Computer and Communications Security* (ASIA CCS), 2013, pp. 13–24; http://dl.acm.org/citation.cfm?id=2484316.
13. W. Klieber et al., "Android Taint Flow Analysis for App Sets," *Proc. 3rd ACM SIGPLAN Int'l Workshop on the State of the Art in Java Program Analysis* (SOAP), 2014; http://dl.acm.org/citation.cfm?id=2614633.
14. A. Fishman and M. Marquis-Boire, "Popular Security Software Came under Relentless NSA and GCHQ Attack," *The Intercept*, 22 June 2015; https://firstlook.org/theintercept/2015/06/22/nsa-gchq-targeted-kaspersky.
15. A.P. Felt et al., "Android Permissions Demystified," *Proc. 18th ACM Conf. Computer and Communications Security* (CCS), 2011, pp. 627–638; http://dl.acm.org/citation.cfm?id=2046779.
16. A.P. Felt et al. "Android Permissions: User Attention, Comprehension, and Behavior," *Proc. Eighth Symp. Usable Privacy and Security* (SOUPS), 2012, article no. 3; http://dl.acm.org/citation.cfm?id=2335360.
17. M. Georgiev et al., "Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks," *Proc. Network and Distributed System Security* (NDSS), 2014; https://www.cs.utexas.edu/~shmat/shmat_ndss14nofrak.pdf.
18. A. Aviv et al., "Smudge Attacks on Smartphone Touch Screens," *Proc. 4th USENIX Workshop on Offensive Technologies* (WOOT), 2010,

pp. 1–7; http://dl.acm.org/citation.cfm?id=1925009.

19. S. Anthony, "How to Bypass an Android Smartphone's Encryption and Security: Put It in the Freezer," *Extreme Tech*, 12 Mar. 2013; www.extremetech.com/computing/150536-how-to-bypass-an-android-smartphones-encryption-and-security-put-it-in-the-freezer.

20. M. Egele et al., "An Empirical Study of Cryptographic Misuse in Android Applications," *Proc. 2013 ACM SIGSAC Conf. Computer & Communications Security* (CCS), 2013, pp. 73–84; http://dl.acm.org/citation.cfm?id=2516693.

21. C. Soghoian and S. Stamm, "Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL," *Financial Cryptography and Data Security*, LNCS, Springer, vol. 7035, 2012, pp 250–259.

22. J. Scahill and J. Begley, "The Great SIM Heist," *The Intercept*, 19 Feb. 2015; https://firstlook.org/theintercept/2015/02/19/great-sim-heist.

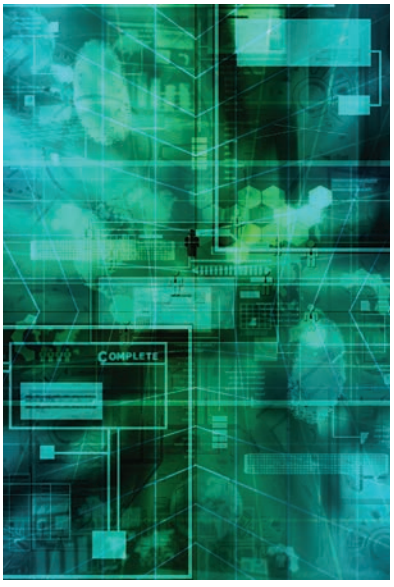
23. S. Pell and C. Soghoian, "Your Secret Stingray's No Secret Anymore: The Vanishing Government Monopoly over Cell Phone Surveillance and Its Impact on National Security and Consumer Privacy," *Harvard J. Law and Technology*, vol. 28, no. 1, 2014.

Lori Flynn is a software security researcher at CERT, in the Software Engineering Institute of Carnegie Mellon University. Contact her at lflynn@cert.org.

Will Klieber is a software security researcher at CERT, in the Software Engineering Institute of Carnegie Mellon University. Contact him at weklieber@cert.org.

This article originally appeared in IEEE Pervasive Computing, *vol. 14, no. 4, 2015.*

EDITOR: Karen Evans, KE&T Partners, karenevans@prodigy.net



Establishing and Maintaining Trust in a Mobile Device

Kristopher Carver, *BlueRISC*
Vincent Sritapan, *US Department of Homeland Security Science and Technology Directorate*
Cherita Corbett, *SRI International*

The mobile device market has grown tremendously. Individuals, businesses, and governments rely on mobile devices to access critical infrastructure and share vital information (banking, medical data, intellectual property, and so on). This growth in adoption has also brought about a parallel surge in attacks. Malware, ransomware, and spyware are targeting mobile platforms to steal sensitive data, access private networks, track users, and do other nefarious activities. Particularly for governments using mobile technology, mobile attacks can disrupt life-saving operations, endanger personnel, and expose government systems to exploitation. Securing mobile devices is no small feat and is therefore a forefront issue to the US Department of Homeland Security Science and Technology Directorate's (DHS S&T's) cybersecurity R&D program.[1]

Roots of Trust

Mobile *roots of trust* (RoT) are highly trustworthy, tamper-evident components that can provide a foundation for building security and trust for mobile devices. RoT is usually provided as a specialized hardware chip (such as a trusted platform module) on desktop or laptop systems. However, mobile devices are resource-constrained and lack dedicated hardware mechanisms for providing RoT. This leaves a single solution—namely, to provide RoT in software. Unfortunately, this is challenging to realize given the sophistication of current threats and the ease with which a mobile device's state and information can be extracted and altered. Moreover, security specifications such as the Trusted Computing Group's Mobile Trusted Module[2] don't address how to support mobile RoT requirements in software, nor do they address dynamic verification of device and software behavior while applications are running.

BlueRISC is developing MobileRoT, a fully software-based dynamic mobile trusted module technology under support from the DHS S&T Cyber Security Division (CSD). MobileRoT measures and verifies a device's static and runtime state (for example, boot loader, operating system, apps, and runtime memory) to enable trust and overall device security. It can be utilized to detect malicious system changes or activity and to ensure that access to critical information and software can only be performed in a trusted state. MobileRoT requires no modifications to the underlying operating system kernel, nor any manufacturer or service provider support for insertion, greatly reducing hurdles to adoption.

MobileRoT Architecture

To overcome the array of surface attacks targeting software-based systems, MobileRoT utilizes a new architecture for enabling transitive trust based on the Core Root of Trust for Measurement (CRTM).[2] The CRTM is hardened code that acts as the RoT for reliable integrity measurements and is the foundation for additional trusted services. The MobileRoT architecture includes a layer of encrypted CRTM code that is tied to a cryptographic key

 Published by the IEEE Computer Society

generated at boot-time. With the CRTM established, the resulting system doesn't require any sensitive information to be stored persistently in an unprotected state, closely mimicking the level of security achievable via dedicated hardware. A secure cryptographic sealing and unsealing procedure tied to the boot-time and runtime measurements performed by the solution enables application and data protection. Because all protected data and applications are sealed, they remain protected even in cases in which an attacker attempts to alter or bypass the MobileRoT technology.

Figure 1 shows an example measurement and verification flow, which illustrates the boot process of an Android mobile device and gives an example of how an RoT can be established within it. In this example, the MobileRoT sits logically between the boot loaders and the Android kernel. Here, it can establish the CRTM and perform backward verification (1), self-verification (2), and forward verification (3) of both privileged components and userland applications.

Traditional solutions focus primarily on boot-time validation, establishing the validity of each component prior to a complete boot, while providing only minimal support for runtime activities. Unfortunately, it is widely known that sophisticated attacks can target applications that are already running, and devices these days are rarely rebooted. To address the shortcomings of one-time static verification, MobileROT provides dynamic verification and attestation by performing runtime measurements of the system state of the device (4 and 5 in Figure 1). These runtime agents harden themselves from attack and modification by creating a self-validating network, which can instantly respond to a threat to the system or the protection technology itself.

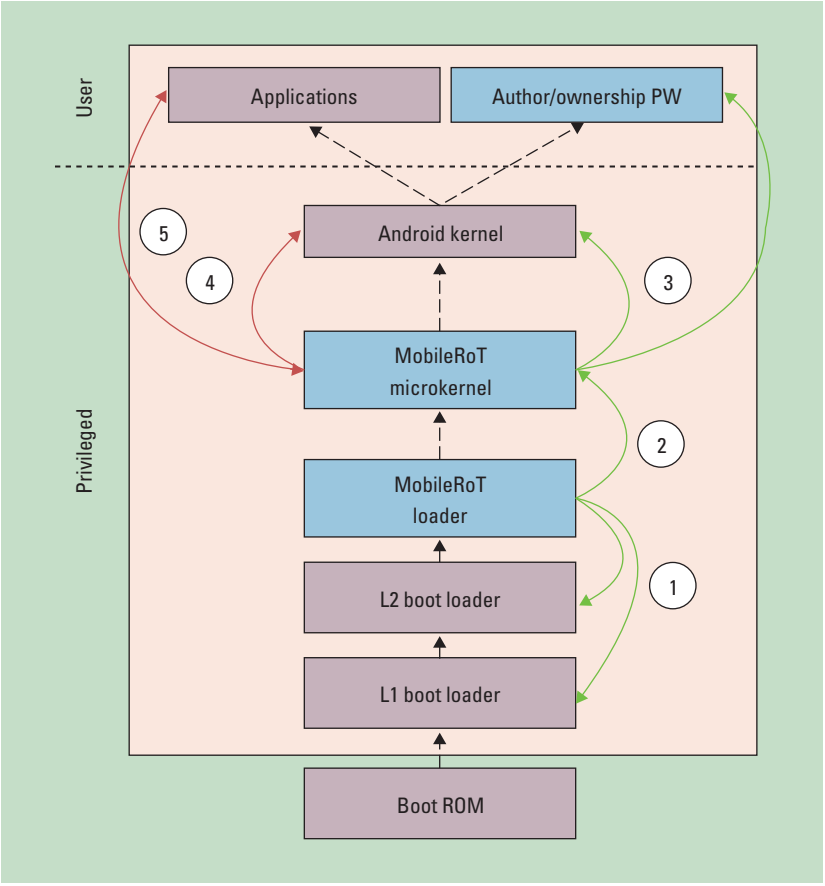


Figure 1. An example measurement and verification flow for an Android mobile device. MobileRoT can perform (1) backward verification, (2) self-verification, (3) forward verification, and (4) and (5) runtime measurement. MobileRoT supports this example flow as well as others while transparently incorporating many additional features.

MobileRoT reliably allows all levels of software, including user applications, to have access to its trusted services through an open API. This enables the creation of secure, off-the-shelf, third-party and proprietary applications and data, and strengthens key management and policy enforcement technology, such as mobile device management (MDM). MobileRoT also provides fine-grained protection integrated directly into an application. For example, a standard Android Calendar application can be modified to support the concept of a "secure event." This secure event is established in cooperation with the MobileRoT and persistently protected. To view a secure event, proper authorization and authentication is required, and the system state must be verified.

Although cybercrime targeting mobile devices is becoming pervasive, mobile RoT can preserve and confirm the integrity of the device while it's at rest or in use. BlueRISC's MobileRoT technology has overcome barriers to bring RoT to a mobile platform, providing a foundation of security features to accelerate the development of secure mobile devices.

Acknowledgments

The views and conclusions contained herein are the authors' and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Department of Homeland Security (DHS) or the US government. The work by BlueRISC was sponsored by the DHS Homeland Security Advanced Research Projects Agency (HSARPA), Cyber Security Division (CSD), via the Small Business Innovation Research Program under contract number D14PC00178. The work by SRI International was funded by the DHS S&T under contract number HSHQDC-10-C-00144.

References

1. D. Maughan et al., "Government-Funded R&D to Drive Cybersecurity Technologies," *IT Professional*, vol. 17, no. 4, 2015, pp. 62–65.
2. *Mobile Phone Work Group Mobile Trusted Module Specification version 1.0*, Trusted Computing Group, Apr. 2010; www.trustedcomputinggroup.org/resources/mobile_phone_work_group_mobile_trusted_module_specification.

Kristopher Carver *is the technical director at BlueRISC, and principal investigator on the MobileRoT project sponsored by the US Department of Homeland Security Science and Technology Directorate. He leads the technical direction of BlueRISC and its system assurance and cybersecurity products and technologies. Carver has more than 10 years of experience in software and hardware protection solutions. He is co-inventor of seven patents. Contact him at kris@bluerisc.com.*

Vincent Sritapan *is a program manager in the US Department of Homeland Security Science and Technology Directorate's Cyber Security Division. His projects focus on mobile software-based roots of trust, mobile device protection, access control, and continuous automated assurance for mobile apps. Sritapan is also an information professional officer in the US Navy Reserves. Contact him at vincent.sritapan@hq.dhs.gov.*

Cherita Corbett *is a senior computer scientist at SRI International. Her research interests include mobile security, self-healing cellular networks, and cyber-physical systems security. Corbett provides subject matter expertise and project management for the US Department of Homeland Security Science and Technology Directorate's cybersecurity R&D program. Contact her at cherita.corbett@sri.com.*

This article originally appeared in IT Professional, *vol. 17, no. 6, 2015.*

Editors: Angela Sasse, a.sasse@cs.ucl.ac.uk | Alessandro Acquisti, acquisti@andrew.cmu.edu

Possessing Mobile Devices

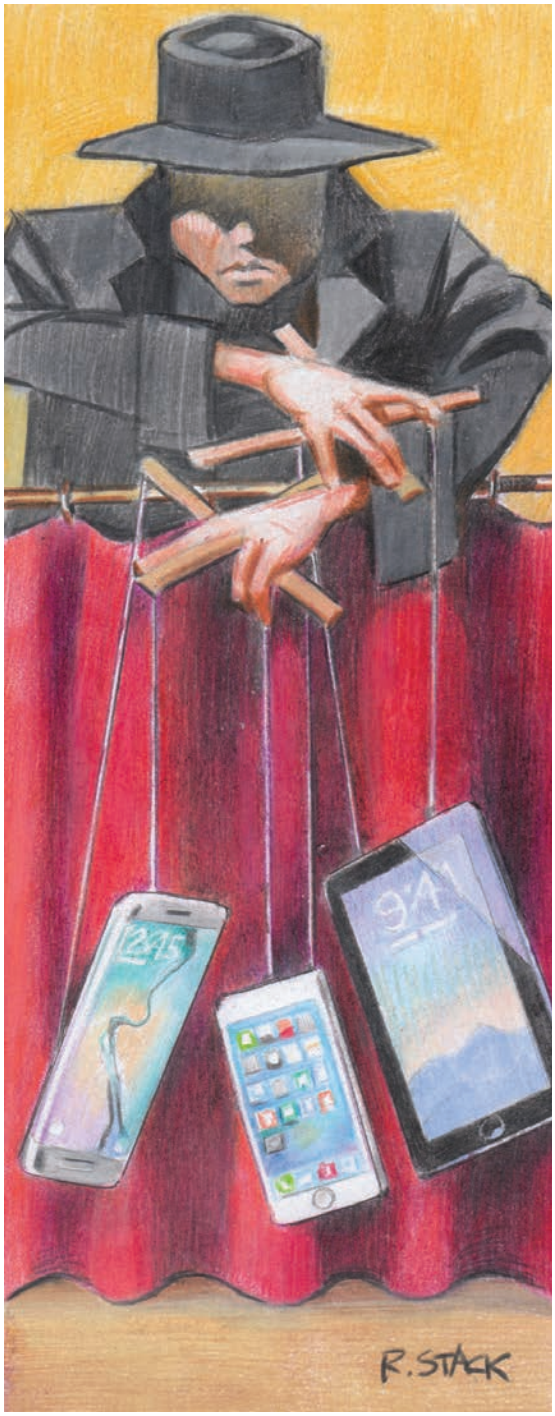
A.A. Adams | Meiji University

Although modern smartphones and tablet computers are at least as powerful as the PCs of a decade or so ago, they're viewed as primarily media consumption and communication devices. As such—and despite the fact that their associated hardware, such as built-in cameras, microphones, accelerometers, and GPS and other location sensors, pose significant privacy risks—such devices' ownership model owes more to home gaming consoles than PCs. According to International Data Corporation and Strategy Analytics, more than 1 billion smartphones running the Android OS were shipped in 2014 (that is, sent out from manufacturers but not necessarily sold to or used by consumers), accounting for more than 80 percent of the year's market.[1,2] Combined with the more than 192 million iPhones shipped (approximately 15 percent of the market), Android and iOS phones accounted for 96 percent of the smartphones shipped in 2014. These figures consider only smartphone shipments, not tablets. In addition, 2014 reports concluded that most online information exchange in the US now occurs via mobile devices.[3,4] Users pay significant amounts of money for these devices and use them to conduct much of their social and business lives. But despite users having paid for the devices, their owners' rights are limited unless they bypass the built-in software restrictions and *root* their Android device or *jailbreak* their iOS device. So, it appears that owners don't truly own these devices containing so much of their personal, sensitive information.

In this article, I explore these ownership issues—their origins; their security, privacy, and autonomy implications for users; and their economic and ecological implications. In particular, I argue that the current smartphone and tablet ownership model violates users' reasonable expectations and fundamental rights without giving them sufficient recompense. Although users buy and therefore supposedly own the devices, the manufacturers or software system integrators, retailers, and network connection providers retain considerable control, prohibiting users from both protecting their privacy and making use of the device's full capabilities. I also argue that the claim that such external control improves users' security is false in multiple ways.

Ownership

Ownership is not as simple a concept as it might first appear. There are legal concepts of ownership that confer both rights and responsibilities to the owners. There are psychological elements such that individuals might feel that their rights—or even their person—have been violated when their legal rights or technical control over their possessions don't match their expectations. There are economic issues driving the market that restrict device owners' actions. In the end, what we think of as ownership is simply shorthand for a bundle of rights in an object. Ownership



2376-113X/16/$33.00 © 2016 IEEE Published by the IEEE Computer Society

rights usually include the right to decide who can use an object, and something owned can usually be sold to another.

However in most countries, we can't sell our body organs, even those without which we could survive (such as one kidney or part of a liver)—despite most people considering themselves owners of their own bodies.[5] Such restrictions are often justified by appeal to a general social benefit, such as avoiding exploitation of the poor as a resource for body parts by the rich. In the case of smartphones and tablets, however, these ownership restrictions seem far from justified when we consider the privacy and security costs to users.

Psychological Attachment to Personal Devices

Smartphones are both phones and computers. Therefore, to understand user expectations of ownership, we must consider the background of ownership rights for both. I focus on the PC era for computers and (mostly) the mobile phone era for telephones. PCs allowed people to have computers not only in their homes but also in their individual office spaces; despite these office PCs being owned by the organization, many people described and felt them to be *personal* devices. As Byron Reeves and Clifford Nass note, people's emotional and psychological attachments to devices are often quite illogical, such as distinguishing between completely fungible devices (identical specification, all data stored on a network) based simply on prior usage of a particular machine.[6]

In many countries, early fixed-line phone networks only allowed devices supplied by the network operator to connect to the network. They claimed that this was to prevent damage to the network, although as the US Carterfone case demonstrated, this was at least partly a spurious claim; in fact, the preservation of sales or rental income on monopoly-provided equipment was the primary reason.[7,8] Pre-smartphone mobile phones in the developed world quickly became objects of deep emotional attachment for their owners.[9]

In the case of smartphones and tablets, ownership restrictions seem far from justified when we consider the privacy and security costs to users.

Given the intense and intimate usage of modern mobile devices, it's unsurprising that users develop strong positive feelings, including trust, toward their devices. However, this trust is misplaced because they actually give up a great deal of control to the real "owners" of the devices: the providers (primarily manufacturers and mobile phone operating companies).

Technical Ownership (Control) of Mobile Phones

Early digital mobile phones had very limited capabilities beyond making phone calls and sending and receiving short text messages. As their capabilities expanded to include digital cameras and connections to networked information services, the hardware and operating systems became more complicated, and interoperability issues between networks and phones and between phones and other devices (particularly PCs) arose. Early feature phones containing information services ran various OSs with different openness levels.

Most early feature phones included limited or no ability to update the system software. In particular, over-the-air (OTA) updates to the phone's firmware (that is, downloading an updated core OS via the mobile network) weren't generally supported. Many phones ran highly customized OSs, and few systems were used by more than one manufacturer.

Firmware-installed OS upgrades, if possible at all, were generally restricted to special-purpose hardware at service centers. Some later phones allowed users to update by downloading new firmware to a PC over the Internet: they had to connect the phone to the PC and run an update program on the PC to rewrite the phone's software. (This was also the update process for iPhones until iOS 5, which introduced OTA updates.)

The road from digital mobile handset to smartphone had many dead ends, byways, and failed highway projects. The smartphone basically combines a digital mobile phone handset and a PDA. The degree of openness of many early smartphones reflected the creators' route—whether they started with a phone and tried to give it PDA functionality and Internet access, or started with a PDA and tried to give it phone functionality and Internet access. For example, Nokia and Microsoft started from the PDA concept, with Nokia creating environments such as the S60 platform and the Symbian system (which superseded S60 at Nokia and was based on the EPOC OS from the UK's Psion PDA maker). Microsoft developed the Windows CE and Windows Mobile systems, both of which had open application development layers and allowed user installation of applications.

In Japan, NTT, the former state fixed-line phone monopoly provider, developed Mobile-Oriented Applications Platform (MOAP) systems—one based on a Symbian kernel and the other on a Linux

kernel—which had neither open third-party development options nor user-installable applications. These systems used NTT's proprietary i-mode system to provide Internet-like services, including translating suitable webpages into a form viewable on the grayscale phone screen and using the keypad for interaction. As with computer gaming consoles, MOAP systems had application development platforms. To access them, development companies were required to enter into contracts with NTT. Application development for these systems was typically done by or under contract to the hardware manufacturer, who sought to compete in the market by offering built-in applications. Japanese rivals au by KDDI and SoftBank Mobile developed phones supporting the Wireless Application Protocol (WAP) standard that allowed access to websites through a stylesheet-like approach. Interactive applications running locally on the device, however, could be produced only using proprietary software development kits. For example, email on these Japanese phones was available only through dedicated apps using the service provider's mail server, or through a WAP-enabled webmail service.

Systems with open application-development environments such as PalmOS, its successor WebOS, Symbian, BlackBerry OS, iOS, and Android have gradually taken much of the market share for mobile devices, including not just smartphones but also larger tablets. Again, devices running these systems are really general-purpose computing devices with mobile networking and integration with POTS (plain old telephone service) via a "phone" app. They're designed to be devices with which software is used, rather than on which software is developed. Although programs can be developed in some applications (such as TerminalIDE for Android), these devices aren't intended as platforms on which to develop apps to run on them. Most development happens on other, more powerful computers running suitable development tools.

There have been and remain many levels of openness in these systems with regard to user control. iOS devices generally only allow applications to be installed from the Apple App Store. Android vendors can preset application sources to be allowed or disallowed. Some distributed versions allow users to switch on other sources, whereas other distributed versions limit application sources to those they have preset. RIM's BlackBerry OS before version 10 (which was a complete rewrite based on the QNX kernel) restricted application installation to only RIM's repository. The BlackBerry 10 system, however, supports Android applications including the ability to install applications from alternative sources like the Amazon Appstore for Android. Only apps from the Windows App store can be installed on Window Phone devices.

Anyone with physical access to a device can, with enough effort, control that device. Physical access restrictions are a standard part of security engineering.[10] However, most people don't have the expertise or equipment to work around devices' built-in control restrictions. Sometimes there are legal restrictions on doing so that make it illegal[11] or more difficult to obtain the required hardware,[12] or that place the user in breach of contract.[13]

Although manufacturers such as Sony and Asus provide instructions and options for users to access full administrative rights (root user or superuser) on some Android-based devices, they do so only with the mobile network provider's agreement, which is often withheld. Many manufacturers and mobile network operators preload Android devices with *bloatware*—(often unwanted) apps that aren't deletable on a nonrooted phone. Many of these apps are set to start on boot, requiring users to manually turn them off after every reboot—the option to not run on boot is usually locked in the user settings.

Interestingly, the Shanghai Consumer Council, a small consumer protection group in China, recently launched a lawsuit against Samsung and Chinese vendor Oppo for violating consumers' rights by selling them devices with undeletable bloatware.[14]

Regulators such as the US Federal Communications Commission are reluctant to require manufacturers and network operators to grant users full control over their own devices. They're concerned that users might misuse software-defined radio capabilities to interfere with other mobile phones and radio communications. However, neither the US Copyright Office's exemption of iPhone jailbreaking and Android rooting[11] from the Digital Millennium Copyright Act's (DMCA's) anticircumvention rules nor the prevalence of these practices by users have persuaded telecom regulators to insist that users be given real ownership of and control over their devices.

Security and Privacy on Possessed Devices

Smartphones and tablets are primarily used for communication (social networking services, photo sharing, messaging, and voice and video calls), although media consumption (games, videos, audio, and text) and information processing (note-taking and self-quantification) are also significant uses. The locked-down model of previous generations' media consumption devices—whereby the manufacturer or other upstream retailer significantly controls the device—seems a poor deal for consumers. Bruce Schneier called this

the "feudal security" model (http://tinyurl.com/b7s2fq4; http://tinyurl.com/k8x5de4). As in the feudal social model, the overlords aren't trustworthy, and the moral hazards of their position without strong external regulation lead them to abusive practices such as secretly spying on users' locations (see for example, Google and Apple[15,16]). Meanwhile, device manufacturers are constantly tweaking proprietary device drivers for their Android phones,[17] shipping binary blobs for attachment to Android's Free Software Linux kernel, all with too little appreciation of the security risks of these often hastily programmed hardware interfaces.

Direct Security Risks of Rooting and Jailbreaking

Steffen Liebergeld and Matthias Lange discuss the risks users run if they root their Android devices,[18] and Kevin Rogers provides a similar discussion of the dangers of jailbreaking iOS devices.[19] Because neither Android nor iOS is designed to run administration accounts, despite both being based on Unix-related kernels (Linux and XNU [X is Not Unix], respectively), once the systems are hacked to expose these administrator-level accounts, they're more vulnerable to external hacking. Although users' privacy and, to some extent, security are always at risk from any application they install (and from other vectors), once they've rooted or jailbroken their device, the applications they install can request root access, which many users will likely grant—just as they grant privacy-invasive privileges to apps such as those to use the camera flash like a flashlight.[20]

Indirect Security Risks of Rooting and Jailbreaking

The hoarding of vulnerabilities by the US National Security Agency and UK Government Communications Headquarters (and probably many other signals intelligence agencies) is condemned by security professionals as putting everyone's security at risk by decreasing the chances that project management becomes aware of vulnerabilities and takes steps to fix them.[21] Similarly, because jailbreaking an iOS device or rooting many Android devices requires breaking their security model, users (particularly highly skilled white hat hackers) have an incentive to prevent system developers from knowing about the vulnerabilities they exploit. These vulnerabilities, in addition to being used by users to gain control over their devices, can also be used by attackers to elevate their privileges as part of a malicious attack.

Preventing users from controlling their own devices encourages them to try to follow instructions on bypassing their devices' security from dubious sources.

In addition, preventing users from controlling their own devices encourages them to try to follow instructions on bypassing their devices' security from dubious sources. Although most online directions about jailbreaking and rooting devices are what they appear, most users don't have the technical expertise to know whether the directions will actually help them achieve their goals or, instead (or in addition), install malware or open up a security hole in their device. Such attacks often target Facebook users; Facebook calls this the *self-cross-site-scripting attack* (www.facebook.com/notes/facebook-security/dont-be-a-self-xss-victim/10152054702905766). Users' willingness to follow somewhat random online advice on breaking their devices' security shouldn't be encouraged, any more than car owners should be encouraged to install updates to their cars' onboard systems using a USB stick delivered to their address without verifying its source as the manufacturer.[22]

Security Risks of Not Rooting and Jailbreaking

Without administrative control of a device, checking the integrity of system files and monitoring the presence and activity of installed applications are very difficult. On both iOS and Android, in fact, ordinary user-space applications aren't supposed to monitor or interfere with other apps. Google and Apple enforce such policies in their respective app stores, although for most Android devices, you can install apps from other sources. Even when such monitoring can be installed as a user-space app, its access to other software's activities is limited.

Lack of administrative control becomes increasingly problematic as smartphone providers (manufacturers, system integrators, telcos, and so on) apparently want users to upgrade their devices more often than some might want to. With the rapid development of iOS, Android, and new models, system providers (typically manufacturers) aren't providing older devices with updates. Even if manufacturers support these older devices, updates are being rolled out far too infrequently. Daniel Thomas and his colleagues recently showed that even though Google is patching the base Android system, many manufacturers are very slow to feed such patches through to users' devices: 87 percent of the Android machines in their study had known unpatched vulnerabilities.[23]

Once updates for the core iOS or Android system stop appearing,

devices often can't run updated versions of various apps, leaving them vulnerable to security problems in the apps' older versions as well as in the OS itself. A very serious version of this problem appeared in January 2015, when Google announced it wouldn't be providing a security fix for a known vulnerability in the WebKit Web browser app, a key element of Android 4.1 to 4.3.[24] (Google did say it would accept and push a patch if offered by a reliable third party). Although it's possible to use alternative browsers such as Mozilla's Firefox, which is updated and available even on the older Android versions, many apps use the WebKit rendering engine for their own HTML parsing and presentation. As I noted, users find it difficult or impossible to know which apps interoperate with which other elements of the system, particularly core elements such as the Web rendering engine.

Unlike, for example, PCs running Windows XP—which Microsoft supported with security patches for more than a decade—Android 4.3.1 was only released in October 2013. Users are completely at the hardware manufacturer's mercy to compile and release a new version of Android for their hardware. So, the manufacturer likely hasn't updated phones released in mid-2013 beyond 4.3.1, which less than 18 months after release, had security vulnerabilities in a core service app that Google decided not to patch and which, even if patched by Google, would probably not be offered as a downstream update by other manufacturers.

Without administrative access, which smartphone providers are reluctant to grant, Android users can't even install an alternative compatible OS such as CyanogenMod. iOS device users face a similar situation, with their reasonably recent devices (sometimes less than two-years old) being left out of the OS upgrade cycle; thus, they're forced to upgrade their hardware or remain vulnerable. Even for a jailbroken iPhone, there appears to be no alternative OS that can be installed to compensate for the lack of an Apple-provided, security-updated iOS.

Limitations should be clearly justified as in the public interest, not simply in the providers' commercial interest.

Privacy Risks of Not Rooting and Jailbreaking

Security and privacy are often represented as oppositional duals: one must give up some privacy to gain some security. Although this might be true in some circumstances, the security of the devices we use is a prerequisite for privacy, not in opposition to it. Being able to see whether unauthorized software is running requires administrator access, as does monitoring and controlling apps' provision of private information. Android applications such as Android Privacy Guard require root access to provide such facilities to users.

So, Who Owns My Device?

So, ownership isn't a single absolute concept granting all possible rights to an item. However, smartphones—whose hardware, such as microphones, cameras, accelerometers, and GPS, and software and data, such as contact listings, photos, social network posts, email, communications, and media consumption, make them so useful but also so risky in terms of privacy and security—aren't primarily owned by their users. Instead, the phone company, hardware manufacturer, and system integrator are these devices' practical owners.

This lack of ownership requires, at the very least, significantly improved consumer rights and privacy protections. As Thomas and his colleagues showed, Android smartphone manufacturers are leaving their users' software vulnerable by not providing regular updates.[23] In the PC world, patching has become one of the standard backbones of ensuring security. System administrators who don't patch their systems are regarded as unprofessional at best, and criminally negligent at worst. Home users are exhorted to keep their systems up to date; in fact, in an effort to preserve the ecosystem's security, Windows 10 Home Edition no longer allows users to defer security updates.

However, there is a long history of software being provided "without warranty." Consumer goods such as cars and drinks used to be outside such negligence claims in most circumstances, but seminal court cases in the early 20th century established a duty of care for manufacturers to not sell dangerous goods into the supply chain, such as cars with faulty brakes (US: *MacPherson v. Buick Motor Co.*)[25] or drinks contaminated with slugs (Scotland: *Donoghue v. Stevenson*).[26] The implications of *MacPherson v. Buick Motor Co.* are likely to gain importance as cars become further informatized and, even without being driverless, increasingly vulnerable to external hacking.[27] Although not usually as physically dangerous, smartphones and tablets are so embedded in our lives that their information is vital to our personal infrastructure, and manufacturers', telcos', and retailers' lack of liability is hard to defend.

At best, the US Copyright Office's exemption of iOS jailbreaking and Android rooting from illegalization under the DMCA[19] should be extended in the US and adopted elsewhere as a clear right of device owners to opt out of external controls by others (whether a person or an organization) on any device; hardware owners should have full visibility of their device's operation and a much greater level of control—that is to say, proper ownership of the device.

Remaining limitations should be clearly justified as in the public interest, not simply in the providers' commercial interest (such as reducing costs by not issuing security updates, charging users for permission to use devices' innate capabilities, or profiting from the invasion of users' privacy). If support for security updates on a device is no longer offered, then restrictions on user access to full control of the device aren't justified. Perhaps at this point, legal liability for failures might shift from providers to users, much as it already does with PCs. Those still running Windows XP have only themselves to blame if their devices invade their privacy or are used as zombies in a botnet.

Unfortunately, events don't appear to be traveling in this direction. In 2012, the US Copyright Office demurred from extending their "right to jailbreak" from iPhones and Android phones to iPads and Android tablets.[28] The latest leaks about the controversial and secretly negotiated Trans-Pacific Partnership include requirements to further lock down devices and to impose harsh penalties—including destruction of the machine—on anyone found circumventing technical protection measures. If this agreement is adopted, in places like the US, Australia, and Japan, a rooted Android or jailbroken iOS device that could bypass digital rights management on music, books, or video files would be subject to confiscation and destruction.[29] ■

Acknowledgments

This work was funded by the following grants: JSPS Kaken (B) 24330127 Organisational and Individual Behaviour, and Personal Information Protection in the Age of Social Media; and JSPS Kaken (B) 15H03385 Easy Security and Privacy.

References

1. "Android and iOS Squeeze the Competition, Swelling to 96.3% of the Smartphone Operating System Market for Both 4Q14 and CY14, According to IDC," International Data Corp., 24 Feb. 2015; http://tinyurl.com/p5mltv4.
2. N. Mawston, "Android Shipped 1 Billion Smartphones Worldwide in 2014. Strategy Analytics Report," 29 Jan. 2015; http://tinyurl.com/om9etpe.
3. S. Perez, "Majority of Digital Media Consumption Now Takes Place in Mobile Apps," TechCrunch, 21 Aug. 2014; http://tinyurl.com/mlvo5el.
4. A. Lella and A. Lipsman, "The US Mobile App Report.comScore White Paper," comScore, 21 Aug. 2014; http://tinyurl.com/pokl2uf.
5. J.W. Harris, "Who Owns My Body," *Oxford J. Legal Studies*, vol. 16, no. 1, 1996, pp. 55–84.
6. B. Reeves and C. Nass, *The Media Equation*, 2nd ed., CSLI, 2002.
7. M.T. Hoeker, "From Carterfone to the iPhone: Consumer Choice in the Wireless Telecommunications Marketplace," *CommLaw Conspectus*, vol. 17, no. 1, 2008, p. 187.
8. T. Wu, "Wireless Carterphone," *Int'l J. Communication*, vol. 1, 2007, pp. 389–426.
9. J. Vincent, "Emotional Attachment to Mobile Phones: An Extraordinary Relationship," L. Hamill, A. Lasen, and D. Diaper, eds., *Mobile World*, Springer London, 2005, pp. 93–104.
10. R. Anderson, *Security Engineering*, 2nd ed., John Wiley & Sons, 2008.
11. T.B. Lee, "Jailbreaking Now Legal under DMCA for Smartphones, but Not Tablets," *Ars Technica*, 25 Oct. 2012; http://tinyurl.com/8os3qn5.
12. B.F. Fitzgerald, "The PlayStation Mod Chip: A Technological Guarantee of the Digital Consumer's Liberty or Copyright Menace/Circumvention Device?," *Media and Arts Law Rev.*, vol. 10, no. 1, 2005, pp. 85–98.
13. M.H. Wolk, "The iPhone Jailbreaking Exemption and the Issue of Openness," *Cornell J. Law and Public Policy*, vol. 19, no. 3, 2009, pp. 795–828.
14. M. Kan, "Samsung Faces Lawsuit in China over Bloatware on Phones," *PCWorld*, 3 July 2015; http://tinyurl.com/pqqddke.
15. R. Chow. "Why-Spy: An Analysis of Privacy and Geolocation in the Wake of the 2010 Google Wi-Spy Controversy," *Rutgers Computers and Technology Law J.*, vol. 39, 2013, pp. 56–93.
16. V. Kumpu, "Privacy and the Emergence of the 'Ubiquitous Computing Society': The Struggle over the Meaning of 'Privacy' in the Case of the Apple Location Tracking Scandal," *Technology in Society*, vol. 34, no. 4, 2012, pp. 303–310.
17. X. Zhou et al., "The Peril of Fragmentation: Security Hazards in Android Device Driver Customizations," *Proc. 2014 IEEE Symp. Security and Privacy* (SP 14), 2014, pp. 409–423.
18. S. Liebergeld and M. Lange, "Android Security, Pitfalls and Lessons Learned," *Proc. 28th Int'l Symp. Computer and Information Sciences* (ISCIS 13), 2013, pp. 409–417.
19. K. Rogers, "Jailbroken: Examining the Policy and Legal Implications of iPhone Jailbreaking," *J. Technology and Law and Policy*, vol. 13, no. 2, 2013; http://dx.doi.org/10.5195/tlp.2013.118.
20. "Flashlight Apps Threat Assessment

Report," SnoopWall, 2014; http://tinyurl.com/pvj3oh3.
21. M.D. Cavelty, "Breaking the Cyber-Security Dilemma: Aligning Security Needs and Removing Vulnerabilities," *Science and Engineering Ethics*, vol. 20, no. 3, 2014, pp. 701–715.
22. N. Ford, "Hacked Jeep USB Software Patch Criticized," IT Governance, 8 Sept. 2015; http://tinyurl.com/qy82237.
23. D.R. Thomas, A.R. Beresford, and A. Rice, "Security Metrics for the Android Ecosystem," *Proc. 5th Ann. ACM CCS Workshop Security and Privacy in Smartphones and Mobile Devices* (SPSM 15), 2015, pp. 87–98.
24. P. Bright, "Google Won't Fix Bug Hitting 60 Percent of Android Phones," *Ars Technica*, 13 Jan. 2015; http://tinyurl.com/o2d5hho.
25. J.W. Wade, "Strict Tort Liability of Manufacturers," *Southwestern Law J.*, vol. 19, 1965, p. 5.
26. R.F.V. Heuston, "Donoghue v. Stevenson in Retrospect," *Modern Law Rev.*, vol. 20, 1957, p. 1.
27. A. Greenberg, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," *Wired*, 21 July 2015; http://tinyurl.com/oaabx46.
28. "Exemption to Prohibition on Circumvention of Copyright Protection Systems for Access Control Technologies," Copyright Office of the US Library of Congress, 2012; http://tinyurl.com/ngewrn2.
29. J. Pearson, "White Hat Hackers Would Have Their Devices Destroyed under the TPP," Motherboard, 9 Oct. 2015; http://tinyurl.com/o3bm553.

A.A. Adams is a professor of information ethics at Meiji University. Contact him at aaa@meiji.ac.jp.

This article originally appeared in IEEE Security & Privacy, *vol. 13, no. 6, 2015.*

Editor: Nayeem Islam ■ Qualcomm ■ nayeem.islam@gmail.com

Concurrency in Mobile Browser Engines

Călin Caşcaval, Pablo Montesinos Ortego, Behnam Robatmili, and Darío Suárez Gracia, Qualcomm Research Silicon Valley

Web browsers are our main window into the wealth of information available on the Internet. All consumer computing platforms, including smartphones and tablets, rely on a browser to provide news, entertainment, and services. We use the term *Web apps* to refer to applications designed and implemented using Web technologies. Some Web apps require users to launch their Web browsers, while others appear to the user as native applications, even though they are just an API layer on top of a browser engine. Using Web technologies as the application back end is a convenient way of building portable applications across a variety of platforms.

However, this presents two main challenges. First, browsers must provide a smooth user experience—fast page load, satisfactory scroll and zoom performance, and uniform behavior regardless of the underlying hardware. The browsers' JavaScript engines thus must provide close-to-native application performance. The second challenge is that when running on mobile devices, browsers must adapt to the related energy and connectivity constraints.

As a result, browsers have been evolving to exploit the underlying hardware. Most current smartphones and tablets have systems on a chip (SoCs), with two to eight cores and powerful GPUs, and they rely on a plethora of techniques to maximize the performance/power ratio. Such techniques include power and clock gating, dynamic voltage and frequency scaling, and offloading work to specialized cores. On the network side, Long-Term Evolution (LTE) offers 100 Mbps bandwidth, yet network latency continues to be high. Web browsers must exploit all available capabilities to address performance and energy challenges.

Here, we focus in particular on how Web browsers can use concurrency to improve per-tab (or per-page) processing. We use the Zoomm browser engine[1] and its MuscalietJS JavaScript engine[2] to illustrate how parallel processing improves performance and hides network latency for faster page loads.

EXPLOITING CONCURRENCY

Desktop browsers, such as WebKit (www.webkit.org) and Firefox (www.mozilla.org/firefox), typically exploit multiple cores by running each tab as a separate collection of processes and relying on the OS scheduler to place processes on different cores. The Zoomm browser architecture was designed with a different goal: take advantage of multicore processing for each browser tab. This is in line with typical mobile device usage, and it lets a more constrained platform meet its performance and energy goals.

A Parallel Browser Architecture

A Web browser has several major components: parsers (HTML, CSS, JavaScript) that create the Document Object Model (DOM), a Cascading Style Sheets (CSS) engine to format and style the DOM, a layout engine to produce the image that will be displayed to the user, a rendering engine to display the page, and a JavaScript engine to enable interactivity and dynamic behavior.

Figure 1 shows the breakdown of execution time by component, excluding the network time. Our measurements, similar to other work,[3] show that the network time is 30–50 percent of the total execution time. As the Web evolves, we're seeing remarkable changes in complexity and dynamic behavior. For example, in 2010, Leo Meyerovich and Rastislav Bodík measured WebKit execution and observed that JavaScript took approximately 5 percent of the execution time.[4] One year later, the fraction of JavaScript execution increased to 30 percent, and for most webpages, it has since plateaued.

Even more significantly, we're observing a major trend to support application development using Web technologies such as HTML5, CSS, and JavaScript. Given this breakdown of computation, it is clear that to optimize the browser execution using concurrent processing, all major components must be addressed, because the gains from optimizing the components in isolation are bounded.

Our goal is to exploit concurrency at multiple levels: parallel algorithms

 Published by the IEEE Computer Society 2376-113X/16/$33.00 © 2016 IEEE

for individual passes to speed up the processing of each component, and overlapping of passes to speed up the total execution time. In addition, we must respect the HTML and JavaScript semantics, even during concurrent execution. The main data structure used by all browser passes is the DOM. The DOM is a tree representing all HTML elements, including their content, relationships, styles, and positions. Web programmers use JavaScript to manipulate the DOM, producing interactive webpages and Web apps. Most communication between browser passes and components happens through the DOM. Unfortunately, even in a concurrent browser, access to the DOM tree (constructed by the HTML5 parser) must be serialized to conform to the HTML5 specification (see http://whatwg.org/html).

This is the biggest limitation Zoomm must contend with, and it significantly influenced the design. In our architecture, we manage access to the DOM through a dispatcher. Most passes have their own private concurrent data structures to allow for greater parallelism inside components, and they send asynchronous DOM updates to the dispatcher for processing. Figure 2 shows the architecture's high-level components, discussed in more detail next.

Zoomm Browser Components

The Zoomm browser consists of a number of loosely coupled subsystems, all of which were designed with concurrency in mind. With the exception of the browser global resource manager and the rendering engine, all subsystems are instantiated once for each page (shown as a separate tab in the user interface).

Resource manager. The resource manager is responsible for managing and preprocessing all network resources, including fetching resources from the network, providing cache management for fetched resources, and notifying other browser components when data from the network arrives.

In our first implementation, all resources are fetched in the order in which they appear, without imposing any priorities. In addition, the resource manager includes other components, such as the HTML prescanner and image decoder. The HTML prescanner quickly determines all external resources in an HTML document, requests their downloading, and, depending on the type of resources, requests further processing. The image decoder component consists of a thread pool that decodes images for later use as the resource manager receives them. These operations are fully concurrent, because each image decode is an independent task.

DOM engine. In Zoomm, each page (tab) instantiates a DOM engine that consists of the DOM dispatcher, HTML parser, CSS parsing and styling, and timers and events. The DOM dispatcher thread schedules DOM updates and serves as the page event loop. It serializes access to the DOM and manages the interaction between components.

The rest of the browser infrastructure dispatches work items to the concurrent DOM dispatcher queue, and the items are then handled one at a time. Work items represent browser passes as well as events from timers and the user interface. The HTML parser receives incoming (partial) data chunks for an HTML document via a DOM dispatcher work item and constructs the DOM tree by executing the HTML5 parsing algorithm. The parser adds external resources (referenced from the HTML document) to the resource manager's fetch queue. The parser also initiates the execution of JavaScript code by calling the JavaScript engine at appropriate times during parsing. The CSS engine calculates the look and feel of the DOM elements for the later layout and rendering stages. Similar to image decoding, the resource manager hands off CSS stylesheets to the CSS engine for parsing and for discovering new resources to request.

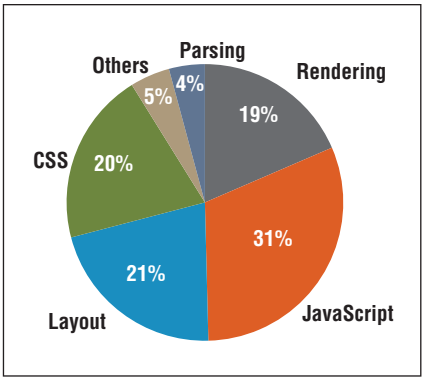


Figure 1. Browser processing times by component, excluding network load time. Profiling results obtained using the WebKit browser on a four-way ARM Cortex-A9 processor. Results are an aggregate of the top Alexa 30 sites as of March 2010.

Rendering engine. Whenever the DOM or CSS stylesheets change—because the fetcher delivered new resources, the HTML parser updated the DOM, or as a result of JavaScript computations—this change needs to be reflected on the screen so that the user can view and interact with it. The layout engine is responsible for transforming the styled DOM tree into geometry and content, which the rendering engine can turn into a bitmap. Ultimately, this bitmap is displayed on the screen by the user interface as a viewable webpage. Normally, the layout and rendering engine takes a snapshot of the DOM information it needs and performs the rest of the work asynchronously; however, it can also be invoked synchronously when JavaScript use APIs that query layout information.

JavaScript engine. The Zoomm employs a novel JavaScript engine, MuscalietJS, for executing all JavaScript code. The engine's design is presented in detail elsewhere (http://github.com/mcjs/mcjs.git).[2] In particular, our engine exploits concurrency by compiling multiple scripts in parallel, as well as compiling scripts asynchronously with the rest of the browser passes.

SMARTPHONES

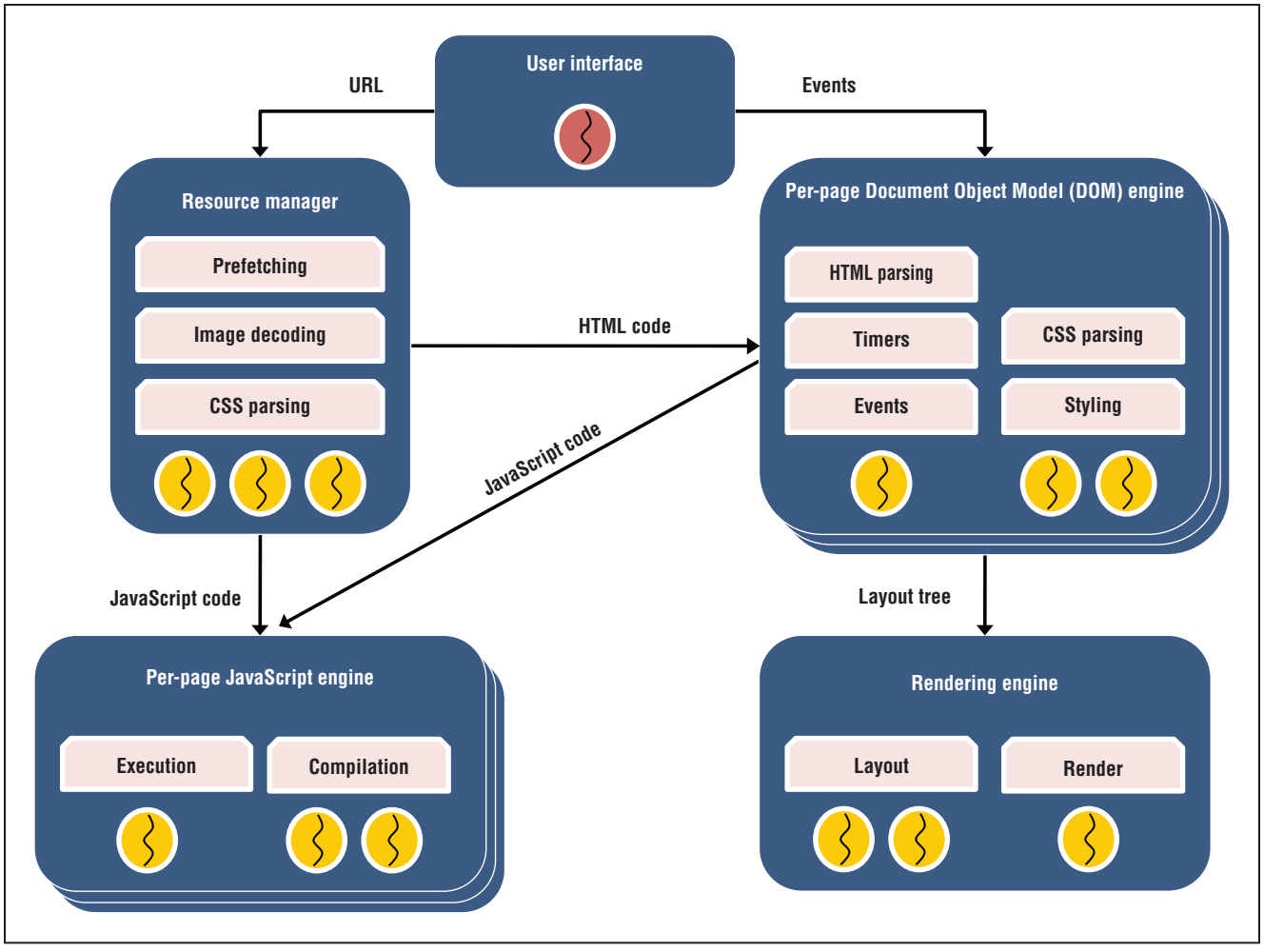


Figure 2. The Zoomm browser architecture. Concurrency is exploited both across components and within each component.

To achieve this, the JavaScript engine uses a thread pool and the just-in-time compiler uses a separate state stored in the metadata of each script. Due to JavaScript semantics, the execution of scripts is performed sequentially in the main engine thread. When the HTML parser or DOM dispatcher (for example, for user interface events) requests the execution of a JavaScript script that has not been compiled already, compilation is initiated. In either case, the engine waits for the compiled result and then executes the script. The goal of the engine is to use available resources on the platform to improve the generated code for JavaScript execution.

Similar to other modern JavaScript engines, MuscalietJS is a multitier execution engine. When the number of times a function has been executed exceeds a certain threshold (in other words, it's "hot"), the engine will promote the function and recompile it at a higher optimization tier. Different tiers include an interpreter, a baseline compiler, and a full compiler. The baseline compiler generates suboptimal code quickly. The full compiler, on the other hand, generates more optimized code for hot functions by performing adaptive JavaScript-specific optimizations, including hidden classes, property lookup, type specialization, and restricted dataflow analysis.

User interface. The Zoomm browser is implemented in platform-agnostic C++. For concurrency, we use a custom asynchronous task library (Qualcomm Multicore Asynchronous Runtime Environment; http://developer.qualcomm.com/mare), optimized for mobile execution. On Android, a thin Java wrapper is used to create the user interface. User interactions, such as touching a link on the display, are translated into Java Native Interface method calls, which ultimately create work items in the DOM dispatcher. Drawing to the display is performed using the Android Native Development Kit, which provides direct access to Android bitmaps. On Linux and Mac OS X, a similar wrapper is implemented in C++ using the Qt interface toolkit (www.qt.io/developers). Although our deployment

TABLE 1
Combined HTML and CSS prefetching initiates the download of most external resources ahead of their discovery by the HTML and CSS parsers with high accuracy ("correct prefetch") and small error ("missed/mistaken prefetch"). "Total resources" denotes the number of referenced resources in a webpage.

	Correct prefetch						Mistaken prefetch					
	HTML		CSS		Missed prefetch		HTML		CSS		Total resources	
Website*	Files	Bytes	Files	Bytes	Files	Bytes	Files	Bytes	Files	Bytes	Files	Bytes
cnn.com	34	979,695	52	409,377	2	372	0	0	5	3,371	93	1,392,815
bbc.co.uk/news	54	610,479	24	407,819	16	468,371	0	0	1	1,277	95	1,487,946
yahoo.com	44	672,595	13	264,603	2	2,016	1	0	0	0	60	939,214
guardian.co.uk	49	1,018,738	14	92,997	7	102,087	1	0	3	11,305	74	1,225,127
nytimes.com	73	1,046,636	9	73,487	13	228,162	1	10,837	1	89	97	1,359,211
engadget.com	128	2,023,135	84	651,030	5	104,320	0	0	9	34,824	226	2,813,309
qq.com	45	485,264	22	167,078	7	39,361	0	0	0	0	74	691,703

*The websites are from the Vellamo benchmark.

targets are Android devices, the Qt implementation allows much easier debugging and testing on desktop-based machines, and the ability to evaluate concurrency beyond what Android devices currently offer.

PARALLEL EXECUTION FOR RESOURCE PREFETCHING

Mobile devices commonly experience high latency when requesting the resources that form an HTML document. To reduce the overall time taken to load a page, fetching all of the dependencies from the network as early as possible is very important.

HTML Prescanning

Due to idiosyncrasies in the HTML5 specification, the HTML5 parser must wait for <script> blocks to finish executing before it can continue parsing. So, if a webpage references an external resource after a script element, fetching the resource can't be overlapped with the waiting. This could delay the completion of page loading.

The Mozilla Firefox browser mitigates such situations by speculatively parsing ahead of script blocks to discover new resources. (It might then be forced to throw away some of that work if, for example, JavaScript inserts new content into the DOM tree via the document.write() API.) Once resources are discovered, network latency can be masked by requesting multiple resources to be fetched in parallel. This strategy also helps use all available bandwidth, and it reduces the overall time spent waiting for resources to arrive.

In Zoomm, we favor concurrency to achieve the same goal by running an HTML *prescanning* component in parallel with a (nonspeculative) HTML parser. The main objective of the HTML prescanner is to quickly determine all external resources in an HTML document and trigger their fetching from the network. The most commonly referenced resources are images, CSS stylesheets, and JavaScript sources. In addition, stylesheets and JavaScript sources can themselves reference further external resources. Furthermore, the prescanner obtains all id, class, and style attributes used in the document.

As network packets of an HTML document arrive, they are given to the prescanner and the actual HTML parser independently. The prescanner can run ahead of the HTML parser because it only has to approximately parse HTML to find resources, thus skipping the complex DOM tree construction phase. More importantly, the prescanner doesn't have to wait for the execution of <script> blocks to finish.

The processing of prefetched resources works as follows. Images are fetched concurrently with the rest of the page processing. Once downloaded, image data is given to a thread pool for decoding concurrently. The decoded image is added to the DOM dispatcher queue, which updates the corresponding img tree node. Then the image is removed from the set of pending images.

CSS Prefetching

CSS stylesheets are dispatched to a thread pool responsible for parsing CSS concurrently. If a CSS rule contains additional external resources, the parser decides whether to initiate prefetching for them, based on the likelihood that they're actually referenced in the HTML document.

It's crucial to download just enough of the referenced resources. Downloading too little means that new resources are discovered only when styling the DOM tree later on, which incurs additional latency penalties. It's common practice among websites to reference many more resources than are actually needed for any given document—for example, by using a site-wide common style file. Downloading

SMARTPHONES

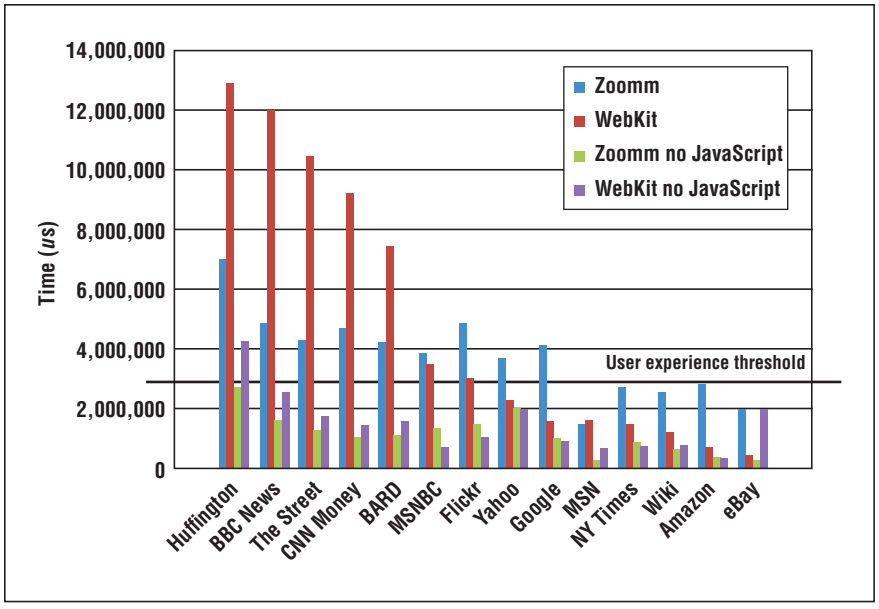


Figure 3. Page load time for several popular sites. Note that users typically expect pages to load in less than 3 seconds.

all resources invariably consumes too much bandwidth and slows down page loading.

In Zoomm, the CSS parser employs the `id` and `class` attributes discovered by the HTML prescanner to determine if a rule is likely to be matched. If all attribute values referenced in a CSS rule selector have been seen by the HTML prescanner, we assume that the rule will match at least one DOM tree element and initiate downloading its resources. This heuristic is simple but effective (see Table 1). Note that wrong decisions don't affect correctness; any missed resources will be discovered during the styling phase, at the cost of additional latency.

Table 1 shows the number of resources that are successfully requested by the prefetching stage, and the number of resources are missed due to use of JavaScript. Note that resources would also count as "missed" if the prefetching algorithms would fall behind the actual HTML and CSS parsers. However, this was never the case in all our experiments. The prefetching components were always fast enough to finish much earlier than the parsers.

Despite the heuristic nature of some of the prefetching decisions, they're quite accurate. In our experiments, 80–95 percent of all externally referenced resources in a document were prefetched correctly, with only a small error rate. Due to bandwidth and power considerations, our heuristics were still conservative—that is, they tend to prefetch too little rather than too much. The "missed prefetch" (not prefetched, but needed for rendering the webpage) numbers were higher than "mistaken prefetch" (prefetched, but not needed for rendering) numbers.

JAVASCRIPT PARALLEL PROCESSING

In modern pages, a significant number of resources (style sheets, images, and other scripts) are dynamically constructed using JavaScript. It's advantageous to discover these resources ahead of time, such that their download doesn't block the page load. HTML5 introduces two attributes for scripts: *async* and *defer* to allow out-of-order processing of scripts. When the HTML parser encounters one of these attributes, it can farm out its compilation and execution to the JavaScript engine immediately. MuscalietJS takes advantage of the asynchronous semantics and compiles and executes these scripts in parallel.

Another technique for exploiting multicore processing for JavaScript is parallel compilation. Almost all current browsers use parallel compilation to either compile multiple scripts concurrently or run an enhanced compiler in a separate thread.[5–7]

Overall, using these parallelization techniques, Zoomm loads pages about twice as fast as WebKit, as shown in Figure 3.

Exploiting parallelism in browsers promises performance and power savings. We believe that Zoomm is just a first step in that direction, and hiding network latency using ahead-of-time processing removes a bottleneck in loading webpages that is beyond the control of browser clients, thus improving the user experience. Optimizations explored in Zoomm and the MuscalietJS engine are being adopted by commercial browsers: the Mozilla Servo project (https://github.com/servo/servo) is using a parallel language (RUST) to implement a concurrent browser architecture similar to Zoomm's. That project puts a larger emphasis on the layout engine to handle all the corner cases of the HTML5 specification, which presents a significant challenge and opportunity.

Browsers such as Chrome and Internet Explorer are implementing parallel JavaScript processing, and recently Chrome has decoupled JavaScript parsing into a concurrent thread.[5] Other researchers are looking at architectural aspects of enabling more concurrency in the browser.

Finally, Web standards are evolving to allow webpage designers to exploit

This article originally appeared in
IEEE Pervasive Computing, *vol. 14, no. 3, 2015.*

concurrency. These include asynchronous and deferred script processing directives in HTML, Web workers, and several efforts to express concurrency in JavaScript. In addition, the declarative nature of CSS makes it ripe for exploiting parallelism through concurrent implementations.

ACKNOWLEDGMENTS

We thank Nayeem Islam and the Qualcomm Research Executive team for the opportunity to build the Zoomm and MuscalietJS engines. We thank Mehrdad Reshadi, Michael Weber, Wayne Piekarski, Seth Fowler, Vrajesh Bhavsar, Alex Shye, and Madhukar Kedlaya for their contributions.

REFERENCES

1. C. Caşcaval et al., "ZOOMM: A Parallel Web Browser Engine for Multicore Mobile Devices," *Proc. 18th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming* (PPoPP), 2013, pp. 271–280.
2. B. Robatmili et al., "MuscalietJS: Rethinking Layered Dynamic Web Runtimes," *Proc. 10th ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments* (VEE), 2014, pp. 77–88.
3. Z. Wang et al., "Why Are Web Browsers Slow on Smartphones?" *Proc. ACM Int'l Workshop on Mobile Computing Systems and Applications*, 2011, pp. 91–96.
4. L.A. Meyerovich and R. Bodík, "Fast and Parallel Webpage Layout," *Proc. Int'l Conf. World Wide Web*, 2010, pp. 711–720.
5. M. Hölttä and D. Vogelheim, "New JavaScript Techniques for Rapid Page Loads," blog, 18 Mar. 2015; http://blog.chromium.org/2015/03/new-javascript-techniques-for-rapid.html.
6. J.-D. Dalton, G. Seth, and L. Lafreniere, "Announcing Key Advances to Javascript Performance in Windows 10 Technical Preview," blog, Oct. 2014; http://blogs.msdn.com/b/ie/archive/2014/10/09/announcing-key-advances-to-javascript-performance-in-windows-10-technical-preview.aspx.
7. J. Ha et al., "A Concurrent Trace-Based Just-in-Time Compiler for Single-Threaded JavaScript," *Workshop on Parallel Execution of Sequential Programs on Multi-Core Architectures* (PESPMA), 2009, pp. 47–54.

Călin Caşcaval is a senior director at Qualcomm Research Silicon Valley. Contact him at cascaval@qti.qualcomm.com.

Pablo Montesinos Ortego is a senior staff engineer/manager at Qualcomm Research Silicon Valley. Contact him at pablom@qti.qualcomm.com.

Behnam Robatmili is a staff research engineer at Qualcomm Research Silicon Valley. Contact him at behnamr@qti.qualcomm.com.

Darío Suárez Gracia is a staff engineer at Qualcomm Research Silicon Valley. Contact him at dgracia@qti.qualcomm.com.

Tracking Cows Wirelessly

Greg Byrd, North Carolina State University

A student team from NC State designed and built a prototype wireless network to monitor the milking and weighing of cows.

To successfully operate any farm, effective livestock management is crucial. Efficient, affordable, and scalable livestock management solutions play an increasingly important role in modern farming, as the number of dairy farms in the US decreases, but the number of dairy cows on each increases. Dairy cows require careful monitoring for milking, weighing, and other activities, so the ability to reliably track these animals in large numbers is particularly important.

Dairy cows are typically identified by visible ear tags. Although tags with embedded RFID devices have been available—allowing them to be scanned electronically—because of cost, most tags use low-frequency (LF) RFID, so the scanner must be within a few inches of the tag. Consequently, farmworkers need to be "up close and personal" with each and every cow for reliable scanning.

Although RFID tagging of cattle has been widely adopted in Europe, US dairy farms are more reluctant to do so because of costs and the lack of national standards. To address the cost and convenience factors, a team of students in the Department of Electrical and Computer Engineering (ECE) at North Carolina State University (Figure 2) designed and built a prototype wireless network that combines long-range ultra-high-frequency (UHF) RFID tags with low-cost wireless and computing components. The long-range RFID allows unmanned scans of multiple tags, and the wireless network provides scalable data collection without costly infrastructure.

NETWORK OVERVIEW

Figure 1 shows an overview of the prototype network using the ZigBee wireless protocol to communicate. The RFID reader is connected to the ZigBee wireless networking node, so when a new RFID tag is detected, the ZigBee node sends a data packet to the controller node. As cows enter and exit the milking station, an RFID reader identifies the cows by their long-range RFID tags, and when the RFID tag is no longer in range, the ZigBee node sends a data packet indicating that the cow left the milking station. (Multiple cows may enter the milking stalls at the same time, so the prototype system can monitor the entry and exit times of up to eight cows simultaneously.)

As the cows leave the milking station, they pass through a weighing station. A floor scale and an RFID

 Published by the IEEE Computer Society

reader are attached to a microprocessor. The cow's weight and identity are passed to a ZigBee node, which sends this information to the controller node, collects the milking and weighing information and the amount of time each cow spends in the milking station, displays it on a user interface, and records it in a spreadsheet.

TAGS, READERS, AND SENSORS

The passive UHF RFID tag is the enabling technology for this project. In a passive RFID system, the reader antenna sends a radio signal at a particular frequency. The RFID tag (also called an RFID transponder) contains an antenna tuned to the same frequency and a microchip. The received signal powers the chip, which modulates a signal that is transmitted back to the reader, and the reader then translates the modulated signal into digital data.

Developed by RFID Sensor Systems, the project's sponsor, the prototype tags have been demonstrated to work from distances of up to 150 feet (45 meters). RFID tags measure approximately 6 inches by 0.75 inches and are mounted to the back of a standard plastic ear tag.

The RFID reader is SkyeTek's Nova module, which includes an ARM Cortex CPU and a UHF transceiver. SkyeTek Protocol software runs on the microprocessor, providing an API for standard RFID-read operations. A serial communications link connects the SkyeTek module to the ZigBee node, which is described in the next section.

For the milking station, the prototype system includes a separate short-range antenna for each stall; multiple antennae can be multiplexed into a single reader. Certain techniques can use a single antenna to distinguish signals from multiple tags, which would be an interesting benefit of the long-range tags provided by RFID

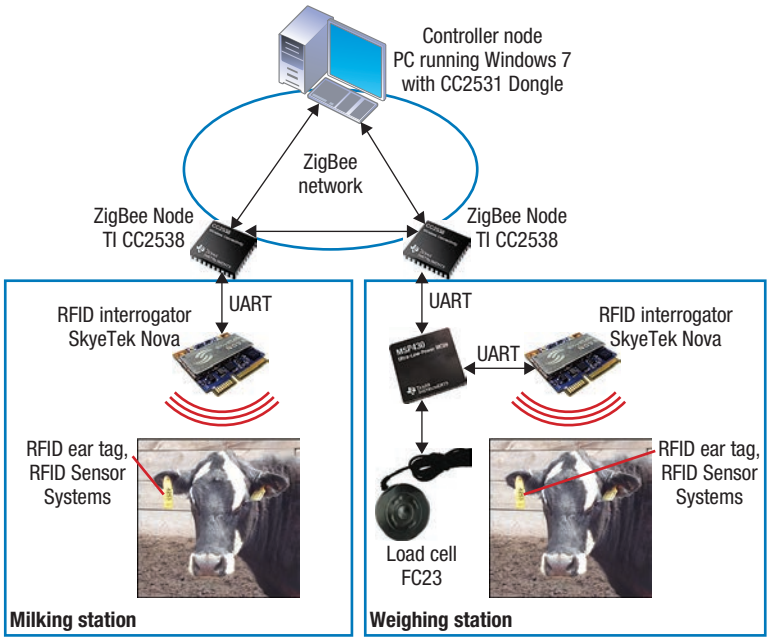


Figure 1. Overview of a prototype wireless network for tracking dairy cows. RFID tags indicate when a cow enters and leaves the milking or weighing station. Timestamps and weight are delivered to the controller node via the wireless mesh network. UART, universal asynchronous receiver/transmitter.

Figure 2. The Department of Electrical and Computer Engineering (ECE) at North Carolina State University design team. Left to right: Bryan Campbell, André Ramos, Youn Chu, and Anthony Laws.

PROJECT DETAILS

- Name: Howling Cow Wireless Network
- School: North Carolina State University
- Department: Electrical and Computer Engineering (ECE)
- Student Participants: Bryan Campbell, Youn Chu, Anthony Laws, André Ramos
- Faculty Mentor: Dr. Rachana Gupta
- Sponsors: Dr. William Carr, RFID Sensor Systems
- Special Thanks: Jim Carlson (ECE), Dr. Jake Adams (ECE), Dr. Dan Poole (Animal Science)

2015 IEEE/IBM WATSON STUDENT SHOWCASE

IEEE and IBM are in search of creative, forward-thinking students to participate in an exciting team-based showcase. Do you want an opportunity to work with IBM's Watson to develop an innovative app? Do you want to develop your cognitive computing skills while earning a chance to win cash prizes? Winning entries will also be featured in this column. The deadline for submitting completed entries has been extended to 30 September 2015, but register your intent to participate now at http://goo.gl/52WUIh so you don't miss this opportunity.

Sensor Systems but was beyond the scope of this project.

For the weighing station, both an RFID reader and a weight (load) sensor are needed. The load sensor is incorporated into a floor-mounted plate and connected to a low-power Texas Instruments MSP430 microprocessor. When the RFID reader detects a cow entering the weighing area, it sends a signal to the microprocessor via the serial port. The microprocessor uses an integrated analog-to-digital converter to read the weight from the load sensor. Both the identification and the weight are sent over a serial connection to a ZigBee node, which sends the appropriate data packet to the controller node.

Both the MSP430 microprocessor and SkyeModule Nova utilize low-power processing cores that have special sleep modes when the CPU is inactive. In addition, the MSP430 FR5738 microprocessor includes 16 Kbytes of integrated ferroelectric nonvolatile RAM (FRAM), which offers lower energy consumption than flash memory for storing instructions and constant data. Although milking station components are likely to have access to electricity, the weighing station and other future sensor stations may require solar power or other energy harvesting and storage solutions.

WIRELESS RADIO

ZigBee is a collection of protocols designed for low-power wireless networks, such as smart homes and sensor networks. The physical layer is based on the IEEE 802.15.4 standard for low-rate wireless networking. Upper-layer protocols add routing and other features to create a mesh network so that a device can relay information through multiple ZigBee nodes to a remote destination node. Other protocol features include security, device discovery, and messaging.

The ZigBee node chosen for this prototype is the Texas Instruments CC2538 system on chip. The chip includes an ARM Cortex-M3 processor, a 2.4-GHz radio transceiver, embedded RAM and flash memories, and a hardware cryptographic accelerator. Like the MSP430 and SkyeModule Nova, the CC2538 features low-power operating modes.

ZigBee is a good match for this application because only a small amount of data is sent per event. The latency also isn't critical. What's needed is a reliable, low-power, inexpensive, and secure network that's scalable to larger farms with multiple milking and sensor stations. An extensive ZigBee mesh network can be deployed, with transmission distances of up to a mile long for a single link.

CONTROLLER NODE SOFTWARE

The controller node is a standard laptop or desktop system equipped with a ZigBee network node. The choice for this prototype is the Texas Instruments CC2531 USB Dongle. It includes an 8051 microcontroller, which runs packet-sniffing software and delivers packets to the host through the USB interface.

As described above, the controller receives data packets from the milking and weighing station nodes. A Java application displays the information for each event (such as when a cow enters or leaves a station) on a user interface window. Events are also recorded in a spreadsheet-compatible file for postprocessing.

DEPLOYMENT ENVIRONMENT

The target deployment environment for the wireless network is the dairy

This article originally appeared in Computer, *vol. 48, no. 6, 2015.*

research and teaching farm at NC State. The farm spans 389 acres, supports a herd of 300 cows, and includes a 20-stall milking station, visitor's center, classroom, and museum. The farm's milk is used in Howling Cow dairy products—including ice cream, milk, and heavy cream—and is processed at the Feldmeier Dairy Processing Lab, which is operated by the Department of Food, Bioprocessing, and Nutrition Science. Howling Cow products are sold at various locations on campus.

The project plan also included an on-the-farm demonstration. Unfortunately, logistical challenges associated with working with live animals have delayed the demonstration. Nonetheless, the components were successfully tested together in the lab.

Our goal was to create a low-cost proof-of-concept system. The retail cost of the system components is around $1,200. With additional development and higher volumes, the cost can be further reduced. For example, the long-range RFID tags would allow multiple cows to be scanned at the milking station with a single reader and antenna. Also, the computational requirements for the load sensor, RFID reader, and ZigBee communication could be consolidated onto a single processor.

When deployed at the farm, researchers and students will use the Howling Cow network to demonstrate how RFID, wireless, and other information technologies can improve the efficiency and productivity of the farm as well as the health of the animals. The farm of the future will employ many emerging technologies: unmanned drones for crop and livestock monitoring (where the long-range RFID tags will be critical), self-driving tractors, 3D printing of replacement parts, and precision agriculture driven by data analytics. Pilot projects like this one will lead the way to show how high-tech can be affordable for farmers in North Carolina and around the world.

SUBMIT A PROJECT

As much as I enjoy writing about our students at North Carolina State University, I'm really interested in hearing about interesting student-led design projects in computer science and engineering everywhere. If you would like to see your project featured in this column, fill out the submission form at: www.computer.org/student-showcase.

GREG BYRD is associate head of the Department of Electrical and Computer Engineering at North Carolina State University. Contact him at gbyrd@computer.org.

A Cloud-Focused Mobile Forensics Methodology

Quang Do
University of South Australia

Ben Martini
University of South Australia

Kim-Kwang Raymond Choo
University of South Australia

THE MODERN SMARTPHONE HAS BECOME THE PRIMARY COMPUTING DEVICE FOR MANY PEOPLE. These devices are used to perform phone-specific tasks, such as texting and making phone calls, as well as other tasks, such as Web browsing and Internet banking, once relegated to their far less portable cousins: desktop and laptop computers. As technology progresses, smartphones will only increase in functionality and adoption rates. In fact, in 2014, more than 1.2 billion smartphones were sold worldwide, with a little over 80 percent of these smartphones running the Android operating system.[1] With the rapid growth of inexpensive smartphones from Chinese vendors, the number of devices sold, and the number of users harnessing the power of the smartphone, will only increase.

Chances are most criminals will have, and heavily utilize, smartphones during the course of their illicit activities. Nielsen found that the average smartphone user spends 30 hours on more than 25 apps each month.[2] Each app might have a specific purpose (for example, cloud storage, communication, or photography) or a general purpose (such as Web browsing), and each app stores its data in specific locations. Because these apps store and transmit sensitive data, along with the innate communicative capabilities of smartphones, *mobile forensics*—the collection of evidential data from a mobile device—has become an important part of many criminal investigations.

Because the number of mobile devices (with different models, makes, and firmware) is immense, and these devices might be running different mobile operating systems, it's infeasible for a forensic practitioner to be familiar with every device. Mobile forensic toolkits that can analyze the attached device, recover digital evidence, and present it in a human-readable way have become the contemporary solution to this challenge. The main drawback is that a forensic practitioner who relies primarily on general-purpose mobile forensic toolkits might find that the product can't obtain all of the relevant evidential data from an offender's smartphone. Furthermore, with the popularity of cloud-based apps, much of this evidential information might never have been present on the device. Thus, there's a need for a forensically sound process that can obtain all evidential data from a smartphone, as well as analyze this data for authentication credentials and other details to obtain cloud-based evidential data.

Snapshot of Existing Mobile Forensic Techniques

Current data collection methodologies for Android devices rely heavily on either flashing an existing

 Published by the IEEE Computer Society 2376-113X/16/$33.00

partition on the device or exploiting the device to obtain operating system root privileges. Timothy Vidas and his colleagues proposed using customized recovery images containing forensic tools, which are flashed over the Android device's "recovery" partition.[3] This gives forensic practitioners access to the device, letting them extract all the data. The major downside to this method is that flashing any image on a modern Android device requires signing the image with the vendor's key. To flash images signed with other keys, the bootloader must be unlocked. Unlocking the bootloader triggers a wipe of the data on the device. In a similar vein, Namheun Son and his colleagues used custom images flashed onto the device's "boot" partition, which allowed the researchers to obtain most of the data on the device.[4] Jeff Lessard and Gary Kessler described a process for collecting a bit-for-bit copy of a device's NAND flash storage that required the device be rooted and have a secure digital (SD) memory card.[5] Sheng-Wen Chen and his colleagues also used an SD memory card in their data acquisition process.[6] They loaded the SD memory card with their own patch for the phone's recovery mode to install.

This approach has two major problems. It requires that the phone be running a third-party recovery that allows for the installation of nonvendor signed patches, and it requires that the device accept an SD memory card. With flagship phones now abandoning SD memory card storage in favor of greater on-board storage, and popular phones once known for allowing additional storage (such as the Samsung Galaxy S6) following suit, SD memory card–based data collection techniques will likely soon become obsolete.

Another risk is that if flashing an existing partition (for example, the boot or recovery partitions) becomes common in forensic procedures, suspects might begin to hide sensitive data in these partitions. When talking about a forensically sound process, rooting a device presents an even greater problem: the vast majority of root exploits are released as closed-source and heavily obfuscated packages, one justification being that smartphone vendors might patch out these exploits. The exploit would have unrestricted access to the exploited device's storage and be capable of performing any number of destructive or incriminating tasks. In a forensic investigation using undocumented root exploits, the evidence could be determined to be tainted or even inadmissible in a court of law. Based on these factors and our review of the literature,[7] we noted a need for a forensically sound methodology to collect cloud-based evidential data from Android devices.

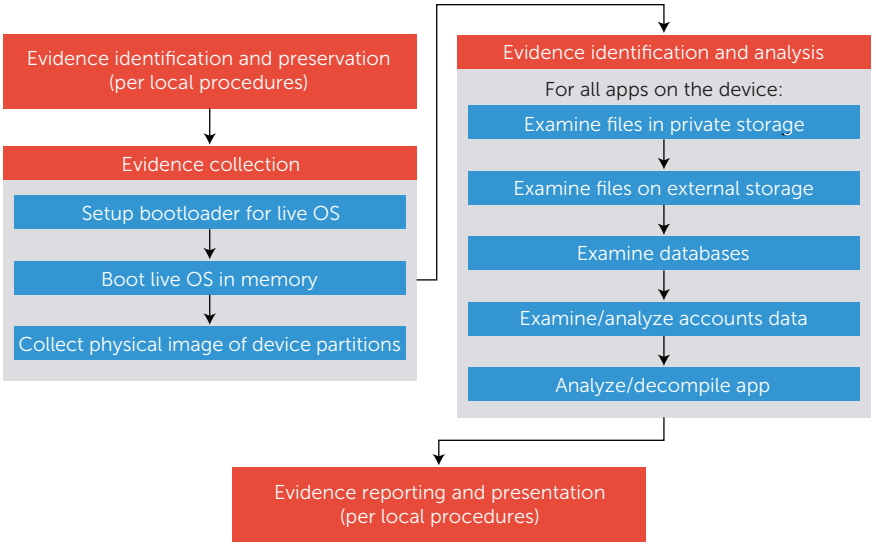


FIGURE 1. An evidence collection and analysis methodology for Android devices (adapted from earlier work[7]). The physical image of the evidential data, obtained via the evidence collection process, is collected via a live OS and then analyzed externally in order to preserve forensic integrity.

A Cloud-Focused Mobile Forensics Methodology

Figure 1 presents our evidence collection and analysis methodology for Android devices that allows for the retrieval and analysis of cloud-based evidential data.[7] The first major step after identifying and physically preserving any evidential devices (for example, Faraday bags and radio-suppressed environments) is evidence collection. This involves exploiting a flaw in the device's bootloader to allow for booting a live operating system. In our experiments, we undertook evidence collection and analysis on a Nexus 4 phone. We discovered a flaw wherein unlocking the bootloader and then booting our image into RAM without rebooting the device let us access all data on the device without the usual device wipe that typically occurs when the bootloader is unlocked. Because the custom image is loaded into the device's volatile RAM, the device's

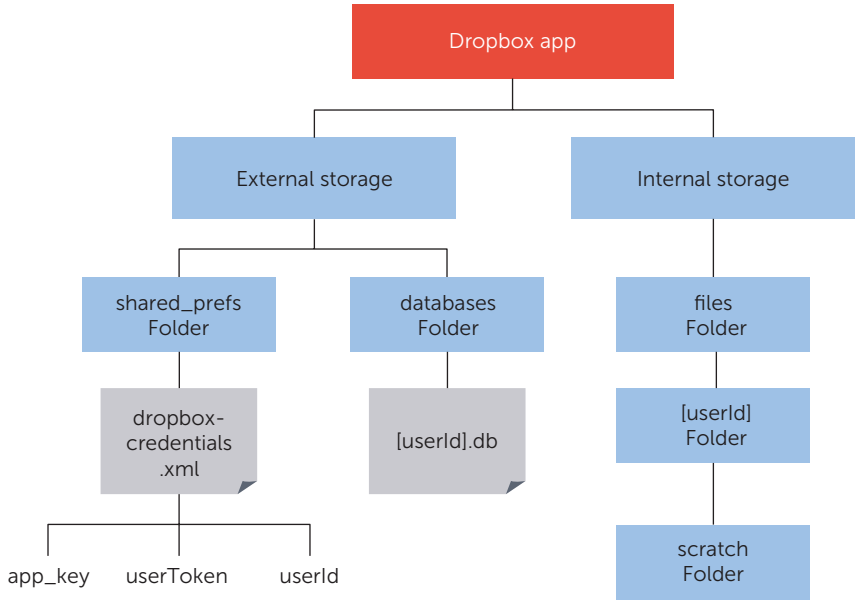


FIGURE 2. Locations where Dropbox stores items of interest. Dropbox primarily utilizes the device's external storage to store files downloaded by the user and the internal storage for user settings and metadata. All of these files are stored unencrypted.

internal partitions aren't modified, so it doesn't violate the principles of forensic soundness.

A physical image of the device partitions is then collected via the custom boot image and transferred to the forensic practitioner's PC. This bit-for-bit copy is then analyzed in the evidence examination and analysis stage. The practitioner must determine what apps are installed on the device, and which apps are of interest in the investigation. In our experiments, we focused on apps providing access to cloud-based services, which included Box, Dropbox, OneDrive, and OneNote.[8] We determined the locations where an app could store data of interest to a forensic practitioner: an app's private storage (internal storage), the device's external storage, and the phone's account data (using the Android AccountManager API). The app's private storage directory often contains user preferences and databases that include data such as OAuth access/refresh tokens and account information. Apps often stored cached copies of files retrieved from cloud-based services on the external storage.

We briefly describe some findings of our study of three of the cloud storage apps.[8]

Dropbox

Using our evidence collection and analysis methodology, we found that Dropbox stores a significant amount of information that might be of general forensic interest in the device's external and internal storage. On Android devices, internal storage refers to an app's private directory and external storage refers to a shared storage space that might be removable (such as an SD memory card) or internal (that is, nonremovable storage).

In the Dropbox app's private app storage directory, located on the device's internal storage, we located an XML configuration file and a SQLite format database of interest. Figure 2 shows these items in the context of their locations.

Dropbox stores three values of interest within its "dropbox-credentials.xml" shared preferences file. First is the app's consumer key, which is used in the header of the URL request to Dropbox's servers to authenticate the user. This is a static key that represents the app that's communicating with the server rather than the user. The "userToken" directive in this XML file contains the current user's OAuth token and half of a secret key used during authentication. In addition to these items of interest, this XML file also contains a numerical string that uniquely identifies the user. This unique identifier is known as the "userId" and represents the user on Android's file system and Dropbox's databases.

Dropbox's "[userId].db" database file contains a significant amount of interesting data. Inside this database is a table containing records for all files stored in the user's Dropbox, including the files' sizes, modification times, and filenames. Also stored within this database is information pertaining to the user's albums, camera uploads, pending uploads, photos uploaded, and cached thumbnails.

On the device's external storage, Dropbox stores a "scratch" folder (see Figure 2), which contains the local cache of files downloaded by the user through the Dropbox app. The files in this directory are unmodified and retain their original file names.

After analyzing the information obtained from the forensic copy, we moved onto the analysis of Dropbox's account information stored on the device. We noted that, because Dropbox stores the majority of its authentication data in its private directory on the internal stor-

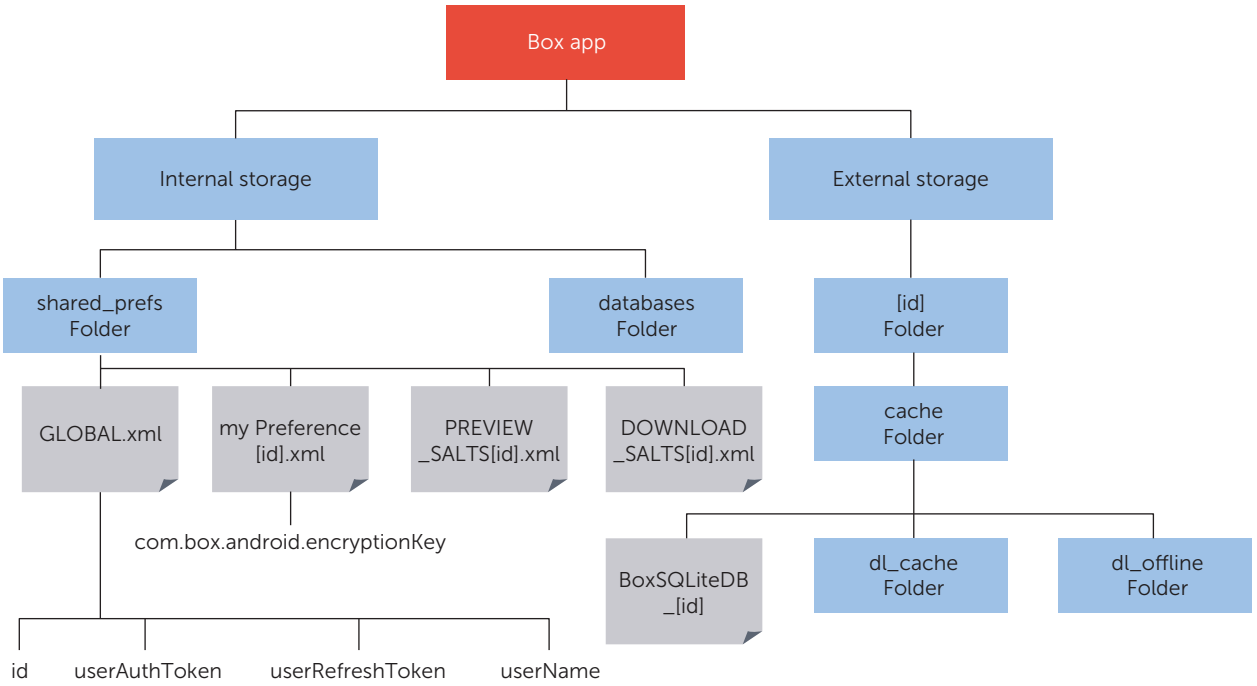


FIGURE 3. Locations where Box stores items of interest. Similar to Dropbox, Box also stores downloaded files on the device's external storage. Unlike Dropbox, however, Box encrypts these files and stores the encryption key in the internal storage.

age, it doesn't store interesting data in the device's accounts. Dropbox stores the user's email address in this location.

Based on our analysis and Dropbox's developer guide,[9] we observed that to authenticate as the device user on Dropbox's servers, we would require the OAuth consumer key, the user's OAuth token, and the OAuth signature. From the analysis of the data contained within the app's directories and device's accounts, we had already obtained all of this information, with the exception of half of the OAuth signature. Because we were unable to obtain this information from the device or Dropbox files, we determined that this string would most likely be statically defined.

Located within Dropbox's heavily obfuscated decompiled code was a function that was entirely self-contained and generated two strings: the OAuth consumer key and the missing half of the OAuth signature we required. Using the obtained information, a forensic practitioner could authenticate as the user and access their files.

Box

Box is a well-known file-syncing storage service often used by organizations. Box uses a device's internal and external storage to store its data, as Figure 3 illustrates.

On the device's internal storage, Box stores data of interest in the "shared_prefs" and "databases" directories. Within its shared preferences directory are four important XML files. "GLOBAL.xml" contains the user's unique numerical identifier (listed as "id" in the file), which the Box servers, Android file system, and Box's databases use to uniquely identify the user. In addition, this XML file contains the user's current access token, refresh token, and email address, which are all used for authentication. The "myPreference[id].xml" file contains, among other things, the 512-bit encryption key that Box uses to encrypt files stored on the device's external storage. Lastly, the "PREVIEW_SALTS[id].xml" file contains the salts of each encrypted preview file stored on the external storage, and the "DOWNLOAD_SALTS[id].xml" file contains the salts of each encrypted file that has been cached on the external storage.

The "BoxSQLiteDB_[id]" database file contains a table ("BoxEvent") listing all actions that have been performed by the app on the user's files. This includes copying a file from one location to another, previewing a file, sharing a file, creating a file, moving a file, and downloading and uploading

a file. Furthermore, this database contains records for each file created by the user (in the "BoxFile" table), each folder created by the user (in the "BoxFolder" table), and files the user has recently accessed (in the "BoxRecentFile" table).

Box uses the device's external storage to store a cache of previews and downloaded files. It stores each file previewed by the user in the "dl_cache" folder, and stores each file downloaded by the user in the "dl_offline" folder. The contents of these files are encrypted using "Box Crypto." To decrypt these files, the 512-bit encryption key obtained from Box's "GLOBAL.xml" file must be used with the file's salt (obtained from the respective SALTS XML file). Bouncy Castle's AES CBC cipher (using PKCS5Padding) is used as Box's encryption cipher. Because Box doesn't use the AccountManager service, it doesn't store any data within the device's accounts.

Further memory-based analysis determined that authenticating as the user to Box's servers only requires a valid access token. Because Box's access tokens expire after 60 minutes, to practically authenticate as the user, we would need to be able to generate new access tokens. We therefore also require a valid refresh token. Box's refresh tokens expire after 60 days, so it's much more likely that a forensic practitioner could obtain a valid refresh token. Obtaining a new access token requires a valid refresh token, the client ID, and the client secret.

From the analysis of the internal storage, we already had a refresh token from the "GLOBAL.xml" shared preferences file. Because the client ID and secret weren't on the device, we concluded that they must be defined statically. We found that these strings were stored within the app's strings resources file. With this information, we believe it would be possible for a forensic practitioner to generate a new valid access token and obtain all of the user's files, given that 60 days hadn't yet passed since the refresh token was generated by Box's servers.

OneDrive

Microsoft OneDrive is another popular file-syncing storage app that also stores a significant amount of data of interest. Within its private app directory, we found several SQLite databases. One of these databases ("metadata") contains information pertaining to each of the user's OneDrive files, including filenames, sizes, and the URL used to download the file (which requires authentication). Another SQLite database file ("cached_files_md.db") contains metadata relating to the files the user has accessed and the OneDrive app has cached.

On the device's external storage, we located several items. For example, the "cache" folder within OneDrive's external storage data path contained a cache of each of the files downloaded by the user. These files were unmodified from the original file stored on the OneDrive servers (as evidenced by the identical hashes for the original and files stored in this directory) and were named with the following convention: "SkyDriveCacheFile_[item's ID].cachedata," with the item's ID being the ID within the "cached_files_md.db" database.

Following this analysis of the OneDrive app, we obtained the data that OneDrive stores in its AccountManager account on the device. OneDrive stores a significant amount of information in this location. This likely explains the lack of authentication details stored in OneDrive's internal and external storage data locations. OneDrive stores, in the OneDrive AccountManager account on the device, a refresh token, an access token, a scope, an account type, the user's ID, and the access token's expiry timestamp.

A further analysis of the memory of the OneDrive app provided us with the URLs for user authentication and access token generation for the user. From our earlier analysis, we found that access tokens expire after 24 hours. This means that in a general forensic scenario, a forensic practitioner would need to generate a new access token. To generate a new access token, the valid refresh token, user ID, and scope would be required, and we were able to locate these items on the device.

PASSWORDS AND USERNAMES WERE ONCE THE DE FACTO STANDARD FOR USER AUTHENTICATION. Today, few services, especially cloud-based services, store usernames and passwords on devices, in an effort to enhance user security. Instead they store a time-limited token and/or a number of separate tokens. This makes it significantly more difficult for practitioners undertaking a forensic investigation due not only to the time-critical nature of these tokens, but also to the number of varied authentication implementations. Our evidence collection and analysis methodology aims to at least partially mitigate these issues and provide forensic practitioners with a clear and forensically sound method to obtain cloud data, both remote and physically present on the seized device. As more and more services become cloud-based, most evidential data might, in fact, be present only on remote servers.

References

1. Gartner, "Gartner Says Smartphone Sales Surpassed One Billion Units in 2014," Gartner press release, 2015; www.gartner.com/newsroom/id/2996817.
2. Nielsen, "Smartphones: So Many Apps, So Much Time," Newswire, 1

July 2014, www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html.
3. T. Vidas, C. Zhang, and N. Christin, "Toward a General Collection Methodology for Android Devices," *Digital Investigation*, vol. 8, supplement, 2011, pp. S14–S24.
4. N. Son et al., "A Study of User Data Integrity During Acquisition of Android Devices," *Digital Investigation*, vol. 10, supplement, 2013, pp. S3–S11.
5. J. Lessard and G. Kessler, "Android Forensics: Simplifying Cell Phone Examinations," *Digital Device Forensics J.*, vol. 4, no. 1, 2010, pp. 1–12.
6. S.-W. Chen, C.-H. Yang, and C.-T. Liu, "Design and Implementation of Live SD Acquisition Tool in Android Smart Phone," *Proc. 5th Int'l Conf. Genetic and Evolutionary Computing*, 2011, pp. 157–162.
7. B. Martini, Q. Do, and K.-K.R. Choo, "Conceptual Evidence Collection and Analysis Methodology for Android Devices," R. Ko and K.-K.R. Choo, eds., *Cloud Security Ecosystem*, Syngress, 2015, pp. 285–307.
8. B. Martini, Q. Do, and K.-K.R Choo, "Mobile Cloud Forensics: An Analysis of Seven Popular Android Apps," R. Ko and K.-K.R. Choo, eds., *Cloud Security Ecosystem*, Syngress, 2015, pp. 309–345.
9. K. Goundan, "Using OAuth 1.0 in the 'PLAINTEXT' Signature Method," blog, 13 July 2012; https://blogs.dropbox.com/developers/2012/07/using-oauth-1-0-with-the-plaintext-signature-method.

QUANG DO *is a PhD candidate in the Information Assurance Research Group at the University of South Australia. His research interests include Android user privacy, forensics, and security. Do has a bachelor of computer science (hons.) from the University of South Australia. Contact him at quang.do@mymail.unisa.edu.au.*

BEN MARTINI *is a research associate in the School of Information Technology and Mathematical Sciences at the University of South Australia. His research interests include cybersecurity and digital forensics, focusing on contemporary technologies such as cloud computing and mobile devices. Martini has a PhD in computer and information science from the University of South Australia. Contact him at ben.martini@unisa.edu.au.*

KIM-KWANG RAYMOND CHOO *is a senior lecturer in the School of Information Technology and Mathematical Sciences at the University of South Australia. His research interests include cyber and information security and digital forensics. Choo has a PhD in information security from Queensland University of Technology, Australia. Contact him at raymond.choo@fulbrightmail.org.*

This article originally appeared in IEEE Cloud Computing, *vol. 2, no. 4, 2015.*

Editor: Yih-Farn Robin Chen • chen@research.att.com

Toward Mobile-Friendly Web Browsing

Feng Qian • *Indiana University Bloomington*

Smartphones' and tablets' rapid proliferation makes content providers publish mobile versions of webpages. But are they indeed mobile-friendly? This article takes a cross-layer investigation of the mobile Web, and reveals why joint efforts in the mobile ecosystem are needed to achieve mobile-friendly Web browsing.

The unprecedented popularity of mobile devices and their ubiquitous access to cellular data networks make surfing the World Wide Web (WWW) on-the-go a common sight. Mobile browsers have become one of the key entities in the smartphone ecosystem, with their generated mobile traffic volume exceeding that of any other application except for video streaming. Moreover, as the standard Web interface, HTTP is used by millions of smartphone apps, and many apps are simply customized programmable browsers.

The term *mobile-friendly* has been used in many contexts including, in particular, UI design of mobile apps and websites. Indeed, many websites do have their appearance tailored to mobile devices' screens. A recent measurement study[1] shows that 65 percent of the Alexa top 500 websites have mobile versions that are specifically designed for handheld devices. However, loading a webpage is a complex procedure involving many subsystems: object downloading, CSS/JavaScript parsing, content rendering, cache management, and so on. Only changing the appearance of a mobile website is therefore often superficial.

To achieve mobile-friendly Web browsing, three factors must be optimized: performance, energy usage, and bandwidth consumption. First, Internet users are sensitive to webpage load time (PLT). For example, with an extra delay of 500 milliseconds, Google will lose up to 20 percent traffic. With a 100 millisecond extra delay, Amazon will lose 1 percent in sales.[2] In the mobile world, achieving fast page loading speed is more challenging due to unpredictable network conditions (for example, due to mobility) and the limited processing capability of handheld devices. Second, battery life has long been an issue for mobile devices. Over the past 15 years, the CPU performance has improved 250 times while the capacity of the li-ion battery has only doubled.[3] In particular, the power-hungry cellular interface (3G Universal Mobile Telecommunications System/High-Speed Packet Access, or UMTS/HSPA, and 4G LTE) worsens the energy issue. Third, bandwidth is also a critical resource for cellular customers who are billed by their data plan usage. Therefore under the constraints of providing a satisfactory user experience, the bandwidth consumption of mobile Web needs to be minimized.

The remainder of this article discusses why today's mobile Web is often not mobile-friendly, and proposes suggestions on improving the state-of-the-art. I will take a top-down approach by describing issues at each layer: website contents, the Web protocol (HTTP), the Secure Sockets Layer/Transport Layer Security (SSL/TLS) encryption, and the transport protocol. In many cases, the inefficiencies aren't caused by a single layer but instead by unexpected cross-layer interactions.

Website Content

Despite its good looks, a professionally designed mobile website might consume an unexpectedly large amount of resources on a mobile device. Typical issues include using unnecessarily high-resolution images, embedding within a single page too much content that few users will read due to having to scroll down to the page's bottom, employing complex CSS and JavaScript, and using excessive redirections that hurt the PLT. As a concrete example,

 Published by the IEEE Computer Society

the height (that is, vertical dimension) of some popular mobile websites' landing pages can reach up to 40 times of a smartphone's screen height, leading to several megabytes of data being transferred during a page loading.[1]

These issues aren't difficult to comprehend, detect, and fix. However, there are trickier problems that can be easily overlooked due to lack of awareness of how cellular radio works. We know that the power consumption characteristics of the cellular interface are quite different from those in Wi-Fi and wired networks. In cellular networks, it's much more energy-efficient to transmit data in a single bundle, instead of sending them slowly and separately. This is because after a data transfer, the radio interface isn't turned off until a fixed timer, called a *tail timer*, expires. Therefore, having multiple transfers taking place intermittently will significantly lengthen the radio-on time, leading to extra battery drainage, as Figure 1 shows.

The cellular tail effect has several implications on mobile Web browsing. As an example, copied from their desktop versions, many mobile sites perform infinite scrolling: when the user scrolls down to the bottom of a page, the browser will load and append more content to the page. This behavior is totally legitimate in wired networks. But in cellular networks, this bursty traffic pattern (see Figure 1) can potentially keep the radio interface always on as the user slowly scrolls the page, leading to energy inefficiencies. Another representative example is that many websites issue periodical pings for tracking users. These periodical pings are usually triggered by third-party JavaScript (for example, Chartbeat.com) that is embedded in the main HTML page. Again due to the tail effect, these periodical requests account for most of the radio energy consumption of loading a page although their sizes are small.

There are several fixes for these issues. Web designers should balance between a large initial loading and many small incremental loadings, which incur a key tradeoff between bandwidth and energy consumption. JavaScript-triggered delayed or periodical transfers should be minimized unless they are really necessary. For delay-tolerant transfers such as user tracking, there is usually some leeway in terms of when to schedule them. Therefore, their transmissions can be shifted to overlap with delay sensitive data to reduce the impact of the tails. Similarly, multiple instances of delay-tolerant transfers can also be batched together. Ideally, both optimizations (called *piggybacking* and *batching*, respectively) need to gain browser support.

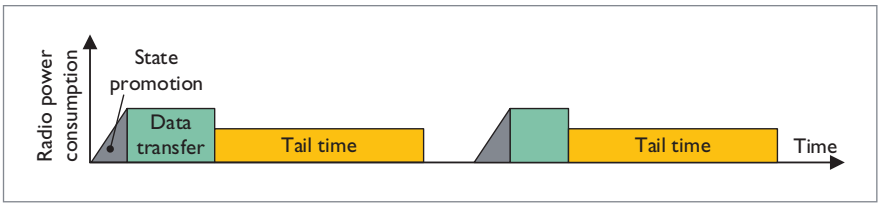


Figure 1. An illustration of cellular radio state transitions. Having multiple transfers taking place intermittently keeps the radio on longer, leading to extra battery drainage.

Caching is another effective mechanism to reduce bandwidth consumption by eliminating redundant data transfers. The effectiveness of caching relies on two aspects: correct caching implementation (browsers must strictly conform to the protocol specification) and good caching semantics (content providers should properly set objects' caching parameters, such as life time). Regarding caching implementation, prior measurement[4] reveals that quite a few HTTP libraries don't perform any caching, and even some popular mobile browsers don't fully support HTTP/1.1 caching. For caching semantics, many professionally designed pages contain objects with a short lifetime (for example, 1 hour), and such objects often belong to images, fonts, and CSS files, that are not expected to change frequently. A similar situation happens with compression, which is often underused for compressible textual objects such as HTML and JavaScript files.

HTTP and Its Interplay with TCP

Now we shift our focus from website contents to the Web protocol. As the key protocol that supports the WWW, HTTP has been stunningly successful. Based on recent measurement studies, HTTP accounts for at least 52 percent of Internet traffic,[5] and 82 percent of the traffic delivered to mobile devices.[6] The percentages are increasing because more and more non-Web applications are using HTTP.

HTTP functions as a request–response protocol. The client, such as a Web browser, sends an HTTP request message to the server asking for a particular resource object (for example, an HTML page or an image). The server then returns with an HTTP response containing the object data. HTTP runs above the Transmission Control Protocol (TCP), which ensures reliable and in-order delivery of the underlying byte stream over the network.

HTTP has been evolving during the past 25 years. The current HTTP version used by the vast majority of today's Web servers is HTTP/1.1, which was standardized in 1999.[7] However, HTTP/1.1 exhibits performance issues as webpages become rich and complex. A modern webpage might consist of hundreds of objects, which are loaded by a large number of short-lived TCP connections in today's HTTP/1.1 scheme. For example, on a Samsung Galaxy S5 smartphone, I conducted an experiment over a commercial LTE network by loading CNN.com, whose 240 objects (1.4 Mbytes' worth of data) from 70 domains were downloaded by 137 connections. The total page load time is 7.7 seconds.

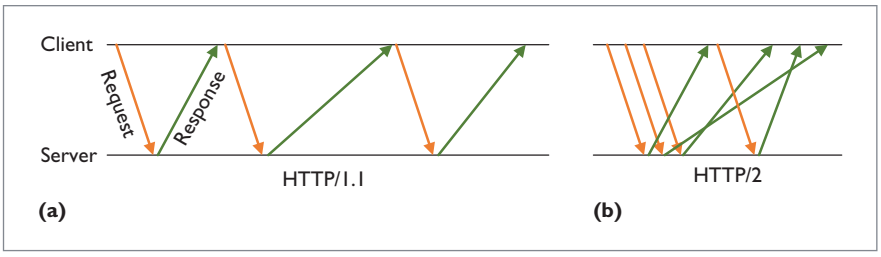


Figure 2. HTTP/1.1 versus HTTP/2. (a) HTTP/1.1 only supports one outstanding request per connection. (b) HTTP/2 allows multiple outstanding requests by multiplexing.

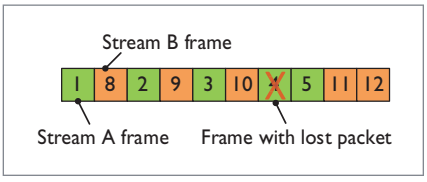


Figure 3. An illustration of HTTP/2 head-of-line blocking. Assume a packet loss occurs in Frame 4 of Stream A. This not only prevents Frame 5 of Stream A from being delivered, but also blocks the delivery of Frames 11 and 12 belonging to Stream B.

However, using LTE to download a single file of 1.4 Mbytes takes less than 1 second under the same network condition. The reasons for such a striking difference are multifold, but the HTTP protocol itself and its interaction with TCP play an important role. In particular, using a large number of short-lived TCP connections is inefficient because each connection incurs bootstrapping overheads of connection establishment, SSL/TLS initialization (if encryption is used), and bandwidth probing. This leads to additional network latency and eventually long PLT. On the other hand, using only one connection is inefficient too because HTTP/1.1 supports only one outstanding request per connection. Several patches such as HTTP pipelining have been proposed to address HTTP/1.1's limitations, but few were widely deployed due to various reasons.

From HTTP/1.1 to HTTP/2

HTTP/2 is the next planned version of HTTP. It aims at overcoming many limitations of HTTP/1.1 by redesigning the data transfer paradigm of HTTP. The Internet Engineering Task Force (IETF) HTTP working group began working on HTTP/2 in 2012. In 2015, the HTTP/2 specification was finally approved by IETF for standardization, and was published as RFC 7540. The design of HTTP/2 draws heavily from SPDY,[8] a recently proposed protocol by Google for improving Web performance. A distinct feature of HTTP/2 is its support for multiple outstanding requests on one TCP connection, as Figure 2 shows. HTTP/2 encapsulates HTTP transactions into streams, such that a stream carries one or more HTTP transactions sequentially, and multiple streams are multiplexed over one TCP connection. Because the number of concurrent connections is reduced from many to one, bootstrapping overheads of short-lived TCP connections in HTTP/1.1 are significantly reduced, leading to more packed traffic on the multiplexed connection. HTTP/2 also supports request prioritization, header compression, and server push.

It's worth mentioning that there are two ways to deploy SPDY: directly connecting to a SPDY server, or using a SPDY proxy. In the former scenario, the client usually establishes one connection for each domain. Many sites today employ a large number of domains with many domains pointing to the same IP address (for example, d1.cnn.com and d2.cnn.com). Known as *hostname sharding*, this makes SPDY behave similarly to HTTP, and is discouraged by the SPDY best practice. In contrast, when a SPDY proxy is used, the browser opens one TCP connection and reuses it across multiple domains. This is a popular and ideal way to use SPDY (for example, configurable in Chrome and used by the Amazon Silk browser). A similar disparity exists in HTTP/2.

SPDY (and HTTP/2) is not without limitations. For example, one known issue is that its performance degrades under conditions of noncongestion packet loss due to the use of a single TCP connection,[9] which aggressively slows down the sending rate upon a loss. In contrast, in HTTP/1.1, a packet loss only affects one of the parallel connections, and the performance of other connections remains unaffected. Furthermore, the connection-level in-order delivery guarantee provided by TCP is too strict for HTTP/2 where only a stream-level in-order delivery guarantee is sufficient. This might cause head-of-line blocking when, for example, a packet loss in one stream prevents data belonging to another stream from being delivered to the upper layer, as Figure 3 shows.

Despite the previously discussed limitations, in wired networks, SPDY has been shown to exceed HTTP/1.1 in most cases with only a few exceptions such as in high packet loss environments.[9] Because cellular networks usually have low noncongestion loss rates, in theory SPDY (and therefore HTTP/2) should also be a winner in the mobile world. However, a recent measurement[10] revealed that in cellular networks, SPDY provides little performance boost, and sometimes might even underperform HTTP/1.1. This counterintuitive observation is attributed to the complex interaction between TCP and the cellular radio layer. More specifically, in cellular networks, when the radio interface state is changed from idle mode to active mode (as triggered by a packet to be transmitted to the base station), it incurs a latency that can last for up to 2 seconds. During this period, which is called the *state promotion delay*[11]

(illustrated in Figure 1), tens of control messages are exchanged between the mobile device and the base station for establishing the data channel. However, at a higher layer, because TCP is not aware of the radio state change, it might simply regard the delay as a signal of packet loss, and therefore retransmit the packet. Such spurious retransmissions cause performance degradation by cutting TCP's slow-start threshold (ssthresh). They occur frequently in 3G UMTS/HSPA networks, and even in LTE.[10]

After this discussion, readers might want to know the final answer to the query, shall we use HTTP/1.1 or HTTP/2 for mobile Web? Thoroughly answering this question requires more study of HTTP/2's behaviors in realistic mobile settings. Nevertheless, given that HTTP/2 is new, it's a promising protocol for high-performance mobile Web. Currently, all main-stream mobile browsers, as well as top content providers such as Google and Facebook, support SPDY and/or HTTP/2. Flywheel,[12] Google's mobile Web proxy that serves millions of customers, also uses SPDY by default, and is migrating to HTTP/2.

Beyond HTTP/2

It might be too early to anticipate what will happen beyond HTTP/2, but researchers have already started working on this issue. Among many proposals, it's worth highlighting the Quick UDP Internet Connections (QUIC) protocol, which Google proposed recently. QUIC has already been deployed at some Google servers.

Similar to HTTP/2, QUIC also multiplexes objects into a single transport connection. However, the most notable feature of QUIC is that it works above UDP instead of TCP, thus eliminating the aforementioned head-of-line blocking issue that is a side effect of TCP's connection-level ordering. Because UDP has no built-in congestion control (CC), QUIC implements a flexible CC framework into which various CC algorithms can be plugged. QUIC's currently implemented CC employs a loss recovery mechanism that's more aggressive than that of the default TCP, thus mitigating the impact of loss on multiplexing. Besides overcoming various limitations in SPDY and HTTP/2, QUIC also introduces several new features. For example, it supports zero-round-trip-time connection setup when the client revisits a server (in contrast, TCP's conventional three-way handshake always takes one round trip); can optionally use forward error correction to better handle losses by adding redundancy to its data transmission; and provides better support for encryption and multipath, which is particularly attractive for mobile devices with multiple network interfaces (for example, Wi-Fi and cellular).

Early measurement using synthetic webpages shows that QUIC outperforms SPDY in many scenarios.[13] However, because QUIC is still experimental, its performance for mobile Web is unclear, and some of its features are potentially not mobile friendly. For example, enabling forward error correction in QUIC consumes up to one-third of available bandwidth even when there is no loss.

HTTPS

TLS is the de facto protocol for securing a TCP connection. Using TLS to transfer data involves two phases: handshake and data transmission. In the handshake phase, TLS uses the Public Key Infrastructure to authenticate the server and to negotiate a symmetric session key, which is subsequently used in the data transmission phase for encryption and decryption. The use of HTTP over TLS (or its predecessor, SSL) is referred to as *HTTP Secure* (HTTPS). Historically, HTTPS was primarily used by Web services involving exchanging sensitive data (for example, a financial transaction). But it's getting increasingly popular, exhibiting a potential trend of HTTPS everywhere. A recent measurement[14] reports that as of 2014, more than 25 percent of server IPs accept HTTPS, which accounts for 50 percent of all HTTP connections. Today, even services such as YouTube use HTTPS.

HTTPS' cryptographic operations incur little energy cost on mobile devices. However, the overheads introduced by the handshake phase are not negligible. First, a full TLS handshake takes at least two round trips. Assuming the average round-trip time in LTE is 70 milliseconds,[15] which translates to 140 ms for a full handshake. Second, the bandwidth consumption of a TLS handshake is not trivial. A TLS handshake consumes on average 4.4 Kbytes of data.[1] This might sound small for a single handshake, but when hundreds of connections are used to load a page in HTTP/1.1 (even SPDY might issue a large number of connections due to domain sharding as mentioned before), the overall penalty could be considerable. As a result, when loading mobile sites using a warm cache, the average bandwidth overhead of TLS is as high as 34 percent.

Two strategies can be leveraged to mitigate the negative impacts incurred by TLS. First, content providers should make fewer HTTPS sessions by, for example, upgrading to HTTP/2 and reducing the number of distinct domains when possible. Doing so facilitates TLS session reuse and mitigates the impact of domain sharding. Second, a Web server should be configured to support TLS Session Identifier or Session Ticket.[16] These would allow lightweight TLS handshakes when the same client connects to the server within a certain time window since its last visit.

So far we have discussed how various aspects at different layers affect mobile-friendly Web browsing. At a high level, we see that achieving mobile-friendly Web browsing is much more than merely tailoring websites' appearance for mobile device screens. It instead requires optimizations on webpage content, Web protocols, transport layers, and

wireless technology. Fueled by joint efforts of all entities in the mobile ecosystem, including content providers, Web browser developers, operating system vendors, and mobile device manufacturers, mobile Web will achieve good performance, a small energy footprint, and low bandwidth consumption.

Acknowledgments

I thank Yih-Farn Robin Chen from AT&T Labs–Research for his valuable feedback.

References

1. F. Qian, S. Sen, and O. Spatscheck, "Characterizing Resource Usage for Mobile Web Browsing," *Proc. Int'l Conf. Mobile Systems, Applications, and Services*, 2014, pp. 218–231.
2. Z. Wang et al., "How Far Can Client-Only Solutions Go for Mobile Browser Speed?" *Proc. Int'l World Wide Web Conf.* (WWW), 2012, pp. 31–40.
3. E. Cuervo et al., "MAUI: Making Smartphones Last Longer with Code Offload," *Proc. Int'l Conf. Mobile Systems, Applications, and Services* (Mobisys), 2010, pp. 49–62.
4. F. Qian et al., "Web Caching on Smartphones: Ideal vs. Reality," *Proc. Int'l Conf. Mobile Systems, Applications, and Services* (Mobisys), 2012, pp. 127–140.
5. C. Labovitz et al., "Internet Inter-Domain Traffic," *Proc. ACM Sigcomm*, 2010, pp. 75–86.
6. J. Erman et al., "To Cache or Not to Cache: The 3G Case," *IEEE Internet Computing*, vol. 15, no. 2, 2011, pp. 27–34.
7. R. Fielding et al., *Hypertext Transfer Protocol-HTTP/1.1*, World Wide Web Consortium (W3C) RFC 2616, 1999; www.w3.org/Protocols/rfc2616/rfc2616.html.
8. *SPDY Protocol-Draft 3.1*, Chromium Project, 2015; www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-1.
9. X.S. Wang et al., "How Speedy Is SPDY?" *Proc. 11th Usenix Symp. Networked Systems Design and Implementation*, 2014; www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-wang_xiao_sophia.pdf.
10. J. Erman et al., "Towards a SPDY'ier Mobile Web," *Proc. Conf. Emerging Networking EXperiments and Technologies* (CoNEXT), 2013, pp. 303–314.
11. F. Qian et al., "Characterizing Radio Resource Allocation for 3G Networks," *Proc. 10th ACM Sigcomm Conf. Internet Measurement* (IMC), 2010, pp. 137–150.
12. V. Agababov et al., "Flywheel: Google's Data Compression Proxy for the Mobile Web," *Proc. 12th Usenix Symp. Networked Systems Design and Implementation*, 2015; www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-agababov.pdf.
13. G. Carlucci, L.D. Cicco, and S. Mascolo, "HTTP over UDP: An Experimental Investigation of QUIC," *Proc. ACM Symp. Applied Computing*, 2015, pp. 609–614.
14. D. Naylor et al., "The Cost of the 'S' in HTTPS," *Proc. Conf. Emerging Networking EXperiments and Technologies* (CoNEXT), 2014, pp. 133–140.
15. J. Huang et al., "An In-Depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance," *Proc. ACM Sigcomm*, 2013, pp. 363–374.
16. J. Salowey et al., *TLS Session Resumption without Server-Side State*, IETF RFC 4507, 2006; https://tools.ietf.org/html/rfc5077.

Feng Qian is an assistant professor in the Computer Science Department, School of Informatics and Computing at the Indiana University Bloomington. His research interests include computer networking, mobile systems, network measurement, and energy efficiency. Qian has a PhD in computer science and engineering from the University of Michigan. Contact him at fengqian@indiana.edu.

This article originally appeared in IEEE Internet Computing, *vol. 19, no. 5, 2015.*

Editor in Chief: **Diomidis Spinellis**
Athens University of Economics and Business, dds@computer.org

Architecture from a Developer's Perspective

Diomidis Spinellis

I CAN STILL remember when, back in 2003, a fellow FreeBSD developer chastised me for an architectural misstep. I had proposed adding a reference to a related C library function in the documentation of a Unix system call. "I believe this is bad practice (a layering violation)," he wrote to me. He was right; I hadn't thought carefully about that small addition. The truth is that as a developer you practice architecture daily, but only rarely do you have time to reflect on your corresponding decisions, actions, and their consequences. Software architecture affects the quality of what you build and how you build it.

The Importance of Software Architecture ...

The most obvious way that architecture affects quality is maintainability. Code that lacks clear boundaries and interfaces is difficult to analyze. It's also brittle and, therefore, difficult to change. A small addition or fix in one place can cause a cascade of additional required work, or worse, bugs. Also, you can't easily test and debug such software because it will lack obvious interfaces where you can apply test probes or add logging functionality.

> Architecture is difficult to learn and practice.

Then comes performance. On a small scale, code jumbled together can be famously efficient: think of tightly coded routines, graphics kernels, and some game engines. However, once the scale increases, the only hope to cope with rising demand comes from parallelism architectures. These let you split your work horizontally (along tasks) or vertically (across multiple clients). They also guide you on how to shard or partition your data. Through such architectures, you increase both your current service capacity and future scalability. Similar approaches can increase your service's reliability. First, you can manage fault tolerance by distributing the work among nodes that can step in to cover each other in the event of a failure. Second, the same nodes can then help the more complex task of recovery. Don't even think about orchestrating recoverability into your service without an architecture to guide the delicate required dance.

Another quality aspect that software architecture aids is portability—a must in an age of rapid innovation and shifting technology alliances. Through clear layering,

IEEE Software Mission Statement: To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

 Published by the IEEE Computer Society 2376-113X/16/$33.00

SER&IP 15 BEST PAPER AWARD

The 2nd Annual Software Engineering Research & Industrial Practice 2015 (SER&IP 15) workshop, held in conjunction with the International Conference on Software Engineering 2015 (ICSE 15), focused on the sometimes problematic interface between the academic and practitioner communities.

Given *IEEE Software*'s mandate to present the practical and impactful work that can help bridge this gap, the magazine was pleased to sponsor the Best Paper Award recognizing some of the outstanding work presented at the workshop. *IEEE Software* editor in chief emeritus Forrest Shull served on the selection committee and helped select the best paper on the basis of the criteria of readability, rigor, and relevance.

The selection committee chose "Principles and a Process for Successful Industry Cooperation— the Case of TUM and Munich Re," by Maximilian Junker, Manfred Broy, Benedikt Hauptmann, Wolfgang Boehm, Henning Femmer, Sebastian Eder, Elmar Juergens, Rainer Janßen, and Rudolf Vaas, for the Best Paper Award on the basis of the soundness of the lessons learned and the balanced treatment of both positive and negative aspects of tech transfer. *IEEE Software* editor in chief Diomidis Spinnellis presented the award on behalf of the magazine. (For more on this paper, see Practitioner's Digest on p. 27.)

Please join us in congratulating the authors for their excellent work and thank all of the authors for taking the time to share their experiences in such a forum.

your software can quickly adapt to new hardware platforms and software interfaces. Proper encapsulation can also make your software easier to install and coexist with other offerings.

Software architecture also affects your main development processes—the way you can split the teams that develop the software, how you can run it across countries and time zones, and how you can maintain it without disruptions. It also helps your ancillary processes. A suitable architecture goes hand-in-hand with effective configuration management tasks, such as versioning, branching, merging, and continuous integration. Software architecture can provide clear boundaries to manage quality efficiently. For instance, it can allow you to tailor quality and processes characteristics for diverse software modules. Regarding testing processes, modern testing frameworks are typically embodiments for corresponding architectural styles.

Finally, software architecture is the key enabler for reusability—processes that span many of your organization's products and services. It can help you create modules that can be reused within your organization, and it can drive software product lines.

… And What to Do

Given software architecture's importance, what should you be doing as a developer? This is a tough question, because architecture is difficult to learn and practice (it's been

described as an old man's art), and its mistakes can be hugely expensive.

My advice is to focus relentlessly on the primary concerns. Smaller ones are important, but the big ones determine success or failure. Look at your software's most common, large, and critical functions; study your software's future evolution path, looking for things that are difficult to change; and determine the key quality attributes. These elements will point toward the important things that your architecture needs to address. Once you have them, invest significant effort in developing a matching architecture. Consider other successful examples, ask around, look for already available modules, prototype, and experiment. Be ready to toss out a solution if something better emerges. Remember, this is what can make or break your software.

I recently withdrew a paper I had submitted and started working almost from scratch on a two-year effort when a much more experienced colleague suggested a drastic improvement in a design. Not all software deserves such sacrifices, but you should be ready to make them when you see the potential.

Then, avoid the temptation to overengineer. The worst architectural sins have been committed by developers keen to demonstrate their (often half-baked) knowledge of design patterns and other elements often misused as architectural crutches. Frameworks, design patterns, and enterprise-scale platforms are all useful, but applying them to the wrong area creates more problems than it solves. The cognitive load of a needlessly complex software architecture is higher than that of a slightly simplistic one. Therefore, avoid designing structures when there isn't a clear demand for them, and choose the simplest solution that can do the work.

This brings me to another principle: be ready to refactor when the need emerges. Your lean and mean software architecture will be pressured as the system evolves, accumulating technical debt. In contrast to an overengineered system, the pressure will quickly become apparent and the pressure points will reveal where refactoring is truly required. Consider yourself lucky at that point: in contrast to green-field development, you have a very clear requirement of where to invest your architectural and refactoring effort. Do it without stinginess or looking back. An interesting example is the evolution of the pipes and filters architecture under Unix. When pipes were introduced to Unix, Bell Labs researchers worked tirelessly to convert all their existing programs into filters that could be connected through them. The rest is history.

> Focus relentlessly on the primary concerns.

Keep in mind that architecture is about more than software code. Consider how your whole system (in the widest possible sense) will be decomposed into processes or services; how data are stored, communicated, and processed; and how all parts fit together to deliver the required functionality, reliability, capacity, scalability, maintainability, and portability. Your decisions here may affect which parts you can purchase, reuse, or outsource. Earlier this year, a team I worked with faced the problem of maintaining a large set of data that would slowly evolve over time, changing through both daily automated processes and human interactions. All changes should be auditable, and it should be possible to rerun the processing starting at an arbitrary point in time.

Initially, we considered as an obvious choice a complex relational database schema encompassing time-stamped records, user authorizations, processing chain identifiers, and an event log. We also considered using file-system directories to implement part of this functionality. Both approaches involved considerable amounts of application code. It then dawned on us that by using a revision control system such as Git to version the data files, we could get most of the required features "for free." As an added bonus, team members could also employ user-friendly Git interfaces to manipulate the data. This simple decision, which took us about a day of deliberation and discussions to agree on, saved us weeks of development effort and debugging.

Finally, when you develop your architecture, you should adhere to sound software design principles:

- abstraction of processes, data, and control structures;
- low coupling and high cohesion;
- separation of concerns, decomposition, modularity, encapsulation, and information hiding;
- separation of interfaces from implementation and of policy from mechanisms; and
- completeness, economy, and simplicity.

Adhering to all these tenets might sound like a tall order. But nobody ever said that architecture is cheap; it's a sound investment for your current needs and future evolution. As Brian Foote and Joseph Yoder once said, "If you think good architecture is expensive, try bad architecture."

This article originally appeared in IEEE Software, *vol. 32, no. 5, 2015.*

See www.computer.org/software-multimedia for multimedia content related to this article.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

The Rise of Multimedia for Online Communication Startups

Rong Yan
Snapchat

Online messaging has been a hot topic in the startup world lately. Last year, a new breed of online messaging tools sprang up and experienced tremendous user growth. According to a recent study by BI Intelligence, the top four messaging apps grew their user numbers at a quarterly rate of at least 15 percent in 2014, leading the study to suggest that "messaging is poised to take over social networks within the next few years."[1]

Although online messaging is similar to text messaging, these next-generation apps usually offer more convenient communication features—in particular, tools for sharing more media types, including images, videos, and voice messages. This is in line with the trend of social media users shifting away from traditional text-based communication and embracing more multimedia-oriented platforms. Popular social networks are also following this trend to enhance their online-messaging services to support new media types. Facebook's purchase of Instagram and Twitter for Vine highlight the potential for social multimedia sharing.

But why is this happening, and who is leading the way? Here, I study the driving forces behind the multimedia evolution of online communication and take a look at some recent startups, exploring the reasons behind their success.

Evolution of Online Communication

The newfound popularity of multimedia content in online messaging is not by accident. About a decade ago, when the first generation of social networks entered the Internet—such as the LinkedIn profile or Facebook News Feed—most users only cared about sharing information. But as sharing became more of a communal experience, a relatively new trend of the "shared experience" began, with users expressing a greater interest in sharing their personal thoughts, feelings, and emotions with as many people as possible. This user evolution of online communication has enabled a new ecosystem of user-created content, supported by multiple media types including video, audio, and photos.

Arguably, multimedia offers the best medium to faithfully deliver what users see, hear, and experience in the real world. According to Mintel's March 2013 survey,[2] 31 percent of US adults who use social networks capture video or images specifically so they can share that content online. Furthermore, 53 percent of networkers said they talk about things they see on social media in face-to-face conversations.

The rise of multimedia adoption has also been profoundly influenced by recent technological advances, which have changed our communication behaviors. In particular, I'd like to stress the following three trends.

Internet Everywhere

Steve Jobs once said,

> When we were an agrarian nation, all cars were trucks. But as people moved more towards urban centers, people started to get into cars. I think PCs are going to be like trucks.

Since the day this quote slipped off Jobs's lips, it has become increasingly clear that that post-PC era everyone has been talking about has arrived. In 2011, two important milestones suggested that we had crossed that imaginary line from the PC era to the post-PC era: smartphone shipments outpaced PCs for the first time ever,[3] and Apple became the world's largest PC maker, counting iPads as PCs.[4] These statistics speak volumes about the state of modern-day computing.

As noted by Chris Jones, a principal analyst at Canalys, the computer-in-your-pocket has moved from being "a niche product segment at

 Published by the IEEE Computer Society

the high end of the mobile phone market to becoming a truly mass-market proposition."[3] With the prevalence of new smart mobile and wearable devices, such as Google Glass and Apple Watch, it has become increasingly convenient for people to access the Internet everywhere, propelling the growing popularity of new media types as the management and uploading of multimedia content becomes simpler than ever. Users can now share their experience anytime, anywhere, and unlike with traditional social media, they do not have to capture the offline world and recreate it online. They simply record it live and communicate at the same time, which has laid the foundation for the success of online messaging apps with multimedia support.

Users are increasingly demanding new forms of cyber communication that more closely resemble the characteristics of real in-person interaction.

Time Fragmentation

Smart devices and apps are now occupying every single corner of our lives. They are the first things we check in the morning and the last things we look at before going to bed. Many young Internet users have grown accustomed to instant gratification and to instantly reaching their friends through social media or messaging apps. They have much shorter attention spans than before[5] and are constantly distracted by an incoming flow of notifications. A report from Pew Internet and American Life Project shows that more than half of us use our cellphones while watching TV.[6] As the adoption of smartphones increases, I am sure we will see more of this.

Today's successful apps must adapt to the increasing difficulties of drawing users' attention and creating a convenient environment for them to achieve their intentions within minutes or even seconds. Multimedia is a perfect tool for this purpose. The success of short-video apps, such as Vine and Snapchat, has exemplified this idea—these apps take only up to a few seconds to convey an idea, make us laugh, or get us to think. Although it might seem as if this approach would be too brief, especially when compared to the longer-form videos hosted by YouTube, these apps make the media-sharing process extremely simple and convenient, and they fit perfectly with the growing fragmentation in user behavior.

Back to the Basics

The advent of social networks has facilitated conservation and information distribution across a large group of users. However, their dominance fundamentally redefined the communication style in the online space, where over-sharing has become the default, and personal information is permanently stored publicly. Users are increasingly demanding new forms of cyber communication that more closely resemble the characteristics of "real" in-person or over-the-phone interaction, which is by nature transient and private.

As a matter of fact, our daily communication is largely comprised of fleeting moments—uttered words are seldom recorded, and the images we perceive with our eyes are rarely saved and replayed. This inspired the creation of ephemeral messaging that no longer keeps the media content forever, and mobile streaming that distributes video experience in real time. Moreover, to regain some sense of privacy, anonymous social networks are emerging that avoid revealing the sender's identity when the content is shared. This need to return to the basic form of communication and privacy protection has created a valuable proposition in this age of social technology.

Online Messaging Startups

To meet with these newest user needs, online communication startups are bubbling up with an astonishing variety of services and products. Fueling this new market are the higher-than-ever inflow of venture capital investments and emergence of basic building blocks that make nimble startup possible, including reusable open source modules made available on the Internet; easy-to-learn programming frameworks; along with cloud-based services that can host startup offerings (Amazon's cloud

These quick bursts of powerful messages are becoming widely accepted among users, catering to the dwindling attention span of online users.

computing), distribute them (Apple's App Store), and market them (Facebook, Twitter). To exemplify, I describe three categories of the latest online messaging startups and explain how multimedia technology plays a vital role in their product offerings.

Ephemeral Messaging and Short Videos

Among the most promising directions for online messaging are ephemeral messaging apps. Their idea sounds extremely simple—users have only a limited amount of time (typically several seconds) to record their images or videos, and the media content self-destructs after a certain expiration time. This form of messaging has achieved tremendous success in recent years.

For example, Snapchat, the leading ephemeral messaging app, has been on a serious growth curve—in 2014, it had the fastest growing audience.[7] More impressively, a recent infographic (https://photoworld.com/how-big-is-snapchat) has shown there are already more photos shared on Snapchat than on main-stream social networks, such as Facebook and Twitter, despite its smaller user base.

To compete in the same space, three other ephemeral messaging startups raised over US$43 million in 2014: Wickr, Frankly, and Cyber Dust. Blink is another similar startup that was acquired by Yahoo for an undisclosed amount, and Facebook had its second attempt at an ephemeral app called Slingshot. Short videos can be popular without being ephemeral. For example, Vine lets users create mini videos up to six seconds long and has attracted 100 million monthly active users after being acquired by Twitter in 2012.

The success of these startups can be traced back to their nature in transmitting temporary, unrecorded, unaltered real-life interactions to the digital world in various media forms. Because the media content is not stored permanently, many users on these platforms are highly engaging, posting tens of images/videos on a daily basis without worrying about information overload for their viewers. In addition, similar to a phone call or in-person conversation, self-destruction means viewers need to pay more attention to the media when watching their friends' messages. This factor further adds up to the popularity of ephemeral messaging apps.

Anonymous Messaging

Recent years have also seen the rise (and fall) of a new type of social messaging apps called anonymous messaging, such as Whisper, Secret, and Yik Yak. These apps draw increasing attentions from users by allowing them to post public messages without revealing their identities, while their friends or friends of friends can respond and search by popularity, topic, or location. Consequently, these apps naturally provide a communal experience among anonymity.

For example, one of the most well-known anonymous apps, Whisper, attracted 10 million active users in 2015.[8] On average, the Whisper app is opened 1 million times an hour, which means its users are highly engaging in this platform. Similarly, Secret amassed 15 million users and raised $35 million in venture capital before its demise.[8] The popularity of these apps can largely be attributed to the so-called "identity fatigue"—that is, Internet users' growing weariness with associating digital communications with their real-world personas, making them susceptible to public scrutiny. As Brooks Buffington, CEO of Yik Yak, pointed out, "Once you have a profile, you are expected to act a certain way. People only post the best, most beautiful parts of their life on Instagram … [For anonymous apps] you just put something out there, and if it doesn't resonate with anyone, it's not a reflection on you."[9]

It is worth noting that multimedia has been widely used in some of these services. For example, Whisper and Secret often ask users to select a relevant picture to feature with the anonymous text message. Users can use an image from the movie *Poltergeist* to illustrate words such as "fear," "ghosts," and "dreams." This is similar to how Google's search engine shows

ads based on the search terms, and it opens up a new research direction for multimedia researchers, where image-recommendation algorithms can be applied to improve the user engagement rate.

However, anonymous messaging still faces a lot of challenges. As a prime example, Secret shut down after just 16 months because of internal strife and uncertainty. A few key mistakes are likely what led to its failure. Its decision of targeting tech-savvy professionals did not pan out as well as Whisper's targeting of teens. Some reports also suggest that Secret's CEO, David Byttow, closed the company partially because the way people were using the app—to spread malicious rumors—was not aligned with his original vision of the app. Secret's quick rise and fall taught us a lesson that founders have to be proactive in understanding customer preferences and must adapt their strategy accordingly over time.

Mobile Live Streaming

Live video streaming is nothing new. These services have been around since the early 2000s, but the older live-streaming platforms (Ustream, Twitch) have been mostly focused on creating a niche segment or courting corporate clients. The new generation of mobile live-streaming apps, Periscope and Meerkat, pursue a more consumer-oriented path by making it extremely simple to broadcast our lives on the fly, shifting the appeal back to the mainstream users. You can consider them as "live YouTube." Both apps let users send live-streamed videos to their followers, such as a product demonstration or a video of taking the dog for a walk.

The live streams on these platforms are ephemeral: Periscope's streams stay visible for 24 hours, while Meerkat's disappear when the recording ends. Both were a breakout hit at this year's South by Southwest technology festival, and both peaked in the highly anticipated heavy-weighted Mayweather-Pacquiao fight in May 2015.[10] Meerkat and Periscope are great indications of the explosive growth in capturing and sharing mobile videos, driven by recent technology advances and user behavior changes. In fact, a 2015 Cisco report predicted that by 2017, video will account for 30 percent of Internet traffic and 70 percent of traffic on mobile devices,[11] and these numbers are likely to grow down the road.

The appearance of live-streaming apps has already impacted the way the media industry interacts with its users. For example, with these new tools, brand advertisers can deliver their marketing messages across multimedia channels in real time. Also, these apps have created a new world of real-time user-generated content available to news reporters. As a complementary channel to other social media services, TV newsrooms are starting to experiment with monitoring these live mobile feeds. With the integrated live comment feed in such apps, producers can quickly communicate with content creators to ask for more context, getting more information sooner than with video sites like YouTube.

Mobile live streaming is still in its infancy. Periscope and Meerkat might be media darlings right now, but that could soon change.

Despite all the hype, as both are also spawning major disputes around alleged privacy violations or infringements on copyrighted content. In April 2015, HBO issued "takedown" notices to Periscope after people streamed the season five premier of its "Game of Thrones" show.[12] The nature of live broadcasting makes it easier to pick up videos of private conversations and copyrighted material without being noticed. Also, unlike the other video-sharing sites, Meerkat and Periscope have very little time to review footage before the content goes out, making it difficult to discover and take down the policy-violated content in time. How to address these concerns and land support from privacy professionals and regulators remains to be a challenging topic for mobile live streaming.

As more and more users can gain access to Internet, have shorter attention spans, and show an inclination to return to basic communication patterns, a new breed of successful online communication startups have emerged to meet with these needs. As multimedia technology continues to evolve, this disruptive

This article originally appeared in IEEE MultiMedia, *vol. 22, no. 4, 2015.*

trend will last in the foreseeable future, presenting a huge, uncharted opportunity for multimedia researchers and industrial participants. **MM**

References

1. D. Smith, "Chart of the Day: Mobile Messaging Is Poised to Overtake Social Networks," *Business Intelligence*, 14 Nov. 2014; www.businessinsider.com/chart-of-the-day-mobile-messaging-is-poised-to-overtake-social-networks-2014–11.
2. B. Hulkower, "Digital Trends Spring—US—March 2013," Mintel, 2013; http://store.mintel.com/digital-trends-spring-us-march-2013.
3. "Smart Phones Overtake Client PCs in 2011," Canalys, Feb. 2012; www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011.
4. "Apple Storms Past HP to Lead Global PC Market," Canalys, Jan. 2012; www.canalys.com/newsroom/apple-storms-past-hp-lead-global-pc-market.
5. L. Watson, "Humans Have Shorter Attention Span than Goldfish, Thanks to Smartphones," *The Telegraph*, 15 May 2015; www.telegraph.co.uk/news/science/science-news/11607315/Humans-have-shorter-attention-span-than-goldfish-thanks-to-smartphones.html.
6. A. Smith, "The Rise of the 'Connected Viewer,'" Pew Research Center, July 2012; www.pewinternet.org/2012/07/17/the-rise-of-the-connected-viewer.
7. J. Mander, "Snapchat Was the Fastest Growing Social App of 2014," blog, 27 Jan. 2105; www.globalwebindex.net/blog/snapchat-was-the-fastest-growing-social-app-of-2014.
8. P. Dave, "Whisper, App with 10 Million Young Users, Attracting Advertisers," *Los Angeles Times*, 30 Apr. 2015; www.latimes.com/business/technology/la-fi-tn-whisper-president-advertisers-20150430-story.html.
9. J. McDermott, "Inside Yik Yak, An Anonymous Sharing App Sweeping through Colleges," *Digiday*, 8 Dec. 2014; http://digiday.com/platforms/inside-yik-yak-anonymous-sharing-app-sweeping-colleges.
10. J. Bracy, "Meerkat and Periscope: Are Privacy Pros Ready for the Consequences of Live Streaming?" IAPP, 2015; https://privacyassociation.org/news/a/meerkat-and-periscope-are-we-ready-for-the-consequences-of-live-streaming.
11. *Cisco Visual Networking Index: Forecast and Methodology*, 2014–2019 White Paper, Cisco, 2015; www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.
12. N. Jarvey, "HBO Criticizes Periscope Over 'Game of Thrones' Live Streams, Issues Takedown Notices," *The Hollywood Reporter*, 14 Apr. 2015; www.hollywoodreporter.com/news/hbo-criticizes-periscope-game-thrones-788734.

Rong Yan is a director of engineering at Snapchat. Contact him at rongyan@snapchat.com.

Editors: Konrad Hinsen, konrad.hinsen@cnrs-orleans.fr | Matthew Turk, matthewturk@gmail.com

Technical Debt in Computational Science

Konrad Hinsen | Centre de Biophysique Moléculaire in Orléans

Technical debt is a recent metaphor that the software industry is rapidly adopting. First used by Ward Cunningham in a 1992 report on a software development project (http://c2.com/doc/oopsla92.html), the term refers to future obligations that are the consequence of technical choices made for a short-term benefit. The standard example is writing suboptimal code under time pressure, knowing that the code will have to be refactored or rewritten later to make the software maintainable. The additional effort for refactoring or rewriting, which doesn't improve the software's utility for its users and therefore doesn't add market value, serves to pay back the debt.

Debt, Interest, Payback, and Default

The word *debt* emphasizes an analogy to monetary debt: both are future obligations incurred in exchange for a short-term benefit. But the analogy goes further: both generate interest. In the example of the hastily written code, any work done on it before refactoring or rewriting, be it to fix bugs or quickly add features, will require more effort than it would for well-written code. It's also probable that much of this work will have to be repeated after paying back the debt—the additional effort is the equivalent of paying interest. Another useful analogy is debt default: defaulting on a technical debt lowers quality standards, indicating that an objective can't be met because of a bad technical choice in the past. For a company, it can mean the end of a product line or, worse, the company itself.

But just like a financial debt, a technical debt isn't necessarily a bad thing. There can be good reasons for cutting corners and fixing the resulting problems later. Being the first company to propose a product on the market is a competitive advantage that can procure long-term benefits. Similarly, a scientist can derive a significant benefit from being the first to publish an important new result. The point of the technical debt metaphor isn't to reprehend such choices but to remind us of the long-term consequences.

2376-113X/16/$33.00 © 2016 IEEE Published by the IEEE Computer Society

Like all analogies, the debt metaphor has its limits. A financial debt is the result of a contract between a borrower and a lender that describes the exact conditions of the debt. Unless you carelessly take a loan without reading the contract, you know what your future obligations are and what short-term benefits you get in return. Technical debt results from a contract with your future self, and its terms usually aren't written down anywhere. An experienced engineer will recognize having incurred a technical debt but might not be able to give a precise estimate of the interest and the final payback. An inexperienced person can even incur technical debt without being aware of it at all, seeing the short-term benefit but not the long-term obligations.

A Case Study: The Python Language

A simple Web search yields many examples of and discussions about technical debt in the context of commercial software development. Much of this applies to scientific software as well, especially to larger and long-lived software projects with multiple developers and some form of project management. However, both the nature of these software projects and of the organizations behind them is much more diverse in scientific computing. In particular, much software development happens in relatively small research groups that have informal collaborations with other such groups, either on a common software package or on distinct but interdependent software packages. In such an organization, anyone's technical debt has an impact on everyone else.

I can illustrate this with examples from the scientific Python ecosystem, the term commonly used to describe the large set of scientific libraries written in the Python language. It has an onion-like structure, with the Python language itself at the core. The next layer contains a small number of scientific infrastructure libraries such as NumPy (array computations) and matplotlib (plotting). The third layer consists of domain-specific libraries that tend to depend on libraries in the infrastructure layer or on other items in the domain-specific layer. Outside of these three layers, we find "client code": scripts and workflows that are specific to a research project but also highly domain-specific software tools with graphical user interfaces.

The transition from Python 2 to Python 3, which started in 2008 and is still going on, is a nice example of paying back technical debt with a partial default. The Python language had continuously evolved over the years, acquiring both new features and new modules in its standard library. The desire to keep each version backward compatible with earlier versions led to redundant features that made the language needlessly complicated. For example, old-style and new-style classes exhibited subtly different behavior. Everyone agreed that new-style was better, but old-style was there before and existing code relied on it. Similarly, the standard library acquired redundant modules, whereas other modules became obsolete in the sense that they relied on no longer maintained libraries or were specific to computing platforms that had long since been transferred to museums.

The reason the transition to Python 3 is partly a repayment and partly a default is that it preserves one objective while violating another. Python started out with the goal of being a simple and easy-to-learn language, an objective that was preserved with the general cleanup that led to Python 3. But publishing a programming language and encouraging people to use it implies the promise of not breaking their code in the future. This tacit promise was broken with Python 3, which is incompatible in many details with earlier versions—the two objectives being contradictory, the only way to maintain both would have been to stop future evolution. Most programming languages face this choice at some time, but most designers choose to continuously accumulate complexity rather than clean up the mess. In other words, they default on the technical debt by giving up simplicity.

Looking at this from the viewpoint of the creators of scientific libraries written in Python, we see how technical debt in Python's development has a direct impact on their work. With the Python development community moving on to Python 3, it will eventually have to abandon Python 2. Library authors thus have to choose: either migrate to Python 3 now or keep the Python 2 platform alive by taking over its maintenance. Both choices involve additional effort. Doing nothing seems like a third option, but given the fast rate of change in computing platforms, today's Python 2 will become effectively unusable within a few years. Moreover, hardly any scientific library is useful in isolation, so everyone's choice depends on the expected behavior of the authors of related libraries. At this time, the core infrastructure libraries and many of the bigger domain-specific offerings have initiated or even completed the transition to Python 3, while still maintaining some level of compatibility with Python 2. Many libraries with a smaller developer base remain in the Python 2 universe, lacking either the means or the motivation to move on.

In terms of the technical debt metaphor, we can say that choosing the Python language, or in fact

choosing to base your work on any dependency or tool controlled by someone else, creates technical debt. The short-term benefit is the immediate availability of a useful software component. The interest is the work required to adapt your own code to changes in the dependencies or alternatively to take on the responsibility of maintaining a version of those dependencies that remains compatible with your own code. Paying back the debt would mean replacing the dependency with your own code, which is rarely done in practice. The technical debt resulting from dependencies is, in most cases, perpetual. Moreover, such debts are practically inevitable because not depending on other people's work—that is, writing everything yourself—isn't a realistic option. After all, even the computer's operating system is a dependency. You can, however, try to minimize risky dependencies as part of a strategy for managing technical debt. Matthew Turk recently wrote about this option in this department.[1]

The kind of technical debt involved here is perhaps the most frequent one in computing, even before the standard example of cutting corners to terminate a project as early as possible. It can be summarized as relying on immature technology. When you choose a programming language that's just a few years old, you should expect that nobody, not even its creator, has sufficient practical experience with it to have made all the right choices. Either the language will remain static and fade from popularity quickly, or it will change and become either messy or incompatible. In all these scenarios, you have a maintenance problem with your code that relies on it. If you want to avoid this, you should choose a programming language that has been around for decades. Indeed, stability is one reason cited for choosing Fortran. Of course, the same principle applies to other dependencies such as libraries. It's probably safe to bet on BLAS being around for many more years without incompatible changes, but the same can't be expected of a recent implementation of today's hottest algorithms. This well-known problem of software becoming unusable because of changes in its dependencies is sometimes called *software rot*. This isn't a good metaphor, however—software doesn't degrade in time. Rather, the foundations on which the software is built change—and not by decaying but as a side effect of improving. The software rot metaphor has led to the equally misleading term *software maintenance* for keeping software usable by adapting it to evolving environments.

In a fast-moving field such as computing, immature technology is the norm rather than the exception. We all work with immature technology every day, and we know it. My computer crashes about once per month, requiring a reboot. It asks me to install software updates, often labeled as security-critical, at least once a week. Broken Web links are a daily experience. It's safe to assume that scientific software is of no better quality, even though the symptoms of bugs are usually more subtle and can go unnoticed. For scientists, who by definition work at the frontiers of knowledge and technology, there's really no way to avoid immature dependencies. We can, however, be aware of it and try to anticipate the consequences, or at the very least, avoid pretending that there aren't any.

Debt in Research

The technical debt metaphor is most frequently applied in software development, but it applies equally well elsewhere. An interesting example is a recent exploration of the impact of data dependencies in applications of machine learning techniques.[2] Such a systems-level view of technical debt is also useful in the context of scientific research.

Science has long-established standards of quality that all scientists have the moral obligation to respect. In particular, they should make a serious effort to verify the results they obtain, actively searching for potential mistakes to overcome confirmation bias, the natural tendency of humans to search for confirmation rather than refutation of their own hypotheses. Moreover, scientists must publish detailed accounts of their work to permit their peers to verify it, attempt to reproduce the findings themselves, and build on it in future research. The respect of these obligations makes the difference between a scientific result and anecdotal evidence.

Verifying your own results and conclusions implies first acquiring a sufficient understanding of your methods and tools prior to using them, as well as ensuring that they're adequate for the task. Computational scientists have traditionally been rather negligent about this. The few prominently public cases of mistakes in scientific results due to bugs in software are probably just the tip of the iceberg,[3] suggesting a widespread lack of testing. Moreover, scientific software is often applied incorrectly, due to a lack of understanding of the computational methods that the software implements.[4] This is partly the fault of scientists using software they don't understand, but also partly the fault of scientific software authors providing insufficient documentation and neglecting the readability of their source code.

The word *negligence* already suggests that basic human tendencies such as laziness are an important cause of these problems, but there's also a technical aspect to it. Scientists increasingly treat computational methods as similar to experimental ones and consider computers and software as the theoretician's equivalent of experimental equipment. This point of view is useful for simulation techniques, which produce data that's analyzed and evaluated in much the same way as experimental measurements, with a strong emphasis on statistical approaches. There is, however, a fundamental difference between computers and instruments used in experiments. Lab instruments, like any physical devices, are subject to inevitable imperfections in manufacture. They're thus designed in such a way that small imperfections can only cause small deviations in the results. Computers, on the other hand, are chaotic dynamical systems. Changing a single bit in a computer's memory can change the result of a computation beyond any predictable bound. Computers are practically usable devices in spite of this sensitivity because of their extreme reliability, compared to other technical artifacts.

Although hardware errors can become a problem with long-running computations on very large machines, for most applications of computers in scientific practice, it's safe to assume that the computer does precisely what the software tells it to do. However, errors in the software or in the input data are amplified with each computational step. Often, we can (and do) ensure that small errors in the input data translate to small deviations in the results via a judicious choice of numerical methods. But we don't yet have good techniques for limiting the impact of software errors. We should therefore add the use of chaotic devices for computation to our technical debt account and accept the effort for carefully testing our software as an inevitable interest payment, hoping to pay back the debt one day by a profound change in the way computers are used in research that limits the impact of chaotic behavior. Because most scientists aren't aware of this fundamental difference between software and the physical devices used in experiments, this particular debt resembles a loan taken without reading the contract.

Reproducibility

The reproducibility requirement of science implies the publication of a sufficiently detailed description of what was done. Computational science has performed very badly in this respect as well. This problem has received a lot of attention recently, and *CiSE* has dedicated two theme issues to it (January/February 2009 and July/August 2012). As with software bugs, there are both human and technical reasons, the latter being cases of technical debt again.

One major reason for the widespread nonreproducibility of computational results is the use of immature technology, which I've already discussed earlier in the context of software development. It means that software must be actively maintained to be usable in the future, making software maintenance a requirement for reproducibility. Unfortunately, active maintenance of all research software down to the tiniest script used for data munging requires more effort than the scientific community can afford to dedicate to such activities. This isn't only a question of affecting the means necessary to do the work—in many cases, only the original author of a script knows what it's supposed to do exactly. If the original author is a PhD student who leaves academic research after the thesis, no one is left to do the maintenance. In practice, we most often prefer to default on this kind of debt, all the more because such a default is still socially acceptable today. The reproducible research movement works toward paying back the debt in two ways: ensuring the sustainability of widely used pieces of scientific software, and preserving more information about the computational environment of a particular research study, to be published alongside its results as essential documentation.

Another technical reason for nonreproducibility is the sheer amount of information required for fully specifying a computation. In theory, any computation is defined by a single computer program. All we have to do is publish that program together with a scientific article, and anyone could rerun it to verify the results. In practice, that program is a complex assembly of a multitude of parts. Typically, we have many libraries, and multiple programs that call functions from these libraries. A compiler and linker create a single unit for each of these programs, specialized for a particular type of computer. We then combine several such programs with input data and an outer algorithmic layer often called a "workflow" to obtain the result. To make it worse, we often launch computational steps interactively, meaning that part of the workflow exists only in our heads. Tools for managing the assembly and execution of such complex computations have been around for a long time—the well-known `make` utility for the Unix family of operating systems was published in 1977. But they've been ignored by most computational scientists until very recently, partly out of ignorance

and partly for not wanting to learn the use of such tools. This debt is in the category of cutting corners for advancing more rapidly. We pay interest in the form of increased manual labor, and we tend to default on the reproducibility aspect.

A final but frequent category of technical debt in computational science results from an obsession with performance. This debt is particularly difficult to deal with because the interest can go unnoticed, and the debt is almost never paid back. Its importance has nevertheless been recognized and is well expressed by the famous D.E. Knuth quote reminding us that "premature optimization is the root of all evil (or at least most of it) in programming."[5] Best practices in software engineering say that you should first write a clear and simple program, and then validate it by extensive testing. In a second step, performance bottlenecks are identified by profiling and eliminated by optimization. Computational scientists often rush for optimization, choosing low-level programming languages for performance and eliminating error checks perceived as too expensive before even having a validated program in which they could look systematically for performance bottlenecks. The consequences are a higher software development effort and more mistakes, leading to less reliable scientific results. Both could be measured in principle, by comparing different software projects using different approaches, but such an evaluation is expensive and in practice almost never done.

As I already mentioned, the main utility of the technical debt metaphor is to remind scientists, science managers, and funding agencies of the long-term consequences of technical choices. On closer inspection, almost every technical choice is associated with some kind of debt, especially when dealing with cutting-edge technology, which is frequent in research. It's useful to analyze major choices in terms of the debt metaphor: Is the debt perpetual, or will it be paid back? What are the interest payments? Is there a chance we'll have to default on the debt? And if so, will we get away with it? The idea is to turn the tacit contract about technical debt with your future self into an explicit one.

Any analysis of the technical debt involved in a typical research project makes the importance of infrastructure evident. Infrastructure is everything not specifically made for one research project. In computational science, it includes shared equipment such as supercomputers but also software made for facilitating research rather than directly conducting it. This includes systems software (operating systems, compilers), programming languages, scientific libraries, and software development tools. For scientists preparing a research project, all of these items represent debt-laden dependencies. The more stable and predictable the computational infrastructure is, the less risky these dependencies are. This ought to be sufficient motivation for science funders to invest in infrastructure. Fortunately, this is starting to happen.

Another good investment for the prevention of debt escalation is education and training. As I've shown, much debt is the result of uninformed choices. In the ideal world, computational scientists would be better prepared to make technical choices, either through better personal education about computing technology or by close collaboration with experts giving advice. Reading *CiSE* is, of course, a good way to improve your technical competence. We also see grassroots movements such as Software Carpentry (http://software-carpentry.org), people who step in for the academic institutions that have failed so far to integrate computational education into the training of young scientists. With a bit of luck, we could avoid a scientific debt crisis. ■

References

1. M. Turk, "Vertical Integration," *Computing in Science & Eng.*, vol. 17, no. 1, 2015, pp. 64–66.
2. D. Sculley et al., "Machine Learning: The High Interest Credit Card of Technical Debt," *Proc. SE4ML: Software Eng. Machine Learning* (NIPS 2014 Workshop), 2014; http://research.google.com/pubs/pub43146.html.
3. Z. Merali, "Computational Science: ...Error," *Nature*, vol. 467, 2010, pp. 775–777.
4. L.N. Joppa et al., "Troubling Trends in Scientific Software Use," *Science*, vol. 340, 2013, pp. 814–815.
5. D.E. Knuth, "Computer Programming as an Art," *Comm. ACM*, vol. 17, 1974, pp. 667–673.

Konrad Hinsen is a researcher at the Centre de Biophysique Moléculaire in Orléans (France) and at the Synchrotron Soleil in Saint Aubin (France). His research interests include protein structure and dynamics and scientific computing. Hinsen has a PhD in theoretical physics from RWTH Aachen University (Germany). Contact him at konrad.hinsen@cnrs-orleans.fr.

This article originally appeared in
Computing in Science & Engineering,
vol. 17, no. 6, 2015.

High-Tech Careers: Finding the Job You Want

As an Intel enterprise architect and technology strategist, Enrique G. Castro-Leon knows a thing or two about computer technology careers. With research interests that include cloud computing and IT-enabled enterprise service innovation, he recently co-authored a guest editors' introduction for the November–December 2015 *IT Professional* special issue on smart systems.

We asked Castro-Leon several questions about computer-related career opportunities, and he shared insights into how to prepare for the rapid changes taking place in technology today.

ComputingEdge: What careers in computing technology will see the most growth in the next several years, and why?

Castro-Leon: Selecting a field because it's in demand or because it's easy is not a formula for success. The student must select a general field she or he would love to work in, keeping in mind that society is moving from a product-oriented to a service-oriented paradigm. In economics, we talk about societies moving from extractive and manufacturing industries to services. Computer technology reflects this as well. There are probably more opportunities for innovation—integrating components and services that already exist to create other services—than for creating the technology components in the first place. In any case, even component creators will need to reach out to other stakeholders, which requires networking skills.

ComputingEdge: What do you consider to be the best strategies for professional networking, and why?

Castro-Leon: Recent graduates may feel discouraged when starting to network. They might think "I don't know how to do it" or "Nothing ever comes back." However, this is only a short-term problem. Here's a helpful analogy. Every leader is like a lighthouse beaming out light. Most of the light gets lost. That's a reality of life. However, some can touch and transform lives. The sender never knows when a beam lands on a ship, but when this happens, the ship uses the light to avoid an accident. Even this is successful networking to

 Published by the IEEE Computer Society

me. Just a handful of ships' crews might thank the lighthouse keeper, but counting only these as successes constitutes a narrow view. There's no right or wrong here. Ultimately it's up to the individual to decide what the networking goal should be. There are two considerations: networking should be multimodal, and it's important to have a role progression from observer to participant to leader. Multimodal means combining traditional personal relationships with both writing and using various social media. It's important not to get discouraged. We get better with practice. These are useful skills.

ComputingEdge: What advice would you give college students to provide them with an advantage over the competition?

Castro-Leon: Professionals with the best chances of success are T-shaped professionals. This is a well-known term in service science. It refers to people who have deep expertise in one area but who also have strong interdisciplinary skills and a level of comfort collaborating with experts in other areas. The lone inventor is I-shaped, does not fit well in an integration society, and is unable to build bridges to other people to make wonderful things happen.

There is no negative in changing careers, as long as the events are part of a coherent strategy.

ComputingEdge: What advice would you give people changing careers midstream?

Castro-Leon: There is no negative in changing careers, as long as the events are part of a coherent strategy. This change can be used as a strength. Give some thought to finding complementary angles. Music and engineering or law and engineering are not necessarily incompatible. In fact, I recommend that a T-shaped person engage in a totally different activity. It brings a broader perspective that can only improve chances of success, even if it's not done professionally. For instance, I'm an electrical engineer and computer scientist by training, but I spend quality time playing classical piano. I love it.

ComputingEdge: What should applicants keep in mind when applying for computer tech jobs?

Castro-Leon: A primary consideration is to break the apparent chicken-and-egg problem of job postings requiring experience but the applicant needing a job to get experience. For students, it's important to have reasonably good grades, but it's not optimal to study to the exclusion of everything else. Participating in certain extracurricular activities, such as journalism, will provide good opportunities to learn communication skills and earn points on the résumé. Seek people who are positive role models. Seek opportunities for giving. Giving could be helping a friend in true need or volunteering. Worrying about the "me" part all the time clouds the mind.

ComputingEdge's Lori Cameron interviewed Castro-Leon for this article. Contact her at l.cameron@computer.org if you would like to contribute to a future *ComputingEdge* article on computing careers. Contact Castro-Leon at enrique.g.castro-leon@intel.com.

CLOUDERA, INC. is recruiting for our Palo Alto, CA office: Build Technical Customer Success Manager: architect enterprise data solutions for large organizations. Mail resume w/job code #36728 to: Cloudera, Attn.: HR, 1001 Page Mill Rd., Bldg. 2, Palo Alto, CA 94304.

CLOUDERA, INC. is recruiting for our New York, NY office: Professional Services Practice Director: manage sales & delivery of professional srvcs (consulting) to Cloudera customers for a defined regional territory. Recruit, retain, manage and mentor a team of technical consultants. Mail resume w/job code #34478 to: Cloudera, Attn.: HR, 1001 Page Mill Rd., Bldg. 2, Palo Alto, CA 94304.

CLOUDERA, INC. is recruiting for our Palo Alto, CA office: Build and Release Engineer: analyze build failures & reduce build failure occurring due to non-product code issues, periodically review the feedback from developers & testers, & deliver ongoing improvement. Mail resume w/job code #36907 to: Cloudera, Attn.: HR, 1001 Page Mill Rd., Bldg. 2, Palo Alto, CA 94304.

BLACKBOARD SYSTEM ADMINISTRATOR: devel. & maintain global LMS; LMS environ. configuration & modification; customize LMS modules; use Oracle, SQL, Unix, JavaScript, and HTML. MS in CS or related + 2 yrs of exp. OR BS+5. Email sarah.ulep@laureate.net w/ Job#10786 in subj. line. Laureate Education, Inc. 7080 Samuel Morse Dr., Columbia, MD 21046. EOE.

JAVA DEVELOPER (E-COMMERCE). Des./dev./implement/test procurement/e-commerce software. Bach. degree (Computer Science) or higher req'd. Min. 2 years' exp. in programmer analyst or software dev. posn's req'd. Prior exp. must incl. dev. using Java 6 language & Hibernate framework. InnerWorkings, Inc., Chicago, IL. Resumes to: Recruiting, InnerWorkings, Inc., 600 West Chicago Avenue, Suite 850, Chicago, IL 60654.

CLOUDERA, INC. is recruiting for our Palo Alto, CA office: Solutions Architect: work on core products by contributing code changes to those products. Travel Required. Mail resume w/job code #36996 to: Cloudera, Attn.: HR, 1001 Page Mill Rd., Bldg. 2, Palo Alto, CA 94304.

PROGRAMMER ANALYST: Design and develop advanced solutions for software applications using knowledge in Apex, Visual Force, Data Loader, HTML, JavaScript, CSS, Workflow rules & Approvals, Reports, Custom Objects, Security Controls, Sandbox data Loading, Data Loader, Custom third party apps configuration and management, REST API and SOAP API based integration. Must be willing to travel & reloc to unanticipated client locations throughout the US. Reqs MS in comp sci, eng or rel. Mail

Cisco Systems, Inc. is accepting resumes for the following positions:

BELLEVUE, WA: Technical Marketing Engineer (Ref.# BEL8): Responsible for enlarging company's market and increasing revenue by marketing, supporting, and promoting company's technology to customers. Travel may be required to various unanticipated locations throughout the United States.

BOXBOROUGH, MA: Customer Support Engineer (Ref.# BOX7): Responsible for providing technical support regarding the company's proprietary systems and software. **Network Consulting Engineer (Ref.# BOX11):** Responsible for the support and delivery of Advanced Services to company's major accounts.

COLUMBIA, MD: Software Engineer (Ref.# COLU1): Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of networking software. **Software Development Manager (Ref.# COLU4):** Lead a team in the design and development of company's hardware or software products.

FORT LAUDERDALE, FL: Software Engineer (Ref.# FL11): Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of networking software. Telecommuting permitted.

ISELIN/EDISON, NJ: Network Consulting Engineer (Ref#: ED9): Responsible for the support and delivery of Advanced Services to company's major accounts. Telecommuting permitted.

JACKSONVILLE, FL: Solutions Architect (Ref.# JAC1): Responsible for IT advisory and technical consulting services development and delivery.

RESEARCH TRIANGLE PARK, NC: Systems Engineer (Ref.# RTP355): Provide business-level guidance to the account team or operation on technology trends and competitive threats, both at a technical and business level. Telecommuting permitted. **Software Engineer (Ref.# RTP3):** Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of networking software. **Network Consulting Engineer (Ref.# RTP2):** Responsible for the support and delivery of Advanced Services to company's major accounts. **Product Manager (Ref.# RTP621):** Create high level marketing strategies and concepts for company solutions for markets and segments worldwide. **IT Manager (Ref.# RTP108):** Design, architect and implement Data Center infrastructure. **Software/QA Engineer (Ref.# RTP4):** Debug software products through the use of systematic tests to develop, apply, and maintain quality standards for company products.

RICHARDSON, TX: Customer Support Engineer (Ref.# RIC1): Responsible for providing technical support regarding the company's proprietary systems and software. **Manager, Technical Services (Ref.# RIC18):** Responsible for leading a team in the delivery of world-class customer support on a line of products or for a targeted group of customers. Telecommuting permitted. **Product Manager, Engineering (Ref.# RIC126):** Responsible for managing the development and implementation of new product introduction engineering activities to meet production launch schedules, quality and cost objectives.

RICHFIELD, OH: Hardware Engineer (Ref.# RICH2): Responsible for the specification, design, development, test, enhancement, and sustaining of networking hardware.

RICHMOND, VA: Network Consulting Engineer (Ref.# RIV21): Responsible for the support and delivery of Advanced Services to company's major accounts. Telecommuting permitted.

SAN DIEGO, CA: Engineering Architect (Ref.# SD3): Work on the cutting edge of a wide range of innovative company's WebEx uses cases.

SAN JOSE/MILPITAS/SANTA CLARA, CA: Engineering Architect (Ref.# SJ851): Responsible for understanding and translating customer requirements combined with state of the art for company's technologies into innovative engineering solutions and products. **Network Consulting Engineer (Ref.# SJ9):** Responsible for the support and delivery of Advanced Services to company's major accounts.

PLEASE MAIL RESUMES WITH REFERENCE NUMBER TO CISCO SYSTEMS, INC., ATTN: M51H, 170 W. Tasman Drive, Mail Stop: SJC 5/1/4, San Jose, CA 95134. No phone calls please. Must be legally authorized to work in the U.S. without sponsorship. EOE.

www.cisco.com

 2376-113X/16/$33.00

resumes to Keypixel Software Solutions LLC, 777 Washington Rd Suite 1 Parlin NJ 08859.

SR. TECHNICAL ANALYSTS L2 sought by IT firm in Herndon, VA. Qualified candidates will have Master degree in Comp. Sci. or related field and 36 months as Technical Analyst or similar position. Experience with Java, JEE, Oracle, Conceptwave 4.1, Eclipse, Weblogic, RSA, Linux, Spring, SOAP Web-Services and Maven required. Send resumes to: Apptium Technologies, LLC. 12950 WorldGate Dr., Ste. 710, Herndon, VA 20170.

SR. DATA ANALYTICS DEVELOPER (OBIEE). Ascension Health-IS, Inc. is seeking a full-time Sr. Data Analytics Developer (OBIEE) in St. Louis, Missouri to work with users to define new application requirements and resolve project issues; code, design and develop data warehouse/analytics toolsets; support toolsets using OBIEE; troubleshoot applications and datasets; monitor and maintain installed systems; and research solutions and technology. Contact Jenna Mihm, Vice President Legal Services & Associate General Counsel, Ascension Health, 4600 Edmundson Road, St. Louis, MO 63134, 314-733-8692, Jenna.Mihm@ascensionhealth.org To apply for this position, please reference Job Number 02.

DATA WAREHOUSE AND ANALYTICS DEVELOPERS (ETL/INFORMATICA) Ascension Health-IS, Inc. is seeking two Data Warehouse and Analytics Developers (ETL/Informatica) in St. Louis, Missouri to code design and development on the data warehouse/analytics Extract Transform Load (ETL) toolset, Informatica PowerCenter; support Informatica toolset; integrate and develop other technologies. Research solutions and technology; participate in testing (e.g. user acceptance testing, unit, system, regression, integration testing); develop test plans and documentation; debug code. Contact Jenna Mihm, Vice President Legal Services & Associate General Counsel, Ascension Health, 4600 Edmundson Road, St. Louis, MO 63134, 314-733-8692, Jenna.Mihm@ascensionhealth.org To apply for this position, please reference Job Number 03.

SENIOR SYSTEM ENGINEER LEAD sought in San Diego, CA area. Master's deg in Comp Sci or related field, & 36 months of exp reqd. Mail resume to Veterans EZ Info Inc, 1901 1st Ave, Ste 192, San Diego, CA 92101.

JUNIPER NETWORKS

Juniper Networks is recruiting for our Sunnyvale, CA office:

Test Engineer #30272: Design, develop and implement testing methods to validate the implementation of the DWDM, Coherent OTN Packet Optical functionality, with focus on system architecture of the product to fit Service Providers' deployments and equipment for all phases of product development and manufacturing.

Resident Engineer Senior Staff #2432: Hold design and information gathering workshops with the customer to understand the customer's existing network design and technical requirements. Assist account and sales team with technical activities in new and existing opportunities. May work at other undetermined locations throughout the U.S.

Software Engineer #31412: Design, develop, troubleshoot and maintain networking, kernel, TCP and IP solutions on Junos, which is a flavor of the FreeBSD UNIX operating System.

Senior Systems Engineer #9050: Develop and deliver detailed technical sales support to customers and business partners. Identify clients and conduct pre-sales meetings and presentations with customers and business partners to demonstrate and showcase company's product portfolio, solutions and services. May work at other undetermined locations throughout the U.S. Travel required. Telecommuting allowed. Fluent in Spanish and English required.

Software Engineer Staff 19474: Design, develop, troubleshoot, debug and implement features and enhancements for Software Defined Network (SDN) solutions in an agile, fast paced manner.

Software Engineer #30892: Design, develop, troubleshoot and debug new tools for mobility services, and software to test company routers.

Software Engineer #27757: Design, develop and debug kernel networking features for JUNOS. Work on high availability features.

Technical Support Engineer #35316: Provide tech support and deliver diagnostics and root-cause analysis for network impacting issues on Juniper routing products to large ISPs & enterprise customers.

Software Engineer #29366: Develop detailed software functional and design specifications.

ASIC Engineer #26133: Perform ASIC verification for large, complex high-speed ASICs for Juniper's next generation of networking products.

Software Engineer #29956: Review Software Functional specification and Unit Tests. Develop Automation framework to support smoke tests and unit test scripting infrastructure.

Software Engineer #29427: Analyze, design, develop, debug, and modify JUNOS features. Interact with PLM to refine requirements. Work with Systest team to fix defects.

Software Engineer Staff #25168: Develop software for Juniper's JUNOS packet forwarding engine. Capture customer requirements and translate them to customer visible features.

Functional Sys Analyst Staff #35757: Partners with business on FICO track. Understands SAP functionality to satisfy business requirements and solve complex business problems.

Solutions Architect #6172: Define, design, and develop solution architecture specifications and improvements that address market and customer requirements and be implemented using company products and solutions.

Technical Support Engineer #37046: Provide technical support to large Internet Service Providers and/or enterprise customers using high level expertise of company specific products.

Software Engineer #17501: Design, develop, implement, troubleshoot, and debug application enhancements and functionality in company's server manager product. Re-design and implement software features and tools in support of the product's infrastructure and platform. Assist in field and customer support of server manager product.

Test Engineer #14613: Responsible for developing and supporting extended unit testing frameworks, writing test plan and test cases to test the various features provided by the operating system.

Juniper Networks is recruiting for our Westford, MA office:

ASIC Engineer #35189: Author test plans for ASIC block level designs. Design and code test bench and test suites in System Verilog and under Unified Verification Methodology (UV M).

Mail single-sided resume with job code # to
Juniper Networks
Attn: MS A.8.429A
1133 Innovation Way
Sunnyvale, CA 94089

Apple Inc. has the following job opportunities in Cupertino, CA:

Hardware Development Engineer (REQ#9CYU46). Dsgn & dev HW for iPhone. Sys intgrtn of analog & digital elctrncs from concept through prdctn. Travel Reqd 25%.

Localization Engineer (REQ# 9EZW5X). Respon for trans, proof, & edit sw products, web content, user interface elements & rel materials from Eng to Danish. Fluency in written & spoken Danish.

Hardware Development Engineer (REQ#9PWUZ3) Dsgn & dvlp pwr electronics & controls.

Software Development Engineer (REQ#9A639A). Dev., debug and test code using Object Oriented concepts.

Software Engineer, Security & Compliance (REQ#9Q6PYK) Resp for the overall security & compliance of the SW apps.

Software Development Engineer (REQ#9BFUZV) Work w/ a team of cellular protocol test & dvlpmnt engineers to help deliver HQ prdct releases.

Software Development Engineer (REQ#9R6TC9) Dsgn & dvlp SW for user mngmnt, workflow, and ticketing syss.

Hardware Development Engineer (REQ#9RGSDY). Desgn, dev & validate electrical HW for iPhone systs, w/ emphasis on Power Management syst. Travel req'd: 15%.

Firmware Manager (REQ#9CYPMM) Mnge an algorithms team in spprt of nxt gen input devices.

ASIC Design Engineer (REQ# 9M2PSZ). Create FPGA-based prototyping syst for bringup and verification/ validation tasks.

Software Quality Assurance Engineer (REQ#9LM3CA). Test multiple srvcs in distrib envt using web & srver-side test methods for the Apple Online Store.

Automation Software Engineer (REQ# 9QJ39D). Des & dev SW tools for Factory Autom Sys to be used in Apple's Manuf process. Travel Req'd: 25%.

Human Interface Designer (REQ# 9D6N27) Conceive, dsgn, & dvlp future enhancements to Apple's prdct & UI experiences.

Hardware Development Engineer (REQ#9FM3RV) Research, dsgn, dvlp, & launch next-gen Sensor Technlgys in Apple prdcts.

Software Engineer Applications (REQ#A2L3AT) Dsgn & implmnt SW & tools for eCommerce syss using the Spring app framework stack.

Mechanical Design Engineering Manager (REQ#9EH3UE) Set up & mnge supply chain to support new prdct dvlpmnt & launch of Apple prdcts. Travel req. 35%.

Software Engineer Applications (REQ#9AYURM). Build and maintain content mgmt. sys. for iTunes Content.

Hardware Development Engineer (REQ#9GX2SK). Dev. new display tech. and implement critical tech. platforms to improve display optical performance.

Systems Design Engineer (REQ# 9FTSZA). Des instrumentation sys for input devices & sensors. Travel req'd 20%.

Software Engineer Applications (REQ#9U229F) Dsgn & Dvlp web service APIs for use by internal clients.

Software Engineer Systems (REQ#9F23FW). Support res & dev of comp vision for mobile devices.

Firmware Engineer (REQ#9LUSCB) Des & dev firmware for wireless audio products.

Software Development Engineer (REQ#9FTNRY). Support new prod intro test environ for factory test. Travel req 35%.

Label Relations Specialist (REQ# 9C33CE). Respons for managing rltshps & max sales w/ direct indep label partners & artist mgrs in the digital music space.

Hardware Development Engineer (REQ#9JFPC3). Des, dev, & launch next-gen Sensing Tech. Travel req'd: 25%.

Senior Software Engineer Applications (REQ#9J7MUX) Architect, Des & dev web & iOS solutions for Apple Retail, Sales and Marketing & wrk across multi proj's.

Software Development Engineer (REQ#9TD3H2). Deliver high quality, polished web apps. that are intuitive and easy to use.

ASIC Design Engineer (REQ# 9MWV5Y). Design complex CPU and SOC microprocessors.

Product Design Engineer (REQ#9USPEH). Design and optimize products affected by air flow, liquid flow, convective heat transfer, conductive heat, and radiative heat transfer using analytical techniques and computational fluid dynamics SW.

Software Engineer Applications (REQ#A5447A). Des & dev SW & tools for large-scale system op's & deployment automation.

Hardware Development Engineer (REQ#9E6374) Dsgn, dvlp, & launch the nxt-gen display and panel dsgn techngly for Apple prdcts. Travel req 20%.

Software Development Engineer (REQ#9FE3WJ) Dsgn, dvlp, & debug SW for intelligent personal assistnt SW (Siri) on mobile dvices.

Software Engineer Applications (REQ#9M4UB6) Run test automation for UI as well as services.

Software Development Manager (REQ#9V4U6W). Lead a team to test, assess, and improve perf. and scalability of Siri.

ASIC Design Engineers (REQ# 9HCV9G) Dvlp DFT logic & insert Scan related logic in a SOC Dsgn flow.

Software Engineer Applications (REQ#9UNTUW) Contribute to building and maintaining REST APIs & implmnt security features.

Systems Design Engineer (REQ# 9EFUXM) [Multiple Positions Open].

Test cell teleph. functionality of iOS devices. Travel req'd: 30%.

Software Development Engineer (REQ#9GHU7V). Des & dev SW & FW for an 802.11 Wi-Fi stack running on mbl pltfrm.

Software Engineer Applications (REQ#9T5UF2). Dev info security tools w/ a focus on infrastructure security.

Technical Program Lead (Operations) (REQ#9D9PVR). Dev & spprt new product intrdctns. Dev final assmbly processes for new prgrms incldng fixtures & automation. Travel Req'd 30%

Software Engineer Applications (REQ#9WY3KP) Dsgn & dev large scale distributed systms to support workflow engines that process & compute large amount of data in various domains.

Engineering Project Lead (REQ# 9TU6N8). Respon for develop aspects of sw projs for Fin & Admin Sys.

Software Development Engineer (REQ#9ZM2F2). Des & dev a digi prsnl asst for mobile dvcs.

Software Development Engineer (REQ#9F7TFL). Rsrch, dsgn, dvlp & maintain Apple's compiler tech.

ASIC Design Engineer (REQ# 9TN2GT). Implement physical design of partitions for highly complex SOC utilizing state of the art process tech.

Information Systems Engineer (REQ#9WEQ7U). Design and dev. scalable, high perf. portal solutions using J2EE and other tech.

Senior Software Engineer (REQ# 9ZTVTH). Dsgn, dev & maintain data ingest/export jobs between relational DBs & hadoop dstrbtd file sys using Apache Sqoop.

ASIC Design Engineer (REQ#9F8T2J) Implmnt complex, high prfmnce & low power microprocessor (CPU) units using gate-lvl logic dsgn, P&R, and HDL synthsis.

Product Design Engineer (REQ# 9VTP2P). Des, test & validate prod cooling systems.

Software Engineer Applications (REQ#9EGQE9). Des & dev rich web apps. Define & architect app prog interfaces.

Software Development Engineer (REQ#9M2PRA) Resp for characterizing & evaluating the end-to-end infrastrctre perfmnce in supporting of newest prdct.

Software Development Engineer (REQ#9N5T7N) Dsgn & implmnt device drivers for peripheral devices across all iOS HW pltfrms.

Software Development Engineer (REQ#9SS5DB) Dsgn & dvlp highly scalable, performant & reliable Java/ J2EE apps for consumer apps.

Engineering Project Manager (REQ# 9QF285) Display Module EPMs define over proj plan & set the prog display qual strategy.

Apple Inc. has the following job opportunities in Culver City, CA:

Mechanical Design Engineer (REQ# 9LUSCQ). Design & develop new consumer audio products. Travel req'd: 25%.

Refer to Req# & mail resume to Apple Inc., ATTN: L.J. 1 Infinite Loop 104-1GM Cupertino, CA 95014.

Apple is an EOE/AA m/f/ disability/vets.

ERICCSON INC. has an opening for the position of: **ENGINEER-RESEARCH** _opening in **PLANO, TX** to develop and integrate Proof Of Concept (POC) projects for the realization of ideas and demonstrate feasibility of the concept. Reference #: 15-TX-2617. A**PPLICATION SUPPORT ANALYST** _opening in **OVERLAND PARK, KS** to ensure that the application is performing adequately & enough capacity is available to meet the business requirements & growth projections of the customer. Reference #: 15-KS-2573. **DATABASE ANALYST** _opening in **ATLANTA, GA** to support issues, outages, & debugging matters of the production support in an Oracle environment. Reference # 15-GA-3615. **ENGINEER - SERVICES SOFTWARE**_ opening in **ATLANTA, GA** to conduct data migration projects and develop and design in accordance with customer requirements. Reference #: 15-GA-3476. **ENGINEER - SOFTWARE** _ opening in **SAN JOSE, CA** to perform system verification of developed & legacy functionality on real equipment for IP routers. Reference #: 15-CA-2908. **ENGINEER - SOFTWARE** _ opening in **SAN JOSE, CA** to develop IP Operating Systems routing software to meet the needs of a diverse set of platforms and applications powering networks. Reference #: 15-CA-3007. **ENGINEER SOFTWARE SERVICES** _ opening in **BELLEVUE, WA** to drive sales strategies and manage solution design and delivery. Reference #: 16-WA-2936. **ENGINEER – SERVICES SOFTWARE** _ opening in **BELLEVUE, WA** to analyze, prepare, implement and verify the configuration and integration of a node, network or system. Reference #: 16-WA-3497. **ACCOUNT MANAGER** _ opening in **HERNDON, VA** to lead sales team in lead generation, proposals, commercial negotiations, & close of sale. 30% domestic/international travel required. Reference #: 15-VA-1433. To apply, please mail resume & include applying for appropriate reference # to: Ericsson Inc. 6300 Legacy Dr., R1-C12, Plano, TX 75024.

DATA SYSTEMS ANALYSTS. Multiple positions available in Amherst, MA. Build services from data to enable enterprise-wide, data-driven decision making. Participate in project setup, discussion of approaches and methods, data strategies, and extract transform and load (ETL) processes. Carry out statistical analysis and programming in R and Python. Design and build visualizations and collaborate across functions and business units. Direct applications to: ATTN: L. Sawtelle, MIP F105, Massachusetts Mutual Life Insurance Company, 1295 State Street, Springfield, MA 01111; Please Reference Job ID 708202100.

WYDE CORP. has multi openings at various levels for the following positions at its office in Bloomington, MN & unanticipated client sites thr/o the US 1. Business Analyst* - Conduct Org. studies & recommend IT solutions. 2. SW Developer* - Design, develop & modify s/w sys. 3. SW Developer Mgr – Manage s/w development project 4. SW Architect* - Develop IT architecture solution 5. SW Architect Mgr – Manage IT architecture development projects 6. Project Manager – Plan & manage project execution. Must have a Bachelor/equiv and prior rel. exp, Master/equiv, or Master/equiv and prior rel. exp. Edu/exp req vary depending on position level/type. Managerial and *Lead positions in this occupation must have Master/equiv+2yr or Bach/equiv+5yr progressive exp. Travel/relo req. Send resume & applied position to: Kristen Kaul, HR North America, Wyde Inc. 3600 American Blvd. W., Suite 330, Bloomington, MN 55431.

SENIOR SERVICES SPECIALIST (NY, NY and unanticipated client sites in US) Provide security & compliance & architecture consulting. Architect & implement security & compliance system controls. Collaborate with partners to distribute & promote products & services. REQS: 5 yrs exp in job &/or rel occup. Must have exp w/ CA Data Protection; Collaborating with channel partners; CA Single Sign On; CA Privileged Indentity Manager. Frequent travel to unanticipated client sites throughout the US; Work from home anywhere in the US. Send resume to: Althea Wilson, CA Technologies, One CA Plaza, Islandia, NY 11749, Refer to Requisition # 118862

ERICSSON INC. has openings for positions in **Plano, TX: SOLUTIONS ARCHITECT** _ to define, analyze & manage customer requirements utilizing Ericsson's OSS & BSS portfolio. Up to 20% domestic travel required. Job ID: 16-TX-3091. **RF ENGINEER** _ responsible for interaction & coordination with RF Eng, Natl. RF Tech Team, Switch Operations, Field Operations, Ntwk Development, & Device Development. Telecommuting is available for this position from anywhere in the US. Job ID: 16-TX-635. **ENGINEER – SERVICES SOFTWARE** _ to participate in software loading, configuration, integration, verification, and troubleshooting of existing solutions. Requires 20% of domestic & international travel. Job ID: 16-TX-3571. **ENGINEER-RESEARCH** _ to develop & integrate Proof Of Concept (POC) projects for the realization of ideas & demonstrate feasibility of the concept. Job ID: 15-TX-2617. **BUSINESS CONSULTANT** _ to support Ericsson business units bridge the gap betw' strategy & implementation by driving the projects focused on Ericsson growth initiatives & provide business case, modeling, & strategic analysis on a variety of projects. Job ID: 15-TX-2620. **PROJECT MANAGER** _ for scheduling, tracking, & implementation of projects supporting key customer deliverables to the highest customer satisfaction, while driving cost, quality, & timeliness. Job ID: 16-TX-2659. To apply please mail resume to Ericsson Inc. 6300 Legacy Dr, R1-C12 Plano, TX 75024 & indicate appropriate Job ID. To apply please mail resume to Ericsson Inc. 6300 Legacy Dr, R1-C12 Plano, TX 75024 & indicate appropriate Job ID.

SENIOR SOFTWARE DEVELOPER: Design & develop new computer software. Work with & supervise developers to ensure seamless integration between backend application & the foreground web application. Liaise with & supervise technical staff to explore & suggest strategic technical solutions for the development of location & mapping technology. Supervise test & Operations teams to troubleshooting & resolve issues throughout the lifecycle. Use technologies being utilized in product development such as Dart, JavaSCript, Java, HTMLS, CSS3 as well as client/server application processes and multi media and internet technology. Use IOT and Building Intelligence Software and protocols such as haystack, MQTT, BACnet. Bachelors degree in Information Systems Engineering or Computer Science or Software Engineering plus 5 yrs exp. req'd. 40 hrs/wk. Job Site & Itvu: Oakland, CA. Send resume to Mr. Mazo at DGLogik, Inc. at e.mazo@dglogik.com.

SR. CONSULTANT/SYSTEMS ADMINISTRATOR F/T (Fishkill, NY) Position involves travel to various unanticipated worksites up to 100% of the time anywhere in the United States. Must have Bach deg or the foreign equiv in Electronic Engg, Engg, Electronics & Communication Engg, Comp Sci, or related with five (5) years of progressive experience Designing, Building or Fixing & Supporting integration interfaces that meets business requirements using QTP. Configuring and managing VMware on Dell, HP. Configuring & Managing Vcenter server cluster with Vsphere HA and DRSsenabled of VMotion, VSwitch and VLAN's in Vcenter server. Managing snapshots, Clones, templates during patch releases and new server deployment. Provide leadership in recommending and implementing continuous process improvement, education and training requirements to management staff. Send resume: Novisync, Inc., Recruiting (VC), 300 Westage Bus Ctr Dr, Ste 350, Fishkill, NY 12524.

CLASSIFIED LINE AD SUBMISSION DETAILS: Rates are $425.00 per column inch ($640 minimum). Eight lines per column inch and average five typeset words per line. Send copy at least one month prior to publication date to: Debbie Sims, Classified Advertising, *Computer* Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 816-2138; fax (714) 821-4010. Email: dsims@computer.org.

In order to conform to the Age Discrimination in Employment Act and to discourage age discrimination, *Computer* may reject any advertisement containing any of these phrases or similar ones: "...recent college grads...," "...1–4 years maximum experience...," "...up to 5 years experience," or "...10 years maximum experience." *Computer* reserves the right to append to any advertisement without specific notice to the advertiser. Experience ranges are suggested minimum requirements, not maximums. *Computer* assumes that since advertisers have been notified of this policy in advance, they agree that any experience requirements, whether stated as ranges or otherwise, will be construed by the reader as minimum requirements only. *Computer* encourages employers to offer salaries that are competitive, but occasionally a salary may be offered that is significantly below currently acceptable levels. In such cases the reader may wish to inquire of the employer whether extenuating circumstances apply.

SR. HYPERION CONSULTANT. Job location: Miami, FL & any other unanticipated locations in U.S. Travel Required. Duties: Design & develop Hyperion forecasting/budgeting appls. Develop hierarchies in Hyperion Data Relationship Mgmt. (DRM). Resp. for developing data processes for Hyperion appls. & designing & developing security processes for Hyperion appls. Requires: M.S. degree in Comp. Sci., Eng. or related field & 3 yrs. exp. in the job offered or 3 yrs. exp. as a Hyperion Developer or Hyperion Analyst. Will accept B.S. (or foreign equiv.) & 5 yrs. exp. in computer ind. in lieu of M.S. & 3 yrs. exp. Concurrent exp. must incl.: 3 yrs. exp. with forecasting & budgeting Hyperion appls.; 3 yrs. exp. with security in Hyperion; & 3 yrs. exp. with DRM. Send resume (no calls) to: Michelle Ramirez, The Hackett Group, Inc., 1001 Brickell Bay Dr., Suite 3000, Miami, FL 33131.

MANAGER. Job location: Miami, FL & any other unanticipated locations in U.S. Travel Required. Duties: Participate in definition, develop., & implementation of info systems based on client requirements. Assist in design, develop., & deployment of Hyperion Planning & Essbase appls. Resp. for full systems develop. lifecycle (SDLC) from requirements gathering through implement. of software arch. solutions. Develop complex financial reports & budget books for clients using Financial Reporting Studio & Oracle BI tools. Resp. for administering, automation, optimization & perform. tuning of Essbase appls., data/metadata processing, upgrading, testing & migrating of cubes between different server environs.Perform automating loading of data in the system & writing scripts to perform calcs. using calc. scripts, Load Rules, MaxL, MDX, batch & shell scripting.Requires: M.S. degree in Comp. Sci, MIS, Eng. or related field & 2 yrs. exp. in job offered or 2 yrs. exp. as a Consultant or Systems Analyst. Concurrent exp. must incl.: 2 yrs. exp. with design, develop. & deployment of Hyperion Planning & Essbase appls. & 2 yrs. exp. developing complex financial reports using Financial Reporting Studio. Send resume (no calls) to: Michelle Ramirez, The Hackett Group, Inc., 1001 Brickell Bay Dr., Suite 3000, Miami, FL 33131.

SANDISK CORPORATION has openings in San Jose, California for Staff Systems Design Engineers to define processes for technical platforms, system specifications, and input/output and working parameters for hardware and software compatibility. Job code: SD582. To apply, reference job code # & mail resume to: SanDisk Corporation, 951 SanDisk Drive, MS: HRGM, Milpitas, CA 95035. EOE.

ADVERTISER INFORMATION • FEBRUARY 2016

Advertising Personnel

Debbie Sims: Advertising Coordinator
Email: dsims@computer.org
Phone: +1 714 816 2138 | Fax: +1 714 821 4010

Chris Ruoff: Senior Sales Manager
Email: cruoff@computer.org
Phone: +1 714 816 2168 | Fax: +1 714 821 4010

Advertising Sales Representatives (display)

Central, Northwest, Southeast, Far East:
Eric Kincaid
Email: e.kincaid@computer.org
Phone: +1 214 673 3742
Fax: +1 888 886 8599

Northeast, Midwest, Europe, Middle East:
David Schissler
Email: d.schissler@computer.org
Phone: +1 508 394 4026
Fax: +1 508 394 1707

Southwest, California:
Mike Hughes
Email: mikehughes@computer.org
Phone: +1 805 529 6790

Advertising Sales Representative (Classifieds & Jobs Board)

Heather Buonadies
Email: h.buonadies@computer.org
Phone: +1 201 887 1703

Help build the next generation of systems behind Facebook's products.

Facebook, Inc.

currently has the following openings in **Menlo Park, CA (various levels/types)**:

Solutions Engineer (SEJ) Combine technical & business skills to make our partners successful & improve Facebook platform. **Data Engineer (3584J)** Build data solutions that help product & business teams at Facebook to make data driven decisions. **UX Researcher (6450J)** Oversee & design the user experience component to generate actionable insights. **Development Operations Engineer (DevOps) (4021J)** Innovate, develop, & operate the next generation of Facebook's Internal Infrastructure serving the productivity needs of the entire company. **Technology Partner (3606J)** Ability to understand revenue recognition rules & interpret & configure those in to revenue applications. **Research Manager (1602J)** Drive the overall strategy & operations of the research team at Facebook, ensuring highly relevant, timely, & effective research. **Security Engineer (3499J)** Research & develop internet security protocols/standards & get them adopted.

Mail resume to: Facebook, Inc. Attn: SB-GIM, 1 Hacker Way, Menlo Park, CA 94025. Must reference job title & job# shown above, when applying.

WhatsApp, Inc.

currently has the following openings in **Mountain View, CA (various levels/types)**:

Mobile Software Developer (5758J) Design and develop software applications for mobile message products.

Mail resume to: WhatsApp, Inc. c/o Facebook Inc. Attn: SB-GMI, 1 Hacker Way, Menlo Park, CA 94025.
Must reference job title and job# shown above, when applying.

TECHNOLOGY

Intuit Inc.

has openings for the following positions in **Santa Clara County, including Mountain View, California** or any office within normal commuting distance: **Data Scientists (Job code: I-537):** Provide guidance and support to Business leaders and stakeholders on how best to harness available data in support of critical business needs and goals. Participate in the full cycle of iterative big data exploration, including hypothesis formulation, algorithm development, data cleansing and testing. **Staff Application Operations Engineers (Job code: I-1828):** Design and develop new software applications, services, features and enhancements, and maintain existing software products. **Product Managers (Job code: I-966):** Gathering requirements, use cases and functional specifications for data products for internal and external customers of Intuit. Prioritize customer needs, analyze landscape and develop product roadmaps for products aligned with Quickbooks data strategy.

Positions located in **San Diego, California**: **Senior Data Engineers (Job code: I-141):** Design, develop, and implement data movement and integration processes in preparation for analysis, data warehousing, and operational data stores, involving very large quantities of data. **Managers, Development (Job code: I-138):** Apply a full understanding of the business, the customer, and the solutions that a business offers to effectively design, develop, and implement operational capabilities, tools and processes that enable highly available, scalable & reliable customer experiences. **Senior Systems Engineers (Job code: I-124):** Exercise senior level knowledge in selecting methods and techniques to design, implement, and maintain servers for Intuit's leading commercial software products. Work on problems of complex scope where analysis of data requires evaluation of multiple factors of the overall product and service.

Positions located in **Plano, Texas**: **Managers 3-Group Research & Analysis (Job code: I-103):** Lead and develop an expanded team of business analysts, technical data analysts and data scientists to provide timely and effective insights to business teams and act as a trusted business partner.

To apply, submit resume to Intuit Inc., Attn: Olivia Sawyer, J203-6, 2800 E. Commerce Center Place, Tucson, AZ 85706.
You must include the job code on your resume/cover letter. Intuit supports workforce diversity.

NEW IN 2015

IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS

SCOPE

The *IEEE Transactions on Multi-Scale Computing Systems* (*TMSCS*) is a peer-reviewed publication devoted to computing systems that exploit multi-scale and multi-functionality. These systems consist of computational modules that utilize diverse implementation scales (from micro down to the nano scale) and heterogeneous hardware and software functionalities; moreover, these modules can be based on operating principles and models that are valid within but not necessarily across their respective scales and computational domains. Contributions to TMSCS must address computation of information and data at higher system-levels for processing by digital and emerging domains. These computing systems can also rely on diverse frameworks based on paradigms at molecular, quantum and other physical, chemical and biological levels. Innovative techniques such as inexact computing, management/optimization of smart infrastructures and neuromorphic modules are also considered within scope.

This publication covers pure research and applications within novel topics related to high performance computing, computational sustainability, storage organization and efficient algorithmic information distribution/processing; articles dealing with hardware/software implementations (functional units, architectures and algorithms), multi-scale modeling and simulation, mathematical models and designs across multiple scaling domains and functions are encouraged. Novel solutions based on digital and non-traditional emerging paradigms are sought for improving performance and efficiency in computation. Contributions on related topics would also be considered for publication.

SUBSCRIBE AND SUBMIT

For more information on paper submission, featured articles, call-for-papers, and subscription links visit:

www.computer.org/tmscs

IEEE computer society | IEEE COMMUNICATIONS SOCIETY | IEEE NANO Nanotechnology Council® | CEDA IEEE Council on Electronic Design Automation | IEEE

IEEE Cloud Computing Call for Papers

Although cloud technologies have been advanced and adopted at an astonishing pace, much work remains. *IEEE Cloud Computing* seeks to foster the evolution of cloud computing and provide a forum for reporting original research, exchanging experiences, and developing best practices.

IEEE Cloud Computing magazine seeks accessible, useful papers on the latest peer-reviewed developments in cloud computing. Topics include, but aren't limited to:

- Cloud architectures (delivery models and deployments),
- Cloud management (balancing automation and robustness with monitoring and maintenance),
- Cloud security and privacy (issues stemming from technology, process and governance, international law, and legal frameworks),
- Cloud services (cloud services drive and are driven by consumer demand; as markets change, so do the types of services being offered),
- Cloud experiences and adoption (deployment scenarios and consumer expectations),
- Cloud and adjacent technology trends (exploring trends in the market and impacts on and influences of cloud computing),
- Cloud economics (direct and indirect costs of cloud computing on the consumer; sustainable models for providers),
- Cloud standardization and compliance (facilitating the standardization of cloud tech and test suites for compliance), and
- Cloud governance (transparency of processes, legal frameworks, and consumer monitoring and reporting).

Submissions will be subject to *IEEE Cloud Computing* magazine's peer-review process. Articles should be at most 6,000 words, with a maximum of 15 references, and should be understandable to a broad audience of people interested in cloud computing, big data, and related application areas. The writing style should be down to earth, practical, and original.

All accepted articles will be edited according to the IEEE Computer Society style guide. Submit your papers through Manuscript Central at https://mc.manuscriptcentral.com/ccm-cs.

If you have any questions, feel free to email lead editor Brian Brannon at bbrannon@computer.org.

www.computer.org/cloudcomputing

CONFERENCES
in the Palm of Your Hand

IEEE Computer Society's Conference Publishing Services (**CPS**) is now offering conference program mobile apps! Let your attendees have their conference schedule, conference information, and paper listings in the palm of their hands.

The conference program mobile app works for **Android** devices, **iPhone**, **iPad**, and the **Kindle Fire**.

For more information please contact **cps@computer.org**

IEEE computer society

CPS Conference Publishing Services

IEEE computer society

ROCK STARS OF RISK-BASED SECURITY

Learn What You Must Know About Risk Assessment and Mitigation

12 April 2016 | Washington, DC Metro Area

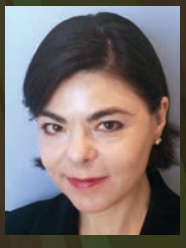
100% Security Solution? Pipedream!

Virtually every company will be hacked, and today, experts accept that a 100% security solution is not feasible. Advanced risk assessment and mitigation is the order of the day.

Rock Stars of Risk-Based Security is the must attend symposium of its kind in 2016 on this critical new reality. What attacks can you expect? How can you be prepared? On April 12, 2016 you'll learn the answers to those questions straight from the people who are driving innovation in risk-based security.

Rock Star Speakers

Scott Borg
Director (CEO) and Chief Economist, U.S. Cyber Consequences Unit



Diana Kelly
Executive Security Advisor, IBM

Jake Kouns
Chief Information Security Officer, Risk Based Security

www.computer.org/rbseast

The Fourteenth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2016

CALL FOR PARTICIPATION

Organizing Committee

General Co-Chairs
Mohan Kumar, Rochester Institute of Technology, USA
Aruna Seneviratne, DATA61,CSIRO, Australia

Program Chair
Christian Becker, Universität Mannheim, Germany

Vice Program Co-Chairs
Frank Dürr, University of Stuttgart, Germany
Jamie Payton, University of North Carolina at Charlotte, USA
Daniele Ribboni, University of Milan, Italy

Workshops Co-Chairs
Chiara Boldrini, IIT-CNR, Italy
Salil Kanhere, University of New South Wales, Australia

Steering Committee Chair
Jadwiga Indulska, The University of Queensland, Australia

Best paper award

The best paper for the prestigious **Mark Weiser Best Paper Award** will be selected on March 16, 2016 at the Best Paper Session.

Papers of particular merit will be considered for a special issue of the Elsevier journal of *Pervasive and Mobile Computing (PMC)*.

Important Dates

Early Registration:	**February 19, 2016**
Pre-Conference Workshops:	**March 14, 2016**
Main Conference:	**March 15-17, 2016**
Post-Conference Workshops:	**March 18, 2016**

IEEE PerCom, now in its fourteenth year, is established as the premier annual scholarly venue in the area of pervasive computing and communications. Pervasive computing and communications has evolved into a highly active areas of research. Research outcomes have found their way to many current commercial systems, due to the tremendous advances in a broad spectrum of technologies and topics including wireless networking, mobile and distributed computing, privacy and security, sensor systems, context modeling and reasoning, ambient intelligence, smart devices and others.

PerCom 2016 will provide a leading edge, scholarly forum for researchers, engineers, and students alike to share their state-of-the art research and developmental work in the broad areas of pervasive computing and communications. The conference will feature a diverse mixture of interactive forums: core technical sessions of high quality cutting-edge research articles; targeted workshops on exciting topics; live demonstrations of pervasive computing in action; inspiring keynote speeches; insightful panel discussions from domain experts; and posters representing emerging ideas.

This year, for the first time PerCom will be held during the summer in Sydney, Australia. We invite you to join us to enjoy a stimulating conference in sunny Sydney.

SPONSORS

IEEE Celebrating 125 Years of Engineering the Future
IEEE computer society TCPP
TCCC Technical Committee on Computer Communications
R·I·T B. Thomas Golisano College of Computing and Information Sciences
DATA 61 CSIRO
NSF
The University Of Queensland AUSTRALIA
ELSEVIER

For additional information, please visit the website www.percom.org **or contact at** percom2016@gmail.com